A step-by-step tutorial on how to create and use Terraform modules with AWS. Terraform modules allow you to organize your code, reuse components, and make your infrastructure as code (IaC) more manageable.

Prerequisites

1. \*\*Terraform Installed\*\*: Ensure that Terraform is installed on your machine.

2. \*\*AWS Account\*\*: You'll need an AWS account and appropriate credentials configured on your machine.

3. \*\*Basic Understanding\*\*: You should have a basic understanding of Terraform, including how to write simple configurations.

Step 1: Set Up Your Directory Structure

Start by setting up a directory structure that is modular. This will help in organizing the Terraform code.

```bash

terraform-modules/

├── main.tf

├── variables.tf

├── outputs.tf

└── modules/

├── vpc/

│ ├── main.tf

│ ├── variables.tf

│ └── outputs.tf

├── ec2/

│ ├── main.tf

│ ├── variables.tf

│ └── outputs.tf

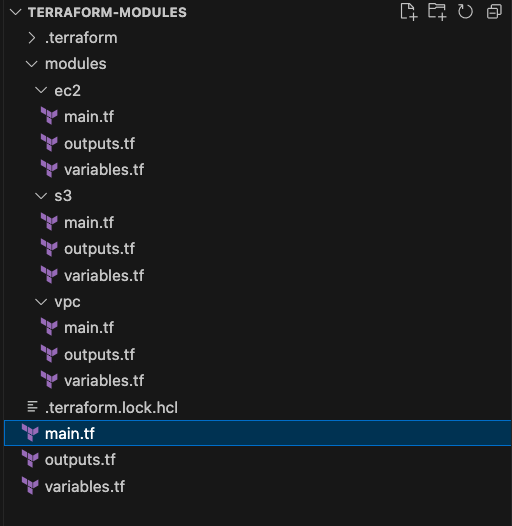
└── s3/

├── main.tf

├── variables.tf

└── outputs.tf

```

****

Step 2: Create a VPC Module

In the `modules/vpc/` directory, create a `main.tf` file for defining a VPC.

resource "aws\_vpc" "this" {

cidr\_block = var.cidr\_block

tags = {

Name = var.vpc\_name

}

}

resource "aws\_subnet" "public" {

count = length(var.public\_subnets)

vpc\_id = aws\_vpc.this.id

cidr\_block = var.public\_subnets[count.index]

availability\_zone = element(var.azs, count.index)

tags = {

Name = "${var.vpc\_name}-public-${count.index}"

}

}

# `modules/vpc/main.tf`

```hcl

resource "aws\_vpc" "this" {

cidr\_block = var.cidr\_block

tags = {

Name = var.vpc\_name

}

}

resource "aws\_subnet" "public" {

count = length(var.public\_subnets)

vpc\_id = aws\_vpc.this.id

cidr\_block = var.public\_subnets[count.index]

availability\_zone = element(var.azs, count.index)

tags = {

Name = "${var.vpc\_name}-public-${count.index}"

}

}

```

# `modules/vpc/variables.tf`

variable "cidr\_block" {

type = string

}

variable "vpc\_name" {

type = string

}

variable "public\_subnets" {

type = list(string)

}

variable "azs" {

type = list(string)

}

```hcl

variable "cidr\_block" {

type = string

}

variable "vpc\_name" {

type = string

}

variable "public\_subnets" {

type = list(string)

}

variable "azs" {

type = list(string)

}

```

# `modules/vpc/outputs.tf`

```hcl

output "vpc\_id" {

value = aws\_vpc.this.id

}

output "public\_subnets\_ids" {

value = aws\_subnet.public[\*].id

}

```

output "vpc\_id" {

value = aws\_vpc.this.id

}

output "public\_subnets\_ids" {

value = aws\_subnet.public[\*].id

}

Step 3: Create an EC2 Module

In the `modules/ec2/` directory, create a `main.tf` file for defining an EC2 instance.

