Lab: Terraform Basics

All interactions with Terraform occur via the CLI. Terraform is a local tool (runs on the current machine). The terraform ecosystem also includes providers for many cloud services, and a module repository. Hashicorp also has products to help teams manage Terraform: Terraform Cloud and Terraform Enterprise.

There are a handful of basic terraform commands, including:

- `terraform init`
- `terraform validate`
- `terraform plan`
- `terraform apply`
- `terraform destroy`

These commands make up the terraform workflow that we will cover in objective 6 of this course. It will be beneficial for us to explore some basic commands now so that work alongside and deploy our configurations.

- Task 1: Verify Terraform installation and version
- Task 2: Initialize Terraform Working Directory: `terraform init`
- Task 3: Validating a Configuration: `terraform validate`
- Task 4: Genenerating a Terraform Plan: `terraform plan`
- Task 5: Applying a Terraform Plan: `terraform apply`
- Task 6: Terraform Destroy: `terraform destroy`

Task 1: Verify Terraform installation and version

You can get the version of Terraform running on your machine with the following command:

```
terraform -version
```

If you need to recall a specific subcommand, you can get a list of available commands and arguments with the help argument.

```
terraform -help
```


Task 2: Terraform Init

Initializing your workspace is used to initialize a working directory containing Terraform configuration files.

Copy the code snippet below into the file called `main.tf` This snippet leverages the random provider, maintained by HashiCorp, to generate a random string.

`main.tf`

```
resource "random_string" "random" {
  length = 16
}
```

Once saved, you can return to your shell and run the init command shown below. This tells Terraform to scan your code and download anything it needs locally.

```
terraform init
```

Once your Terraform workspace has been initialized you are ready to begin planning and provisioning your resources.

> Note: You can validate that your workspace is initialized by looking for the presence of a `.terraform` directory. This is a hidden directory, which Terraform uses to manage cached provider plugins and modules, record which workspace is currently active, and record the last known backend configuration in case it needs to migrate state. This directory is automatically managed by Terraform, and is created during initialization.

Task 3: Validating a Configuration

The terraform validate command validates the configuration files in your working directory.

To validate there are no syntax problems with our terraform configuration file run a

```
terraform validate
```

```
Success! The configuration is valid.
```

Task 4: Genenerating a Terraform Plan

Terraform has a dry run mode where you can preview what Terraform will change without making any actual changes to your infrastructure. This dry run is performed by running a `terraform plan`.

In your terminal, you can run a plan as shown below to see the changes required for Terraform to reach your desired state you defined in your code. This is equivalent to running Terraform in a “dry” mode.

```
terraform plan
```

If you review the output, you will see 1 change will be made which is to generate a single random string.

```
Terraform will perform the following actions:

  # random_string.random will be created
  + resource "random_string" "random" {
      + id          = (known after apply)
      + length      = 16
      + lower       = true
      + min_lower   = 0
      + min_numeric = 0
      + min_special = 0
      + min_upper   = 0
      + number      = true
      + result      = (known after apply)
      + special     = true
      + upper       = true
    }

Plan: 1 to add, 0 to change, 0 to destroy.
```

> Note: Terraform also has the concept of planning out changes to a file. This is useful to ensure you only apply what has been planned previously. Try running a plan again but this time passing an -out flag as shown below.

```
terraform plan -out myplan
```

This will create a plan file that Terraform can use during an `apply`.

Task 5: Applying a Terraform Plan

Run the command below to build the resources within your plan file.

```
terraform apply myplan
```

Once completed, you will see Terraform has successfully built your random string resource based on what was in your plan file.

Terraform can also run an `apply` without a plan file. To try it out, modify your `main.tf` file to create a random string with a length of 10 instead of 16 as shown below:

```
resource "random_string" "random" {
  length = 10
}
```

and run a `terraform apply`

```
terraform apply
```

Notice you will now see a similar output to when you ran a `terraform plan` but you will now be asked if you would like to proceed with those changes. To proceed enter `yes`.

```
Terraform used the selected providers to generate the following execution plan. Resource
following symbols:
-/+ destroy and then create replacement

Terraform will perform the following actions:

  # random_string.random must be replaced
-/+ resource "random_string" "random" {
      ~ id          = "XW>5m{w8Ig96d1A&" -> (known after apply)
      ~ length      = 16 -> 10 # forces replacement
      ~ result      = "XW>5m{w8Ig96d1A&" -> (known after apply)
        # (8 unchanged attributes hidden)
    }

Plan: 1 to add, 0 to change, 1 to destroy.

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

  Enter a value:
```

Once complete the random string resource will be created with the attributes specified in the `main.tf` configuration file.

Task 6: Terraform Destroy

The `terraform destroy` command is a convenient way to destroy all remote objects managed by a particular Terraform configuration. It does not delete your configuration file(s), `main.tf`, etc. It destroys the resources built from your Terraform code.

Run the command as shown below to run a planned destroy:

```
terraform plan -destroy
```

```
Terraform will perform the following actions:

  # random_string.random will be destroyed
  - resource "random_string" "random" {
      - id          = "1HIQs)moC0" -> null
      - length      = 10 -> null
      - lower       = true -> null
      - min_lower   = 0 -> null
      - min_numeric = 0 -> null
      - min_special = 0 -> null
      - min_upper   = 0 -> null
      - number      = true -> null
      - result      = "1HIQs)moC0" -> null
      - special     = true -> null
      - upper       = true -> null
    }

Plan: 0 to add, 0 to change, 1 to destroy.
```

You will notice that it is planning to destroy your previously created resource. To actually destroy the random string you created, you can run a destroy command as shown below.

```
terraform destroy
```

```
Terraform will perform the following actions:

  # random_string.random will be destroyed
  - resource "random_string" "random" {
      - id          = "1HIQs)moC0" -> null
      - length      = 10 -> null
      - lower       = true -> null
      - min_lower   = 0 -> null
      - min_numeric = 0 -> null
      - min_special = 0 -> null
      - min_upper   = 0 -> null
      - number      = true -> null
      - result      = "1HIQs)moC0" -> null
      - special     = true -> null
      - upper       = true -> null
    }

Plan: 0 to add, 0 to change, 1 to destroy.

Do you really want to destroy all resources?
```


```
Terraform will destroy all your managed infrastructure, as shown above.
There is no undo. Only 'yes' will be accepted to confirm.

Enter a value:
```

> Note: As similar to when you ran an apply, you will be prompted to proceed with the destroy by entering “yes”.