Top 60 Terraform Interview Questions and Answers

by FOSS TechNix

In this article we are going to cover Terraform Interview Questions and Answers | Terraform Scenario based Interview Questions and Answers | Terraform Troubleshooting Interview Questions and Answers | Terraform Interview Questions and Answers for DevOps Engineer | Terraform advanced Interview Questions and Answers

What is Infra as a Code?

- Infrastructure as code codifies and manages underlying IT infrastructure as software.
- Traditionally managing IT infrastructure was a manual and difficult process, person had to physically install and configure servers. It was a expensive, slow and inconsistent.
- IaaC enables the devs or operation teams to automatically install, configure & manage infra.

What are advantages of Infrastructure as code?

Infrastructure as Code (IaC) is a software engineering approach that allows you to manage and provision infrastructure resources using code and automation. There are several advantages to using IaC:

1. **Automation**: IaC enables the automation of infrastructure provisioning and management. This reduces manual and error-prone tasks, streamlines processes, and allows for consistent and reproducible deployments.
2. **Version Control**: IaC code can be version-controlled using tools like Git. This provides a history of changes, facilitates collaboration, and allows you to roll back to previous configurations if issues arise.
3. **Consistency**: IaC ensures that your infrastructure is consistent across different environments, such as development, testing, and production. This reduces configuration drift and minimizes deployment-related problems.
4. **Scalability**: IaC makes it easier to scale your infrastructure up or down as needed. You can adapt to changing workloads and requirements by adjusting code rather than manually configuring resources.
5. **Speed and Efficiency**: With IaC, you can provision and modify infrastructure quickly. This is particularly beneficial in agile and DevOps environments where rapid development and deployment are essential.
6. **Self-Documenting**: IaC code serves as documentation for your infrastructure. It provides a clear and concise description of the resources and their configurations, making it easier for teams to understand and maintain the infrastructure.
7. **Reusability**: You can reuse IaC code for similar components or services, reducing duplication of effort and saving time.

8. **Testing**: IaC code can be tested and validated before deployment. This allows you to catch potential issues early, improving the reliability of your infrastructure.
9. **Collaboration**: IaC promotes collaboration between development and operations teams. Developers can define infrastructure requirements in code, and operations teams can manage the deployment and maintenance.
10. **Security**: IaC can help enforce security best practices by codifying security configurations and policies. This reduces the risk of misconfigurations and security vulnerabilities.
11. **Cost Control**: IaC can help you manage and optimize infrastructure costs by allowing you to control resource provisioning based on demand and automatically clean up unused resources.
12. **Auditing and Compliance**: IaC facilitates auditing and compliance efforts by providing a detailed record of infrastructure changes and configurations. This is valuable for regulatory and compliance requirements.
13. **Disaster Recovery**: With IaC, you can quickly rebuild your infrastructure in case of failures or disasters, reducing downtime and data loss.
14. **Multi-Cloud and Hybrid Cloud Support**: IaC is often cloud-agnostic, meaning you can use it to manage infrastructure across various cloud providers and on-premises environments.
15. **Easier Replication**: You can replicate entire environments or infrastructure setups in different regions or for different projects by reusing IaC code.

In summary, Infrastructure as Code offers numerous benefits, including automation, consistency, scalability, speed, efficiency, and improved collaboration. It has become a fundamental practice in modern IT and DevOps operations, helping organizations streamline their infrastructure management and deployment processes.

What is Infrastructure Provisioning?

- It’s the automated process of setting up and configuring IT resources, such as servers, networks, and storage, to support applications and services. This ensures rapid, consistent, and scalable resource deployment, reducing manual effort and errors.
- Infrastructure provisioning allows for easy scaling of resources, ensuring that you can quickly adapt to changing demands while optimizing resource utilization for cost-efficiency.

Terraform Interview Questions and Answers

Why choose Terraform for infrastructure provisioning?

- Multi-Cloud: Provision resources across different cloud providers.
- Declarative: Describe what you want; Terraform handles the “how.”
- Code Reusability: Modular configurations for efficient development.
- Community Support: Vast community and extensive provider ecosystem.

What is Terraform, and how does it differ from other infrastructure-as-code tools like Ansible or Puppet?

Terraform is an open-source, cloud-agnostic tool that helps users build, change, and version infrastructure. It's used primarily by DevOps teams to automate infrastructure tasks, such as provisioning cloud resources.

