**Terraform is an open source infrastructure as code tool created by HashiCorp that lets you provision Google Cloud resources with declarative configuration files**   
  
**IaC configuration workflow**

**Scope**

Confirm the

resources required

for a project.

**Author**

Author the

configuration files

based on the scope.

**Initialize**

Download the

provider plugins and

initialize directory

**Plan**

View execution plan

for resources

created, modified,

or destroyed.

**Apply**

Create actual

infrastructure

resources.

What is infrastructure as code (IaC)?  
Infrastructure as code means and what problems it solves. Infrastructure as code, as the name indicates, involves managing and provisioning the cloud infrastructure as code through a configuration language. In other words, instead of clicking around a web UI or using SSH to connect to a server and manually executing commands, with IaC you can write code in files to define, provision, and manage your infrastructure. Gone are the days where a system administrator has to manually configure hundreds of servers, networks, and firewall rules by interacting with a UI. With IaC, you only need to declare the desired end state of the infrastructure, and IaC tools manage provisioning the infrastructure.  
  
**Terraform features**Multi-cloud and multi-API

Open core with enterprise support

Large community

Infrastructure provisioning

advantages   
Declarative   
Specify the desired state of infrastructure, not updates.

Code management  
Commit, version, trace, and collaborate, just like source code.

Auditable  
Compare infrastructure between desired state and current state.

Portable  
Build reusable modules across an organization.  
  
Problems that IaC can solve  
● Inability to scale rapidly: High business demand requires the rapid scaling of

IT infrastructure across industries.

● Operational bottlenecks: Due to the rapid scaling of IT infrastructure, Ops

teams need to overcome new organizational and technical bottlenecks, such

as managing infrastructure consistently in scale.

● Disconnected feedback loops: Whenever the infrastructure is changed, the

dev and Ops teams struggle to collaborate and audit changes. Both closing

the communication gap and auditing changes between the software and IT

teams are imperative for successful deployments.

● Manual errors: Increased quantity and scale has led to greater human error

with the potential for significant impacts.  
Provisioning and configuring are two terms that are sometimes misinterpreted

**Infrastructure as code**

● Used for provisioning and managing cloud

resources.   
● Example: Creating and provisioning a VM

instance.  
 ● Referring to frameworks that manipulate

Google Cloud APIs to deploy the

infrastructure.

**What is Terraform?**

A: Terraform is an open-source Infrastructure as Code (IaC) tool developed by HashiCorp. It allows users to define and provision infrastructure resources, such as virtual machines, networks, and storage, using declarative configuration files.

**Q: What is the purpose of Terraform?**

A: The primary purpose of Terraform is to automate the process of provisioning and managing infrastructure resources in a cloud environment. It enables users to define infrastructure configurations in code, version control them, and apply changes predictably and consistently across different environments.

**Q: How does Terraform work?**

A: Terraform works by reading configuration files (written in HashiCorp Configuration Language or HCL) that define the desired state of infrastructure resources. It then compares this desired state with the current state of the target environment and generates an execution plan to bring the infrastructure into the desired state.

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

A: Some key features of Terraform include:

Infrastructure as Code (IaC)

Declarative Configuration

Resource Graph

Plan & Apply Workflow

State Management

Provider Ecosystem

**Q: What is the difference between Terraform plan and apply?**

A: Terraform plan generates an execution plan that shows what actions Terraform will take to change the infrastructure to match the desired state specified in the configuration files. Terraform apply then executes the actions specified in the plan to make the changes to the infrastructure.

**Q: What is Terraform state?**

A: Terraform state is a JSON file that stores the current state of managed infrastructure resources. It contains information about the resources provisioned by Terraform, their attributes, dependencies, and metadata. Terraform uses state files to track changes and manage the lifecycle of resources.

**Q: How do you manage secrets in Terraform?**

A: Secrets in Terraform can be managed using techniques such as environment variables, Terraform input variables marked as sensitive, external secret management systems (e.g., HashiCorp Vault, AWS Secrets Manager), or encryption at rest for sensitive data stored in state files.

**Q: What are Terraform providers?**

A: Terraform providers are plugins that enable Terraform to interact with specific cloud providers, services, or infrastructure platforms. Providers translate Terraform configuration into API calls to provision and manage resources in the target environment.

**Q: How do you organize Terraform configurations?**

A: Terraform configurations can be organized into modules, which are reusable units of infrastructure configurations. Modules encapsulate related resources and allow for modular, maintainable, and scalable infrastructure codebases.

**Q: What is the difference between Terraform state and Terraform configuration?**

A: Terraform configuration refers to the declarative configuration files (written in HCL) that define the desired state of infrastructure resources. Terraform state, on the other hand, refers to the JSON file that stores the current state of managed infrastructure resources and tracks changes made by Terraform.

