Terraform Hands-on Exam and Q&A - Answers Kobi Kuzi

Terraform Fundamentals:

1. Terraform is a tool for provisioning, managing, and deploying infrastructure resources. One of the key differences is terraform uses a declarative approach and not an imperative, using a state file to track infrastructure and resources. It's modular and supports multiple cloud providers. It does not require an agent running on managed machines.
2. Terraform uses a declarative approach, meaning you define the desired state of your infrastructure and terraform handles how to archive that state. terraform maintains a state file (terraform.tfstate) to track resource, infrastructure.
3. The purpose of terraform is to help automate the deployment, configuration and management of remote servers. Terraform can manage both existing service providers and custom in-house solutions.
4. Terraform configuration dependencies can be established through implicit and explicit dependent declaration. Implicit dependencies are declared through expression refs, while explicit dependencies are specified by using the “depends_on” meta arg.
5. Terraform config file consists of several key components that define and manage infrastructure using HCL and stored in .tf files. A few of those components are: Provider block(which defines the cloud provider or service), Resource block(which defines a resource to be managed like S3 or EC2), Variable block(which defines input variables to make the config more reusable), Output block(which display value after deployment), Terraform block(which specifies terraform settings like versions or providers), Data block(which fetches data from existing resource), Module block(which encapsulates terraform config into reusable modules).

State Management & Backend Configuration:

1. Terraform refresh commands reads the current settings from all managed remote objects and updates the terraform state to match, while terraform apply command creates an execution plan which lets us preview the changes that terraform is planning to make and terraform apply executes the action proposed in the command plan.
2. local would allow us to run commands on our own machine to check certain things while remote would allow us to run commands on a remote machine.
3. Terraform would lock the file which is being used/edited so other people working on the file would not overwrite each other.

Terraform Modules & Reusability:

1. It encapsulates the terraform config and creates a more reusable and modular code.
2. You could pass a variable through the module block while referencing the variable inside the modules/ folder, to insert the given u can use the terraform command in the cli - terraform apply -var “<specific_varibale>”...-var<...>”.
3. Count is used when creating a fixed number of identical resources while for_each is used when creating multiple resources with unique properties.
4. You can source a terraform module from git using the source arg in the module block, it will look something like that “git::https://<github_rep>”.

Terraform with AWS:

1. you can create a EC2 instance using the resource aws_instance.
2. There are no required field t define VPC according the aws_vpc
3. IAM policies is defined with aws_iam_policy and could be attached using aws_iam_role_policy_attachment
4. Terraform allows you to create and configure aws lb using aws_lb.

Debugging & Error Handling:

1. Terraform validate command will check for syntax errors in your .tf files.
2. You can debug terraform effectively using the commands plan, validate and going through your code, you could also create mock files and build your code step by step.
3. The ignore_changes lifecycle policy in Terraform prevents specific attributes of a resource from being modified by Terraform when a configuration is applied
4. To import an existing AWS resource into Terraform, use the terraform import command followed by the resource type and AWS identifier.