Terraform Interview Questions Cheat Sheet:-

Basic Questions:-

What is Terraform, and how does it differ from other IaC tools?
Answer: Terraform is an open-source Infrastructure as Code (IaC) tool used to define, provision, and manage infrastructure across multiple cloud providers. Unlike other IaC tools like AWS CloudFormation, Terraform is multi-cloud, meaning it supports AWS, Azure, Google Cloud, and other providers.

What are the main components of Terraform?
Answer:

Provider: Defines the cloud provider (AWS, Azure, etc.).
Resource: Defines the infrastructure components (e.g., EC2 instance, S3 bucket).
Module: A collection of resources used to organize and reuse infrastructure code.
State: A file where Terraform tracks the current state of your infrastructure.
Backend: Specifies where the state is stored (e.g., S3 for remote state).

How does Terraform ensure idempotency?
Answer: Terraform ensures idempotency by maintaining a state file that tracks the current infrastructure. When a `terraform apply` is run, it compares the desired state (as per the configuration) with the actual state (in the state file) and only makes necessary changes.

- **What file formats does Terraform use, and what is the role of `.tfstate`?**
  Answer: Terraform configuration files use the `.tf` extension for defining infrastructure (e.g., `main.tf`). The `.tfstate` file stores the current state of the infrastructure, ensuring Terraform knows what resources it has created or modified.

Explain the Terraform lifecycle commands.
Answer:

- **`terraform init`:** Initializes the working directory, downloads provider plugins.
- **`terraform plan`:** Creates an execution plan showing what actions Terraform will take.
- **`terraform apply`:** Applies the changes described in the plan.
- **`terraform destroy`:** Destroys the infrastructure managed by Terraform.

What is the purpose of a `provider` in Terraform?
Answer: A provider in Terraform is responsible for understanding API interactions with a specific cloud service (like AWS or Azure). It allows Terraform to create, read, update, and delete resources on the cloud provider.

How do you manage sensitive data in Terraform?
Answer: Sensitive data can be managed by using variables marked with `sensitive = true` or by storing secrets securely in environment variables or secret management systems like AWS KMS, HashiCorp Vault, or SSM Parameter Store.

What is the difference between `terraform apply` and `terraform refresh`?
Answer:

- **`terraform apply`:** Applies changes to infrastructure based on the configuration.
- **`terraform refresh`:** Updates the state file by querying the actual infrastructure to reflect any changes made outside Terraform.

What is a `terraform lock file`, and why is it used?
Answer: The `terraform.lock.hcl` file locks the version of providers used in the configuration. This ensures that when the infrastructure is applied on different machines or by different team members, the same provider version is used.

How do you define variables in Terraform?
Answer: Variables in Terraform are defined using the `variable` block:

```
variable "instance_type" {
  type    = string
  default = "t2.micro"
}
```

Intermediate Questions:-

What are Terraform modules, and how do you create and use them?

Answer: Modules in Terraform are reusable units of infrastructure code that encapsulate resources and can be shared or reused. A module is typically stored in a directory with its own configuration files.

```
module "network" {
  source = "./modules/network"
  vpc_id = "vpc-123"
}
```

How does Terraform manage state, and why is it important?

Answer: Terraform stores the current state of resources in a state file (`terraform.tfstate`). This file is crucial because it tracks the real-world resources managed by Terraform. Without it, Terraform wouldn't know which resources exist and would create new ones or fail to track changes.

What are backends in Terraform? Name a few examples.
Answer: Backends in Terraform define where the state file is stored. Examples:

- **S3 (AWS)**
- **Azure Blob Storage**
- **Consul**

- **Terraform Cloud**
- **Explain the difference between `local` and `remote` state management in Terraform.**
  Answer:

- **Local state:** The state file is stored locally on your machine.
- **Remote state:** The state file is stored in a remote location (e.g., AWS S3, Terraform Cloud), allowing collaboration among teams and ensuring consistency.

- **What is `terraform import`, and when would you use it?**
  Answer: The `terraform import` command allows you to bring existing infrastructure under Terraform management by importing resources into the state file. For example, importing an EC2 instance:

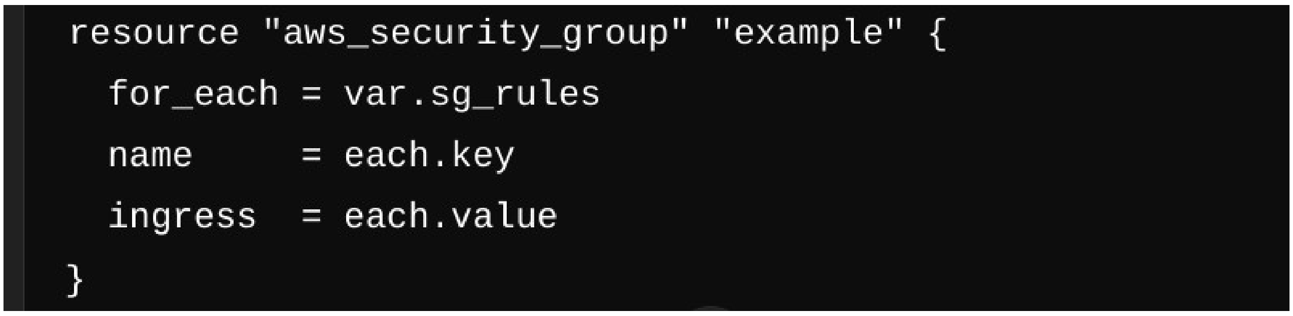
How do you handle drift detection in Terraform?
Answer: Drift detection can be done by running `terraform plan`. This command will show the differences between the current infrastructure and the Terraform state, allowing you to detect any configuration drift.

