Section 4

Mastering the Terraform CLI

The Terraform provisioners

Terraform is an infrastructure provisioning tool and not a configuration management tool.

Provisioners address this limitation and perform some actions after the resource has been created such as executing a script, commands for example

Provisioners ONLY to be used as a last resort

There are a few types of provisioners:

- Local -exec
- Remote-exec
- file

They are plugins built-in to the Terraform Core so no need to download a separate bin.

The provisioner block is nested in the resource block

```
resource “resource_type” “name”{
    .....
  provisioner “local-exec”{
        <identifier> = <expression>
        <identifier> = self.<parent_attribute>
    }
}
```

Expressions need to use self to refer to the parent attributes.

The local exec provisioner

The local exec provisioner as its names entails allows to execute commands or scripts in the local system.

A good use case is when executing commands locally to a provisioned resource or a script or combining Ansible configuration management capabilities with Terraform provisioned resources.

```
resource “aws_instance” “mytestVM” {
    ...
   provisioner “local-exec” {
       command = “echo ${self.id} >> vmid”
          }
}

resource “aws_instance” “mytestVM1” {
...
   provisioner “local-exec” {
          when = destroy
          command = “mytestVM has been deleted”
        }
 }
```

The remote exec provisioner with the null resource.

A null resource does not do anything. It will install a null provider when configured that does not do anything as well.

You can associate a provisioner with a null resource and trigger the provisioner by referring to an attribute of another resource.

You can associate the provisioner using the null resource with multiple resources.

The remote exec provisioner will invoke a script like BOOTSTRAP, commands or a configuration management tool on a remote resource after it is created by terraform

2 connections types supported:

- ssh
- winrm

Can be used with the create and destroy provisioner.

So...What does actually happens when a provisioner fails?

When a provisioner action fails during a Terraform apply, the resource with the provisioner block will be tainted.

The resource will be created and provisioners only run once during the creation of a resource. tainting the resources is the only way to run the provisioner again.

Importing your infrastructure to Terraform

Whether it is someone that configured manually a resource or you are getting started with IaC, it is possible to import existing resources in a provider into Terraform.

This is accomplished by creating the config in terraform and then using the ID of the resource in the provider as all resources have an id.

The following is command:

```
Terraform import resource_type.name id
```

For GCP authentication you can set the environment variables for a GCP service account:

```
export GOOGLE_APPLICATION_CREDENTIALS="[PATH]"
```

Download the service account key and save in a directory on your local machine. The PATH will be the path to that key.

Understanding Local workspaces and how to best use them

Local Workspaces to run multiple state files associated with the same configuration in the same root module.

Prevents repeated code in multiple root modules in line with the coders DRY principle

Resources cannot viewed or shared across workspaces.

${terraform.workspace} interpolation can be used in the config with resource names or tags...

The default workspace is always available and cannot be deleted.
Additional workspaces can be added and named.

- Terraform workspace new workspace_name
- Terraform workspace select workspace_name
- Terraform workspace delete workspace_name

A directory is created in the root module to save each workspace state file.

```
/terraform.tfstate.d/<workspace_name>
```

Could be used with a version control system like GitHub or Gitlab for a Terraform configuration to test a Terraform configuration

Map a VCS branch to a workspace for example

Managing the terraform state

Terraform state is saved in the terraform.tfstate file

There may come a time that the state needs to be modified but it should no be done by editing the state file.

The terraform state command is the way to go that

```
$ terraform state <subcommand>
$ terraform state list
```

This is ideal command to list all the resource presently provisioned in the provider in an easy to read output without all the curly braces.

To filter out specific resources or modules using resource_type.name or module_type.name

The infrastructure provider resource ID can be used for specific lookups using -id option as the default will list all resources

```
    $Terraform state list -id <realworld_resource_ID>
```

```
$ terraform state mv
```

The mv option will allow you to rename resources in the state file or just move resources to other state files or modules.

```
$ terraform state rm
```

Rm will allow you to delete specific entries from the terraform state file

```
$ terraform state pull
```

When using a remote backend to store the state file in a remote location like S3 or artifact, the pull subcommand will allow to download a copy of that state file to your local Terraform root directory