Terraform was created by HashiCorp in 2014. It's written in the Go language. Users define and provide data center infrastructure using a declarative configuration language known as HashiCorp Configuration Language (HCL), or optionally JSON.

What are Benefits of Terraform?

- **CollaborationTeams:** can define and manage infrastructure using version control, which makes it easier for multiple people to collaborate and work on the same codebase.
- **Full-stack deployment:** You can have Amazon instances running Kubernetes containers with your workloads and manage the whole system from one tool.
- **Management of external resources:** Terraform manages external resources (network appliances, software as a service, platform as a service, etc.) with "providers".
- **Tracking resource changes:** Terraform's state allows you to track resource changes throughout your deployments.
- **Reducing the amount of code:** You can create a module and reference it multiple times, passing different parameters.
- **Drift detection:** The drift detection feature in Terraform Cloud is designed to identify and manage configuration drift in your infrastructure deployments.
- **Security practices and governance:** With Terraform Cloud, you get Sentinel and OPA policies to enforce security practices and governance throughout your workflow

Why Terraform?

Think of Terraform as a magical blueprint for building and managing the infrastructure that supports your software applications. Here's why DevOps engineers need Terraform in simple terms

- **Infrastructure Wizardry:** Terraform is like a wizard's spellbook for creating and managing servers, databases, and other infrastructure components. It allows DevOps engineers to describe their desired infrastructure in a simple, human-readable language.
- **Consistency Enforcer:** Just as a recipe ensures that you make the same delicious dish every time, Terraform ensures that your infrastructure is consistent. It creates and configures resources exactly as you specify, reducing errors and surprises.
- **Efficiency Booster:** Terraform automates the process of creating, modifying, and destroying infrastructure. It's like having an army of helpers who can set up servers and services in minutes instead of hours or days.
- **Multi-Cloud Harmony:** Terraform is cloud-agnostic, meaning it works with different cloud providers like AWS, Azure, and Google Cloud. It lets DevOps engineers manage infrastructure across multiple clouds with a single set of commands.
- **Version Control Friend:** Just as you save different versions of your document, Terraform lets you version-control your infrastructure. This means you can track changes over time and easily roll back to previous configurations if needed.

- **Collaboration Facilitator:** Terraform enables teamwork. Multiple DevOps engineers can work on the same infrastructure code, and Terraform helps merge their changes and maintain consistency.
- **Risk Minimizer:** Like a safety net, Terraform can help recover from disasters. If something goes wrong, you can use your Terraform code to rebuild your infrastructure exactly as it was before.
- **Security Sentinel:** Terraform helps ensure that your infrastructure is configured securely. It can enforce security policies and best practices, reducing vulnerabilities. It makes sure everything is built correctly, saves time, and keeps everything organized and consistent, whether you're working in one cloud or many.

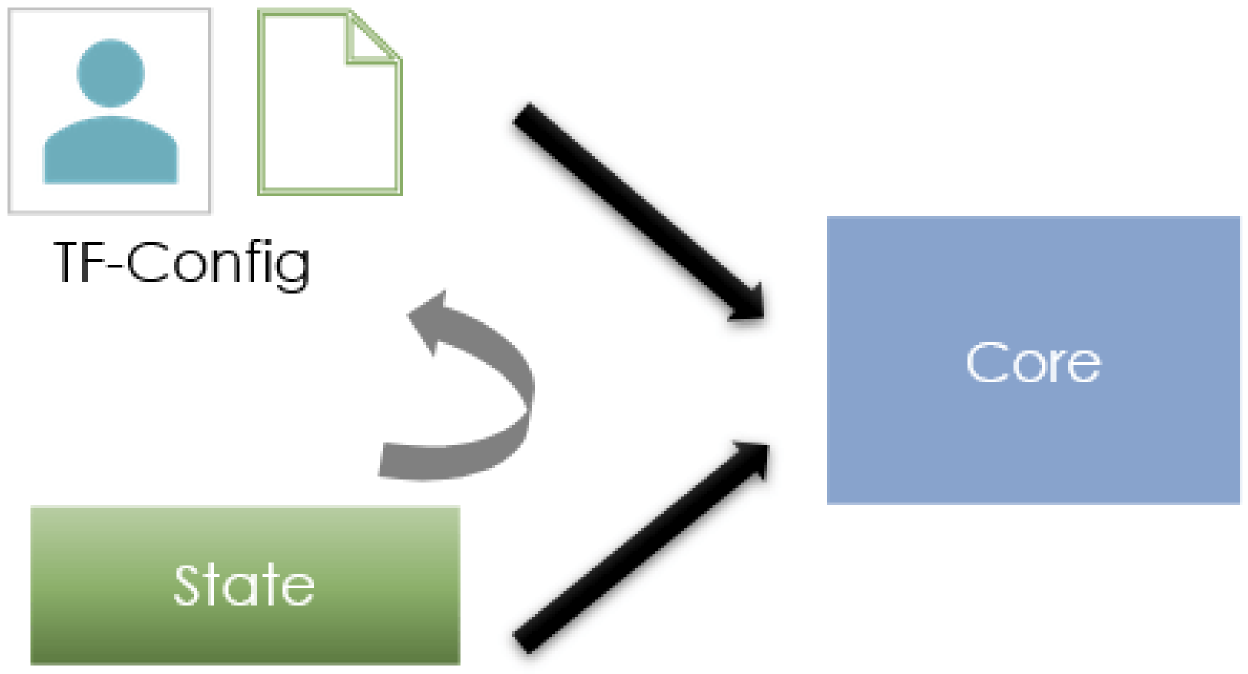
What are **Difference Between Terraform and other tools like Ansible, Chef**.

