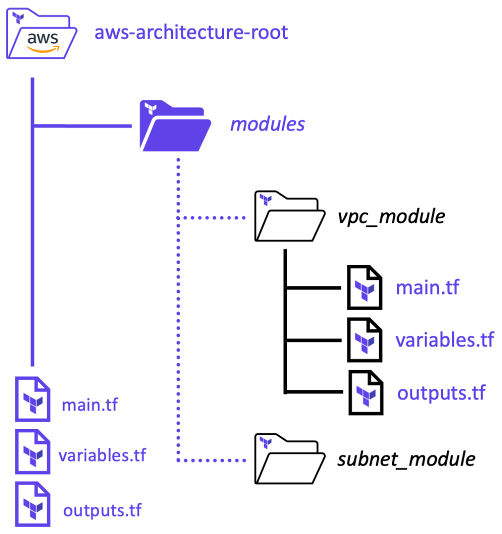
Terraform Modules Inputs and Outputs

To make a Terraform module configurable you can add input parameters to the module. These are defined within the module using input variables. A module can also return values to the configuration that called the module. These module returns or outputs are defined using terraform output blocks.

- Task 1: Refactor module using code organization principles
- Task 2: Required and Optional Module inputs
- Task 3: Module outputs and returns
- Task 4: Reference module outputs

Task 1: Refactor module using code organization principles

You can create a module with a single .tf file, or use any other file structure you like. Modules can be a bit daunting at first. We are suddenly working with a number of files in different file hierarchies, and those files are referencing each other in a few different ways. An example of this can be found in the diagram below:



The example modules contain the following files:

`main.tf` will contain the main set of configuration for your module. You can also create other configuration files and organize them however makes sense for your project.

`variables.tf` will contain the variable definitions for your module. When your module is used by others, the variables will be configured as arguments in the module block. Since all Terraform values must be defined, any variables that are not given a default value will become required arguments. Variables with default values can also be provided as module arguments, overriding the default value.

`outputs.tf` will contain the output definitions for your module. Module outputs are made available to the configuration using the module, so they are often used to pass information about the parts of your infrastructure defined by the module to other parts of your configuration.

Step 1.1 - Refactor the server module

Our `server` module is very simplistic, and we want to incorporate the code organization principles to follow the file structure above. Create three new files inside the `server` folder: `main.tf`, `variables.tf`, `outputs.tf`

Copy the appropriate code out of the `server.tf` and place it into these appropriate files:

`main.tf`

```
resource "aws_instance" "web" {
  ami                    = var.ami
  instance_type          = var.size
  subnet_id              = var.subnet_id
  vpc_security_group_ids = var.security_groups

  tags = {
    "Name"        = "Server from Module"
    "Environment" = "Training"
  }
}
```

`variables.tf`

```
variable "ami" {}
variable "size" {
  default = "t2.micro"
}
variable "subnet_id" {}
variable "security_groups" {
  type = list(any)
}
```

`outputs.tf`

```
output "public_ip" {
  value = aws_instance.web.public_ip
}

output "public_dns" {
  value       = aws_instance.web.public_dns
}
```

You can run a `terraform init` along with a `terraform plan` and there should be no changes if we refactored the module correctly.

```
terraform init
terraform plan
```

Task 2: Module required and optional inputs

Variables within modules work almost exactly the same way that they do for the root module. When you run a Terraform command on your root configuration, there are various ways to set variable values, such as passing them on the commandline, or with a .tfvars file. When using a module, variables are set by passing arguments to the module in your configuration. You will set some of these variables when calling this module from your root module's main.tf.

Variables defined in modules that aren't given a default value are required, and so must be set whenever the module is used. Comment out the `size` variable within the `server` module to make it required.

`variables.tf`

```
variable "ami" {}
variable "size" {
  # default = "t2.micro"
}
variable "subnet_id" {}
variable "security_groups" {
  type = list(any)
}
```

Run a `terraform validate` from the root module/working directory and notice that the module now requires the `size` argument to be passed into the module block, because it is now required.

```
terraform validate

│Error: Missing required argument
│
│   on main.tf line 316, in module "server":
│ 316: module "server" {
```

```
│
│The argument "size" is required, but no definition was found.
```

Update the `server` module block to know specify the new required argument: `size`

```
module "server" {
  source          = "./modules/server"
  ami             = data.aws_ami.ubuntu.id
  size            = "t2.micro"
  subnet_id       = aws_subnet.public_subnets["public_subnet_3"].id
  security_groups = [aws_security_group.vpc-ping.id, aws_security_group.ingress-ssh.id,
}
```

Validate that the all required inputs into the `server` module block have been satisfied.

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

Task 3: Module outputs and returns

Like variables, outputs in modules perform the same function as they do in the root module but are accessed in a different way. A module's outputs can be accessed as read-only attributes on the module object, which is available within the configuration that calls the module. You can reference these outputs in expressions as `module.<MODULE NAME>.<OUTPUT NAME>`.

Outputs are required for common modules that define resources. Variables and outputs are used to infer dependencies between modules and resources. Without any outputs we cannot properly order your module in relation to their Terraform configurations.

We can add a return to our `server` module by adding an output block within the module. Add a `size` variable within the `outputs.tf` of our `server` module.

`output.tf`

```
output "public_ip" {
  description = "IP Address of server built with Server Module"
  value       = aws_instance.web.public_ip
}

output "public_dns" {
  description = "DNS Address of server built with Server Module"
  value       = aws_instance.web.public_dns
}

output "size" {
  description = "Size of server built with Server Module"
  value       = aws_instance.web.instance_type
}
```

Add an output inside the `main.tf` of our working directory to reference the new output provided by the module.

```
output "size" {
  value = module.server.size
}
```

```
terraform init
terraform apply
```

Outputs should have meaningful descriptions you should make an effort to output all the useful values root modules would want to reference or share with modules. Particularly for open source or heavily used modules, expose all outputs that have potential for consumption.

Task 4: Reference module outputs

In order to reference items that are returned by modules (by the module's `outputs.tf` file) you must use the interpolation syntax referring to the output name returned by the module. Eg: `module.server.public_ip`

`outputs.tf` of server module

```
output "public_ip" {
  description = "IP Address of server built with Server Module"
  value       = aws_instance.web.public_ip
}
```

Refering to the `public_ip` of the server module within the root module:

`main.tf` of root module:

```
output "public_ip" {
  value = module.server.public_ip
}
```

Notice that both output blocks have an output named `public_ip`, but their values reference a different interpolation. The output within the module itself references the value of the `public_ip` argument of the resource itself, while the output within the root module/working directory's configuration references the value of `public_ip` returned by the module.