Table of Contents

[**SECTION 1: GIT** 1](#_Toc135226054)

[**Section 2: Terraform** 2](#_Toc135226055)

[**Section 3-b: GCP** 3](#_Toc135226056)

# **SECTION 1: GIT**

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

* **Git Stash** commands locally stores all the recently made changes in the workspace and returns to the previous commit state, code stored in local stash will not be visible to other developers.
* **Git staging** area basically stores all the changes that you have made and added to staging area using git add command and when you run git commit command, it takes the changes from staged area and commits them, it allows you to fine tune your commits, you can add and remove the changes from your staging area, git doesn’t use any specified files for staging area instead it uses a index file which tracks the file changes in three area working directory, staging area, and repository and when ever any changes made into staging area git updates in index file about the changes.

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

* **Git Rebase** allows linear process for merging, it is used to apply sequence of commits from distinct branches to final commit, it merges different commits one by one, for example if you have 2 commits in master branch and 3 commits in test branch, it will be merged in a linear fashion, you can rebase any of the branches.

If you want to rebase the test branch to master branch after you made some changes in master, switch to test branch, do git rebase master.

* If two branches have not diverged and there is a direct linear path from source branch to target branch , git runs a fast forward merge, I have a feature branch and master branch and currently they are pointing to same commit , If I switch to feature branch and do some commits and need bring changes to master branch, it’s a linear path, all git has to do is to change the pointer of master forward. This is where we use **git fast-forward merg**e.
* **Git fetch** is used to update local repository with latest commits from remote repository, it updates latest commits changes in repo but doesn’t merge with the current head commit id, git push is used to send your local changes to remote repository.

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

* Run git log command to see the latest commits performed.
* To undo the commit run **git revert [commit hash]** (commit hash you get through git log command), it removes changes associated with this hash id only.

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

* Cherry picking allows you to pick commit from one branch and apply it to another branch, for example you want to run commit from feature branch to master branch got to master branch by running git checkout master, run **git cherry-pick** [commit reference that you can get through git log], additionally you use –-no-commit option with cherry pick command which will update the changes into working directory of current branch instead of making a new commit.
* If we use **git reset –-hard [commit hash]** when using the hard reset on a specific commit, it forces the HEAD to get back to that commit and deletes everything else after that. The --hard option is typically used when a local commit has gone wrong, and we simply want a fresh start from a previous committed state, git reset is similar to git revert, but approach is slightly different, instead of removing all the commits in its way, the revert ONLY undoes a single commit by taking you back to the staged files before the commit.
* The outcome of HEAD reference commit Id in case of git cherry pick would be no changes as we are not committing any changes while doing cherry pick hence head would remain the same if you are using –no-commit option.

In case of git reset, the head would change to the hash ID of the commit specified while running the command.

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

* Git clone is used to create a copy of remote git repo on your local system ex git clone <repository\_URL>, it will clone the repo from mentioned URL into a new directory where you run this command with the same name as remote repo.
* Git remote command is used to manage remote repository, which is associated with your local repo, allowing you to view, add, rename and remove remote repo.

# **Section 2: Terraform**

1. **What is the difference between terraform count and for\_each meta data function? and give a scenario-based example to use them?**

* When you want to create multiple similar resources instead of writing the resource block multiple times, we can specify the count argument and give the count as required, terraform **count** meta-argument accepts numeric expressions only.

Ex- if I want to spin up 10 compute engine instances with exact same configurations, instead of writing 10 resource blocks, I can give count argument and spin 10 instances.

resource "google\_compute\_instance" "web\_server"{

    name = var.web\_server\_instance\_name[count.index]

    count = 10

    machine\_type = ""

    zone = "var.zone"

    boot\_disk {

      initialize\_params {

          image = ""

      }

    }

    network\_interface {

      network = ""

    }

}

* **For\_each** meta-argument accepts map or a set of strings, for example you want to spin up 10 compute engine instances but with different configurations.

Ex-I want to spin up 3 instances with different machine types and zone, instead of writing the resource blocks again and again I can make use of for\_each meta-argument.

resource "google\_compute\_instance" "vm" {

for\_each = {

"vm1" = { vm\_size = "e2-small", zone = "us-central1-a" }

"vm2" = { vm\_size = "e2-medium", zone = "us-central1-b" }

"vm3" = { vm\_size = "f1-micro", zone = "us-central1-c" }

 }

name = each.key

machine\_type = each.value.vm\_size

zone = each.value.zone

    boot\_disk {

      initialize\_params {

          image = ""

      }

    }

    network\_interface {

      network = ""

    }

}

1. **What is Terraform taint? When to use it? When would you use terraform state rm vs terraform taint?**

* The terraform taint command will manually mark a Terraform-managed resource as tainted, forcing it to be destroyed and recreated on the next apply command-> **terraform taint resource.id**. This command will not modify infrastructure but does modify the state file in order to mark a resource as tainted, once a resource is marked as tainted, the next plan will show that the resource will be destroyed and recreated and the next apply will implement this change.
* The **terraform state rm ‘resource.id’** is used to mark the resource as unmanaged from terraform end forget about it and remove any bindings by excluding it from the state file without destroying or deleting the resource.
* Terraform state rm will be used when you want resource to be managed through terraform while it continues to exist on cloud console while terraform taint will be used when you want to recreate the resource with the same configuration which usually in cases where the resource is damaged or degraded and then terraform proposes to it replace it in new plan.

