First thing to do is **Authentication**

Once new code added to terraform (tf)
First thing we have to do is

1. • terraform init

2. • terraform plan
   To see the tf's upcoming plan

3. terraform apply

- The tf file will have the provider section with access key and secret key.

- With **terraform init** it downloads the required dependencies into **.terraform** folder

• terraform destroy — To destroy all the resources created in that folder

To destroy specific tf files and resources you can specify the target

- terraform destroy -target aws_instance.myec2

- We can also destroy the resource in terraform Just by commenting it out

• Terraform State file

- Terraform creates a state file to keep a track of resources

This files are saved with extention .tfstate

- The statefile will have more information on the resource other than mentioned info.

Desired state & Current State

- Desired state - is a state which is desired by the system.

- Current state -

- Terraform's primary function is to create, modify and destroy infrastructure resources to match the desired state described in a TF's configuration.

  >= 3.0 means in range of 3.X any.

If we specify version that info is stored inside `terraform.lock.HCL` file

Once lock file has the HCL we can either delete lock file to change the version or use upgrade to take new constraints.

- `terraform init -upgrade`

Dependency lock file - is used to lock to a specific version of the provider.

- `terraform refresh` is dangerous command try to **avoid** doing in manually.
- Depricated in newer versions
  - `-refresh-only` option is always there.

- `terraform plan` already has terraform refresh in it so no need to run in explicitly

* Whenever there is a change which is done manually and which is not mentioned in the terraform explicitly then terraform will have no update over that.

ex. We changed security group from default to custom
But if our terraform file has not mentioned security group then terraform will do nothing about that SG.

Best Practice → Try to add as much as information in the terraform file. for granular control.

- Cross-referencing attribute

- Terraform Variables -

- Data Source - Way to fetch already existing resources that were not created by tf itself.

Terraform logging -

We can use attribute `TF_LOG`

To export it to file

export `TF_LOG_PATH` = /tmp/terra.log

`TRACE` is one of the default logging

Terraform validate -

To validate your code.

- Production tip - try to have seperate files for each resource, providers etc.

Dynamic Blocks - To dynamically construct repetable nested blocks

Terraform taint →

`terraform apply -replace="aws_instance.myec2"`

This will destroy and recreate the aws resources again to revert back to the changes where everything was working based on the terraform file.

- In previous versions it was called as terraform taint.

- Splat Expression

It allows us to get list of all attribute

- Terraform Graph

- Terraform graph and we can get that .dot file and create a visualization of that

- Terraform Plan file

`terraform plan -out=path`
↑ filename

- you can save this plan to specific file

and we can use the same "path" file to restore the same terraform resources

`terraform apply path`
↑ filename

• Terraform output

Three ways to get the output
① By looking into statefile
② By running a output fonction in tf
③ By running `terraform output name` command in CLI

• Provisioners

Provisioners are used to execute scripts on local or remote machine as part of resource creation or destruction

ex. once EC2 is created running application script on it.

Two types

→ Allows us to invoke local executables after resource creation

① local exec

② Remote exec

→ Runs on remote resources

command to ssh to aws using cmd.

```
ssh -i terraform-key.pem ec2-user@pub-ip.
```

terraform provisioner

and it should be running inside resource

* provisioners are not recommended way for remote exec rather than that use EOF

Terraform Settings

- Dealing with the longer files make sure to have seperate files.

on Large infrastructure to reduce the API calls we can

① We can stop the refresh using

① `-refresh = False Flag`

This will reduce API call

② We can also target

`-target = resource`

Zipmap

- Constructs a map from a list of keys and corresponding list of values.

Comments in the code - # , // or /* */

Provisioner types

① creation-time provisioner

- only run during creation, not during updating or any other lifecycle.

If creation-time provisioner fails, the resource is marked as tainted.

② Destroy-time provisioner -
runs before resource is destroyed

`When = destroy`

`terraform apply -auto-approve`

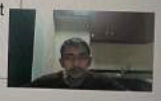
6. Terraform Modules & Workspaces

Acessing child module

- module.<MODULE NAME>.<OUTPUT NAME>

Requirements for Publishing Module

Requirement	Description
GitHub	The module must be on GitHub and must be a public repo. This is only a requirement for the public registry.
Named	Module repositories must use this three-part name format terraform-<PROVIDER>-<NAME>
Repository description	The GitHub repository description is used to populate the short description of the module.
Standard module structure	The module must adhere to the standard module structure.
x.y.z tags for releases	The registry uses tags to identify module versions. Release tag names must be a semantic version, which can optionally be prefixed with a v. For example, v1.0.4 and 0.9.2

