Terraform - DAY-4

Presented By:
Rashi Rana

Terrafor Backend

Basics of Backends

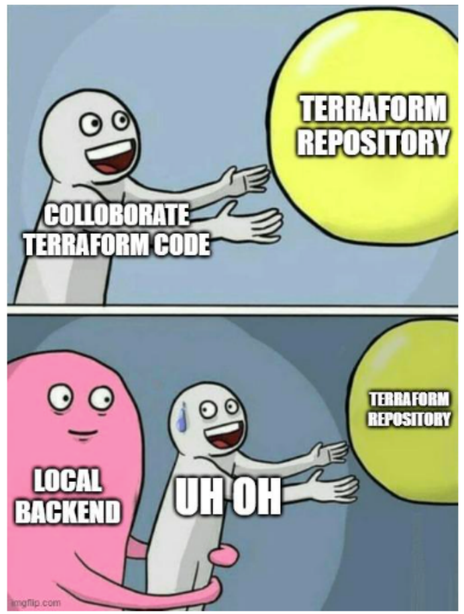
Backends primarily determine where Terraform stores its state.

By default, Terraform implicitly uses a backend called local to store state as a local file on disk.

Challenge with Local Backend

Nowadays Terraform project is handled and collaborated by an entire team.

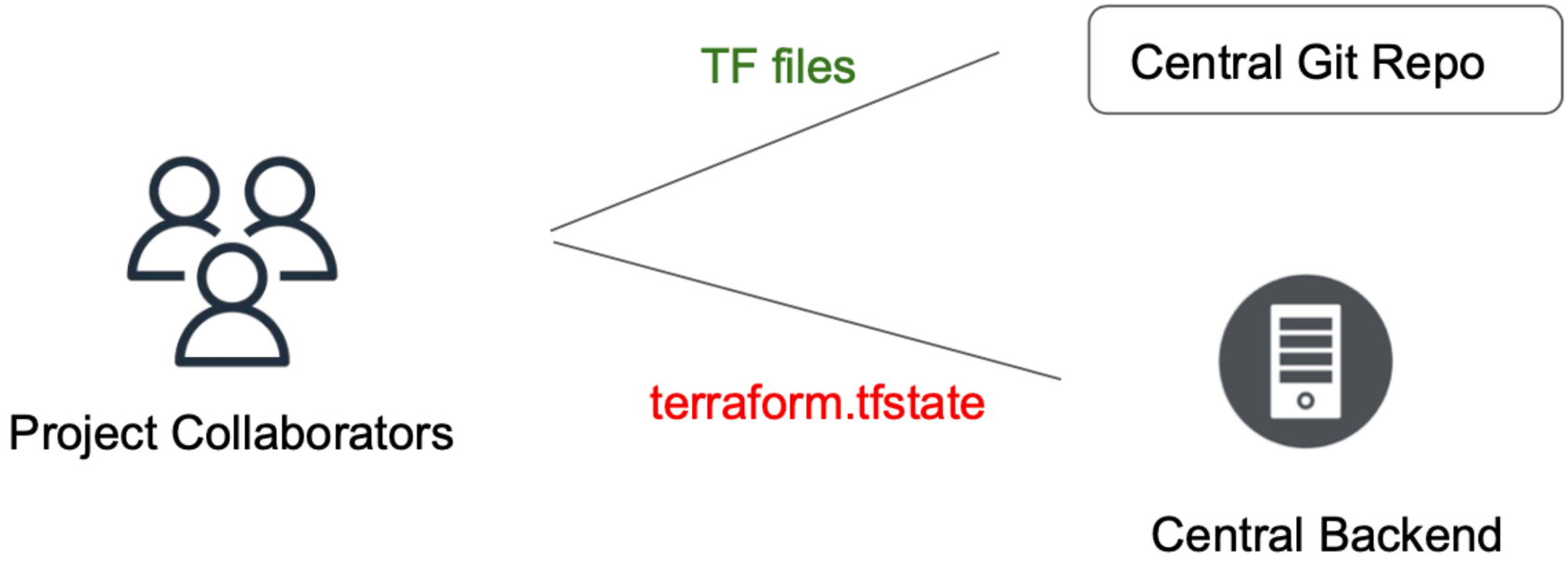
Storing the state file in the local laptop will not allow collaboration.



