HashiCorp Terraform Workspaces

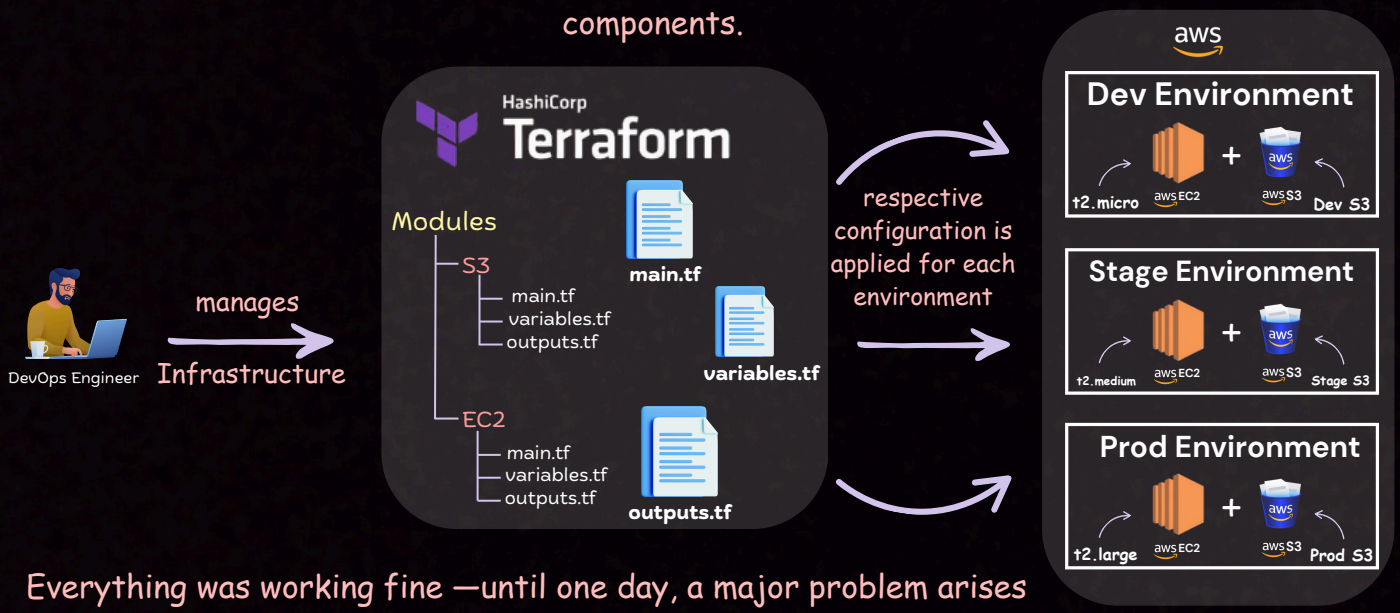
Terraform Workspaces

Rishabh Mishra

Imagine you are a DevOps Engineer and Developer Team 1 requests you to create resources EC2 and S3 in AWS

Now , this request has been generated frequently and also by different Teams

Instead of provisioning these resources manually each time, you decide to make your life easier by using Terraform modules to create reusable infrastructure components.



Everything was working fine —until one day, a major problem arises

PROBLEM :

The Development team needed an EC2 instance (t2.micro)
The Development team needed an EC2 instance (t2.medium)
The Development team needed an EC2 instance (t2.large)

Initially, you configured the Terraform files with t2.micro for staging.
But later, when the production team requested t2.large,
you updated the same Terraform configuration and applied the changes.

VPC
- main.tf
- variables.tf
- outputs.tf
- terraform.tfstate

overwritten every time!

Instead of creating a separate t2.large instance for production, Terraform replaced the existing t2.micro instance.

Solution → HashiCorp Terraform Workspaces

HashiCorp Terraform Workspaces

Terraform Workspaces

Rishabh Mishra

To solve this problem of managing statefile , you decided to use Terraform Workspaces,

Terraform Workspaces are a built-in feature of Terraform that allow you to manage multiple environments (such as development, staging, and production) within the same Terraform configuration.

How Terraform Workspaces Helped DevOps Engineer ?

STEP 1 Initialize Terraform :

First, you initialize Terraform in your working directory

`terraform init` ← This sets up the Terraform backend and prepares it for further commands.

