KPLABS Course

HashiCorp Certified: Terraform Associate

Domain 5 - Remote State Management

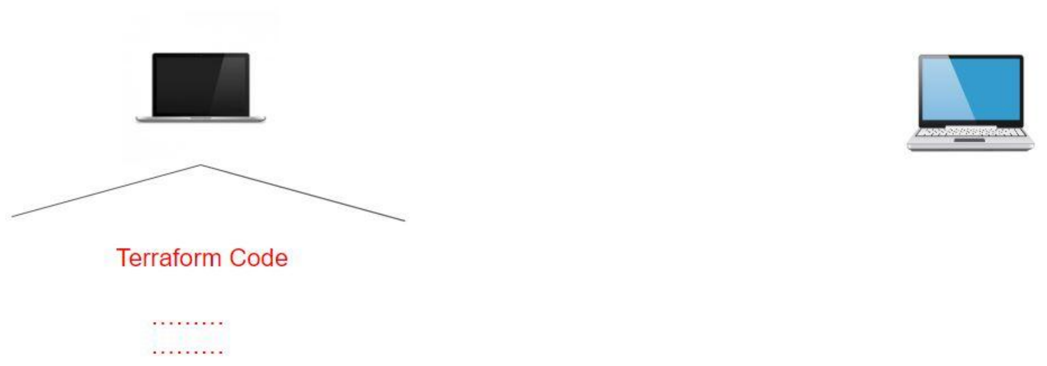
ISSUED BY
Zeal

REPRESENTATIVE
instructors@kplabs.in

Domain 5 - Remote State Management

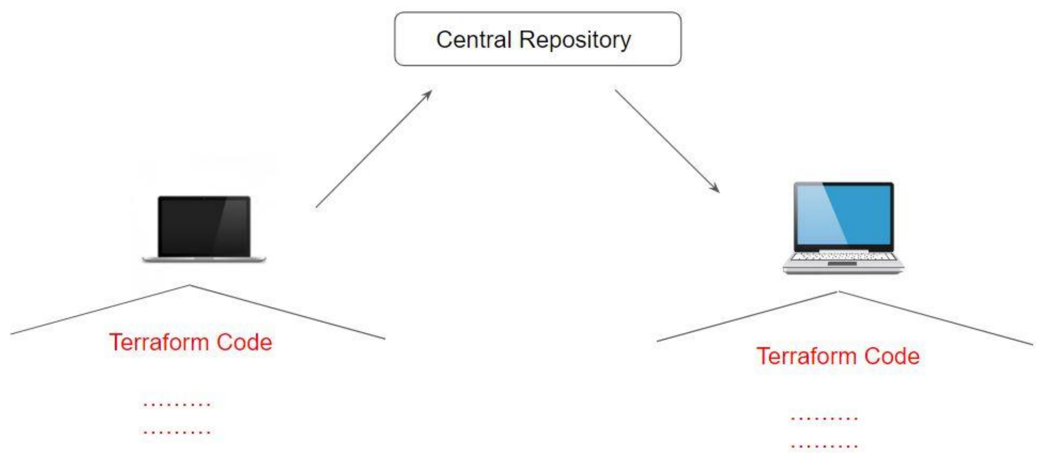
Module 1: Integrating with GIT for team management

Till now, we have been working with terraform code locally.



However, storing your configuration files locally is not always an idea specifically in the scenario were other members of the team are also working on Terraform.

For such cases, it is important to store your Terraorm code to a centralized repository like in Git.



Module 2: Module Sources in Terraform

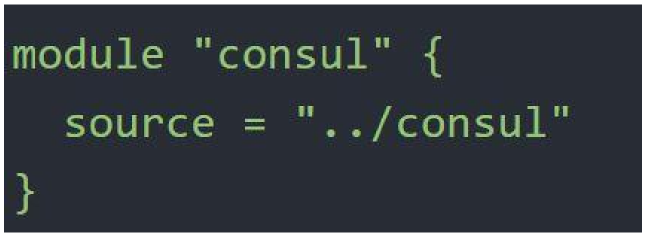
The source argument in a module block tells Terraform where to find the source code for the desired child module.

