# QATIP Intermediate AWS Lab04 Terraform Modules

## Contents

## Lab Objectives

1. Develop reusable Terraform modules for AWS Virtual Private Clouds (VPCs).
2. Parameterize the module using variables.
3. Deploy multiple VPCs by reusing the module with parameters.
4. Utilize module outputs in the root module to create dependent resources.

## Before you begin

5. Ensure you have completed Lab0 before attempting this lab.
6. In the IDE terminal pane, enter the following command...

```
cd ~/environment/aws-tf-int/labs/04
```

7. This shifts your current working directory to labs/04. Ensure all commands are executed in this directory
8. Close any open files and use the Explorer pane to navigate to and open the labs/04 folder.

## Try It Yourself

### Challenge

9. Using **labs/04** as your root directory, create a reusable module structure to deploy two AWS VPCs in the same AWS region, **us-east-1**. As you deploy the module, expose outputs from it and use these outputs in the root module to create a subnet on each VPC.

### Key Requirements

- Create **main.tf** in **aws-tf-int/labs/04**
- Create **main.tf**, **variables.tf**, and **outputs.tf** files in **aws-tf-int/labs/04/modules/vpc**
- Deploy two virtual networks, **vpc-01** with CIDR **10.0.0.0/16** and **vpc-02** with CIDR **10.1.0.0/16**, using a vpc module structure
- Pass output values from the module back to the root configuration

- Use the outputs from the module to create a subnet in each network; **subnet-01** on **vpc-01** using CIDR **10.0.1.0/24** and **subnet-02** on **vpc-02** using CIDR **10.1.1.0/24**

## Solution

10. A proposed solution to this challenge is at **aws-tf-int/labs/solutions/04**

## Step-by-Step Instructions

### Step 1: Set Up the Project Directory

11. Ensure you have moved to the lab directory:

```
cd ~/environment/aws-tf-int/labs/04
```

12. Create subdirectories for the module:

```
mkdir modules
cd modules
mkdir vpc
cd vpc
```

### Step 2: Create the VPC Module

13. You should now be in directory **aws-tf-int/labs/04/modules/vpc**. Create the following files:

**main.tf**:

```
resource "aws_vpc" "lab_vpc" {
  cidr_block = var.cidr_block
  enable_dns_support = true
  enable_dns_hostnames = true
  tags = {
    Name = var.vpc_name
  }
}
```

**variables.tf**:

```
variable "vpc_name" {}
variable "cidr_block" {}
```

**outputs.tf**:

```
output "vpc_id" {
  value = aws_vpc.lab_vpc.id
}
output "vpc_cidr" {
  value = aws_vpc.lab_vpc.cidr_block
}
```

## Step 3: Create the Root module Configuration

14. Each subnet in AWS must reside in a specific Availability Zone (AZ), which represents an isolated data center within a region. For predictable, repeatable deployments, it is best practice to explicitly specify the availability_zone when creating a subnet. You can use the aws_availability_zones data source to dynamically select valid AZs (e.g. data.aws_availability_zones.available.names[0]) instead of hardcoding AZ names.
15. In **aws-tf-int/labs/04**, create **main.tf**

```
provider "aws" {
  region = "us-east-1"
}

data "aws_availability_zones" "available" {}

module "vpc1" {
  source     = "./modules/vpc"
  vpc_name   = "vpc-01"
  cidr_block = "10.0.0.0/16"
}

module "vpc2" {
  source     = "./modules/vpc"
  vpc_name   = "vpc-02"
  cidr_block = "10.1.0.0/16"
}

resource "aws_subnet" "subnet1" {
  vpc_id     = module.vpc1.vpc_id
  cidr_block = "10.0.1.0/24"
```

```
 availability_zone = data.aws_availability_zones.available.names[0]
 tags = {
  Name = "subnet-01"
 }
}

resource "aws_subnet" "subnet2" {
 vpc_id     = module.vpc2.vpc_id
 cidr_block = "10.1.1.0/24"
 availability_zone = data.aws_availability_zones.available.names[1]
 tags = {
  Name = "subnet-02"
 }
}
```

## Step 4: Initialize, Review and Apply

16. Initialize terraform and deploy resources…

```
terraform init
terraform plan
terraform apply
```

## Step 5: Verify the Deployment

17. Use **terraform state list** and the Azure Portal to verify the deployment of 2 VPCs in us-east-1, each with a single subnet .