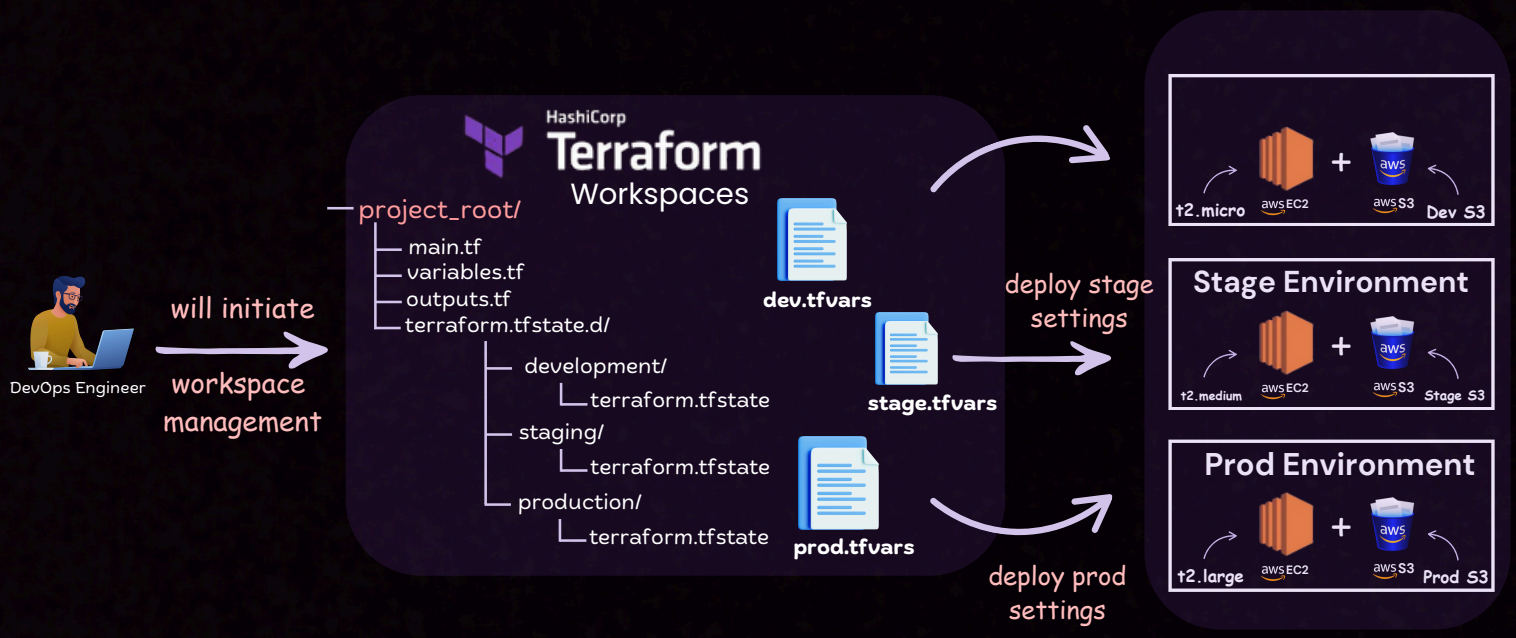
STEP 2 Create Separate Workspaces :

Then you create a seperate workspace for Dev,Stage and Prod Environment

- `terraform workspace new dev` ← Creates a workspace for Development
- `terraform workspace new staging` ← Creates a workspace for Staging
- `terraform workspace new production` ← Creates a workspace for Production

STEP 3 Verify the Active Workspaces :

`terraform workspace show` ← Then you check which workspace is active (e.g., dev, staging, production)



HashiCorp Terraform Workspaces

Terraform Workspaces

Rishabh Mishra

STEP 4 Use Dynamic Variables :

```
variable "instance_type" {
  type    = string
  default = "t2.micro"
}
```

This declares a variable named instance_type with a default value of "t2.micro" which can also be used for dev environment

If terraform apply is run without specifying - var="instance_type=t2.large", it will default to t2.micro

```
resource "aws_instance" "example" {
  ami           = "ami-12345678"
  instance_type = var.instance_type
  tags = {
    Name = "my-ec2-instance-${terraform.workspace}"
  }
}
```

Since terraform.workspace automatically returns the current workspace name,
the instance names become unique for each environment
(e.g., my-ec2-instance-staging, my-ec2-instance-production).

```
resource "aws_s3_bucket" "example" {
  bucket = "my-bucket-${terraform.workspace}"
  acl    = "private"
}
```

Creates an S3 bucket with a unique name for each workspace.

Uses terraform.workspace to ensure that each environment gets a separate S3 bucket.
(e.g., my-bucket-staging, my-bucket-production).

STEP 5 Apply Configuration Per Workspace :

This command switches the active workspace to staging

```
terraform workspace select staging
terraform apply -var="instance_type=t2.medium"
```

This applies Terraform changes in the staging workspace.
The variable instance_type is set to "t2.medium",
meaning the EC2 instance in staging will be of type t2.medium.

This command switches the active workspace to staging

```
terraform workspace select production
terraform apply -var="instance_type=t2.large"
```

This applies Terraform changes in the production workspace.
The variable instance_type is set to "t2.medium",
meaning the EC2 instance in staging will be of type t2.medium.

CONCLUSION

Use Case	Terraform Workspaces	Separate Directories
Minimal configuration differences between environments	✅	❌
Completely different configurations per environment	❌	✅
Strict isolation of state files	✅	✅