1. **How would you show a diagram of all terraform resources in the state file? When is this useful?**

* In order to get a visual representation of the configuration of the resources or plan in the terraform state file we can use **terraform graph** command, output is generated in DOT format and GraphViz can be used to generate the charts.

1. **Solve this expression:**

count                  = var.run\_remote\_environment ? var.TFC\_RUN\_ID !=["Yes"]) : null

* Set count meta-argument is equal to variable “run\_remote\_environment” if TFC\_RUN\_ID variable is not equal to yes, if it is equal to yes set count meta-argument is equal to none.

1. **How would you apply terraform to multiple accounts simultaneously? We want to ensure this follows security best practices.**

* If you have two accounts one is Aws and one in GCP and want to create resources in both simultaneously then you can define multiple providers in the providers file and then define the provider in the resource block as well.

provider aws {

region = var.region

}

provider “google” {}

* If you would like to create resources simultaneously to two AWS accounts, you can make use of provider aliases. Create a providers file and put the providers name in it.

provider aws {

  region = var.region

}

provider aws {

  alias = "aws\_02"

  region = var.region

  profile = "aws\_profile" #this should be same name of profile as in AWS credential file.

}

 refrencing the provider in resource blocks

 resource "aws\_instance" "server" {

   provider = "aws"

 }

 resource "aws\_instance" "server" {

   provider = "aws.aws\_02"

 }

# **Section 3-b: GCP**

1. **What are different network connectivity options to connect from On-prem or another cloud to GCP?**

* We can connect to GCP from On-prem or another clous through Cloud VPN, Cloud Interconnect and Cloud Router.
* Cloud VPN allows you to establish an encrypted IPsec VPN tunnel between your on-prem and VPC network in cloud in a single region.
* Cloud Interconnect has two options Dedicated Interconnect and Partner Interconnect, in dedicated interconnect there is a direct connection to google, we can choose from 10 GBPs or GBPs circuits with flexible VLAN attachments capacities from 50 mbps to 50 gbps, in partner interconnect instead of direct connection you connect through one of the Google’s service providers, traffic flows through a service provider not on the public internet, flexible capacity from 50mbps to 50gbps.
* To connect other cloud providers to GCP we can also use Managed VPN services with both AWS and Azure to have an encrypted channel between VPCs in both cloud environments to transfer data only using private IP addresses.
* Partner interconnect also lets you connect a GCP VPC to other cloud providers networks through the network of select partners who provide direct multi cloud solutions.
* You can also connect AWS and Azure using dedicated interconnect by using one or more physical routing devices at a common facility between google cloud and other cloud providers.

1. **Where & how are the FW rules managed in GCP when using shared VPC architecture ?**

* Firewall rules are managed in host project in shared VPC architecture since firewall rules at defined at network level and all the firewall rules defined in host project will be applicable in-service projects as well.

1. **How do you connect to GKE cluster from GCP cloud shell?**

* If you are using a public GKE cluster, then follow the below steps, and for private GKE cluster you will not be able to connect to GKE via cloud shell.
* Open the Google console->Cloud Shell
* Set your project if Kubernetes cluster you are trying to access is in a specific project.
* Retrieve the cluster credentials with following command -> **gcloud container clusters get-credentials [CLUSTER\_NAME] –zone [ZONE or REGION]**
* Verify the connection -> **kubectl cluster-info**
* Once you have connected to Kubernetes cluster through cloud shell, you can interact with cluster using kubectl commands.

1. **How is GCP VPC networking different than AWS VPC networking?**

|  |  |
| --- | --- |
| **GCP** | **AWS** |
| VPC is a global resource, routing traffic between regions is automatic. | VPCs are regional resources; one has to add additional resources to route traffic between regions, e.g., [vpc peering](https://docs.aws.amazon.com/vpc/latest/peering/what-is-vpc-peering.html" \t "_blank) |
| Subnets are regional resources. | Subnets are zonal. |
| Routes are associated with the VPC. They can be restricted to instances via instance tags (or service accounts) | Routes reside in a subnet’s routing table. Controlling the routing of an instance’s traffic involves placing it in the appropriate subnet |
| Firewall rules can have both allow and deny rules | Security groups have only allow rules. Network access control lists (NACL) associated with subnets have both allow and deny rules. Also, unlike the GCP firewall rules and AWS security groups, NACLs are stateless firewalls. |
| Firewall rules can be automatically applied to all instances via network tags. Also, there is an implied egress firewall rule to allow all egress traffic to all destinations. Likewise, there is an implied ingress firewall rule to deny all ingress traffic from all sources | Security groups must be associated with an instance to take effect |

1. **Explain high level steps to create a service project in GCP org with Shared VPC enabled?**

* Shared VPC architecture is where single VPC is shared amongst multiple projects and the project where VPC resides is called host project and project which are using the VPC are called as service projects.
* In order to create service projects, got to IAM & ADMIN section-> Create a project.
* Provide the project name, enter a description and click on create.
* Once you have created the host project and shared VPC in the VPC section go to **give permission** section of VPC and select the project that you have created to attach as the service project and In the **Select users by role** section, add Service Project Admins. These users will be granted the IAM role of “compute.networkUser” for the shared subnets
* Click on Save.