Lab 16 – Terraform Workspaces (Managing Multiple Environments: Dev, QA, Prod)

Creator: Sandeep Kumar Sharma

Learning Objectives

- Understand what Terraform **workspaces** are.
- Learn why workspaces are used in multi-environment setups.
- Learn how to create, select, and manage Terraform workspaces.
- Deploy separate infrastructure for Dev, QA, and Prod using workspaces.
- Learn the difference between workspaces and directories.

Learning Outcome

By the end of this lab, the learner will be able to: - Create and switch Terraform workspaces. - Use `terraform.workspace` to dynamically decide resource names. - Maintain multiple environments with a single Terraform configuration. - Avoid code duplication between Dev/QA/Prod.

Concept Explanation (Natural Style)

Imagine your company has: - **Dev environment** for developers - **QA environment** for testers - **Prod environment** for customers

All these environments need **similar infrastructure**: - EC2 - S3 - Security groups - VPC

But traditionally, people copy-paste folders:

```
terraform-dev/
terraform-qa/
terraform-prod/
```

This becomes a nightmare to manage.

Terraform provides a smarter solution → **Workspaces**.

👉What is a Terraform Workspace?

A workspace allows you to run **multiple environments** from the **same Terraform code**, but maintain **separate state files**.

You can have: - default (Dev) - qa - prod

Each workspace has its **own tfstate**, meaning resources stay separate.

👉Why use workspaces?

- Avoid duplicate Terraform folders
- Same code for all environments
- Separate state files for safety
- Easy switching between environments

This is perfect for real enterprise use.

😠Part 1 – Setup Project

```
mkdir terraform-lab16-workspaces
cd terraform-lab16-workspaces
```

Create file:

```
touch main.tf
```

😠Part 2 – Terraform Code Using Workspace Name

Open **main.tf** and add:

```
provider "aws" {
  region = "ap-south-1"
}

# ---------------------------------
# Use workspace to decide instance name
# ---------------------------------
```

```
locals {
  instance_name = "lab16-ec2-${terraform.workspace}"
  instance_type = terraform.workspace == "prod" ? "t3.medium" : "t2.micro"
}

resource "aws_instance" "lab16" {
  ami           = "ami-052c08d70def0ac62"
  instance_type = local.instance_type

  tags = {
    Name        = local.instance_name
    Environment = terraform.workspace
  }
}

output "instance_name" {
  value = local.instance_name
}

output "environment" {
  value = terraform.workspace
}
```

😠Explanation

👉🏽terraform.workspace

This fetches the workspace name: - dev - qa - prod

We used it to: - choose instance type - set name dynamically - tag the EC2

👉🏽Different Workspaces = Different State Files

Each workspace keeps **its own infrastructure separate**.

😠Part 3 – Initialize Terraform

```
terraform init
```

😠Part 4 – Create Workspaces

Create a QA workspace

```
terraform workspace new qa
```

Create a Prod workspace

```
terraform workspace new prod
```

List all workspaces

```
terraform workspace list
```

You will see:

```
* default
  qa
  prod
```

😠Part 5 – Deploy in Dev (default)

```
terraform apply
```

It will create:

```
Name = lab16-ec2-default
Type = t2.micro
Environment = default
```

😠Part 6 – Deploy in QA

```
terraform workspace select qa
terraform apply
```

QA will get:

```
Name = lab16-ec2-qa
Type = t2.micro
Environment = qa
```

😠Part 7 – Deploy in Prod

```
terraform workspace select prod
terraform apply
```

Prod will get:

```
Name = lab16-ec2-prod
Type = t3.medium  (because of condition)
Environment = prod
```

Now you have **3 different EC2 instances**, all from the same code, but using different workspaces.

😠Part 8 – Validate in AWS Console

Search EC2 Instances: - lab16-ec2-default - lab16-ec2-qa - lab16-ec2-prod

Each will have: - Different name - Different tags - Different instance types (prod uses bigger machine)

This is the real power of workspaces.

😠Part 9 – Destroy Each Environment Separately

Dev:

```
terraform workspace select default
terraform destroy
```

QA:

```
terraform workspace select qa
terraform destroy
```

Prod:

```
terraform workspace select prod
terraform destroy
```

Summary

You learned: - What Terraform workspaces are - How to create Dev/QA/Prod environments from the same code - How Terraform keeps different states for each workspace - How to use workspace names inside resource logic - How to deploy and destroy environment-specific infrastructure

Workspaces are extremely important in real-world DevOps and Terraform automation.

In the next lab, we will explore **Terraform lifecycle rules (create_before_destroy & prevent_destroy)** to control resource behavior.

End of Lab 16