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**

Stretch Challenge

19. In this stretch challenge, you will design and deploy a multi-region network foundation using Terraform modules and AWS best practices. You'll create an infrastructure-as-code configuration that automatically deploys two VPCs — one in the US East (N. Virginia) region and another in the US West (Oregon) region. Each VPC will contain a single subnet deployed in a valid availability zone that is discovered dynamically rather than hard-coded.

Challenge Objectives

20. By the end of this challenge, you will have:
    - Defined a reusable VPC module that can deploy a VPC and a subnet given customizable input parameters.
    - Configured your root module to:
        - Use multiple provider configurations to work in more than one AWS region at the same time.
        - Query available availability zones dynamically for each region to avoid hard-coded AZ names.
        - Call the same VPC module twice to deploy vpc-east in us-east-1 and vpc-west in us-west-2.
    - Verified your deployment by outputting the VPC and subnet IDs for both regions.

Key Concepts Practiced

Provider Aliasing: Using alias to manage multiple AWS regions in the same run.

Data Sources: Using aws_availability_zones to dynamically select an AZ instead of relying on static values.

Module Reuse: Encapsulating VPC + subnet creation logic in a clean reusable module and calling it multiple times with different inputs.

Outputs: Using Terraform outputs to expose created resource IDs for validation.

Scenario

21. Your team has decided to expand application deployments to serve users from both the US East and West coasts for lower latency and higher availability. You've been asked to prototype the base networking layer as infrastructure-as-code. Using Terraform, you will deploy consistent, well-tagged VPCs and subnets in both regions to establish a robust foundation for future compute or service deployments.

Success Criteria

22. Your solution will be complete when:

    Both VPCs exist in their respective regions.

    Each VPC contains exactly one subnet.

    The subnets are placed in valid, available AZs discovered at runtime.

    The output displays the VPC and subnet IDs for east and west clearly.

Run, Verify, Clean Up

23. Use terraform init, terraform plan, and terraform apply to deploy.
24. Use terraform destroy to clean up when finished.

Solution

25. A solution can be found in **aws-tf-int/labs/solutions/04/stretch**

Demo Lab

26. 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.

27. 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

28. 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.
29. 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.
30. 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.
31. This structure ensures that the root module acts as the orchestrator, passing data into and consuming outputs from the modules below it.

32. 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)
33. 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.
34. 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.
35. The security group module (sg) takes two inputs:
    - sg_name, typically derived from the VPC name
    - vpc_id, passed from the vpcs module output
36. 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

37. 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

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

Congratulations, you have completed this lab