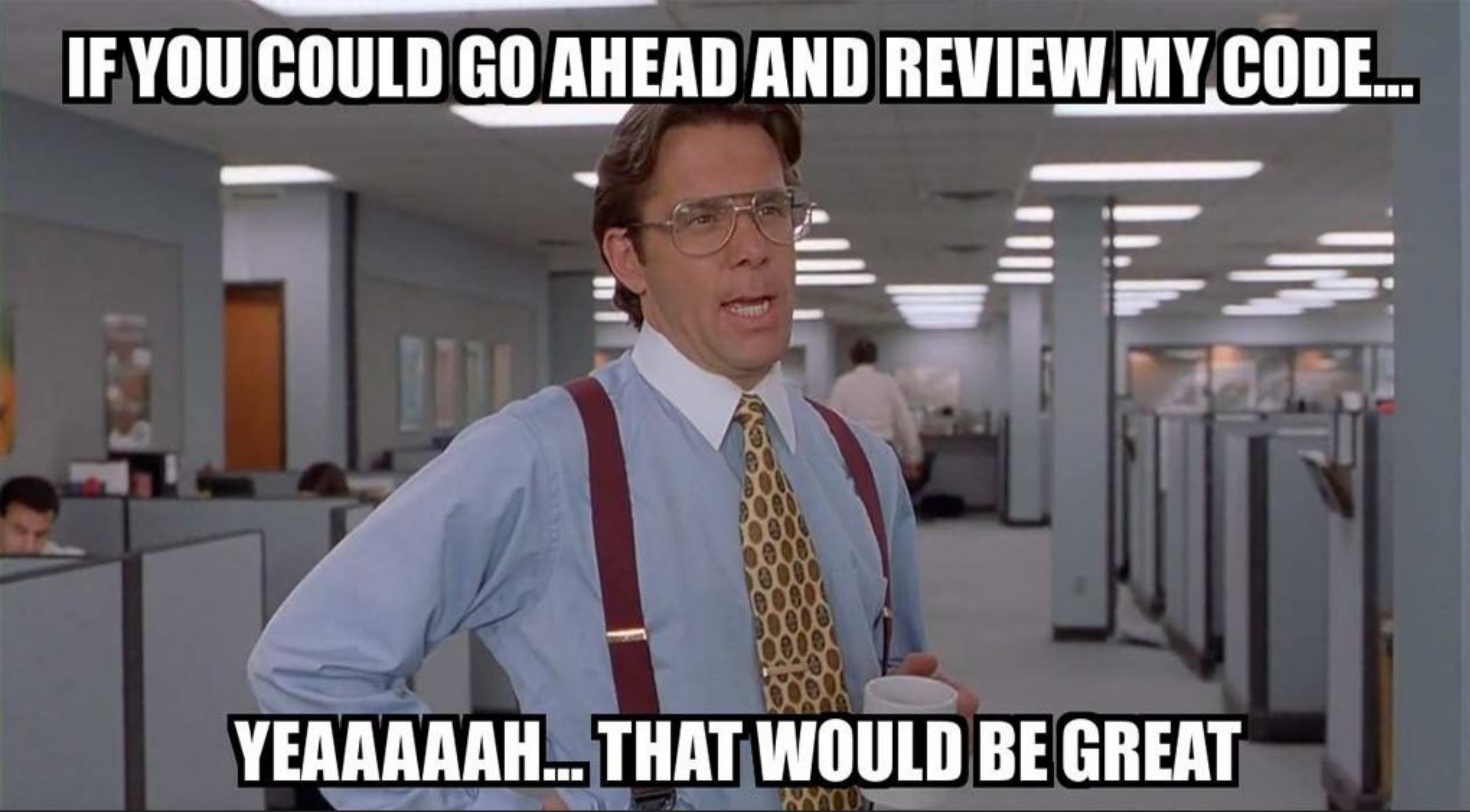
```
$ terraform state push
```

The push subcommand will manually upload a tfstate file to a remote location. Remote usually refers to a storage outside of your local machine

$ terraform state show

This is another useful command to show the output of the state file in JSON.

Format your Terraform code!!


IF YOU COULD GO AHEAD AND REVIEW MY CODE...
YEAAAAAH... THAT WOULD BE GREAT

There are Terraform syntax style conventions to improve readability and review of your code..these are for 12 and they may change between TF releases:

- Indent 2 spaces
- Meta-arguments on top and meta-arguments last within a config block
- One blank line between arguments and nested blocks
- Group similar arguments together and separate with blank line between logical groups.
- Arguments should be declared first and then nested blocks within a block.

```
$ terraform fmt
```

An easy to read code is essential for all developers for collaboration and review.

Terraform fmt command will rewrite all the .tf files in a canonical format in the root working directory in order to have consistency of syntax for all the configuration files.

Tainting your resources...

Tainting resources is commonly used with software managed environment to make a resource radioactive sort to speak..

Terraform will taint a resource when a terraform apply failed or if an infra provider was unable to create or if a provisioner fails during creation..no roll-back

Taint can be used to rerun a provisioner, if someone make a manual config change to a Terraform managed resource or selective rebuilds..

It will only flag for deleting and recreation and not modify the real world. It will NOT destroy and recreate >> -/+ by default

Terraform untaint will undo the taint

Please be aware of dependencies as the destruction of a resource will have an impact on dependent resources.

**Consider using the lifecycle arguments

Using the Terraform console

```
$terraform console
```