**SECTION 1: GIT**

1. **If you using git stash, where will it save data? What is diff b/w index and staging area?**

When we use git stash, the changes are saved in a hidden stash stack in our local repository. We can later retrieve these changes using git stash apply or git stash pop. The terms "index" and "staging area" refer to the same concept in Git. Both are a temporary area where we can stage our changes before we commit them to the repository. The index and the staging area serve as an intermediary between our working directory and the Git repository.

1. **When would individuals use git rebase, git fast-forward, or a git fetch then push?**

- Git Rebase: when we want to make our branch up-to-date with another branch, like `master`. It will look like you made all your changes recently.

- Git Fast-Forward: when we want to update our branch to match another branch but don't want to create a new commit for it.

- Git Fetch Then Push: when we want to first get the latest updates from the remote repository and then push your changes.

1. **How to revert already pushed changes ?**

To revert the already pushed changes, we can use git revert <commit\_hash> to create a new commit that undoes the changes made in the specified commit. Then, we need to push this new commit to update the repository.

git revert commit hash

git push origin <branch\_name>

1. **What is the difference between cherry picking commits vs trying a hard reset. What is the final outcome of the head reference?**

- Cherry-Picking: It will let us to pick a specific commit from one branch and apply it onto another branch. It creates a new commit with those changes on the current branch.

- Hard Reset: It will move our branch pointer back to a previous commit, erasing all the commits that came after it.

- After cherry-picking, the HEAD points to the new commit we’ve just created.

- With a hard reset, the HEAD will move back and point to the commit we’ve reset to.

1. **Explain the difference between git remote and git clone?**

- Git Remote: When we add a remote using `git remote add`, we’re telling Git where the server version of our repository is stored. We can then push to or fetch from that remote.

- Git Clone: When we use to copy an existing Git repository from a server to your local machine. When we clone, Git automatically sets up a remote called "origin" pointing back to the cloned repository.

**SECTION 2: TERRAFORM**

1. **Difference Between `count` and `for\_each` with Scenario-Based Example:**

- Count: Creates multiple instances of a resource based on a count. Cannot be used with complex structures.

- Example: Create 5 EC2 instances

resource "aws\_instance" "example" {

count = 5

// other configuration

}

- Example: Create EC2 instances with different types

resource "aws\_instance" "example" {

for\_each = { "small" = "t2.small", "medium" = "t2.medium" }

instance\_type = each.value

// other configuration

}

1. **Terraform Taint & When to Use terraform state rm vs terraform taint:**

- Terraform Taint: It will the mark a resource for recreation during the next apply, When we want to force-recreate a resource without changing its configuration, we can use this taint.

- Terraform State RM: It will removes a resource from the state file, When the resource is manually deleted or corrupted, and we want to sync the state file, we can use this.

1. **Diagram of Terraform Resources & When Useful:**

- We can use the terraform graph command to generate a visual representation of our resources. This is useful for debugging complex dependencies or for documentation.

1. **Expression:**

* This expression has some syntax issues. A corrected version might look like:

**count = var.run\_remote\_environment ? (var.TFC\_RUN\_ID != "Yes" ? 1 : 0) : null**

1. **Applying Terraform to Multiple Accounts Following Security Best Practices:**

- We can use different workspace environments for each account.

- We can implement Role-Based Access Control (RBAC) for different accounts.

- We can use remote state backends like S3 with encryption and versioning.

- We can apply changes using a CI/CD pipeline that has minimal but sufficient permissions.

**SECTION 3: AWS**

1. **Creating and Sharing Encrypted Volume from Unencrypted Volume:**

- We can create a snapshot of the unencrypted volume and then we can create a new encrypted volume from the snapshot.

- In order to share this encrypted volume with another account we can modify snapshot permissions to share with the other AWS account ID and ensure KMS key policies are adjusted to allow the other account to use the key for decryption.

1. **Implementing Service Control Policy (SCP):**

- SCPs are used in AWS Organizations to set permission boundaries for member accounts and they can be applied at the root, organizational unit, or individual account levels.