Ideal Architecture

Following describes one of the recommended architectures:

1. The Terraform Code is stored in Git Repository.
2. The State file is stored in a Central backend.



Backends Supported in Terraform

Terraform supports multiple backends that allows remote service related operations. Some of the popular backends include:

- S3
- Consul
- Azurerm
- Kubernetes
- HTTP
- ETCD

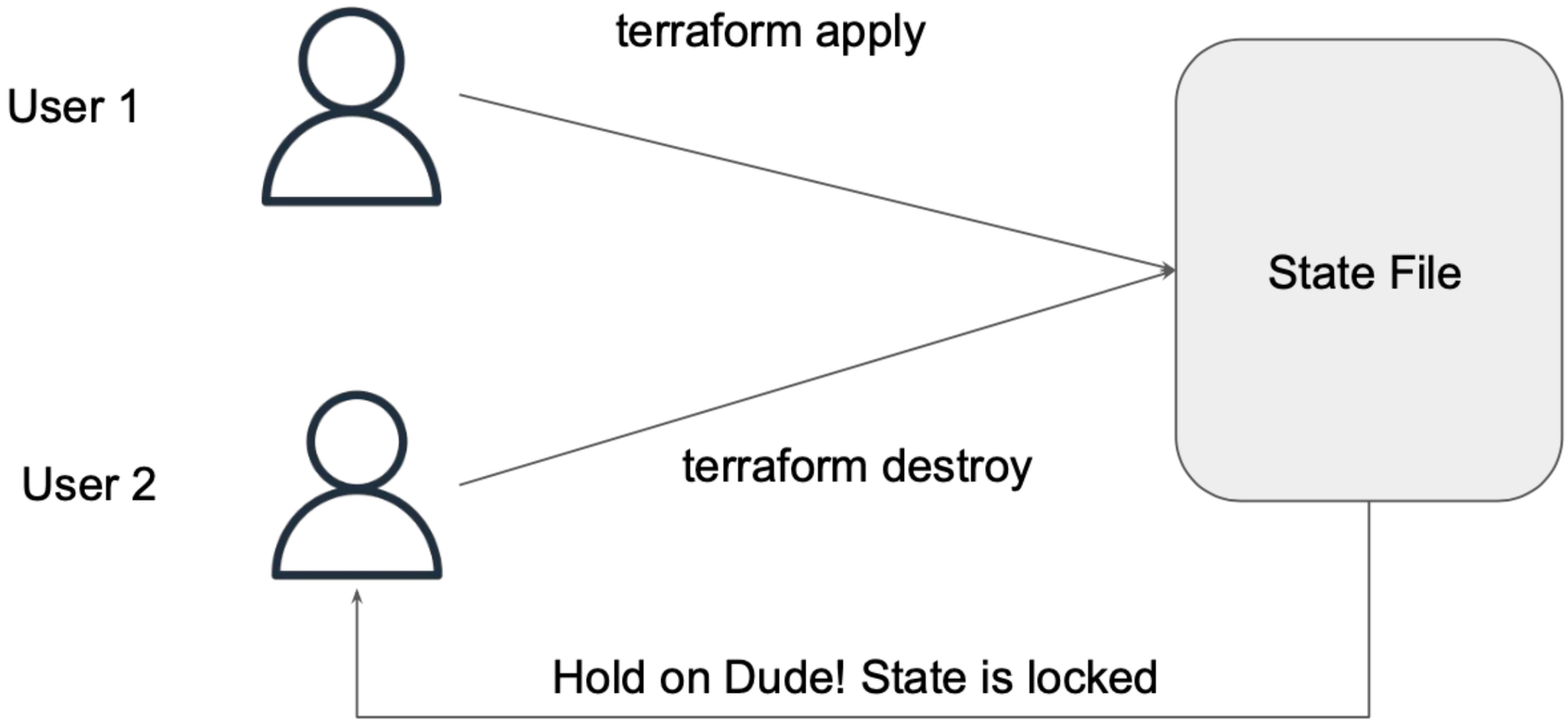
State Locking

Understanding State Lock

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 can corrupt your state file.

Basic Working



Important Note

State locking happens automatically on all operations that could write state. You won't see any message that it is happening

If state locking fails, Terraform will not continue

Not all backends support locking. The documentation for each backend includes details on whether it supports locking or not.

Force Unlocking State

Terraform has a force-unlock command to manually unlock the state if unlocking failed.

If you unlock the state when someone else is holding the lock it could cause multiple writers.