Explain the use of `count` and `for_each` in Terraform. Provide examples.
Answer:

- **`count`:** Allows creating multiple instances of a resource based on a number

```
resource "aws_instance" "example" {
  count = 3
  ami   = "ami-123"
  instance_type = "t2.micro"
}
```

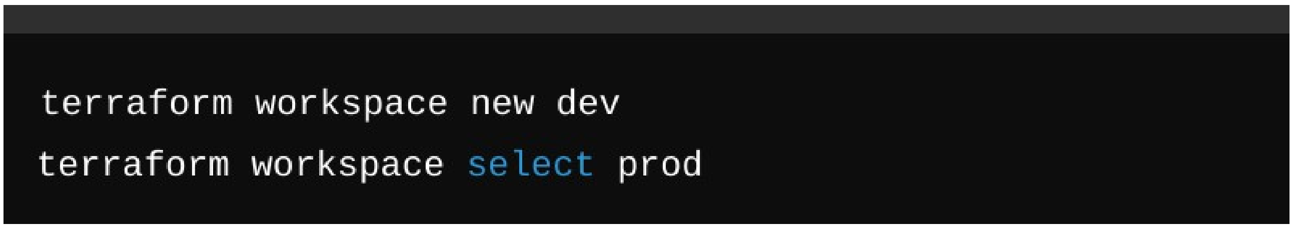
`for_each`: Used to create resources from a map or set:



```
resource "aws_security_group" "example" {
  for_each = var.sg_rules
  name     = each.key
  ingress  = each.value
}
```

How can you manage dependencies between resources in Terraform?
Answer: Terraform automatically handles resource dependencies based on references. Explicit dependencies can also be created using the `depends_on` attribute.

What is a workspace in Terraform, and how do you use it?
Answer: Workspaces allow you to manage different environments (e.g., dev, staging, prod) with the same configuration. Each workspace has its own state file.



```
terraform workspace new dev
terraform workspace select prod
```

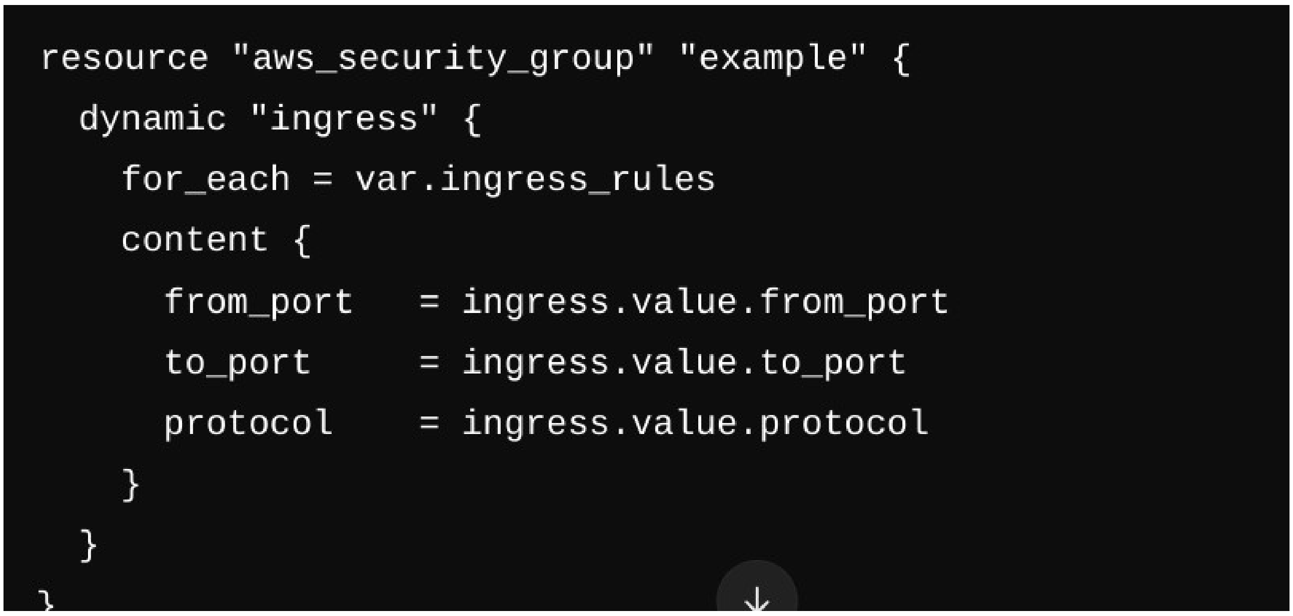
How do you version Terraform configurations, and why is it important?
Answer: Terraform configurations should be versioned using Git to ensure that changes are tracked, and multiple environments can be managed efficiently.