Aspect	Terraform	Ansible	Chef
Purpose	Infrastructure provisioning and management (IaC).	Configuration management and automation.	Configuration management and automation.
Declarative vs. Imperative	Declarative: Defines the desired infrastructure state.	Declarative: Defines desired system state in YAML.	Imperative: Defines tasks should be executed step by s
Domain	Infrastructure (cloud, on-premises) provisioning and management.	Server, network, and application configuration.	Server and applica configuration.
State Management	Maintains a state file to track infrastructure state.	Stateless: Does not track system state.	Stateless: Does no system state.
Language	HashiCorp Configuration Language (HCL).	YAML for playbooks.	Ruby for recipes.
Agent/Agentless	Agentless: No agents on managed servers.	Agentless: No agents on managed servers.	Agent-based: Requ Chef agent.
Ecosystem	Extensible using providers for various infrastructure services.	Extensive library of modules and roles for various use cases.	Cookbook commur sharing recipes.
Orchestration	Primarily used for infrastructure provisioning and changes.	Supports both configuration management and application deployment.	Supports configura management and deployment.
Community Support	Strong community support and official providers for many services.	Large and active community with extensive roles and modules available.	Active community w cookbook repositor

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

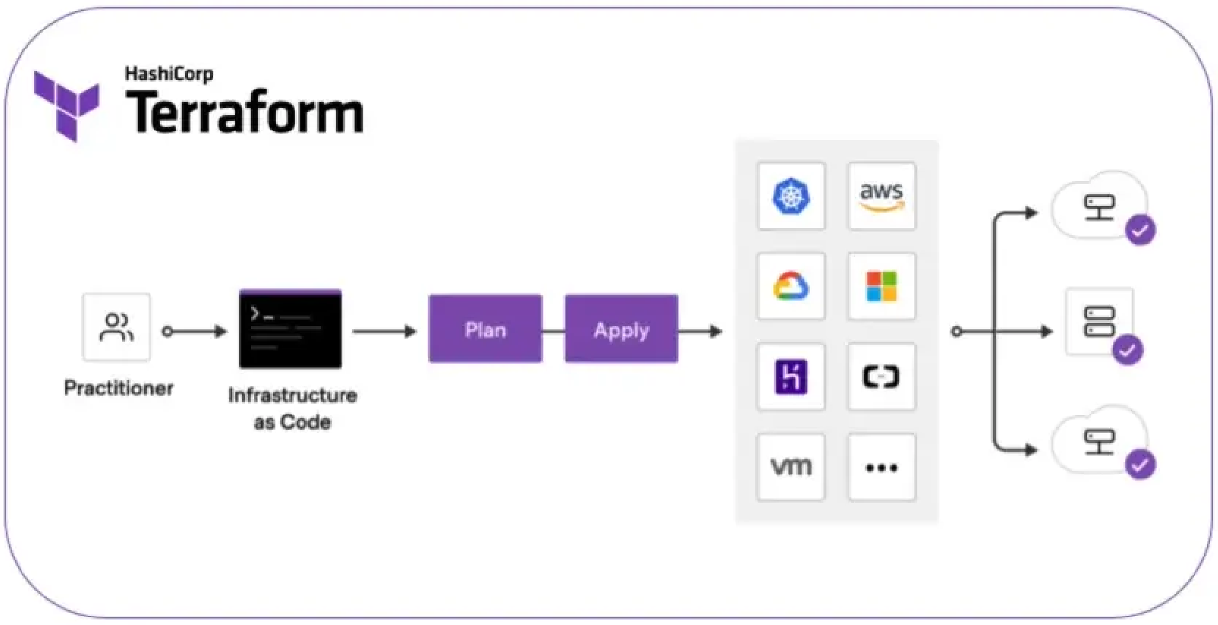
A Terraform provider is a plugin that allows Terraform to interact with a specific cloud or service API. Providers are responsible for authenticating with the service, creating and managing resources, and handling state management. Terraform provides a variety of built-in providers for popular cloud platforms like AWS, Azure, and GCP, as well as other services like Docker and Kubernetes. You can also develop custom providers for services not covered by the built-in ones.