resource "aws\_instance" "this" {

ami = var.ami

instance\_type = var.instance\_type

subnet\_id = var.subnet\_id

key\_name = var.key\_name

tags = {

Name = var.instance\_name

}

}

# `modules/ec2/main.tf`

```hcl

resource "aws\_instance" "this" {

ami = var.ami

instance\_type = var.instance\_type

subnet\_id = var.subnet\_id

key\_name = var.key\_name

tags = {

Name = var.instance\_name

}

}

```

# `modules/ec2/variables.tf`

variable "ami" {

type = string

}

variable "instance\_type" {

type = string

}

variable "subnet\_id" {

type = string

}

variable "key\_name" {

type = string

}

variable "instance\_name" {

type = string

}

```hcl

variable "ami" {

type = string

}

variable "instance\_type" {

type = string

}

variable "subnet\_id" {

type = string

}

variable "key\_name" {

type = string

}

variable "instance\_name" {

type = string

}

```

# `modules/ec2/outputs.tf`

```hcl

output "instance\_id" {

value = aws\_instance.this.id

}

```

output "instance\_id" {

value = aws\_instance.this.id

}

Step 4: Create an S3 Module

In the `modules/s3/` directory, create a `main.tf` file for defining an S3 bucket.

resource "aws\_s3\_bucket" "this" {

bucket = var.bucket\_name

tags = {

Name = var.bucket\_name

}

}

# `modules/s3/main.tf`

```hcl

resource "aws\_s3\_bucket" "this" {

bucket = var.bucket\_name

tags = {

Name = var.bucket\_name

}

}

```

# `modules/s3/variables.tf`

```hcl

variable "bucket\_name" {

type = string

}

```

variable "bucket\_name" {

type = string

}

# `modules/s3/outputs.tf`

```hcl

output "bucket\_arn" {

value = aws\_s3\_bucket.this.arn

}

```

output "bucket\_arn" {

value = aws\_s3\_bucket.this.arn

}

Step 5: Use the Modules in Your Root Module

provider "aws" {

region = "us-east-1"

}

module "vpc" {

source = "./modules/vpc"

cidr\_block = "10.0.0.0/16"

vpc\_name = "my-vpc"

public\_subnets = ["10.0.1.0/24", "10.0.2.0/24"]

azs = ["us-east-1a", "us-east-1b"]

}

module "ec2" {

source = "./modules/ec2"

ami = "ami-0c55b159cbfafe1f0"

instance\_type = "t2.micro"

subnet\_id = element(module.vpc.public\_subnets\_ids, 0)

key\_name = "my-key"

instance\_name = "my-ec2-instance"

}

module "s3" {

source = "./modules/s3"

bucket\_name = "my-aug23-bucket-name"

}

Now, use the modules you created in your root module (`main.tf` in the root directory).

# `main.tf`

```hcl

provider "aws" {

region = "us-east-1"

}

module "vpc" {

source = "./modules/vpc"

cidr\_block = "10.0.0.0/16"

vpc\_name = "my-vpc"

public\_subnets = ["10.0.1.0/24", "10.0.2.0/24"]

azs = ["us-east-1a", "us-east-1b"]

}

module "ec2" {

source = "./modules/ec2"

ami = "ami-0c55b159cbfafe1f0"

instance\_type = "t2.micro"

subnet\_id = element(module.vpc.public\_subnets\_ids, 0)

key\_name = "my-key"

instance\_name = "my-ec2-instance"

}

module "s3" {

source = "./modules/s3"

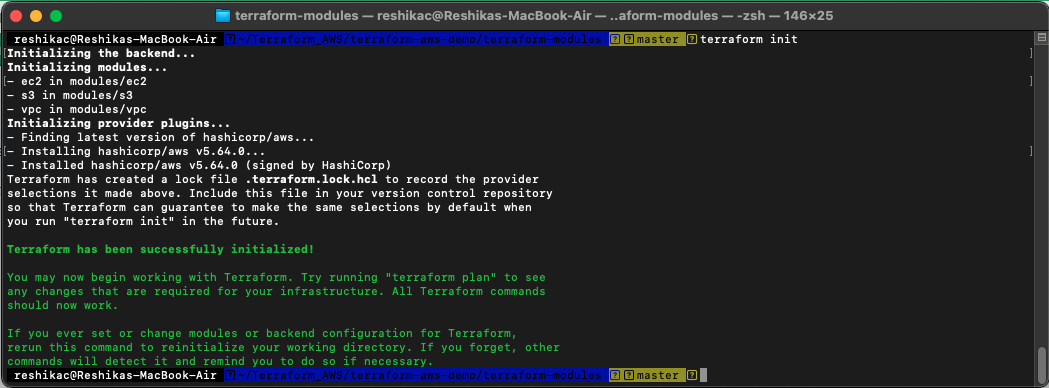
bucket\_name = "my-unique-bucket-name"

}