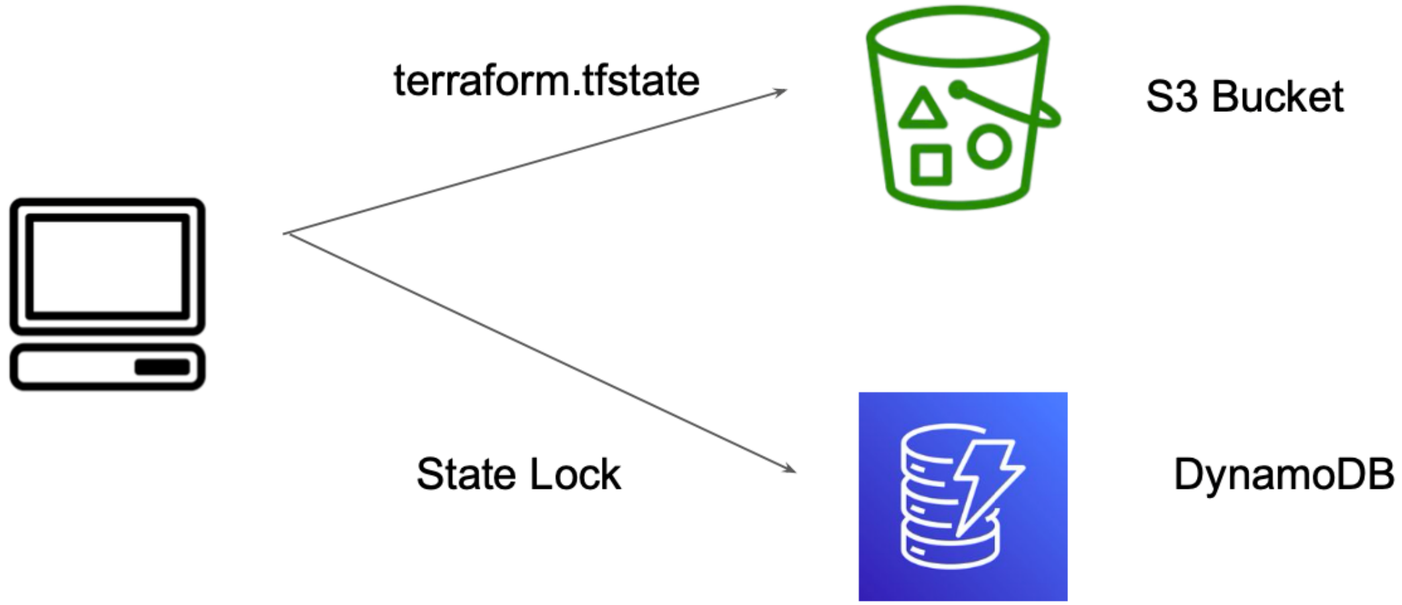
Force unlock should only be used to unlock your own lock in a situation where automatic unlocking failed.

State Locking in S3 Backend

State Locking in S3

By default, S3 does not support State Locking functionality.

You need to make use of DynamoDB table to achieve state locking functionality.



Terraform State Management

Overview of State Modification

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.

Overview of State Modification

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.

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
```

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

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.

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.

Sub Command - Show

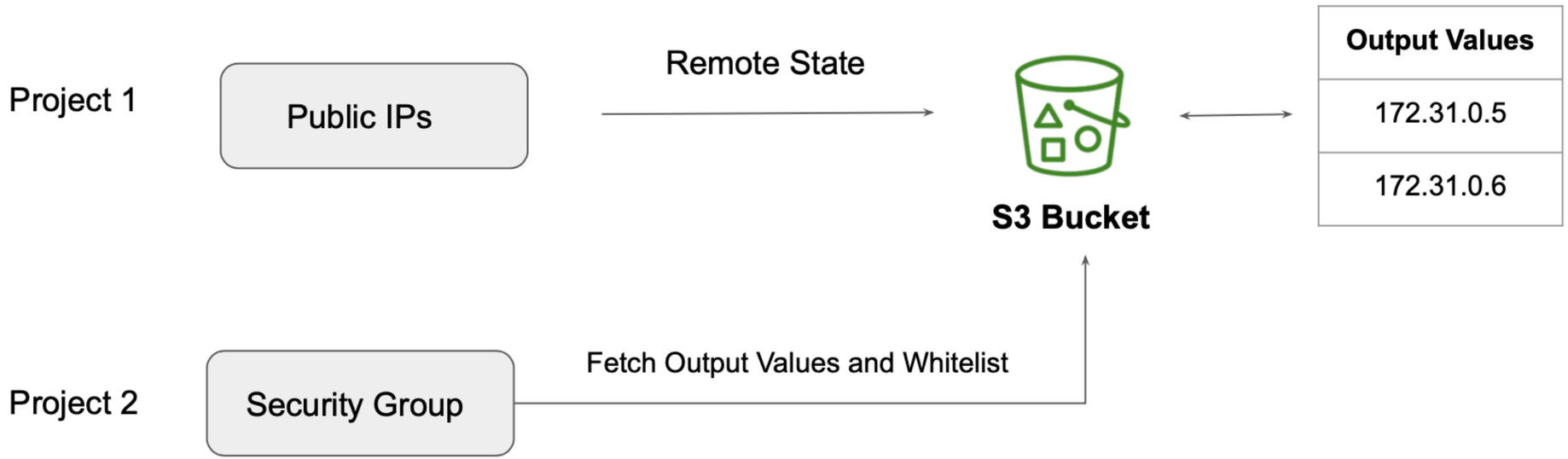
The terraform state show command is used to show the attributes of a single resource in the Terraform state.

