**Infrastructure AS Code (IAC)**  
  
**Challenge :**We have to deploy hundreds of micro-service on several instances in the micro-service world. These instances might have different software and server installed as microservice could build on different technologies. It is time-consuming and error-prone while provisioning instances and installing software manually. So IAC comes in the picture.  
  
**Teraaform** is cloud agnostic for IAC. AWS has **cloud** **formation** and Azure has **ARM** Azure Resource manager to create instances. These tools make the resource provisioning automatic and build in the pipeline.  
**Ansible/ Chef/ Puppet** tools are used to install and configure the software and are known as configuration manager software in IAC pipeline.  
Once servers are provisioned and software are configured you can deploy application using **Jenkins** or **Azure DevOps**

**Terraform installation:  
  
Install Terraform on your machine from the below URL:**<https://learn.hashicorp.com/tutorials/terraform/install-cli>

Or  
Install terraform using Chocolatey. Chocolatey is an open source package manager to install software in Windows.  
Run below command from power shell with admin rights  
**choco install terraform**

Check terraform version if it is installed on your system using power shell or command prompt

Terraform version  
terraform –version

Working With TerraForm  
  
Create folder named as terraform  
Terrfaorm file has extension .tf  
Create basic.tf file in folder and open folder in visual Code

**Providers :** Terraform works for multiple providers. When we install terraform it does not install any provider and we need to mention provider name in .tf file like below and then run command.

provider "google" {

project = "my-project-id"

region = "us-central1"

}

Run command in powershell under terraform project folder  
**terraform init**

The above command will download the Google provider for terraform

Terraform has three main commands given below.

**Terraform init :** This is used to download the required provider to provide the resources.  
**terraform plan :** It runs before apply . It check plans before execution means when you ready with you script this command will check if all the required supporting resources are mentioned and what resources will be created.  
**terraform apply :** This command runs to generate the resources.  
**terraform destroy:** It will delete all the resource created by .tf file

**For Google provider:**Need to have **service account** for authentication. Service account is a identity to authenticate application which will work on Google cloud. So, we need to create service account in GCP then generate its Key. It is highly recommended theay never store access and secrete key in .tf file because .tf will be checked in repositoty and keys can ne leaked. So, we need to export keys in JSOn on our local machine. Place Json file in terraform source code folder and mention credential as key and file name as value in .tf file.  
provider "google"{

region = "us-central1"

project = "arboreal-vision-358115"

**credentials = file("account.json")**

zone = "us-central1-a"

}

Refer below url to generate compute engine on Google through Terraform.

<https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/compute_instance>

**For AWS Provider:**

Same with AWS , never mention access key and secrete access key in .tf file. Run export command for access key and secrete access key to export in Local. It will set environoment variables. AWS required keys and secrets of named profile.

**export AWS\_ACCESS\_KEY\_ID="anaccesskey"**

**export AWS\_SECRET\_ACCESS\_KEY="asecretkey"**Refer below link:  
<https://registry.terraform.io/providers/hashicorp/aws/latest/docs>

State in Terrform:

Terraform create a file .tfstate which has known state. Terraform compare .tf file with .tfstate file at the time of execution and tells if any change is not included in .tf state that was previously executed then it will destroy.

**Desired State:** It is what written in .tf file

**Known State:** It is what written in .tfstate file

**Actual State:** It is what present in AWS or any cloud

Using below command you can start working on terraform console.  
**terraform console**

**resource "google\_storage\_bucket" "static\_sitee"{**

**name ="priyankastore1"**

**location = "US"**

**}**

In this, google keyword shows provider is Google, “google\_storage\_bucket” is the name of resource which needs to be created. “static\_sitee” is the name/id of resource which would be internally known by terraform.

When we want to return some output after apply command then add output variable at bottom of .tf file. You can add multiple output fields.

Output mystorage\_details {

Value = google\_storage\_bucket.static\_sitee

Sometime you need to download plan in local and then apply this specific plan. It will not compare the desired state with actual state.  
**terraform plan –out store.tfplan  
terraform apply “store.tfplan”**

Terraform folder automatic generates two files .tfstate and .tfstate.backup when we apply .tf file to create resource.  
.**tfstate** is having known state means what we executed through .tf file.  
.**tfstate.backup** has one previous known state.  
terraform compares the desired state by comparing .tfstate and actual state on cloud and then produce plan.  
If you delete .tfstate file and run apply command of .tf file then terraform treat it as fresh state or record and could not compare with existing real state.  
name is very important for terraform . It is like ID and compares with find resource with its name only.  
google\_storage\_bucket.**static\_sitee here static site is internal name for Terraform**

.tf file has multiple resource code written but if you want to execite any specific resource the run below command using target keyword with apply command. **terraform apply –target= “google\_storage\_bucket" "static\_sitee”**

**Do Not checkin confidential files in git/gitlab:**

When you are working in team then you need to share your files across. It is highly recommended that never checkin .**tfstate** and .**tfstate.backup** file in git to prevent leakage and apply security. For sharing you can store them in storage on cloud like S3 and then developer get files from there. Add these files in **gitignore** folder to be ignored by Git.