[

**What is the purpose of Terraform modules?**

A: Terraform modules are reusable units of Terraform configuration that encapsulate related resources and configuration logic. They allow you to abstract and modularize infrastructure code, promote code reusability, and simplify the management of complex infrastructure deployments.

**Q: How do you manage multiple environments (e.g., development, staging, production) with Terraform?**

A: You can manage multiple environments with Terraform by using separate configuration files, variables, and state files for each environment. Additionally, you can leverage Terraform workspaces or environment-specific variable files to parameterize configurations and differentiate between environments.

**Q: What are Terraform workspaces?**

A: Terraform workspaces are a feature that allows you to manage multiple distinct sets of state files from a single configuration. Each workspace maintains its own state, enabling you to manage different environments (e.g., development, staging, production) or configurations independently within the same Terraform project.

**Q: How do you handle dependencies between resources in Terraform?**

A: Terraform automatically manages dependencies between resources based on their defined relationships in the configuration. You can specify dependencies explicitly using the depends\_on attribute or implicitly through resource references, and Terraform will ensure that resources are created and destroyed in the correct order.

**Q: What is the difference between Terraform apply and Terraform refresh?**

A: Terraform apply applies the changes specified in the Terraform configuration to the target environment, creating, updating, or deleting resources as necessary. Terraform refresh, on the other hand, reconciles the state of managed resources in the target environment with the state defined in the Terraform configuration, updating the state file without making any changes to resources.

**Q: How do you manage third-party Terraform modules?**

A: You can manage third-party Terraform modules using various methods, including module registries (e.g., Terraform Registry), version control systems (e.g., Git), or by hosting modules internally in your organization. Terraform makes it easy to reference and use modules from external sources in your configurations.

**Q: What is the recommended approach for testing Terraform configurations?**

A: The recommended approach for testing Terraform configurations includes using automated testing frameworks, infrastructure testing tools (e.g., Terratest), and integration testing environments to validate configuration changes, detect errors, and ensure the reliability and correctness of infrastructure deployments.

Q**: How do you manage state file locking in Terraform?**

A: Terraform automatically handles state file locking to prevent concurrent access and potential corruption of state files. By default, Terraform uses a local locking mechanism, but you can configure remote backends (e.g., Terraform Cloud, AWS S3) for distributed locking and improved concurrency control in collaborative environments.

Q: **What is the purpose of Terraform providers?**

A: Terraform providers are responsible for managing the lifecycle of resources within a specific infrastructure platform or service provider. They translate Terraform configuration into API calls required to provision, update, and delete resources in the target environment.

Q**: How do you handle Terraform state management in a team environment?**

A: In a team environment, Terraform state management can be handled by using remote state backends, such as Terraform Cloud, AWS S3, or Azure Blob Storage. These backends provide centralized storage, locking mechanisms, and access controls for Terraform state files, facilitating collaboration and concurrency control among team members.

Q: **What are Terraform data sources?**

A: Terraform data sources allow you to query and retrieve information from external systems or resources at runtime. They provide a mechanism for dynamically populating Terraform configurations with data from sources such as AWS, Azure, Google Cloud, or external APIs.

Q**: How do you handle secrets or sensitive data in Terraform configurations?**

A: Secrets or sensitive data in Terraform configurations can be managed using techniques such as environment variables, Terraform input variables marked as sensitive, or integration with external secret management systems (e.g., HashiCorp Vault, AWS Secrets Manager). Encryption at rest for state files can also help protect sensitive information stored within Terraform.

Q**: What is the recommended approach for managing infrastructure changes in Terraform**?

A: The recommended approach for managing infrastructure changes in Terraform involves using version control systems (e.g., Git) to track changes to configuration files, following best practices for code review and collaboration, and leveraging Terraform workflows such as plan, apply, and versioning to ensure consistency and reliability of deployments.

Q: **How do you manage Terraform state across multiple projects or environments?**

A: Terraform allows you to manage state across multiple projects or environments by using separate state files and workspaces for each project or environment. You can organize state files in remote backends, namespace them by project or environment, and use Terraform workspaces to switch between different sets of state.

Q**: What is the difference between Terraform provisioners and modules?**

A: Terraform provisioners are used to execute scripts or commands on local or remote machines during resource creation or destruction. They are typically used for tasks such as bootstrapping, configuration management, or post-deployment tasks. Modules, on the other hand, are reusable units of Terraform configuration that encapsulate related resources and configuration logic for easier management and reuse.

Q: **How do you handle Terraform configuration drift and divergence?**

A: Terraform configuration drift and divergence can be managed by regularly applying Terraform plan and Terraform apply to reconcile the desired state with the current state of infrastructure resources. Implementing infrastructure testing, monitoring, and automated workflows can also help detect and mitigate configuration drift or divergence over time.

