**Building And Testing a Basic Terraform Module**

**Introduction**

Terraform modules are a good way to abstract out repeated chunks of code, making it reusable across other Terraform projects and configurations. In this hands-on lab, we'll be writing a basic Terraform module from scratch and then testing it out.

**Solution**

Log in to the lab server using the credentials provided:

ssh cloud\_user@<Terraform-Controller>

**Create the Directory Structure for the Terraform Project**

1. Check the Terraform status using the version command:

terraform version

Since the Terraform version is returned, you have validated that the Terraform binary is installed and functioning properly.

**Note:** If you receive a notification that there is a newer version of Terraform available, you can ignore it — the lab will run safely with the version installed on the VM.

1. Create a new directory called terraform\_project to house your Terraform code:

mkdir terraform\_project

1. Switch to this main project directory:

cd terraform\_project

1. Create a custom directory called modules and a directory inside it called vpc:

mkdir -p modules/vpc

1. Switch to the vpc directory using the absolute path:

cd /home/cloud\_user/terraform\_project/modules/vpc/

**Write Your Terraform VPC Module Code**

1. Using Vim, create a new file called main.tf:

vim main.tf

1. In the file, insert and review the provided code:
2. provider "aws" {
3. region = var.region
4. }
5. resource "aws\_vpc" "this" {
6. cidr\_block = "10.0.0.0/16"
7. }
8. resource "aws\_subnet" "this" {
9. vpc\_id = aws\_vpc.this.id
10. cidr\_block = "10.0.1.0/24"
11. }
12. data "aws\_ssm\_parameter" "this" {
13. name = "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86\_64-gp2"

}

1. Press **Escape** and enter :wq to save and exit the file.
2. Create a new file called variables.tf:

vim variables.tf

1. In the file, insert and review the provided code:
2. variable "region" {
3. type = string
4. default = "us-east-1"

}

1. Press **Escape** and enter :wq to save and exit the file.
2. Create a new file called outputs.tf:

vim outputs.tf

1. In the file, insert and review the provided code:
2. output "subnet\_id" {
3. value = aws\_subnet.this.id
4. }
5. output "ami\_id" {
6. value = data.aws\_ssm\_parameter.this.value

}

**Note:** The code in outputs.tf is critical to exporting values to your main Terraform code, where you'll be referencing this module. Specifically, it returns the subnet and AMI IDs for your EC2 instance.

1. Press **Escape** and enter :wq to save and exit the file.

**Write Your Main Terraform Project Code**

1. Switch to the main project directory:

cd ~/terraform\_project

1. Create a new file called main.tf:

vim main.tf

1. In the file, insert and review the provided code:
2. variable "main\_region" {
3. type = string
4. default = "us-east-1"
5. }
6. provider "aws" {
7. region = var.main\_region
8. }
9. module "vpc" {
10. source = "./modules/vpc"
11. region = var.main\_region
12. }
13. resource "aws\_instance" "my-instance" {
14. ami = module.vpc.ami\_id
15. subnet\_id = module.vpc.subnet\_id
16. instance\_type = "t2.micro"

}

**Note:** The code in main.tf invokes the VPC module that you created earlier. Notice how you're referencing the code using the source option within the module block to let Terraform know where the module code resides.

1. Press **Escape** and enter :wq to save and exit the file.
2. Create a new file called outputs.tf:

vim outputs.tf

1. In the file, insert and review the provided code:
2. output "PrivateIP" {
3. description = "Private IP of EC2 instance"
4. value = aws\_instance.my-instance.private\_ip

}

1. Press **Escape** and enter :wq to save and exit the file.

**Deploy Your Code and Test Out Your Module**

1. Format the code in all of your files in preparation for deployment:

terraform fmt -recursive

1. Initialize the Terraform configuration to fetch any required providers and get the code being referenced in the module block:

terraform init

1. Validate the code to look for any errors in syntax, parameters, or attributes within Terraform resources that may prevent it from deploying correctly:

terraform validate

You should receive a notification that the configuration is valid.

1. Review the actions that will be performed when you deploy the Terraform code:

terraform plan

In this case, it will create 3 resources, which includes the EC2 instance configured in the root code and any resources configured in the module. If you scroll up and view the resources that will be created, any resource with module.vpc in the name will be created via the module code, such as module.vpc.aws\_vpc.this.

1. Deploy the code:

terraform apply --auto-approve

**Note:** The --auto-approve flag will prevent Terraform from prompting you to enter *yes* explicitly before it deploys the code.

1. Once the code has executed successfully, note in the output that 3 resources have been created and the private IP address of the EC2 instance is returned as was configured in the outputs.tf file in your main project code.
2. View all of the resources that Terraform has created and is now tracking in the state file:

terraform state list

The list of resources should include your EC2 instance, which was configured and created by the main Terraform code, and 3 resources with module.vpc in the name, which were configured and created via the module code.

1. Tear down the infrastructure you just created before moving on:

terraform destroy

1. When prompted, type *yes* and press **Enter**.

**Conclusion**

Congratulations — you've completed this hands-on lab

