QATIP Intermediate
AWS Lab04
Terraform Modules

Lab Objectives

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

Before you begin

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

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

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

Try It Yourself

Challenge

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
- 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

Step-by-Step Instructions

Step 1: Set Up the Project Directory

Ensure you have moved to the lab directory:

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

Create a subdirectory for the module:

```
mkdir modules/vpc
```

Step 2: Create the VPC Module

Navigate to the new vpc directory and 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

In **labs/04**, create **main.tf**

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

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"
```

```
  tags = {
    Name = "subnet-01"
  }
}

resource "aws_subnet" "subnet2" {
  vpc_id     = module.vpc2.vpc_id
  cidr_block = "10.1.1.0/24"
  tags = {
    Name = "subnet-02"
  }
}
```

Step 4: Initialize, Review and Apply

terraform init

terraform plan

terraform apply

Confirm with **yes** when prompted.

Step 5: Verify the Deployment

Terraform state list

Verify in the Azure Portal.

Step 6: Clean Up Resources

terraform destroy confirmed with **yes**

Solution Review

This lab follows best practices by utilizing reusable Terraform modules to deploy two AWS VPCs and creating subnets in the root module based

on module outputs. The root module orchestrates the deployment by calling the VPC module twice, each time with different parameters.

Module-Based Approach

Instead of defining VPCs directly in the root module, this lab uses a modular approach, which provides:

- Scalability: Easily extendable to deploy additional VPCs.
- Reusability: The same module can be reused with different parameters.
- Maintainability: Reduces duplication and simplifies future modifications.

Exposing and Utilizing Module Outputs

A critical part of the lab is exposing outputs from the module and utilizing them in the root module. The output block in outputs.tf ensures that the VPC ID and CIDR block are accessible after deployment and can be referenced in subnet creation.

Terraform Modules Demo Lab

Overview

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 highlight the benefits of dynamic and reusable infrastructure by leveraging Terraform modules and outputs.

In your IDE, navigate to **/labs/04_demo**

Review the provisioned files.

This demonstration shows how to structure a Terraform deployment using modules to define reusable, parameterized components. It reinforces key skills such as:

- Separating concerns using modules (`vpc`, `sg`)
- Dynamically creating resources with `for_each`
- Passing variables and using outputs between modules
- Managing dependencies (e.g., subnets inside VPCs, SGs in VPCs, peering between VPCs)

Directory Structure

```
├ 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 Overview

a. Root Module (main.tf):

- Defines AWS provider and region
- Uses `for_each` over a map of VPCs (`vpc1`, `vpc2`) to deploy VPC + subnets via module `vpcs`
- Creates a Security Group in each VPC via module `sgs`
- Establishes one VPC peering connection between the VPCs

b. VPC Module:

- Creates a VPC using `vpc_name` and `cidr_block`
- Iterates over subnets using `for_each`
- Outputs `vpc_details` and `subnets`

c. SG Module:

- Creates a security group using input `vpc_id` and `sg_name`
- Outputs the security group ID

Data Flow Between Modules

- VPC inputs come from `var.vpcs` and `var.subnets`
- SG module references `module.vpcs[each.key].vpc_details.id`
- Peering uses `.vpc_details.id` from each module
- Outputs are structured for referencing across modules

How to Deploy

Run the following commands:

terraform init

terraform plan

terraform apply

Expected output:

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

Validation Steps

- Check VPCs in AWS Console → VPC Dashboard
- Validate subnet CIDR ranges
- Confirm SGs are attached to correct VPCs
- Verify peering connection is active
- (Optional) Use AWS CLI: ` aws ec2 describe-vpc-peering-connections`

Lab Clean Up

Destroy all resources with **terraform destroy** confirmed with **yes**

Congratulations, you have completed this lab