**what is Terraform and what problem does it solve?**

Terraform is an open-source Infrastructure as Code (IaC) tool used for provisioning and managing infrastructure resources in a declarative manner. It helps automate the process of setting up and configuring infrastructure components such as servers, networks, and storage.

**How does Terraform differ from traditional infrastructure provisioning methods?**

Unlike traditional manual provisioning methods, Terraform allows you to define infrastructure configurations in code, enabling automation, repeatability, and version control of infrastructure deployments.

**What are the key components of a Terraform configuration?**

A Terraform configuration consists of resources, providers, variables, outputs, and optionally modules. Resources define the infrastructure components to be provisioned, providers specify the target environment, variables allow for parameterization, and outputs provide useful information after provisioning.

**What is a Terraform provider and how does it work?**

A Terraform provider is responsible for managing the lifecycle of resources within a specific infrastructure platform or service provider, such as AWS, Azure, or Google Cloud. It interacts with the provider's API to create, update, and delete resources based on the Terraform configuration.

**How do you initialize a Terraform configuration?**

To initialize a Terraform configuration, you run the terraform init command in the directory containing your Terraform files. This command downloads provider plugins and initializes the working directory with any necessary configuration.

**What is a Terraform state file and why is it important?**

The Terraform state file (terraform.tfstate) is a JSON file that stores the current state of managed infrastructure resources. It keeps track of the relationships between resources, their attributes, dependencies, and metadata. The state file is essential for Terraform to manage and update resources accurately.

**How do you apply changes to infrastructure using Terraform?**

To apply changes to infrastructure using Terraform, you run the terraform apply command in the directory containing your Terraform configuration files. Terraform generates an execution plan, prompts for confirmation, and then applies the changes to the target environment.

**What is the difference between Terraform plan and Terraform apply?**

Terraform plan generates an execution plan that shows the proposed changes to infrastructure resources without actually applying them. Terraform apply, on the other hand, executes the actions specified in the plan and applies the changes to the target environment.

**How do you manage variables in Terraform configurations?**

Variables in Terraform configurations can be defined using the variable block and referenced throughout the configuration files. You can specify variable values directly in the configuration, pass them via command-line arguments, or load them from environment variables.

**What is the purpose of Terraform modules?**

Terraform modules are reusable units of Terraform configuration that encapsulate related resources and configuration logic. They allow you to abstract and modularize infrastructure code, promote code reusability, and simplify the management of complex infrastructure deployments.

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

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.

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.

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)

What are modules in Terraform?

A jug for many resources 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.

**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.

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

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.

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.

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

1. Ab output variable is defined in resource configuration.
2. Declare the output variable of module\_A.
3. Create a file variable.tf for module B.
4. Establish the input variable inside this file having the same name as the key defined in module\_B.
5. Replicate the process for making variable available to other modules

What is a Terraform state file and why is it important?

The Terraform state file (terraform.tfstate) is a JSON file that stores the current state of managed infrastructure resources. It keeps track of the relationships between resources, their attributes, dependencies, and metadata. The state file is essential for Terraform to manage and update resources accurately.

How do you apply changes to infrastructure using Terraform?

To apply changes to infrastructure using Terraform, you run the terraform apply command in the directory containing your Terraform configuration files. Terraform generates an execution plan, prompts for confirmation, and then applies the changes to the target environment.

What is the purpose of Terraform modules?

Terraform modules are reusable units of Terraform configuration that encapsulate related resources and configuration logic. They allow you to abstract and modularize infrastructure code, promote code reusability, and simplify the management of complex infrastructure deployments.

What is the difference between Terraform provisioners and modules?

A: Terraform provisioners are used to execute scripts or commands on local or remote machines during resource creation or destruction. They are typically used for tasks such as bootstrapping, configuration management, or post-deployment tasks. Modules, on the other hand, are reusable units of Terraform configuration that encapsulate related resources and configuration logic for easier management and reuse.

How do you manage Terraform state across multiple projects or environments?

A: Terraform allows you to manage state across multiple projects or environments by using separate state files and workspaces for each project or environment. You can organize state files in remote backends, namespace them by project or environment, and use Terraform workspaces to switch between different sets of state

How do you handle secrets or sensitive data in Terraform configurations?

A: Secrets or sensitive data in Terraform configurations can be managed using techniques such as environment variables, Terraform input variables marked as sensitive, or integration with external secret management systems (e.g., HashiCorp Vault, AWS Secrets Manager). Encryption at rest for state files can also help protect sensitive information stored within Terraform.

How do you manage state file locking in Terraform?

A: Terraform automatically handles state file locking to prevent concurrent access and potential corruption of state files. By default, Terraform uses a local locking mechanism, but you can configure remote backends (e.g., Terraform Cloud, AWS S3) for distributed locking and improved concurrency control in collaborative environments.

What is the difference between Terraform apply and Terraform refresh?