**Creating Separate .tf files:**

Separate .tf Files: It is not good idea to write every code in single .tf file. You can create outputs.tf and write all output code what you want to return after apply command. At runtime, terraform find all files having extension .tf and combined them and then execute.

**Creating multiple resource of same type with index in resource ID:**Specify count of resource and pass index in resource ID to generate dynamic IDs

resource "google\_storage\_bucket" "static\_sitee"{  
count = 2

name ="priyankastore1\_${count.index}"

location = "US"

}

**Getting values in terraform console:**

Open terraform console using below commands in power shell. You can get values of resource what is available in .tfs file after executing apply command.

**Terraform console**

When we create multiple resources of same type then it returns properties of each resource in array. We find it by using index  
**google\_storage\_bucket.static\_sitee[0].name**

Use below command to know if any error in terraform script before apply

**Terraform validate**

Use below to format terraform script:  
**terraform fmt**

Use command in human readable format to know the current state of resources so it gets current state from .tfstate file.

**Terraform show**

**Recovering from Error:**  
It is easy to know the error and recover from it. Let us if you apply the script with same user name as IAM user then it will throw error and will only create first IAM user. It will not show error in plan command but throw error in apply command.

If you don’t want to refresh the state and directly want to run apply command then run below. But it is not recommended.

**Terraform apply –refresh=false**

**Variable Declaration:**  
When you want to pass resource properties dynamic and pass at runtime then declare variable in .tf file and pass variable name in creation script.  
You can pass variables in four ways with given priorites if same variable declared in all ways.  
declaring variables within .tf file - 4(least)  
using environment file - 3  
create new file (.tfvars) to define all variables. Default file name is terraform.tfvars  
passing variable from command line. – 1 (Highest)

**Variables with in same .tf file**

Variable “priyankastore1” {

Type = string ,#any,number,bool, list, map, set,object,tuple /These are supported type.Default is any

Default= “priyankastorage”

}

resource "google\_storage\_bucket" "static\_sitee"{  
count = 2

name ="**${var.priyankastore1}**\_${count.index}"

location = "US"

}

**Passing variable through command line.**

Terraform apply –var=”variableName” = variableValue

**When we want to create multi resource with different names.**

Variable “names”{

**Default =[mystorage”, “priyastorage”, “prodstorage”]**

}

resource "google\_storage\_bucket" "static\_sitee"{  
 count = **names.length**

name = **var.names[count.index]**"

location = "US"

}

When you want to add more values in variable name and add in start then will not work properly because it works in index as it will find new value for each indexes. Below approach is better way to set and pass variable values.

Variable “names”{

**Default =[mystorage”, “priyastorage”, “prodstorage”]**

}

resource "google\_storage\_bucket" "static\_sitee"{  
 **for\_each = toset(var.names)**

**name = each.value**

location = "US"

}

**You would have different folder/project for each terraform as every environment or resource has different requirement or scope. You need ot run terraform init command first time when you run new terraform project**

**Maps:**

Map is used to store multiple properties of multiple resources in variable and get their values using for\_each and each.key

Variable “names”{

**Default ={**

**Mystorage :{country: “US”}**

**priyastorage:{country: “US”}**

**prodstorage:{country: “US”}**

**}**

}

resource "google\_storage\_bucket" "static\_sitee"{  
 **for\_each = var.names**

**name = each.key**

**tags ={**

**country: each.value.country**

location = "US"

}

**Terrform State:**

Used to provide mapping of state of resource with cloud resource,

provides metadata to understand dependencies,

gives performance so that need to connect with cloud always and store in cache.

**Immutable Servers:**  
When we create compute engine through terraform, we might need to do some tweak in server configuration again and again for ex security patching. Rather than doing on same instance what we created, we need take same setting and configuration from existing one and add new configuration and then create new instance. This is the concept of immutable server.

**Note**: When you create resource like EC2 always mention dynamic values like VPC, subnet,AMI(Virtual image), CPU type etc.

Resource “aws\_default\_vpc” “default”{

}  
In AWS, aws\_default\_vpc is the default VPC. Terraform can not create/destroy this VPC. It will get the id and other information of this VPC using above syntax.

Data Provider: Data provider in terraform is used to get the values from cloud to pass in resource creation

**Terraform Graph:** Terraform graph command produce an output in **Dot** format which you can paste in **Graphviz** online and create diagram. Dot is Graph description language.

**Workshpace:** When there are multiple environments like dev/prod.test then Terraform creates multiple workspace for each environment and create the same resource across all env by executing same script .tf file. However, first you need to run Init command. Default workspace name is default.

Below are workspace commands

Terraform workspace show -- Display the current selected env.  
terraform workspace new prod-env ---Creates new env.

Terraform workspace select default ---Used to select the env.  
terraform workspace list --display all env in list