Lab: Terraform Enterprise - Workspaces

Those who adopt Terraform typically want to leverage the principles of DRY (Don't Repeat Yourself) development practices. One way to adopt this principle with respect to IaC is to utilize the same code base for different environments (development, quality, production, etc.)

Workspaces is a Terraform feature that allows us to organize infrastructure by environments and variables in a single directory.

Terraform is based on a stateful architecture and therefore stores state about your managed infrastructure and configuration. This state is used by Terraform to map real world resources to your configuration, keep track of metadata, and to improve performance for large infrastructures.

The persistent data stored in the state belongs to a Terraform workspace. Initially the backend has only one workspace, called "default", and thus there is only one Terraform state associated with that configuration.

- Task 1: Using Terraform Workspaces (Open Source)
- Task 2: Create a new Terraform Workspace for Production State
- Task 3: Deploy Infrastructure within the Terraform production workspace
- Task 4: Changing between Workspaces
- Task 5: Utilizing the ${terraform.workspace} interpolation sequence within your configuration

Task 1: Using Terraform Workspaces (Open Source)

Terraform starts with a single workspace named "default". This workspace is special both because it is the default and also because it cannot ever be deleted. If you've never explicitly used workspaces, then you've only ever worked on the "default" workspace.

You can check the Terraform workspace you are in with the `terraform workspace` command.

```
terraform workspace show
```

```
default
```

To see a list of the options you have within the terraform workspace command issue a `terraform workspace -help`

```
terraform workspace -help
```

```
Usage: terraform [global options] workspace
```


```
  new, list, show, select and delete Terraform workspaces.

Subcommands:
    delete    Delete a workspace
    list      List Workspaces
    new       Create a new workspace
    select    Select a workspace
    show      Show the name of the current workspace
```

We will utilize the current **default** workspace to store the state information for our us-west-2 aws region.

Task 2: Create a new Terraform Workspace for Production State

Leveraging the same code base let's perform a production deployment into the us-east-1 region while not affecting the infrastructure that was built out in our **default** workspace.

To begin, let's create a new terraform workspace called prod

```
terraform workspace new prod
```

```
Created and switched to workspace "prod"!

You're now on a new, empty workspace. Workspaces isolate their state, so if you run "ter
for this configuration.
```

You can validate that we are no longer in the **default** workspace by issuing a terraform workspace show command

```
terraform workspace show
```

```
prod
```

You can also see that the state is empty for this workspace by issuing a terraform show

```
terraform show
```

```
No state.
```

Task 3: Deploy Infrastructure within the Terraform production workspace

Modify your main.tf to change the region of the aws provider block to us-east-1.

`main.tf`

```
# Configure the AWS Provider
provider "aws" {
  region = "us-east-1"
  default_tags {
    tags = {
      Owner       = "Acme"
      Provisoned = "Terraform"
    }
  }
}
```

Save your file and issue a `terraform plan` to see Terraform's dry run for execution

```
terraform plan
```

```
Plan: 26 to add, 0 to change, 0 to destroy.
```

Because the state information is new for the `prod` workspace, Terraform will go and deploy all of the infrastructure declared in the configuration now into `us-east-1` which is our production region.

```
terraform apply
```

```
Do you want to perform these actions in workspace "prod"?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes
```

Task 4: Changing between Workspaces

To move between terraform workspaces you can use the `terraform workspace` command.

To move back to the infrastructure residing in `us-west-2` issue:

```
terraform workspace select default
```

To move back to the infrastructure residing in `us-east-1` issue:

```
terraform workspace select prod
```

You can issue a `terrafrom show` in either workspace to see the resources that each of the workspaces is managing.

Task 5: Utilizing the ${terraform.workspace} interpolation sequence within your configuration

Now that we see the benefit of isolating our resource state information using terraform workspaces, we may wish to reflect the workspace name information into our code base.

Within your Terraform configuration, you may include the name of the current workspace using the ${terraform.workspace} interpolation sequence.

Modify the environment default tag for `main.tf` inside the AWS provider to reflect the terraform workspace name

```
provider "aws" {
  region = "us-east-1"
  default_tags {
    tags = {
      Environment = terraform.workspace
      Owner       = "TF Hands On Lab"
      Project     = "Infrastructure as Code"
      Terraform   = "true"
    }
  }
}
```

```
  # aws_vpc.vpc will be updated in-place
  ~ resource "aws_vpc" "vpc" {
        id                                   = "vpc-00d26110d48784808"
      ~ tags                                 = {
          ~ "Environment" = "demo_environment" -> "prod"
            # (2 unchanged elements hidden)
        }
      ~ tags_all                             = {
          ~ "Environment" = "demo_environment" -> "prod"
            # (2 unchanged elements hidden)
        }
        # (14 unchanged attributes hidden)
    }
```

Reference

Terraform Workflow