```
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"
```

Connecting Remote States

Basics of Terraform Remote State

The **terraform_remote_state** data source retrieves the root module output values from some other Terraform configuration, using the latest state snapshot from the remote backend.

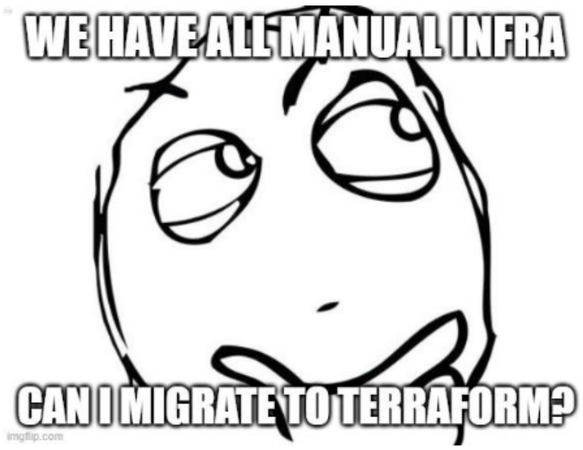


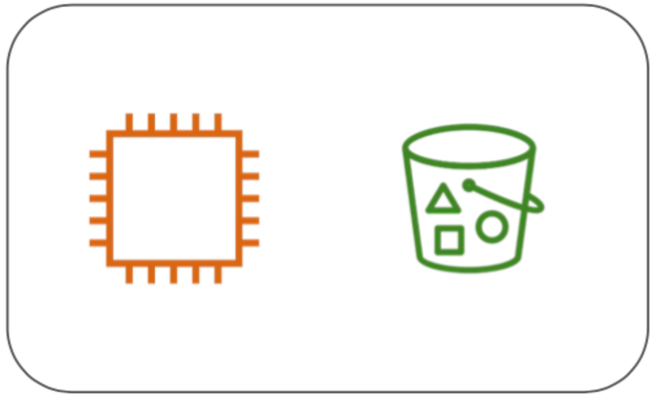
Terraform Import

Typical Challenge

It can happen that all the resources in an organization are created manually.

The organization now wants to start using Terraform and manage these resources via Terraform.



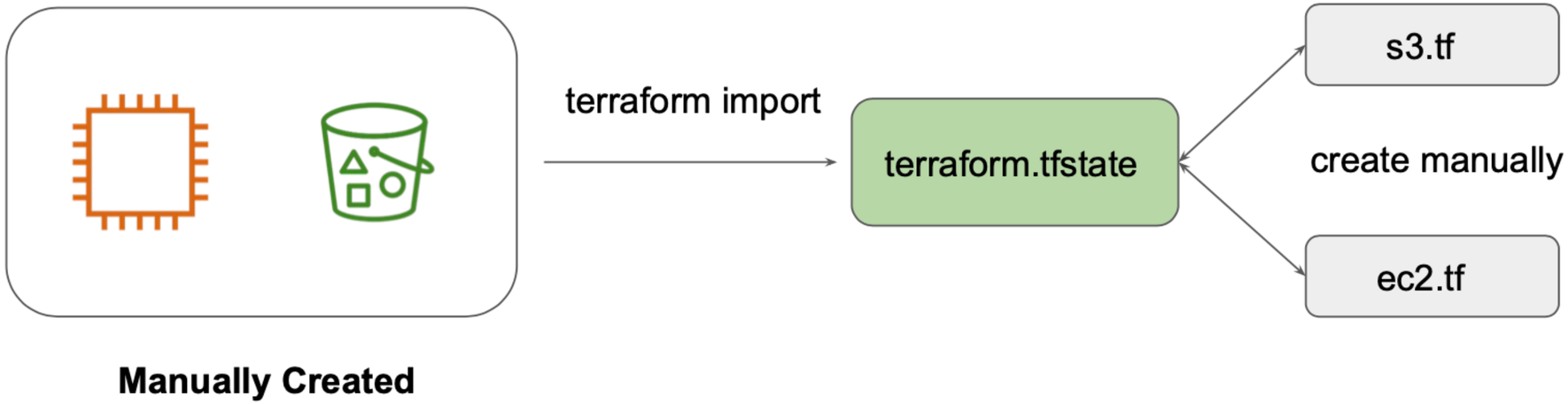


Manually Created

Terraform Import

Terraform is able to import existing infrastructure.

This allows you take resources you've created by some other means and bring it under Terraform management.



Important Pointer

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.

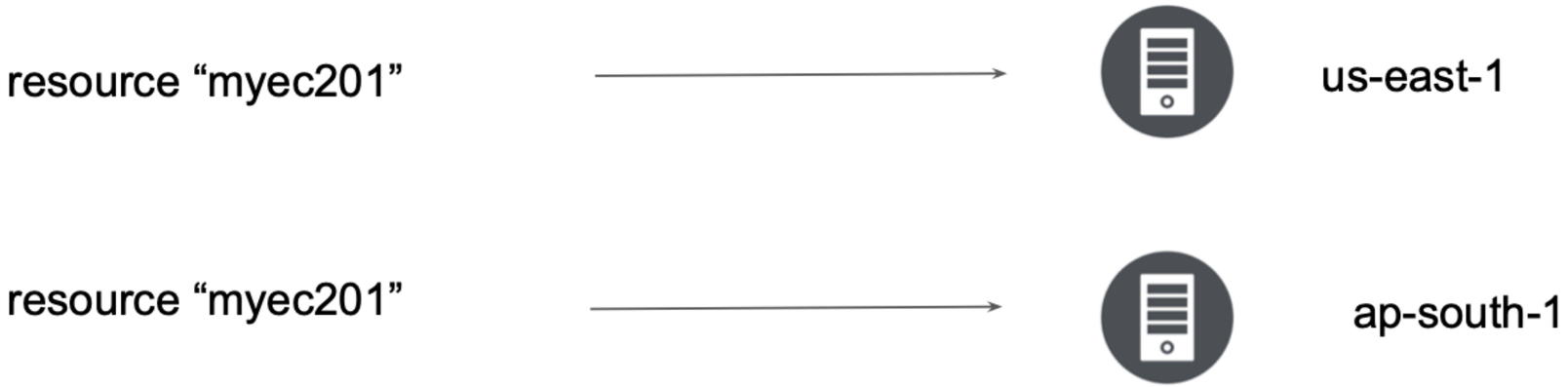
Because of this, prior to running terraform import it is necessary to write manually a resource configuration block for the resource, to which the imported object will be mapped.

Provider Configuration

Single Provider Multiple Configuration

Till now, we have been hardcoding the aws-region parameter within the providers.tf

This means that resources would be created in the region specified in the providers.tf file.



Single Provider Multiple Configuration

resource “myec201” →  Account 01

resource “myec201” →  Account 02

Sensitive Parameter

Overview of Sensitive Parameter

With organization managing their entire infrastructure in terraform, it is likely that you will see some sensitive information embedded in the code.

When working with a field that contains information likely to be considered sensitive, it is best to set the Sensitive property on its schema to true