- Local paths
- Terraform Registry
- GitHub
- Bitbucket
- Generic Git, Mercurial repositories
- HTTP URLs
- S3 buckets
- GCS buckets

Let us explore some of the supported module sources.

2.1 Local Path

A local path must begin with either ./ or ../ to indicate that a local path is intended.



```
module "consul" {
  source = "../consul"
}
```

2.2 Git Module Source

Arbitrary Git repositories can be used by prefixing the address with the special git:: prefix.

After this prefix, any valid Git URL can be specified to select one of the protocols supported by Git.

.

```
module "vpc" {
  source = "git::https://example.com/vpc.git"
}

module "storage" {
  source = "git::ssh://username@example.com/storage.git"
}
```

2.3 Referencing to a Branch

By default, Terraform will clone and use the default branch (referenced by HEAD) in the selected repository.

You can override this using the ref argument:

```
module "vpc" {
  source = "git::https://example.com/vpc.git?ref=v1.2.0"
}
```

.
The value of the ref argument can be any reference that would be accepted by the git checkout command, including branch and tag names.

Module 3: Terraform & GitIgnore

The .gitignore file is a text file that tells Git which files or folders to ignore in a project.

.gitignore
conf/
*.artifacts
credentials

Depending on the environment, it is recommended to avoid committing certain files to GIT.

Files to Ignore	Description
.terraform	This file will be recreated when terraform init is run.
terraform.tfvars	Likely to contain sensitive data like usernames/passwords and secrets.
terraform.tfstate	Should be stored in the remote side.
crash.log	If terraform crashes, the logs are stored to a file named crash.log

Module 4: Remote State Management

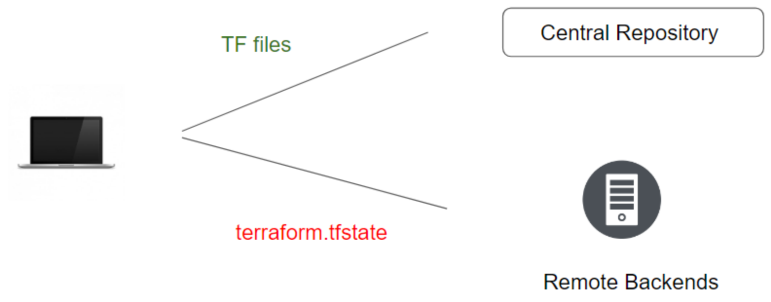
Terraform supports various types of remote backends which can be used to store state data.

As of now, we were storing state data in local and GIT repository.

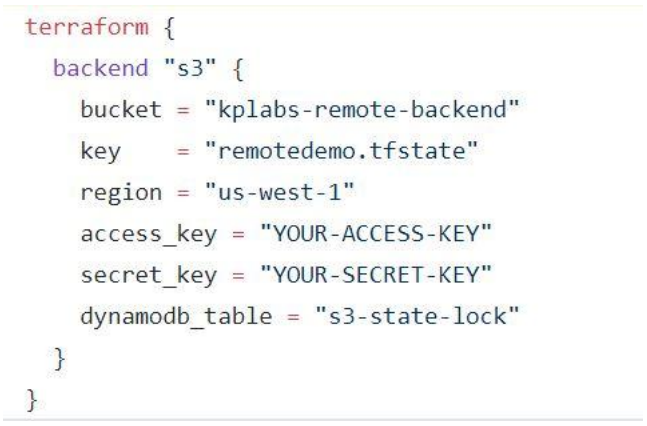
Depending on remote backends that are being used, there can be various features.

- Standard BackEnd Type: State Storage and Locking
- Enhanced BackEnd Type: All features of Standard + Remote Management

In the ideal scenario, your terraform configuration code should be part of the centralized Git repositories and state file should be part of the remote backends.



During our demo, we had made use of S3 Backend to store our state file. Following is the sample configuration file for the S3 backend:



```
terraform {
  backend "s3" {
    bucket = "kplabs-remote-backend"
    key    = "remotedemo.tfstate"
    region = "us-west-1"
    access_key = "YOUR-ACCESS-KEY"
    secret_key = "YOUR-SECRET-KEY"
    dynamodb_table = "s3-state-lock"
  }
}
```

Module 5: State File Locking

Whenever you are performing a write operation, terraform would lock the state file.