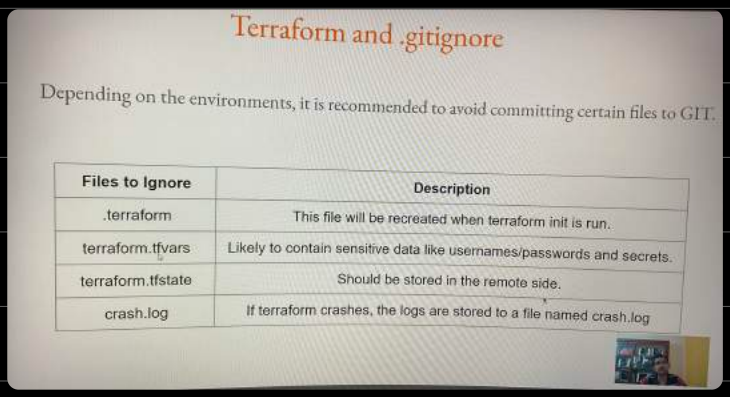


Terraform Workspace -

- just like windows workspace for seperation of Work.

7. Remote state management

.gitignore
files to ignore
- .terraform
- terraform.tfvars → may have sensitive info
- terraform.tfstate →
- crash.log → log file



Terraform state file is being used by a team so storing it locally is not a good option instead of that we can store them on the Backend server so everyone can access them.

But that file should have locking mechanism to protect the file being changed by multiple users at a time which may create inconsistency.

Example of Central Backend - AWS S3 Bucket

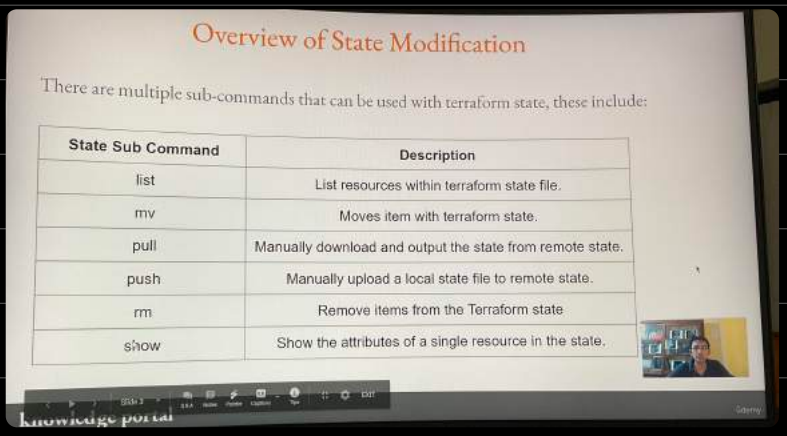
State locking - Whenever we are performing terraform operation then state file will be locked to protect from corruption of file.

* Not All Backends supports state locking.

- force-unlock → to manually unlock

- S3 does not support the state locking
- With the help of DynamoDB we can state lock

Terraform state Management



• Terraform Import

- With this we can manually import Architecture into terraform.

To do that we create the same tf file as the config we have running and map that to the tf file.

8. Security Primer

- To have multiple providers for different resources we create alias in terraform and we can put the new provider block in the resource with the alias to run that resource in that alias.

• Terraform Cloud

for running TF on cloud
- can add variables
- Secure TF state file
- Auto plans on changes
- & many more

- Sentinals

- Policy as code.

ex. Block any EC2 with no tags
Block All traffic sg's

* When doing Remote execution in the Terraform plan the statefile and other files are stored inside a remote cloud while logs are stored locally.

provisioners - They allows you to execute scripts or commands on resource after it has been created or updated.
primarily used for task such as bootstrapping, config management.

- ① remote-exec

ex. Accessing EC2 instance using TF and running nginx command on remote m/c

② local-exec - To run scripts locally.

? can modules be stored on github
modules versioning providers -
- not on public registery.

Notes

- state are saved in terraform.tfstate

- All terraform state subcommands that modify state will write a backup file.