## Step 6: Clean Up Resources

18. Use **terraform destroy** confirmed with **yes**

# Optional Stretch Challenge

19. This optional stretch challenge builds on the previous lab, adding complexity by introducing multiple AWS regions. You are required to deploy two VPCs—each in a different region—via a reusable module structure. Outputs from each module will then be used to create a subnet in the appropriate region.

## Challenge Objectives

- Develop a reusable Terraform module for creating VPCs.
- Use provider aliasing to deploy resources in multiple regions.
- Deploy two VPCs, one in us-east-1 and one in us-west-2.
- Use module outputs to create region-specific subnets.

## Before You Begin

20. Ensure you are in the correct working directory for this stretch challenge:

    **cd ~/environment/aws-tf-int/labs/04**
    **mkdir stretch**
    **cd stretch**

## Try It Yourself

21. Using **aws-tf-int/labs/04/stretch** as your root directory, create a multi-region setup that deploys:
    - A VPC named **vpc-east** in **us-east-1** with CIDR **10.0.0.0/16**
    - A VPC named **vpc-west** in **us-west-2** with CIDR **10.1.0.0/16**
    - A subnet in each VPC:
        - **subnet-east** with CIDR **10.0.1.0/24** in any az in **us-east-1**
        - **subnet-west** with CIDR **10.1.1.0/24** in any az in **us-west-2**

## Solution

22. A solution can be found in **aws-tf-int/labs/solutions/04/stretch**
23. Destroy all resources created with **terraform destroy**

# Optional Demo Lab

24. This demonstration walks you through a more complex use of modules to deploy VPCs (Virtual Private Clouds) with subnets,

Security Groups (SGs), and VPC Peering. The goal is to further highlight the benefits of dynamic and reusable infrastructure by leveraging Terraform modules and outputs.

25. In your IDE, navigate to **/labs/04/demo** and review the provisioned files...

```
04/demo/
  main.tf         # Root Terraform configuration
  variables.tf    # Root input variables
  modules/
    vpc/
      main.tf         # VPC and subnets
      variables.tf    # Inputs: vpc_name, cidr_block, subnets
      outputs.tf      # Outputs: vpc_details, subnets
    sg/
      main.tf         # Security group resource
      variables.tf    # Inputs: sg_name, vpc_id
      outputs.tf      # Output: sg_id
```

## Module Logic

26. At the top level, the root module defines the AWS provider and statically assigns the deployment region (us-east-1). It declares a variable called vpcs, which is a map of VPC definitions (e.g., vpc1, vpc2). Each entry in this map includes metadata like the VPC name and CIDR block.
27. The root module then uses a for_each loop to iterate over these VPC definitions, deploying each one by calling the vpcs module. In parallel, it looks up a separate subnets variable — a nested map that lists subnets per VPC — and transforms this into a list of subnet objects to pass into the module.
28. The root module also contains logic to:

    - Create a security group for each VPC by looping over the deployed vpcs and invoking the sgs module.

- Establish two-way VPC peering connections between VPCs, again leveraging the output from the vpcs module.

29. This structure ensures that the root module acts as the orchestrator, passing data into and consuming outputs from the modules below it.
30. The vpcs module is designed to be reusable. It accepts three inputs:
    - vpc_name
    - cidr_block
    - A list of subnet definitions (each with a name and cidr_block)
31. Within the VPC module, a new VPC is created using the provided CIDR and tagging the resource with its name. The module then loops over the list of subnet definitions using for_each to create multiple aws_subnet resources, each tied to the newly created VPC.
32. Two key outputs are exposed from the module:
    - A vpc_details map containing the VPC's id and name — useful for peering or referencing in other modules.
    - A subnets map containing subnet names and their respective subnet IDs — useful for attaching downstream resources like instances, route tables, or load balancers.
33. The security group module (sg) takes two inputs:
    - sg_name, typically derived from the VPC name
    - vpc_id, passed from the vpcs module output
34. It creates a basic security group and outputs the group ID. While minimal in its content, the security group can be extended to define ingress/egress rules or support tagging and environments.

## Deploy the resources

35. Run the following commands:
    **terraform init**
    **terraform plan**
    **terraform apply**

    Expected output:
    Plan: 9 to add, 0 to change, 0 to destroy.

## Lab Clean Up

36. Validate the resources have been created and then clean up with **terraform destroy** confirmed with **yes**

**## Congratulations, you have completed this lab ##**