This is very important as otherwise during your ongoing terraform apply operations, if others also try for the same, it would corrupt your state file.

Example:

- Person A is terminating the RDS resource which has associated rds.tfstate file
- Person B has now tried resizing the same RDS resource at the same time.

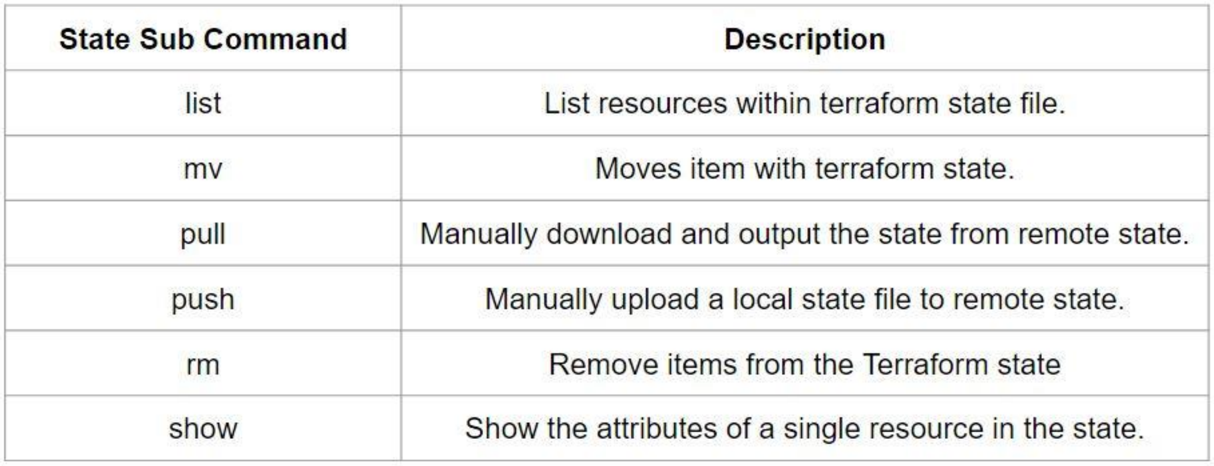
For the S3 backend, you can make use of the DynamoDB for state file locking functionality.

Module 6: Terraform State Management

As your Terraform usage becomes more advanced, there are some cases where you may need to modify the Terraform state.

It is important to never modify the state file directly. Instead, make use of terraform state command.

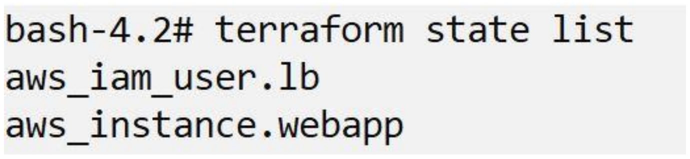
There are multiple sub-commands that can be used with terraform state, these include:



State Sub Command	Description
list	List resources within terraform state file.
mv	Moves item with terraform state.
pull	Manually download and output the state from remote state.
push	Manually upload a local state file to remote state.
rm	Remove items from the Terraform state
show	Show the attributes of a single resource in the state.

6.1 Sub Command - List

The terraform state list command is used to list resources within a Terraform state.



```
bash-4.2# terraform state list
aws_iam_user.lb
aws_instance.webapp
```

6.2 Sub Command - Move

The terraform state mv command is used to move items in a Terraform state.

This command is used in many cases in which you want to rename an existing resource without destroying and recreating it.

Due to the destructive nature of this command, this command will output a backup copy of the state prior to saving any changes

Overall Syntax:

terraform state mv [options] SOURCE DESTINATION

6.3 Sub Command - Pull

The terraform state pull command is used to manually download and output the state from a remote state.

This is useful for reading values out of state (potentially pairing this command with something like jq).

6.4 Sub Command - Push

The terraform state push command is used to manually upload a local state file to remote state.

This command should rarely be used.

