Section 1: Q&A (20 Questions
Terraform Hands-on Exam and Q&A

Exam Overview

Duration: **3-5 hours**

- **Section 1: Multiple Choice & Short Answer Questions (20 questions)** (~1 hour)
- **Section 2: Hands-on Practical Assignments** (~2-4 hours)
    - Task-based questions requiring Terraform configuration
    - Infrastructure provisioning and troubleshooting

Exam Structure

Section 1: Q&A (20 Questions)

- **Terraform Fundamentals (5 questions)**

    - What is Terraform and how does it differ from other IaC tools?

Terraform is an Infrastructure as Code (IaC) tool created by HashiCorp that uses a declarative language (HCL) to provision and manage infrastructure across multiple providers (AWS, Azure, GCP, etc.). The main differences are it's agentless, uses a state file and others often specialize in configuration management or platform-specific tasks while Terraform focuses on infrastructure provisioning across multiple clouds.

    - Explain Terraform's declarative nature and state management.

Declarative, meaning you describe the desired end state, not the steps.
It is stored in Terraform.tfstate and enables dependency tracking, diffing, and safe updates.

    - What is the purpose of the Terraform provider?

A Terraform provider is a plugin that enables Terraform to communicate with a specific service or cloud platform's API. It allows Terraform to create, read, update, and delete resources defined in the configuration.

    - How does Terraform handle dependency resolution?

Terraform resolves dependencies by building a graph that includes implicit dependencies inferred from resource references and explicit dependencies defined with depends_on, ensuring resources are applied in the correct order.

- What are the key components of a Terraform configuration file?

The key components of a Terraform configuration file are the terraform block, provider, resource, variable, output, data, module, and locals blocks, all written in HCL. These blocks define Terraform's behavior, required providers, infrastructure resources, inputs, outputs, reusable modules, data lookups, and local values.

- **State Management & Backend Configuration (3 questions)**

  - Explain the difference between `terraform refresh`, `terraform plan`, and `terraform apply`.

terraform refresh updates the state file to match the real infrastructure without making changes, terraform plan shows the execution plan by comparing the current state with the desired configuration, and terraform apply executes the plan to create, update, or destroy resources.

  - What is the difference between local and remote backends?

A local backend stores the Terraform state file on the local machine, which is simple but not suitable for team collaboration, while a remote backend stores the state in a shared remote location (such as S3 or Terraform Cloud), enabling state locking, collaboration, and safer multi-user workflows.

  - How can you prevent state corruption when multiple engineers work on the same infrastructure?

State corruption is prevented by using a remote backend with state locking so only one Terraform operation can run at a time, typically with services like S3 + DynamoDB or Terraform Cloud. Additional safeguards include enabling state versioning, avoiding local state files, and running terraform apply through CI/CD pipelines to control and serialize changes.

- **Terraform Modules & Reusability (4 questions)**

  - What are the benefits of using Terraform modules?

Terraform modules provide reusability, consistency, and maintainability by allowing you to group and organize infrastructure code into logical, reusable components. They also make configurations scalable, easier to test, and shareable across projects or teams.

  - Explain how to pass variables to a Terraform module.

Variables are passed to a Terraform module by defining input variables in the module and specifying their values when calling the module.

  - What is the difference between `count` and `for_each`?

count creates multiple resource instances based on an integer and identifies them by index (e.g., [0]), making it best for simple, identical items. for_each, on the other hand, iterates over maps or sets to create distinct instances identified by unique keys, providing greater stability and flexibility when items are added, removed, or changed.

- How do you source a module from a Git repository?

You source a Terraform module from a Git repository by setting the module's source attribute to the Git URL.

- **Terraform with AWS (4 questions)**
  - How do you create an EC2 instance with Terraform?

You create an EC2 instance in Terraform by defining an aws_instance resource and specifying required attributes such as the AMI ID, instance type, subnet, and security groups using this basic template:

```
resource "aws_instance" "example" {
  ami  = var.AMIS[var.AWS_REGION]
  instance_type = "t2.micro"
}
```

  - What are the required fields for defining a VPC in Terraform?

The required field for defining a VPC in Terraform is the cidr_block, which specifies the IP address range for the VPC.

  - Explain how Terraform manages IAM policies in AWS.

Terraform manages IAM policies in AWS by defining IAM resources such as aws_iam_policy, aws_iam_role, aws_iam_user, and their attachments in code, then tracking their relationships in the state file. This allows Terraform to create, update, and attach policies consistently and manage changes through plans and applies.

  - How do you use Terraform to provision and attach an Elastic Load Balancer?

You provision and attach an Elastic Load Balancer in Terraform by creating the load balancer resource, defining a target group, registering targets (or using an Auto Scaling Group), and configuring a listener to forward traffic to the target group.

- **Debugging & Error Handling (4 questions)**

  - What does the `terraform validate` command do?

The terraform validate command checks Terraform configuration files for syntax and internal consistency, ensuring they are structurally valid. It is a local check that catches errors early without contacting providers or making any infrastructure changes.

  - How can you debug Terraform errors effectively?

Terraform errors can be debugged by carefully reading error messages, running terraform plan to preview changes, using detailed logs with TF_LOG, and inspecting the state with commands like terraform state list or terraform show.

- What is Terraform's `ignore_changes` lifecycle policy used for?

Terraform's ignore_changes lifecycle policy is used to tell Terraform to ignore changes to specific resource attributes during updates, preventing Terraform from modifying them even if they differ from the configuration.

- How do you import existing AWS infrastructure into Terraform?

You can import existing AWS resources into Terraform using the terraform import command, which maps a real resource to a Terraform resource in your configuration.
eg. terraform import aws_instance.web i-a1b2c3d4e5

Section 2: Hands-on

Section 2: Hands-on Practical Assignments

Task 1: Create a Custom VPC with Public & Private Subnets

- Create a VPC with CIDR block `10.0.0.0/16`.
- Define two subnets:
  - **Public Subnet**: `10.0.1.0/24`, allows outbound internet access.
  - **Private Subnet**: `10.0.2.0/24`, restricted from direct internet access.
- Attach an Internet Gateway to the VPC and associate it with the public subnet.
- Define routing tables for both subnets.

Task 2: Deploy an EC2 Instance in the Public Subnet

- Launch an EC2 instance with the following specifications:
  - AMI: Ubuntu 22.04
  - Instance Type: `t2.micro`
  - Assign a **public IP address**
  - Security Group: Allow SSH (port 22) and HTTP (port 80)
- Use Terraform outputs to display the public IP of the instance.

Task 3: Convert the VPC and EC2 Configuration into a Terraform Module

- Refactor the VPC and EC2 configuration into a reusable module.
- The module should allow users to specify:
  - VPC CIDR range
  - Subnet count
  - Instance type
  - Whether a public IP should be assigned
- Deploy the module in a separate Terraform root configuration.

Task 4: Deploy an Application Load Balancer with Auto Scaling

- Create an **Application Load Balancer (ALB)**.
- Attach a **target group** and add the EC2 instances.
- Configure **auto-scaling** with a minimum of 1 and a maximum of 3 instances.
- Use Terraform outputs to display the ALB DNS name.

Evaluation Criteria

- **Correctness**: Infrastructure must be properly provisioned.
- **Efficiency**: Solutions should use best practices (e.g., modules, variables, lifecycle policies).
- **Reusability**: Modularization and maintainability of the Terraform code.

- **Security Considerations**: Proper IAM role assignments, subnet configurations, and security group rules.
- **Troubleshooting Skills**: Ability to debug and resolve Terraform errors efficiently.