- We can use SCPs to whitelist or blacklist AWS services or actions that accounts can perform.

1. **Converting a Public Subnet to a Private Subnet:**

- We can remove the route in the route table that points to the Internet Gateway and optionally, we can associate it with a new route table that doesn't have Internet Gateway routes.

1. **Default Route for a Newly Created Route Table:**

- The default route is a local route, enabling communication within the VPC (`Destination: VPC CIDR, Target: local`).

1. **Ensuring Routes in Route Table Do NOT Use Local Routes:**

- We cannot remove the local route in AWS as it's mandatory for VPC internal communication.

**SECTION 4: Programming Exercise:**

Exercise 1 (AWS) :

Terraform Code to create the below Infrastructure:

• 1 VPC in us-east-1 region. This should be flexible based on region. If no region is provided, this should be built in us-east-1.

• 2 Subnets with high availability supported in 2 zones

• 1 Route table not including the default one. Routes should not be routed using the local route.

• Autoscaling group with a flexible cool down, deregistration delay, instance warm up.

• 2 EC2 instances created from the autoscaling group

• ALB to load-balance the app servers. Ensure the port is flexible based on the application.

• IAM roles should only be used by the account owner**.**

provider "aws" {

region = var.region != "" ? var.region : "us-east-1"

}

# Variables

variable "region" {

default = ""

}

variable "cooldown" {

default = 300

}

variable "deregistration\_delay" {

default = 300

}

variable "instance\_warm\_up" {

default = 300

}

variable "application\_port" {

default = 80

}

# VPC

resource "aws\_vpc" "main" {

cidr\_block = "10.0.0.0/16"

}

# Subnets

resource "aws\_subnet" "subnet" {

count = 2

cidr\_block = element(["10.0.1.0/24", "10.0.2.0/24"], count.index)

availability\_zone = element(

flatten([for az in flatten([list(aws\_availability\_zones.available.names)]) : list(az)]),

count.index,

)

vpc\_id = aws\_vpc.main.id

}

# Route table

resource "aws\_route\_table" "route\_table" {

vpc\_id = aws\_vpc.main.id

route {

cidr\_block = "0.0.0.0/0"

gateway\_id = aws\_internet\_gateway.igw.id

}

}

# Internet Gateway

resource "aws\_internet\_gateway" "igw" {

vpc\_id = aws\_vpc.main.id

}

# Autoscaling group

resource "aws\_autoscaling\_group" "asg" {

min\_size = 2

max\_size = 2

desired\_capacity = 2

vpc\_zone\_identifier = aws\_subnet.subnet[\*].id

launch\_configuration = aws\_launch\_configuration.launch\_config.name

lifecycle {

default\_cooldown = var.cooldown

instance\_warm\_up = var.instance\_warm\_up

}

}

# Launch Configuration

resource "aws\_launch\_configuration" "launch\_config" {

image\_id = "ami-0c55b159cbfafe1f0" # Update with your AMI ID

instance\_type = "t2.micro"

}

# Application Load Balancer

resource "aws\_lb" "alb" {

load\_balancer\_type = "application"

enable\_deletion\_protection = false

listener {

port = var.application\_port

protocol = "HTTP"

default\_action {

type = "forward"

target\_group\_arn = aws\_lb\_target\_group.alb\_tg.arn

}

}

}

# Target Group

resource "aws\_lb\_target\_group" "alb\_tg" {

port = var.application\_port

protocol = "HTTP"

deregistration\_delay = var.deregistration\_delay

}

# IAM Role for account owner

resource "aws\_iam\_role" "owner\_role" {

name = "account-owner-role"

assume\_role\_policy = jsonencode({

Version = "2012-10-17",

Statement = [

{

Action = "sts:AssumeRole",

Effect = "Allow",

Principal = {

AWS = "arn:aws:iam::ACCOUNT\_ID:root" # Replace ACCOUNT\_ID with the actual account ID

}

}

]

})

}