6.5 Sub Command - Remove

The terraform state rm command is used to remove items from the Terraform state.

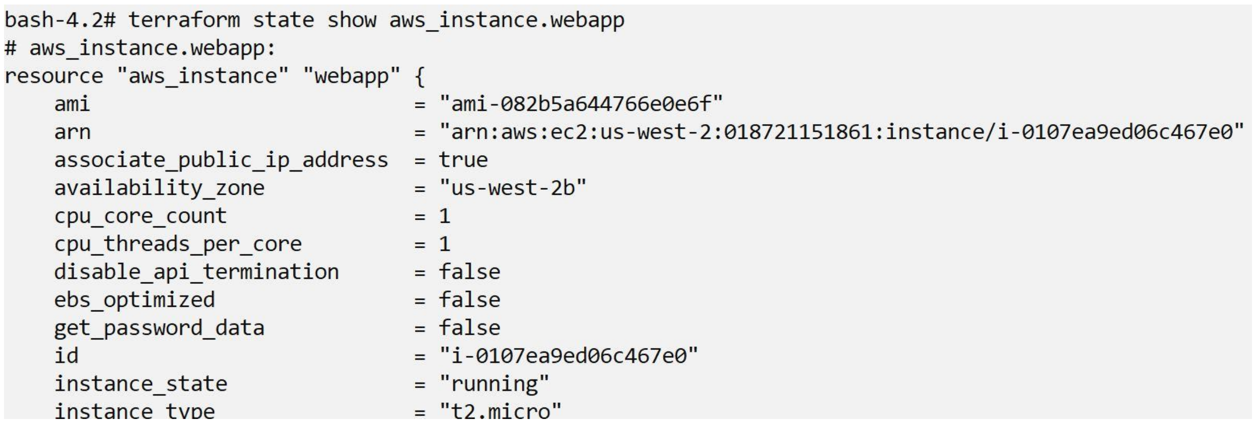
Items removed from the Terraform state are not physically destroyed.

Items removed from the Terraform state are only no longer managed by Terraform

For example, if you remove an AWS instance from the state, the AWS instance will continue running, but terraform plan will no longer see that instance.

6.6 Sub Command - Show

The terraform state show command is used to show the attributes of a single resource in the Terraform state.

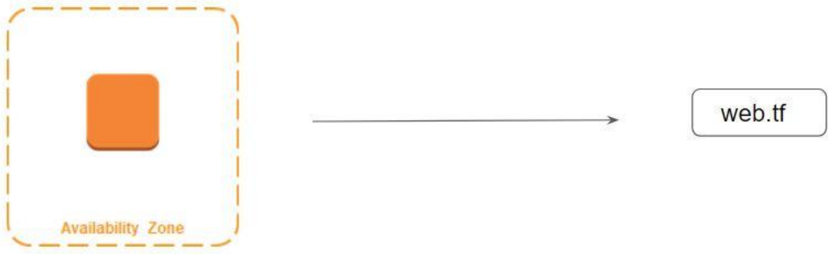


```
bash-4.2# terraform state show aws_instance.webapp
# aws_instance.webapp:
resource "aws_instance" "webapp" {
    ami                          = "ami-082b5a644766e0e6f"
    arn                          = "arn:aws:ec2:us-west-2:018721151861:instance/i-0107ea9ed06c467e0"
    associate_public_ip_address  = true
    availability_zone            = "us-west-2b"
    cpu_core_count               = 1
    cpu_threads_per_core         = 1
    disable_api_termination      = false
    ebs_optimized                = false
    get_password_data            = false
    id                           = "i-0107ea9ed06c467e0"
    instance_state               = "running"
    instance_type                = "t2.micro"
```

Module 7: Terraform Import

It might happen that there is a resource that is already created manually.

In such a case, any change you want to make to that resource must be done manually.



Terraform is able to import existing infrastructure. This allows you to take resources you've created by some other means and bring it under Terraform management.

The current implementation of Terraform import can only import resources into the state. It does not generate configuration. A future version of Terraform will also generate configuration.

Because of this, prior to running terraform import it is necessary to write manually a resource configuration block for the resource, to which the imported object will be mapped.