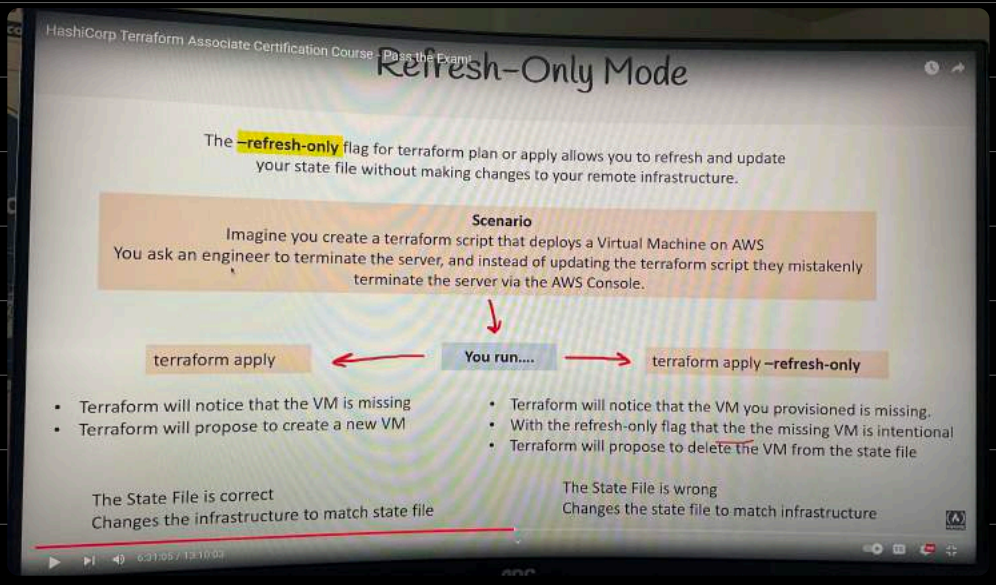
  - Terraform will take the current state and store it in a file called terraform.tfstate.backup.

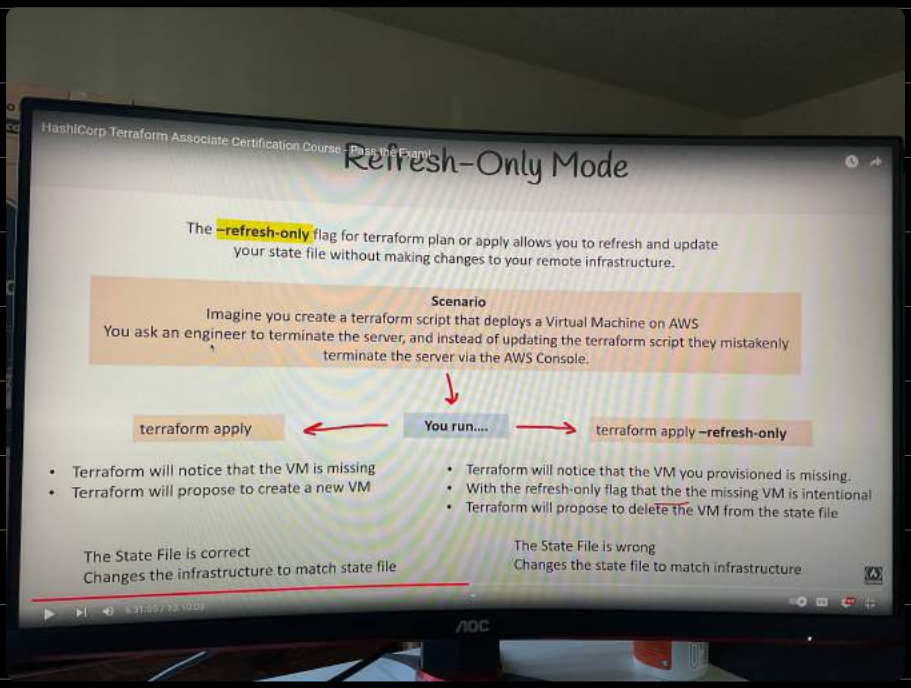
- Backups cannot be disabled.
  To get rid of the backup file you need to manually delete the file.

terraform init

- Downloading plugin dependencies
- create a .terraform directory
- create a dependency lock file to enforce expected version to plugin and tf.

- first cmd to run
- modify or changed dependencies ? →
  run tf init





Terraform init will create following

- .terraform directory
- .terraform.lock.hcl file

- Does terraform plan creates an state file?
- - locks an state file?

TF validate works when tf plan & tf apply

- TF init is required to download tf module
- TF validate doesn't catch any errors related to value ex. ami = "Sam"
- tf providers to get the list of current providers
- Modules can be stored into your private registery.
- By default you are using the backend state.
- tf cloud requires you to have workspaces.
- When running tf cloud credentials needs to be configured on the cloud Environmental variable block.
- We can set partial settings for the Backends using the -backend-config flag for terraform init

ex. terraform init -backend-config=backend.hcl

- Terraform remote state allows that state to be stored on the cloud.

