Lab: Initializing Terraform with `terraform init`

The `terraform init` command is used to initialize a working directory containing Terraform configuration files. This is the first command that should be run after writing a new Terraform configuration or cloning an existing one from version control.

- Task 1: Initialize a Terraform working directory
- Task 2: Re-initialize after adding a new provider
- Task 3: Re-initialize after adding a new module
- Task 4: Re-initialize after modifying a Terraform backend
- Task 5: Other initialization steps/considerations

Task 1: Initialize a Working Directory

You must run `terraform init` to initialize a new Terraform working directory. 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_pet" "server" {
  length = 2
}
```

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 2: Re-initialize after add a new provider

You must run terraform init to initialize a new Terraform working directory, and after changing provider, module or backend configuration. You do not need to run terraform init before each command and it always safe to run multiple times.

Let's add anew provider for use within our `terraform.tf` file. Let's add the `Azure` provider:

`terraform.tf`

```
terraform {
  required_version = ">= 1.0.0"
  required_providers {
    aws = {
      source = "hashicorp/aws"
    }
    http = {
      source  = "hashicorp/http"
      version = "2.1.0"
    }
    random = {
      source  = "hashicorp/random"
      version = "3.1.0"
    }
    local = {
      source  = "hashicorp/local"
      version = "2.1.0"
    }
    tls = {
      source  = "hashicorp/tls"
      version = "3.1.0"
    }
    azurerm = {
      source = "hashicorp/azurerm"
      version = "2.84.0"
    }
  }
}
```

Before re-initializing our working directory, let's verify that the `Azure` provider is not yet in use:

```
terraform providers
```

```
Providers required by configuration:
.
|-- provider[registry.terraform.io/hashicorp/aws]
|-- provider[registry.terraform.io/hashicorp/http] 2.1.0
```


```
|-- provider[registry.terraform.io/hashicorp/random] 3.1.0
|-- provider[registry.terraform.io/hashicorp/local] 2.1.0
|-- provider[registry.terraform.io/hashicorp/tls] 3.1.0
```

Now let's re-initialize our working directory to add the `azurerm` provider.

```
terraform init
```

Notice that the azurerm provider has now been installed and initialized.

```
Initializing provider plugins...
- Reusing previous version of hashicorp/random from the dependency lock
    file
- Reusing previous version of hashicorp/local from the dependency lock
    file
- Reusing previous version of hashicorp/tls from the dependency lock file
- Finding hashicorp/azurerm versions matching "2.84.0"...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Reusing previous version of hashicorp/http from the dependency lock file
- Installing hashicorp/azurerm v2.84.0...
- Installed hashicorp/azurerm v2.84.0 (signed by HashiCorp)
- Using previously-installed hashicorp/aws v3.63.0
- Using previously-installed hashicorp/http v2.1.0
- Using previously-installed hashicorp/random v3.1.0
- Using previously-installed hashicorp/local v2.1.0
- Using previously-installed hashicorp/tls v3.1.0
```

Task 3: Re-initialize after adding a new module

During init, the configuration is searched for module blocks, and the source code for referenced modules is retrieved from the locations given in their source arguments.

Add a Terraform module to your configuration.

`main.tf`

```
module "s3-bucket_example_complete" {
  source  = "terraform-aws-modules/s3-bucket/aws/examples/complete"
  version = "2.10.0"
}
```

In order for Terraform to use this module, it must first retrieve the module. This is done by running a `terraform init`.

Verify that the module has not yet been retrieved:

```
terraform providers
```

Retrieve the module by performing a `terraform init`

```
terraform init
```

```
Initializing modules...
Downloading terraform-aws-modules/s3-bucket/aws 2.10.0 for s3-
    bucket_example_complete...
- s3-bucket_example_complete in .terraform/modules/s3-
    bucket_example_complete/examples/complete
- s3-bucket_example_complete.cloudfront_log_bucket in .terraform/modules/
    s3-bucket_example_complete
- s3-bucket_example_complete.log_bucket in .terraform/modules/s3-
    bucket_example_complete
- s3-bucket_example_complete.s3_bucket in .terraform/modules/s3-
    bucket_example_complete
```

Validate that the module has been retrieved and initialized

