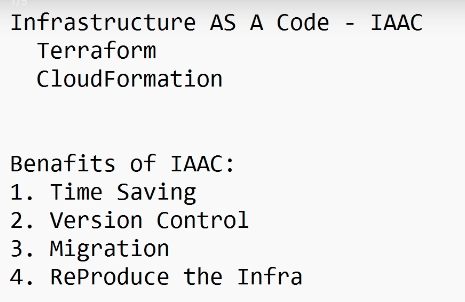
# PART-1 --> Learn Terraform with AWS from scratch

* Terraform install in google -->HashiCorp -->terraform documentation



## EC2

* sudo su - root
* sudo yum install -y yum-utils
* sudo yum-config-manager --add-repo <https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo>
* sudo yum -y install terraform
* Permission to access aws services --> refer AWS - IAM notes
  + access key and secret acces key
  + IAM role attach to EC2
* terraform -v --> check installed or not
* aws s3 ls

### main.tf

* terraform init --> initializing
* terraform plan → Shows what Terraform is going to do before making any changes.
* terraform apply --> Applies the planned changes to your infrastructure, creating, updating, or deleting resources as defined in your Terraform configuration
* cat terraform.tfstate --> terraform memory
* terraform destroy → Deletes all the infrastructure resources

### Terraform vs CloudFormation

* Terraform is cloud-agnostic, so the same configuration can manage resources across AWS, Azure, GCP, or on-prem environments
* CloudFormation is AWS-specific, meaning it can only manage infrastructure resources within the AWS cloud using AWS templates
* Terraform → HCL (HashiCorp Configuration Language)
* CloudFormation → YAML or JSON
* CloudFormation has automatic rollbacks
* Terraform does not have automatic rollbacks; if a deployment fails, we need to identify and fix the issue manually.

### Terraform vs Ansible

* Terraform --> civil Engineer
* Ansible --> Interior Designer
* Terraform provisions cloud resources like EC2, VPCs, and load balancers.
* Ansible is used to install packages on EC2 instances.
* Terraform has terraform.tfstate to store the current state of infrastructure
* Ansible does not have a state file; it does not store the state of managed resources.

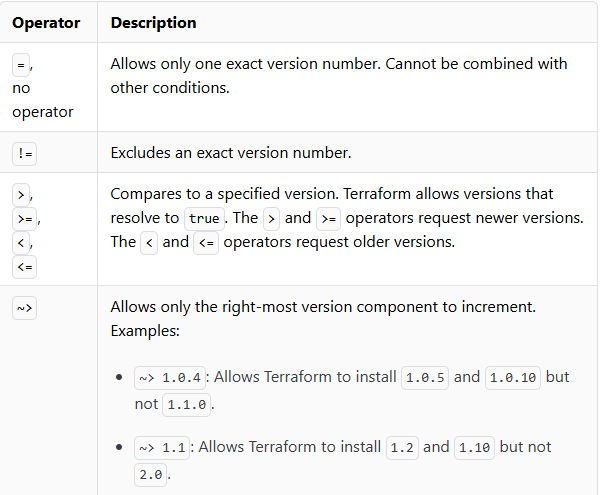
### Chef and Puppet

* Chef is imperative and Ruby-based, giving more flexibility but requiring coding.
* Puppet is declarative with its own DSL, easier to enforce configurations but less flexible.

# PART-2 --> Learn Advanced Terraform Count, meta arguments, for each

### Required Version --> To set version lock

* terraform init
* terraform init -upgrade

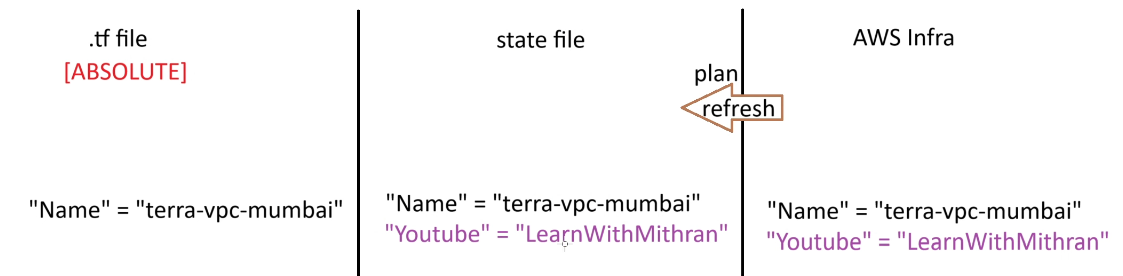
 

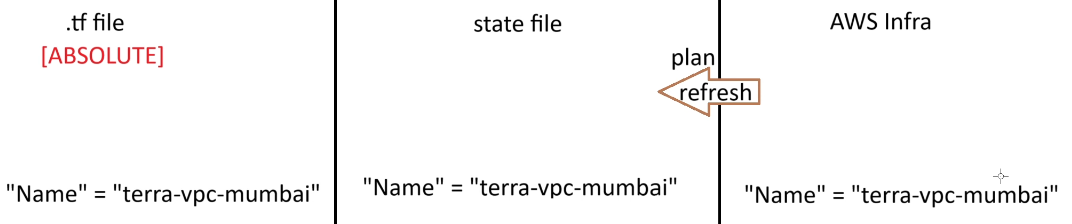
### Multiple Provider --> Multi regions or multi accounts

* terraform plan
* terraform apply
* terraform destroy --auto-approve --> auto approve means skips the interactive confirmation step.
* check .tf files for referenece

### Mutable and Immutable infrastructure

* terraform init
* terraform apply --auto-approve
* Terraform code .tf --> **Absolute**
* Mutable: Can be modified or updated in-place, like tags.
* Immutable: Cannot be changed; any update requires recreating the resource, like a CIDR block.





* terraform refresh --> aws infra manual changes to statefile update
* terraform plan --> removed aws infra manual changes in statefile based on tf abolute code

### Meta-Arguments

* Provider → Terraform plugin that manages resources on a specific platform or service, e.g., AWS, GCP, Azure.
* count --> to create multiple copies of a single resource
* for\_each --> Key-value pair (map) --> value (unmap) with toset
* lifecycle --> true
  + create\_before\_destroy -->no downtime in the gap
  + prevent\_destroy --> not allow the resource to be destroyed
  + ignore\_changes = [tags,] --> it will ignore if changes done in aws infra --> it won’t be remove in state file.
  + depends\_on=[aws\_s3\_bucket.mys3bucket] --> **Explicit dependency**
    - **Implicit dependency** → Created automatically when one resource **uses another’s attribute**.
    - **Explicit dependency** → Created manually using depends\_on to force order.

# PART-4 --> Master Terraform Workspaces, Providers, Provisioners