Explain Terraform Architecture?



- Terraform has two main components :-
  Core and State
- Core uses two input source :-
  TF-Config and the state
- Core takes input and compares current state and the desired state (TF-Config file) and figures out what need to be done to get the desired state.
- The core creates execution plan.
- Finally the core execute the plan with the providers (AWS, AZURE etc.).

Can you please Explain Terraform Workflow?



To establish an effective Terraform workflow, you can follow these general steps:

1. **Install Terraform**: Ensure you have Terraform installed on your local machine. You can download it from the official website or use a package manager if available for your platform.
2. **Create a Terraform Configuration**: Start by creating a directory for your Terraform project. In this directory, create a `.tf` or `.tf.json` file to define your infrastructure. This is where you'll declare the resources you want to provision, their configurations, and any dependencies.
3. **Initialize the Working Directory**: Run the `terraform init` command in your project directory. This command initializes your working directory by downloading the necessary provider plugins and modules specified in your configuration.
4. **Write Terraform Configuration**: Define your infrastructure using Terraform configuration language (HCL). Create resource blocks, variables, and data sources as needed. You can also organize your code into modules for reusability.
5. **Plan Your Changes**: Run `terraform plan` to preview the changes Terraform will make to your infrastructure. This step helps you validate your configuration and understand what Terraform intends to do.
6. **Apply Changes**: After reviewing the plan and ensuring it's what you want, run `terraform apply` to provision or modify the infrastructure. Terraform will prompt you for confirmation before making changes.
7. **Maintain State**: Terraform keeps track of the state of your infrastructure in a state file. Ensure that you manage this state file securely and consistently, as it's crucial for tracking the actual state of your infrastructure.
8. **Use Variables and Input Data**: Utilize Terraform variables and data sources to parameterize your configurations and make them more flexible.
9. **Version Control**: Store your Terraform configuration and related files in a version control system like Git. This enables collaboration, change tracking, and history management.
10. **Use Modules**: Organize your Terraform code into reusable modules to promote consistency and reduce duplication in your infrastructure definitions.

11. **Automate Workflows**: Integrate Terraform into your CI/CD pipeline to automate the provisioning and management of your infrastructure.
12. **Test and Validate**: Implement automated tests to validate your Terraform configurations, ensuring that they work as expected and remain compliant.
13. **Monitor and Maintain**: Regularly monitor your infrastructure for changes and updates, and apply changes as needed. Terraform can help you evolve your infrastructure over time.
14. **Destroy Resources** (with caution): Use `terraform destroy` to remove infrastructure resources when they are no longer needed. Be cautious when doing this in production to avoid unintended data loss.
15. **Document Your Infrastructure**: Maintain documentation for your Terraform configurations and infrastructure to aid in understanding, troubleshooting, and onboarding new team members.
16. **Security and Access Management**: Implement proper security practices, including IAM roles and policies, to ensure that only authorized personnel can make changes to your infrastructure.
17. **Backup and Disaster Recovery**: Have a strategy in place for data backup and disaster recovery, as Terraform is primarily focused on provisioning resources, not managing data.

Remember that Terraform is not a one-size-fits-all tool, and your workflow may need to be adjusted to suit the specific needs of your project and organization. It’s essential to continually refine and adapt your Terraform workflow to meet evolving infrastructure requirements.

