Lab: Introduction to the Module Block

A module is used to combine resources that are frequently used together into a reusable container. Individual modules can be used to construct a holistic solution required to deploy applications. The goal is to develop modules that can be reused in a variety of different ways, therefore reducing the amount of code that needs to be developed. Modules are called by a `parent` or `root` module, and any modules called by the parent module are known as `child` modules.

Modules can be sourced from a number of different locations, including remote, such as the Terraform module registry, or locally within a folder. While not required, local modules are commonly saved in a folder named `modules`, and each module is named for its respective function inside that folder. An example of this can be found in the diagram below:

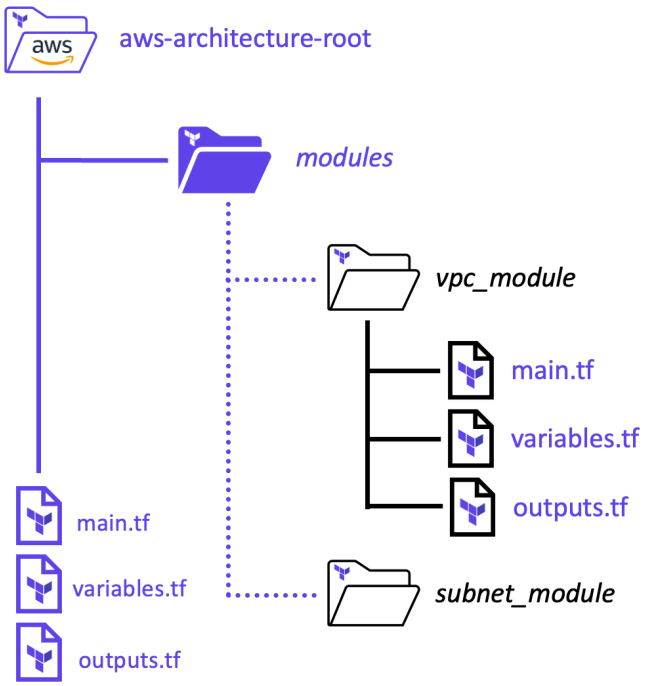

Figure 1: Module Structure

Modules are defined in a `module` block with a unique name for each module. Within the module block, the `source` indicates the local path of the module or the remote source where Terraform should download the module. You can also specify the `version` of the module to use, along with inputs that are passed to the child module.

Template

```
module “<MODULE_NAME>” {
  # Block body
  source = <MODULE_SOURCE>
  <INPUT_NAME> = <DESCRIPTION> #Inputs
  <INPUT_NAME> = <DESCRIPTION> #Inputs
}
```

Example

```
module "website_s3_bucket" {
  source = "./modules/aws-s3-static-website-bucket"

  bucket_name = var.s3_bucket_name
  aws_region = "us-east-1"

  tags = {
    Terraform   = "true"
    Environment = "certification"
  }
}
```

- Task 1: Create a new module block to call a remote module

In order to call a module, let’s create a new module block. In the `main.tf` file, add the following code:

```
 module "subnet_addrs" {
  source  = "hashicorp/subnets/cidr"
  version = "1.0.0"

  base_cidr_block = "10.0.0.0/22"
  networks = [
  {
    name     = "module_network_a"
    new_bits = 2
  },
  {
    name     = "module_network_b"
    new_bits = 2
  },
 ]
}
```


```
output "subnet_addrs" {
  value = module.subnet_addrs.network_cidr_blocks
}
```

Task 1.1

Now that we've added the new module block, let's first run a `terraform init` so Terraform can download the referenced module for us.

```
$ terraform init
Initializing modules...
Downloading hashicorp/subnets/cidr 1.0.0 for subnet_addrs...
- subnet_addrs in .terraform/modules/subnet_addrs

Initializing the backend...

Initializing provider plugins...

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to
   see
any changes that are required for your infrastructure. All Terraform
   commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget,
   other
commands will detect it and remind you to do so if necessary.
```

Task 1.2

Now that the module has been downloaded, let's apply our new configuration. Keep in mind that this module only calculates subnets for us and returns those subnets as an output. It doesn't create any resources in AWS. Run a `terraform apply -auto-approve` to apply the new configuration.

```
$ terraform apply -auto-approve

No changes. Your infrastructure matches the configuration.
```


```
Terraform has compared your real infrastructure against your configuration
    and found no differences, so no changes are needed.

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.

Outputs:

subnet_addrs = tomap({
  "module_network_a" = "10.0.0.0/24"
  "module_network_b" = "10.0.1.0/24"
})
```