Terraform DevOps Interview Questions and Answers

Trivial Questions

1. **What is Terraform, and how does it work?**
   - **Answer:** Terraform is an Infrastructure as Code (IaC) tool that allows you to define and provision infrastructure using declarative configuration files. It uses a `plan` and `apply` workflow to create, update, or delete resources.
2. **What is a Terraform provider?**
   - **Answer:** A provider is a plugin that interacts with APIs of cloud platforms (e.g., AWS, Azure, GCP) or other services. It defines resource types and data sources for managing infrastructure.
3. **What is a Terraform state file, and why is it important?**
   - **Answer:** The state file (`terraform.tfstate`) stores the current state of your infrastructure. It maps Terraform configuration to real-world resources and is used to track changes and dependencies.
4. **What is the difference between `terraform plan` and `terraform apply` ?**
   - **Answer:**
     - `terraform plan` generates an execution plan showing what changes Terraform will make.
     - `terraform apply` executes the plan and applies the changes to the infrastructure.
5. **What are Terraform modules, and why are they useful?**
   - **Answer:** Modules are reusable, encapsulated sets of Terraform configurations. They help organize code, promote reusability, and simplify complex infrastructure management.
6. **What is the purpose of `terraform init` ?**
   - **Answer:** `terraform init` initializes a working directory by downloading providers, modules, and setting up the backend for storing the state file.
7. **What is a Terraform workspace?**
   - **Answer:** Workspaces allow you to manage multiple environments (e.g., dev, staging, prod) within the same Terraform configuration. Each workspace has its own state file.
8. **What is the difference between `local` and `remote` state in Terraform?**
   - **Answer:**
     - **Local state** is stored on the local filesystem.
     - **Remote state** is stored in a shared backend (e.g., S3, Terraform Cloud) for team collaboration and state locking.
9. **What is Terraform state locking, and why is it important?**
   - **Answer:** State locking prevents multiple users from making changes to the state file simultaneously, avoiding conflicts and corruption. It is enforced when using remote backends.
10. **What is the purpose of `terraform destroy` ?**
    - **Answer:** `terraform destroy` removes all resources managed by the current Terraform configuration.

Scenario-Based Questions

1. **You need to manage infrastructure for multiple environments (dev, staging, prod) using Terraform. How would you approach this?**
   - **Answer:**
     - Use **Terraform workspaces** to manage separate state files for each environment.
     - Use **input variables** to customize configurations for each environment.
     - Store the state file in a **remote backend** (e.g., S3) for collaboration and state locking.
2. **Your team is working on a large Terraform project. How would you organize the code for better maintainability?**
   - **Answer:**
     - Use **modules** to encapsulate reusable components (e.g., VPC, EC2 instances).
     - Separate environments into different directories or workspaces.
     - Use **variables** and **outputs** to make configurations dynamic and reusable.
     - Store the state file in a **remote backend** for team collaboration.
3. **You accidentally deleted a resource manually in the cloud console. How would Terraform handle this situation?**
   - **Answer:**
     - Terraform will detect the drift between the state file and the actual infrastructure during the next `terraform plan` .
     - Running `terraform apply` will recreate the deleted resource to match the desired state defined in the configuration.
4. **You want to share a Terraform module across multiple projects. How would you do this?**
   - **Answer:**
     - Publish the module to a **Terraform Registry** (public or private).
     - Alternatively, store the module in a **version-controlled repository** (e.g., GitHub) and reference it using the `source` argument in the module block.
5. **Your Terraform state file has become too large and difficult to manage. How would you optimize it?**

- **Answer:**
  - Use **modules** to break down the configuration into smaller, reusable components.
  - Use the `terraform state rm` command to remove unnecessary resources from the state file.
  - Consider using **partial configuration** or `-target` for specific resource management.

6. **You need to provision resources across multiple cloud providers (e.g., AWS and Azure). How would you do this in Terraform?**
   - **Answer:**
     - Use **multiple providers** in the same configuration.
     - Define provider blocks for each cloud (e.g., `aws` and `azurerm`).
     - Use provider-specific resources and data sources to manage infrastructure.
7. **Your team is experiencing state file conflicts. How would you resolve this issue?**
   - **Answer:**
     - Use a **remote backend** (e.g., S3 with DynamoDB for state locking) to store the state file.
     - Ensure team members pull the latest state file before making changes.
     - Use `terraform refresh` to sync the state file with the actual infrastructure.
8. **You want to enforce policies on Terraform configurations (e.g., restrict instance types). How would you achieve this?**
   - **Answer:**
     - Use **Terraform Sentinel** (in Terraform Cloud/Enterprise) to enforce policies.
     - Alternatively, use **validation rules** in input variables or custom scripts to validate configurations.
9. **You need to provision a complex infrastructure with dependencies between resources. How would you handle this in Terraform?**
   - **Answer:**
     - Use **implicit dependencies** by referencing resource attributes (e.g., `aws_subnet.id`).
     - Use the `depends_on` argument for explicit dependencies.
     - Test the configuration with `terraform plan` to ensure correct resource ordering.
10. **You want to roll back to a previous version of your infrastructure. How would you do this in Terraform?**
    - **Answer:**
      - Use **version control** (e.g., Git) to track changes to Terraform configurations.
      - Revert to a previous commit and run `terraform apply` to roll back the infrastructure.
      - Ensure the state file is consistent with the desired configuration.

Advanced Questions

1. **What is the difference between `count` and `for_each` in Terraform?**
   - **Answer:**
     - `count` creates multiple instances of a resource using a numeric value.
     - `for_each` creates multiple instances using a map or set, allowing for more flexibility and named resources.
2. **How do you handle sensitive data (e.g., passwords) in Terraform?**
   - **Answer:**
     - Use the `sensitive` argument to mark variables as sensitive.
     - Store sensitive data in **environment variables** or a **secrets manager** (e.g., AWS Secrets Manager).
     - Avoid hardcoding sensitive data in Terraform configurations.
3. **What is the purpose of `terraform import`, and how do you use it?**
   - **Answer:** `terraform import` is used to import existing infrastructure into the Terraform state file. It allows you to manage resources that were not created by Terraform.
4. **What is the difference between `output` and `local` values in Terraform?**
   - **Answer:**
     - `output` values expose information to other configurations or the CLI.
     - `local` values are used for intermediate calculations within a configuration and are not exposed externally.
5. **How do you handle Terraform versioning in a team environment?**
   - **Answer:**
     - Use a `.terraform-version` file or `required_version` in the configuration to specify the Terraform version.
     - Use tools like **tfenv** or **Terraform Cloud** to manage version consistency.

Behavioral Questions

1. **Describe a time when you used Terraform to solve a complex infrastructure problem.**
   - **Answer:** (Tailor this to your experience. Example: "I used Terraform modules to provision a multi-region architecture, reducing deployment time and ensuring consistency.")
2. **How do you ensure collaboration and avoid conflicts when working with Terraform in a team?**

- **Answer:** (Example: "We use a remote backend with state locking, enforce code reviews, and follow Git workflows to manage changes.")

3. **What challenges have you faced with Terraform, and how did you overcome them?**
   - **Answer:** (Example: "We faced state file conflicts, which we resolved by migrating to a remote backend and implementing state locking.")
