Terraform Hands-on Exam and Q&A

Name: Hen Cohen

ID: 300713591

Mail: hencohen888@gmail.com

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 open-source Infrastructure as Code (IaC) tool used to define, provision, and manage cloud and on-premises infrastructure. Unlike many other IaC tools, Terraform uses a declarative language (HCL) and creates an execution plan that shows what changes will be made before applying them, allowing the same infrastructure to be reliably recreated.

  - **Explain Terraform's declarative nature and state management.**

    Terraform is declarative, meaning you define the desired end state of your infrastructure rather than the steps to create it. Terraform uses a state file to track the current state of resources, compare it with the desired configuration, and determine what changes are required to reach and maintain that state.

  - **What is the purpose of the Terraform provider?**

    A Terraform provider is a plugin that allows Terraform to interact with external services such as AWS, Azure, or Google Cloud. Providers define resource types and translate Terraform's declarative configuration into CRUD (Create, Read, Update, Delete) API calls to manage infrastructure.

- **How does Terraform handle dependency resolution?**

  Terraform handles dependencies by building a **Directed Acyclic Graph (DAG)** of all resources. Dependencies can be implicit (based on resource references) or explicit using depends_on. The DAG determines the correct execution order and allows Terraform to provision independent resources in parallel.

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

  A Terraform configuration file, written in HCL, consists of several key blocks:

  terraform – configures Terraform settings such as required versions, providers, and backend

  provider – defines provider-specific configuration (e.g., AWS, Azure)

  resource – declares infrastructure resources to be created and managed

  variable – defines input variables to make configurations flexible and reusable

  output – displays values from the infrastructure after terraform apply

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

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

    These commands serve different roles in the Terraform workflow:

    terraform refresh - updates the state file to match the real infrastructure without changing resources.

    terraform plan - performs a dry run, comparing the current state with the configuration and showing the proposed changes.

    terraform apply - executes the plan and makes actual changes 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 filesystem and is commonly used for personal or small-scale projects.
  A remote backend stores the state remotely (e.g., S3, Terraform Cloud), enabling team collaboration, state locking, and improved security and consistency.

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

  State corruption can be prevented by using a remote backend with state locking enabled, which ensures only one Terraform operation can modify the state at a time. Additional best practices include using version control, restricting access to the state file, and avoiding manual state edits.

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

  - **What are the benefits of using Terraform modules?**

    Terraform modules improve reusability by allowing infrastructure code to be shared across projects. They enforce consistency through standardized components, simplify maintenance, and help organize complex configurations into manageable units.

  - **Explain how to pass variables to a Terraform module.**

    Variables are passed to a Terraform module by first defining input variables in the child module, then providing values for those variables as arguments in the module block of the parent configuration. This creates a clear interface, similar to passing arguments to a function.

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

  The difference between count and for_each is how Terraform tracks multiple resource instances.
  count uses numeric indices and is best for simple lists, while for_each uses unique keys from maps or sets, making it more predictable and resilient when the input data changes.

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

  A module can be sourced from a Git repository by using the source argument in a module block and prefixing the repository URL with git::. Terraform supports both HTTPS and SSH URLs. When using SSH, Terraform relies on the locally configured SSH keys for authentication.

- **Terraform with AWS (4 questions)**

  - **How do you create an EC2 instance with Terraform?**

    An EC2 instance is created in Terraform by configuring the AWS provider and defining an aws_instance resource with required attributes such as the AMI and instance type. The infrastructure is then provisioned using the Terraform CLI commands terraform init, terraform plan, and terraform apply.

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

    The required fields are specific to the cloud provider (AWS, Azure, etc.) for example, for the AWS provider's aws_vpc resource, the primary required field is the cidr_block. This defines the IPv4 address range for the VPC.

    ```
    resource "aws_vpc" "main" {
     cidr_block = "10.0.0.0/16"
     tags = {
      Name = "Project VPC"
     }
    }
    ```

  - **Explain how Terraform manages IAM policies in AWS.**

    Terraform manages IAM policies in AWS using Infrastructure as Code by defining IAM resources declaratively in HCL. Resources such as aws_iam_policy, aws_iam_role, and aws_iam_policy_attachment are used to create, manage, and attach permissions in a consistent and repeatable way

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

    o provision and attach an Elastic Load Balancer using Terraform, you define the load balancer, a target group, and a listener, then attach compute instances to the target group. The core resources involved are aws_lb, aws_lb_target_group, aws_lb_listener, and aws_lb_target_group_attachment.

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

  - **What does the terraform validate command do?**

    The terraform validate command checks whether Terraform configuration files are syntactically valid and internally consistent. It does not access remote services or make any changes to infrastructure.

  - **How can you debug Terraform errors effectively?**

    Terraform errors can be debugged by carefully reading error messages, running terraform plan to preview issues, using terraform validate to check configuration syntax, and enabling detailed logs with the TF_LOG environment variable when needed.

- **What is Terraform’s ignore_changes lifecycle policy used for?**

  The ignore_changes lifecycle policy is used to tell Terraform to ignore specific attribute changes on a resource during updates. This is useful when certain values are managed externally or change frequently and should not trigger resource recreation

- **How do you import existing AWS infrastructure into Terraform?**

  Existing AWS infrastructure can be imported into Terraform using the terraform import command or the declarative import {} block, which maps existing resources to Terraform resource definitions. After importing, the configuration files must be updated to match the imported infrastructure.

Section 2: Hands-on Practical Assignments

Tasks 1-4 solutions located on the link

https://github.com/HenCohen888/TerraformExam/tree/main