```
terraform providers
```

```
Providers required by configuration:
|-- provider[registry.terraform.io/hashicorp/tls] 3.1.0
|-- provider[registry.terraform.io/hashicorp/azurerm] 2.84.0
|-- provider[registry.terraform.io/hashicorp/aws]
|-- provider[registry.terraform.io/hashicorp/http] 2.1.0
|-- provider[registry.terraform.io/hashicorp/random] 3.1.0
|-- provider[registry.terraform.io/hashicorp/local] 2.1.0
|-- module.s3-bucket_example_complete
    |-- provider[registry.terraform.io/hashicorp/aws] >= 3.60.0
    |-- provider[registry.terraform.io/hashicorp/random] >= 2.0.0
    |-- module.cloudfront_log_bucket
        |-- provider[registry.terraform.io/hashicorp/aws] >= 3.50.0
    |-- module.log_bucket
        |-- provider[registry.terraform.io/hashicorp/aws] >= 3.50.0
    |-- module.s3_bucket
        |-- provider[registry.terraform.io/hashicorp/aws] >= 3.50.0
```

Re-running init with modules already installed will install the sources for any modules that were added to configuration since the last init, but will not change any already-installed modules. Use -upgrade to override this behavior, updating all modules to the latest available source code.

```
terraform init -upgrade
```

```
Upgrading modules...
```


```
Downloading terraform-aws-modules/s3-bucket/aws 2.10.0 for s3-
    bucket_example_complete...
- s3-bucket_example_complete in .terraform/modules/s3-
    bucket_example_complete/examples/complete
- s3-bucket_example_complete.cloudfront_log_bucket in .terraform/modules/
    s3-bucket_example_complete
- s3-bucket_example_complete.log_bucket in .terraform/modules/s3-
    bucket_example_complete
- s3-bucket_example_complete.s3_bucket in .terraform/modules/s3-
    bucket_example_complete
```

Task 4: Re-initialize after modifying changing a Terraform backend

During init, the root configuration directory is consulted for backend configuration and the chosen backend is initialized using the given configuration settings. By default terraform uses a `local` backend and saves its state file to a `terraform.tfstate` file located in the working directory.

You can validate that your state file lives in the current directory by looking for the presence of a `terraform.tfstate` file. You may also see a backup that was created for this state file

```
|-- main.tf
|-- myplan
|-- terraform.tf
|-- terraform.tfstate
|-- terraform.tfstate.backup
|-- terraform.tfstate.d
|-- |-- development
      |-- terraform.tfstate
      |-- terraform.tfstate.backup
|-- variables.tf
```

You an modify the default backend that terraform uses by updating the terraform configuration block. Update the `terraform.tf` configuaration to move terraform state to a different directory by including a `backend` block to the configuration.

```
terraform {
  backend "local" {
    path = "mystate/terraform.tfstate"
  }
}
```

Create a new subdirectory called `mystate` for terraform to use with the new backend.

```
terraform init
```


If your state file is empty then you would be presented with the following when re-initializing:

```
Initializing the backend...

Successfully configured the backend "local"! Terraform will automatically
use this backend unless the backend configuration changes.
```

If your state file is not empty then you would be presented with the following when re-initializing:

```
Initializing the backend...
Do you want to copy existing state to the new backend?
  Pre-existing state was found while migrating the previous "local"
     backend to the
  newly configured "local" backend. No existing state was found in the
     newly
  configured "local" backend. Do you want to copy this state to the new "
     local"
  backend? Enter "yes" to copy and "no" to start with an empty state.

  Enter a value: yes

Successfully configured the backend "local"! Terraform will automatically
use this backend unless the backend configuration changes.
```

Terraform will automatically copy the state information from the default location to your new directory during the re-initialization process.