A: Terraform apply applies the changes specified in the Terraform configuration to the target environment, creating, updating, or deleting resources as necessary. Terraform refresh, on the other hand, reconciles the state of managed resources in the target environment with the state defined in the Terraform configuration, updating the state file without making any changes to resources.

How do you handle dependencies between resources in Terraform?

A: Terraform automatically manages dependencies between resources based on their defined relationships in the configuration. You can specify dependencies explicitly using the depends\_on attribute or implicitly through resource references, and Terraform will ensure that resources are created and destroyed in the correct order.

What are Terraform workspaces?

A: Terraform workspaces are a feature that allows you to manage multiple distinct sets of state files from a single configuration. Each workspace maintains its own state, enabling you to manage different environments (e.g., development, staging, production) or configurations independently within the same Terraform project.   
--------------------------------------------------------------------------------------------------------------------------------------------------------------  
  
**How to Store Sensitive Data in Terraform?**

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

**How to lock Terraform module versions?**   
We can use the ‘version’ attribute in module of the Terraform configuration file

A proven way of locking Terraform module version is using the Terraform module registry as a source.. As the Github repository is being used as a source, we need to specify versions, branch, and query

**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/).  
 **Does Terraform support multi-provider deployments?**

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

**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

**Can two persons at a time run terraform apply simultaneously.? Yes or No. Explain**No, two persons should not run terraform apply simultaneously on the same Terraform configuration in the same environment. Running terraform apply simultaneously by multiple users can lead to conflicts and race conditions, potentially resulting in inconsistent or unpredictable infrastructure changes.  
  
**How to add an existing resource in terraform**

Import the Resource:

Use the terraform import command to import the existing resource into your Terraform state. The command syntax is:

 terraform import RESOURCE\_TYPE.NAME ID

**What are Built-in functions in terraform?**   
Terraform language includes a number of built-in functions that you can call from within expressions to transform and combine values

Numeric Functions.

String Functions Collection Functions

Encoding Functions

Filesystem Functions

Date and Time Functions

Hash and Crypto Functions

IP Network Functions

**What do Terraform Provisioners do?**   
Terraform provisioners are used to execute scripts or commands on local or remote machines during resource creation or destruction.

What are the ways to authenticate to GCP from terraform?  
json file and access tokens  
  
**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.

**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.

**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.

**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.

**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.

**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

How can we create multiple resources in terraform ?   
**for each resource we can configer or we can use modules**What functions or blocks you'll use to avoid repeatition?  
dynamic block  
When would you use sentinel policies in Terraform  
sentinel policy are rules whch are enforce on terraform run to validate that the plan and corresponding resource are in compliance with company policies  
What is the hard & soft mandatory sentinel enforcement?  
Hard mandatory enforcement is suitable for critical policies where compliance must be strictly enforced, and any deviation is unacceptable.

Soft mandatory enforcement is useful for policies where some flexibility is allowed, but violations should still be brought to attention for review or corrective action

What does Terraform’s external data block represent?  
Terraform, the external data block is used to fetch data from external sources and use it within your Terraform configuration. It allows you to integrate Terraform with external systems or scripts, enabling you to incorporate dynamic or computed data that cannot be directly managed by Terraform itself.

How can we export data from one module to another?  
In Terraform, you can export data from one module and consume it in another module using output variables. Output variables allow you to define values that should be exposed to other parts of your Terraform configuration, including other modules

How can two people using the Terraform cloud can create two different sets of infrastructure using the same working directory?  
we have to use github from git hub we can clone to local repo so that we can perform   
Use Workspaces (Optional):If you're using Terraform Cloud/Enterprise, utilize workspaces to create separate environments for each user. This allows each user to work independently without interfering with each other's changes

How do you ensure your team uses certain version in terraform  
in provider block we have version terraform lock file we can see the version   
How can you use the same provider in Terraform with different configurations?

terraform, you can use the same provider with different configurations by defining multiple provider blocks, each with its own set of configuration options. This allows you to interact with the same provider (e.g., AWS, Azure, Google Cloud) multiple times within your Terraform configuration, each time with different settings. Here's how you can do it:

# Provider configuration 1

provider "gcp" {

  region = "us-west-1"

  access\_key = "ACCESS\_KEY\_1"

  secret\_key = "SECRET\_KEY\_1"

}

# Provider configuration 2

provider "gcp" {

  region = "us-east-1"

  access\_key = "ACCESS\_KEY\_2"

  secret\_key = "SECRET\_KEY\_2"

}

# Define resources using provider 1

resource "g\*\*\*\_c\*\*\*\_instace" "instance\_us\_west" {

  # Configuration for resource using provider 1

}

# Define resources using provider 2

resource ""g\*\*\*\_c\*\*\*\_instace " "instance\_us\_east" {

  # Configuration for resource using provider 2

}