- You can disable lock with `-lock` flag

- tf When using `save planned` it will not preempt you to confirm and will out like `auto-approve`

- tf apply is also running tf validate

- Migrating tf Backend

- difference between public modules and private modules

- public modules format

<Namespace>/<NAME>/<PROVIDER>

ex. `hashicorp/consul/aws`

private modules

- `<HOSTNAME>/<NAMESPACE>/<NAME>/<PROVIDER>`

ex. app.terraform.io/example/vpc/aws.

publishing Modules follows -
- versioning
- Readme's
- Well Documentation
- show examples
- automatically generated documentation

· public modules are managed via public git repo on Github.
It should follow all the tf rules.

· verified modules are reviewed by Hashicorp but managed by users.

Modules directory consist of Nested modules

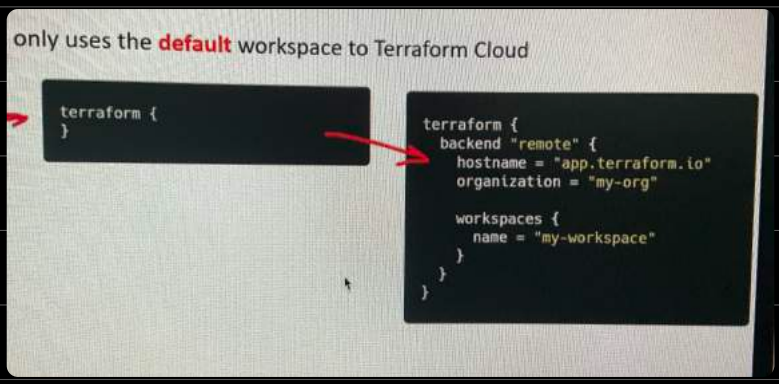
· private module registry supports
- module versioning
- searchable and filterable list
- a configuration designer

- "cost estimation" feature is only available for teams and Governance plan and above.

- Workspaces
  - You can choose any tf versions for worksps
  - You can share state globally.
  - You can choose to auto approve runs.

Migrating from default workspace to tf cloud

- create a workspace in tfCloud
- Replace your terraform configuration with a remote backend



- Run terraform init

Workspaces
- CLI Workspaces
- Terraform cloud workspaces

tf show -json → to get output in json

- credentials should be saved in
  - ① Environmental variables
  - ② credentials file

- Modules are stored in `.terraform/modules`

- `Source` argument is neccessary to call child modules

  Modules can use following meta-arguments
  - for_each
    depends_on
    count
    provider

- Two complex types in terrafer
  - ① A collection type
  - ② A structural type.

① collection types
  - ① List -
  - ② set -
  - ③ Map -

② structural
  - Tuples
    objects
    resources
    Modules

- Whenever configuration backend changes you must run `terraform init`

- default backend - local
- You cannot load additional backends as plugins.

- You don't need to configure backend for tf cloud.

- terraform / terraform.tfstate file contains the backend configuration

* partial configuration is allowed

* When new backend is detected by terraform at the time of reinitialization

* To ensure correct operation & destroy terraform retains copy of the most recent set of dependencies within the state.

* tf stores metadata.

*tf state increases performance
- state file's are cloud agnostic - means irrelevant to particular cloud provider

* Refreshing state can be made optional for large state file or can use -target

*

Terraform cloud.

- Workspaces in terraform cloud and terraform CLI Works differently.
- Workspaces are required in tf cloud.
- Terraform plan will write data to the terraform.tfstate file.
- Terraform refresh — to refresh state file to actually align with real-world

① Single Workspace can have multiple state files.

private registery modules has Hostname mentioned

- syntax — <Hostname>/<NAMESPACE>/<NAME>/<PROVIDER>

Terraform is immutable, declarative IaC

providers plugins downloaded are stored inside
`.terraform/providers`

- Terraform workspace select dev ← To switch wp

- Github is not a supported backend type.
- invalid is a valid variable name in terraform

- Terraform open source stores local state at
`terraform.tfstate.d/<workspace name>`

- cloud agnostic — can run on any cloud platform

- Terraform uses a configuration file to understand resource order and if it is missing then order cannot be determined with config. alone.
so it stores that into State file.

- You can tf code without providers block. It is not mandatory to have terraform block.