```
output "db_password" {
  value       = aws_db_instance.db.password
  description = "The password for logging in to the database.
  sensitive   = true
}
```

Overview of Sensitive Parameter

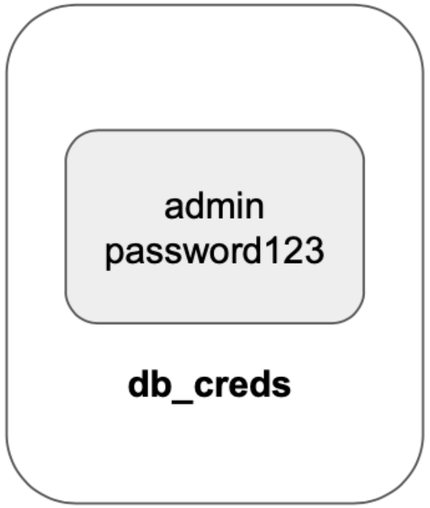
Setting the sensitive to “true” will prevent the field's values from showing up in CLI output and in Terraform Cloud

It will not encrypt or obscure the value in the state, however.

Vault Provider

Vault Provider

The Vault provider allows Terraform to read from, write to, and configure HashiCorp Vault.



Vault

Inject in Terraform

```
provider "vault" {
  address = "http://127.0.0.1:8200"
}

data "vault_generic_secret" "demo" {
  path = "secret/db-creds"
}
```

Important Note

Interacting with Vault from Terraform causes any secrets that you read and write to be persisted in both Terraform's state file.