15 Days of Terraform – Curated by Shaik Hari Sadia Anjum

Terraform Workspaces

Terraform workspaces allow you to manage multiple instances of a given set of Terraform configurations. Each workspace has its own state file, which makes it useful for managing environments like dev, staging, and prod without duplicating code.

🎯 Use Case Example

Imagine you have a main.tf that creates an S3 bucket. Instead of copying this code for each environment, you can use workspaces to manage dev, stage, and prod using a single Terraform configuration, each with isolated state.

How Workspaces Work

- **Default workspace**: Every Terraform project starts with a default workspace.
- Each workspace **has a separate state file**, stored under .terraform/ (local) or using a suffix in remote backends.

⚙️ Basic Workspace Commands

Command	Description
terraform workspace list	**Lists all workspaces**
terraform workspace new dev	**Creates a new workspace named dev**
terraform workspace select dev	**Switches to the dev workspace**
terraform workspace show	**Shows the current workspace**
terraform workspace delete dev	**Deletes the dev workspace**

Example Usage

1. Create and Select Workspace:

terraform workspace new dev

terraform workspace select dev

2. Reference Workspace in Code:

Use terraform.workspace in your config to apply conditional logic.

```
resource "aws_s3_bucket" "example" {

 bucket = "my-bucket-${terraform.workspace}"

 acl    = "private"

}
```

So, in the dev workspace it becomes my-bucket-dev, and in prod, it becomes my-bucket-prod.

Workspaces with Remote Backends

With remote backends (like S3 + DynamoDB), Terraform appends the workspace name to the state file path:

```
terraform {

 backend "s3" {

  bucket = "my-terraform-states"

  key    = "env/terraform.tfstate" # workspace will append here

  region = "us-east-1"

 }

}
```

In dev workspace, the actual key becomes env/dev/terraform.tfstate.

Benefits

- **Avoids code duplication across environments.**
- **Keeps state isolated by environment.**
- **Works well with automation pipelines and remote backends.**

Limitations:

- Not a full substitute for **separate Terraform projects** in large-scale deployments.
- Limited use in modules – doesn't allow completely separate variables or providers unless handled carefully.
- Workspaces only isolate **state**, not variables – you must still manage variable overrides per workspace manually (e.g., via *.tfvars).

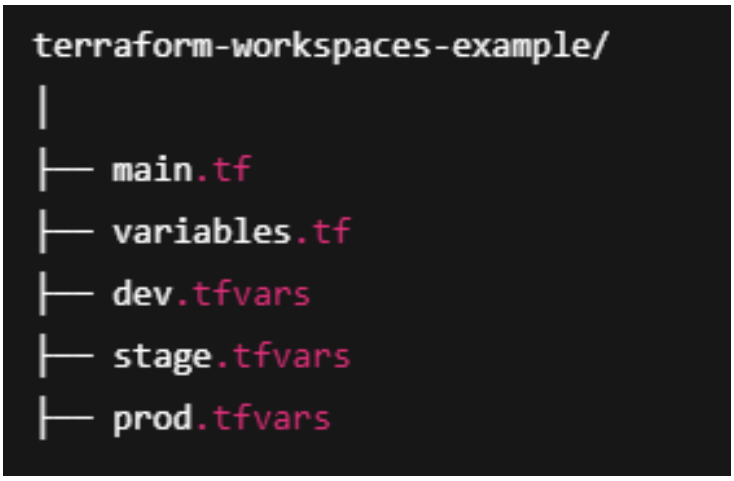
Best Practices

- Use for **simple environment separation** (dev/stage/prod).
- Use terraform.workspace for naming resources dynamically.
- Combine with terraform.tfvars files and automation to control variable differences.

Here's a complete example project showing how to use Terraform workspaces to manage multiple environments (dev, stage, prod) using:

- **Workspaces (terraform.workspace)**
- **Variable files (dev.tfvars, stage.tfvars, prod.tfvars)**
- **A single shared configuration (main.tf)**

Project Structure:



main.tf:

```
provider "aws" {
 region = var.aws_region
}

resource "aws_s3_bucket" "workspace_bucket" {
 bucket = "demo-${terraform.workspace}-bucket-12345"
 acl    = "private"

 tags = {
  Environment = terraform.workspace
 }
}
```

variables.tf:

```
variable "aws_region" {
  description = "AWS Region"
  type        = string
}
```

dev.tfvars:

```
aws_region = "us-east-1"
```

stage.tfvars:

```
aws_region = "us-west-1"
```

prod.tfvars:

```
aws_region = "eu-west-1"
```

How to Use It

Step 1: Initialize Terraform

terraform init

Step 2: Create and switch to a workspace

terraform workspace new dev

terraform workspace select dev

Step 3: Apply configuration with the right vars

terraform apply -var-file="dev.tfvars"

This will create an S3 bucket named something like:
demo-dev-bucket-12345 in region us-east-1

Repeat for Other Environments

```
terraform workspace new stage
terraform workspace select stage
terraform apply -var-file="stage.tfvars"

terraform workspace new prod
terraform workspace select prod
terraform apply -var-file="prod.tfvars"
```

Each workspace will get a separate bucket with its own state file and configuration.

Cleanup

You can destroy individual environments like this:

terraform workspace select dev

terraform destroy -var-file="dev.tfvars"