- Variables declared inside modules cannot be called directly
→ you can call variables outside of module directly with the help of output block type

- OSS stores its state file in local backend

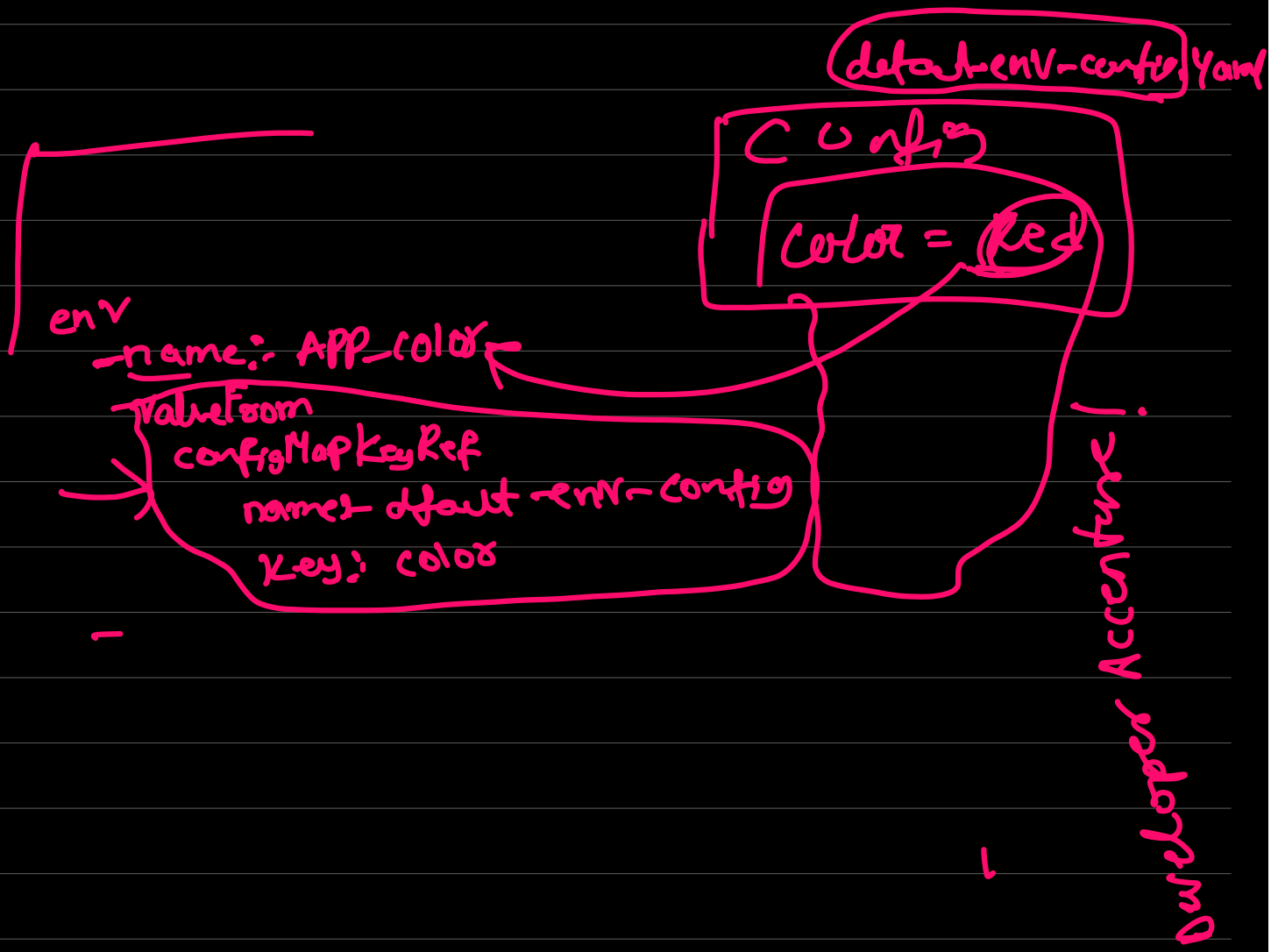
- Better way to secure the configuration
  - credential file
  - environmental variable

- To set backend in terraform we can use

```
terraform {
    backend "name" { }
}
```

- We can only configure one backend block.

- You cannot install terraform providers from source code.


default-env-config.yaml
Config
color = red
env
- name: APP_color
valueFrom
configMapKeyRef
name: dfault-env-config
key: color
Overlapse Accenture