```
.
|-- MyAWSKey.pem
|-- main.tf
|-- myplan
|-- mystate
  |-- terraform.tfstate
|-- terraform.tf
|-- terraform.tfstate
|-- terraform.tfstate.backup
|-- variables.tf
```

When modifying terraform's backend you must re-initialize the working directory. Re-running init with an already-initialized backend will update the working directory to use the new backend settings. Either -reconfigure or -migrate-state must be supplied to update the backend configuration.

```
  -reconfigure            Reconfigure the backend, ignoring any saved
                          configuration.

  -migrate-state          Reconfigure the backend, and attempt to migrate
     any
                          existing state.
```


Migrate the state to the `mystate` directory

```
terraform init -migrate-state
```

Now the state information (when the next `terraform apply` is run) will be updated in the `/mystate` directory.) Let's configure terraform back to the default current directory by removing the `backend` block within our `terraform.tf` and reinitalizing.

```
terraform init -migrate-state
```

```
Initializing the backend...

Terraform has detected you're unconfiguring your previously set "local"
    backend.
Do you want to copy existing state to the new backend?
  Pre-existing state was found while migrating the previous "local"
      backend to the
  newly configured "local" backend. No existing state was found in the
      newly
  configured "local" backend. Do you want to copy this state to the new "
      local"
  backend? Enter "yes" to copy and "no" to start with an empty state.

  Enter a value: yes

Successfully unset the backend "local". Terraform will now operate locally
    .
```

Once succesfully migrating back to the default you can delete the `mystate` directory.

Task 5: Other initialization steps/considerations

For providers that are published in either the public Terraform Registry or in a third-party provider registry, `terraform init` will automatically find, download, and install the necessary provider plugins. If you cannot or do not wish to install providers from their origin registries, you can customize how Terraform installs providers using the provider installation settings in the CLI configuration.

To check the provider versions installed via `terraform init` run the `terraform version` command.

```
terraform version
```


```
Terraform v1.0.10
on darwin_amd64
+ provider registry.terraform.io/hashicorp/aws v3.64.2
+ provider registry.terraform.io/hashicorp/azurerm v2.84.0
+ provider registry.terraform.io/hashicorp/http v2.1.0
+ provider registry.terraform.io/hashicorp/local v2.1.0
+ provider registry.terraform.io/hashicorp/random v3.1.0
+ provider registry.terraform.io/hashicorp/tls v3.1.0
```

You can modify the version line of any provider to be as specific or general as desired. Update the `version` line within your `terraform.tf` file for the `azurerm` provider installed earlier in the lab to an older provider version:

`terraform.tf`

```
  azurerm = {
      source  = "hashicorp/azurerm"
      version = "2.57.0"
    }
```

Utilize the `terraform init -upgrade` command to update the provider version for terraform to use. The `-upgrade` subcommand for `terraform init` will cause Terraform to take the newest available version matching the configured version constraints. If the `-upgrade` option is not issued Terraform will continue to use version installed or specified within the dependency lock file.

```
terraform init
```

```
 Error: Failed to query available provider packages

  Could not retrieve the list of available versions for provider hashicorp
     /azurerm: locked provider registry.terraform.io/hashicorp/azurerm
  2.84.0 does not match configured version constraint 2.57.0; must use
     terraform init -upgrade to allow selection of new versions
```

```
terraform init -upgrade
```

```
- Installing hashicorp/azurerm v2.57.0...
- Installed hashicorp/azurerm v2.57.0 (signed by HashiCorp)
```

```
terraform version
```

```
Terraform v1.0.10
on darwin_amd64
+ provider registry.terraform.io/hashicorp/aws v3.64.2
+ provider registry.terraform.io/hashicorp/azurerm v2.57.0
```


```
+ provider registry.terraform.io/hashicorp/http v2.1.0
+ provider registry.terraform.io/hashicorp/local v2.1.0
+ provider registry.terraform.io/hashicorp/random v3.1.0
+ provider registry.terraform.io/hashicorp/tls v3.1.0
```