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?
  - Explain Terraform's declarative nature and state management.
  - What is the purpose of the Terraform provider?
  - How does Terraform handle dependency resolution?
  - What are the key components of a Terraform configuration file?
- **State Management & Backend Configuration (3 questions)**
  - Explain the difference between `terraform refresh`, `terraform plan`, and `terraform apply`.
  - What is the difference between local and remote backends?
  - How can you prevent state corruption when multiple engineers work on the same infrastructure?
- **Terraform Modules & Reusability (4 questions)**
  - What are the benefits of using Terraform modules?
  - Explain how to pass variables to a Terraform module.
  - What is the difference between `count` and `for_each`?
  - How do you source a module from a Git repository?
- **Terraform with AWS (4 questions)**
  - How do you create an EC2 instance with Terraform?
  - What are the required fields for defining a VPC in Terraform?
  - Explain how Terraform manages IAM policies in AWS.

  - How do you use Terraform to provision and attach an Elastic Load Balancer?
- **Debugging & Error Handling (4 questions)**
  - What does the `terraform validate` command do?
  - How can you debug Terraform errors effectively?
  - What is Terraform’s `ignore_changes` lifecycle policy used for?
  - How do you import existing AWS infrastructure into Terraform?

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.