Explain the difference between a Terraform resource and a data source

In Terraform, a resource is an entity that you want to create, modify, or delete, such as a virtual machine, network, or database. A data source, on the other hand, is used to fetch information about existing resources. Data sources are read-only and are commonly used to retrieve details about resources that you want to reference in your Terraform configuration, like obtaining the IP address of an existing instance.

What is Terraform state, and why is it important?

Terraform state is a representation of the resources managed by Terraform. It includes information about resource IDs, attributes, and the relationships between resources. State files are crucial for Terraform to understand the current state of your infrastructure and track changes during plan and apply operations. Storing and managing the state correctly is essential for collaboration and preventing conflicts in a team environment.

Explain the difference between the `terraform plan` and `terraform apply` commands.

The `terraform plan` command is used to preview the changes that Terraform will make to your infrastructure based on your configuration. It provides a detailed summary of what resources will be created, updated, or destroyed. The `terraform apply` command, on the other hand, is used to execute the changes outlined in the plan and actually create or modify resources in your infrastructure.

How do you handle sensitive data, such as API keys or passwords, in Terraform configurations?

Sensitive data should never be stored in plain text within Terraform configurations. Instead, you can use Terraform variables and secrets management solutions like HashiCorp Vault, AWS Secrets Manager, or Azure Key Vault to securely store and retrieve sensitive information. You can reference these variables in your configuration without exposing the actual values.

Explain the concept of remote state in Terraform

Remote state is the practice of storing Terraform state files in a remote, centralized location rather than on a local machine. This approach enables collaboration and helps prevent state file conflicts. Popular remote state backends include Amazon S3, Azure Blob Storage, and HashiCorp Terraform Cloud. Terraform commands like `terraform init`, `terraform plan`, and `terraform apply` can be configured to use a remote state backend.

What is the difference between Terraform and Terragrunt?

Terragrunt is a wrapper around Terraform that simplifies and enhances the management of Terraform configurations. It helps manage remote state, enforce best practices, and maintain infrastructure code across multiple environments. Terragrunt is particularly useful for large and complex Terraform projects, as it provides features for DRY (Don’t Repeat Yourself) configuration and makes it easier to manage multiple environments.

How can you create reusable modules in Terraform, and why are they important?

Terraform modules are a way to encapsulate and reuse infrastructure code. Modules allow you to define, parameterize, and version infrastructure components so that they can be easily reused in different projects. Reusable modules improve code organization, maintainability, and reduce duplication across different parts of your infrastructure.

What is the “Terraform Apply” workflow, and how does it relate to the Infrastructure as Code (IaC) philosophy?

The “Terraform Apply” workflow is a fundamental part of Terraform and IaC. It aligns with the IaC philosophy by allowing you to define your infrastructure in code, version it, and apply changes consistently. The workflow involves creating or updating infrastructure resources by running `terraform apply` after defining and reviewing changes with `terraform plan`. This ensures that infrastructure is provisioned and modified in a controlled and repeatable manner.

Remember to tailor your answers to your specific experience and expertise, and be prepared to provide examples from your previous work with Terraform. Additionally, you may encounter more advanced questions depending on the specific requirements of the DevOps engineer role you’re interviewing for.

Terraform Scenario Based Interview Questions

1. what components did you create using Terraform?
2. How do changes in already created services in AWS using Terraform?
3. what tfstate contains and How do you keep it safe?
4. what are proviosioners in terraform?
5. How to take action if you lose tfstate file?
6. what are the features of terraform?
7. Terraform validate command is used for the?
8. what does terraform init command do?
9. How do restrict users not to write at the same time in the tfstate file?
10. what is the lifecycle block in tf?
11. what you will lose anisible or terraform and why?
12. How to destroy a specific resource?
13. How to keep AWS credentials safe while using tf?
14. what are modules in Terraform? & types of modules?
15. what is the remote backend in Terraform?
16. what are commands used in Terraform will you elaborate?
17. In How many ways, we can provide the variable values in terraform?

Conclusion:

We have covered Terraform Interview Questions and Answers | Terraform Scenario based Interview Questions and Answers | Terraform Troubleshooting Interview Questions and Answers | Terraform Interview Questions and Answers for DevOps Engineer | Terraform advanced Interview Questions and Answers