Advanced Questions:-

How do you write reusable Terraform modules? What are the best practices?
Answer: Reusable Terraform modules should be written to abstract common resources, such as VPC, EC2, or RDS. Best practices include using input variables for customization, output variables to expose values, and keeping module logic clean and simple.

How do you integrate Terraform with CI/CD pipelines?
Answer: Terraform can be integrated into CI/CD pipelines (Jenkins, GitLab CI) by automating the `terraform init`, `terraform plan`, and `terraform apply` commands. Store sensitive data (e.g., AWS credentials) securely in the CI/CD system.

What are dynamic blocks, and how are they used in Terraform?
Answer: Dynamic blocks allow for the generation of nested blocks in Terraform based on variables or conditions.



```
resource "aws_security_group" "example" {
  dynamic "ingress" {
    for_each = var.ingress_rules
    content {
      from_port   = ingress.value.from_port
      to_port     = ingress.value.to_port
      protocol    = ingress.value.protocol
    }
  }
}
```

1. **What is Sentinel in Terraform, and how can it enforce policies?**
   Answer: Sentinel is a policy-as-code framework integrated into Terraform Enterprise and Cloud that allows you to enforce rules before resources are provisioned (e.g., ensuring resources meet security or compliance standards).
2. **How does Terraform handle multi-cloud provisioning?**
   Answer: Terraform can handle multi-cloud provisioning by using multiple provider blocks in the same configuration. Resources from different cloud providers can be managed in a single plan.
3. **What are the differences between `data` and `resource` blocks in Terraform?**
   Answer:
   - **`data` block:** Retrieves data from an external source (e.g., data from a cloud provider).
   - **`resource` block:** Defines infrastructure that Terraform will manage and create.

4. **Explain the concept of graph theory in Terraform and how it optimizes execution.**
   Answer: Terraform uses a directed acyclic graph (DAG) to determine resource dependencies and execution order. This graph allows Terraform to execute independent resources in parallel for faster provisioning.
5. **What are preconditions and postconditions in Terraform 1.3+, and how do they enhance validations?**
   Answer: Preconditions ensure certain conditions are met before a resource is created (e.g., an AMI exists), while postconditions validate the state after resources are created.
6. **How do you handle secrets management in Terraform without exposing them in configuration files?**
   Answer: Use encrypted backends (e.g., AWS SSM, Vault) to store secrets and environment variables for sensitive data instead of hardcoding them in `.tf` files.

Scenario-Based Questions:-

1. **You have a Terraform module for deploying EC2 instances. How would you handle deploying to multiple environments (dev, staging, prod) with minimal code duplication?**
   Answer: Use Terraform workspaces, or create separate `tfvars` files for each environment (e.g., `dev.tfvars`, `prod.tfvars`) to manage different configurations while reusing the same module.
2. **What would you do if you accidentally deleted the `.tfstate` file? How would you recover?**
   Answer: If the `.tfstate` file is deleted, attempt to restore it from a backup. If unavailable, use `terraform import` to bring resources back under Terraform management.
3. **Your team is using Terraform, but two developers simultaneously ran `terraform apply`. How would you avoid this conflict?**
   Answer: Use remote state with state locking (e.g., using DynamoDB for S3 backend) to prevent concurrent changes to the state file.
4. **How would you migrate existing infrastructure into Terraform?**
   Answer: Use `terraform import` to import existing resources into Terraform's state, and then define them in Terraform configuration files.
5. **A production deployment failed because of a misconfiguration in Terraform. How would you debug and resolve it?**
   Answer: Run `terraform plan` to identify discrepancies, check the state file for inconsistencies, and use Terraform logs for troubleshooting. Rollback if necessary.
6. **How would you design a multi-region architecture in Terraform with disaster recovery in mind?**
   Answer: Use modules to manage VPC, EC2, and RDS resources across multiple regions, along with conditional logic (e.g., `count` or `for_each`) to enable failover scenarios.

7. **How do you optimize Terraform configurations for large-scale infrastructure?**
   Answer: Use modules for reusable components, break down infrastructure into smaller chunks, and use backend storage for managing state efficiently.
8. **What would you do if a resource you want to delete in Terraform is still required by another team's infrastructure?**
   Answer: Use `lifecycle` blocks with `prevent_destroy` to protect critical resources from accidental destruction.