```

Step 6: Initialize and Apply

Initialize Terraform and apply the configuration:

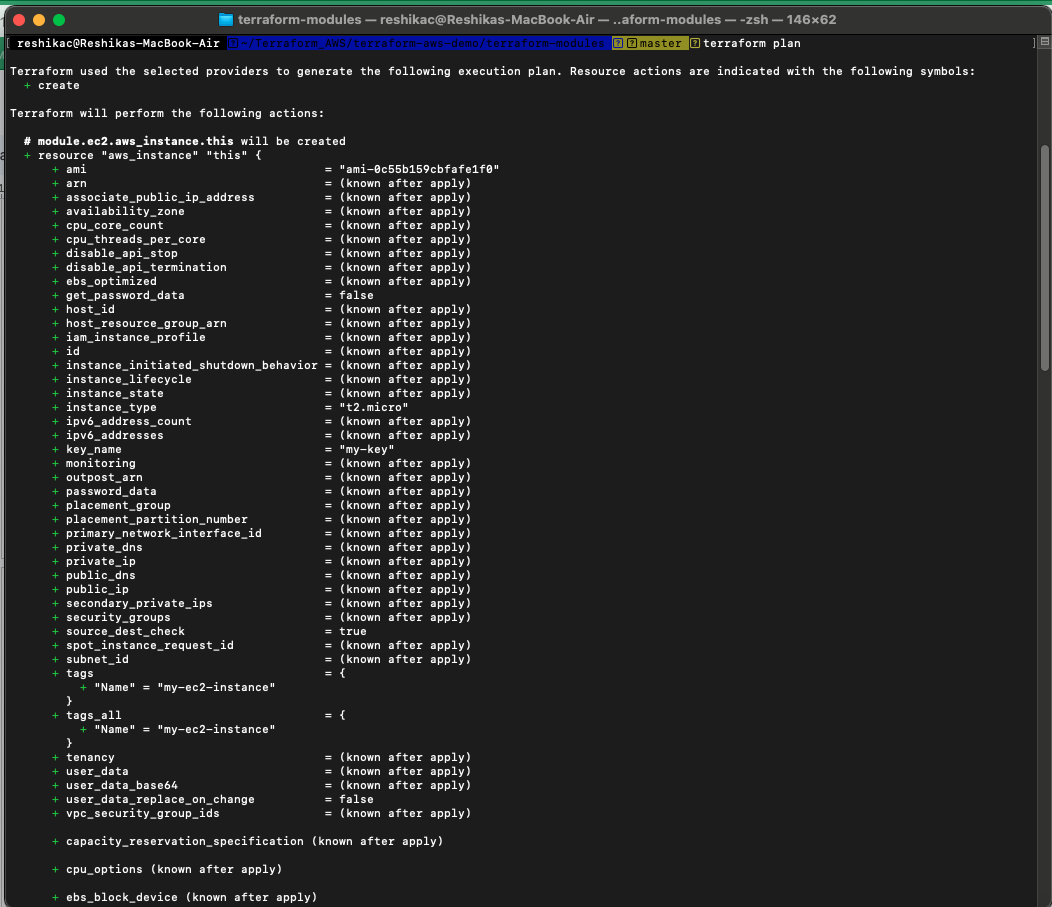


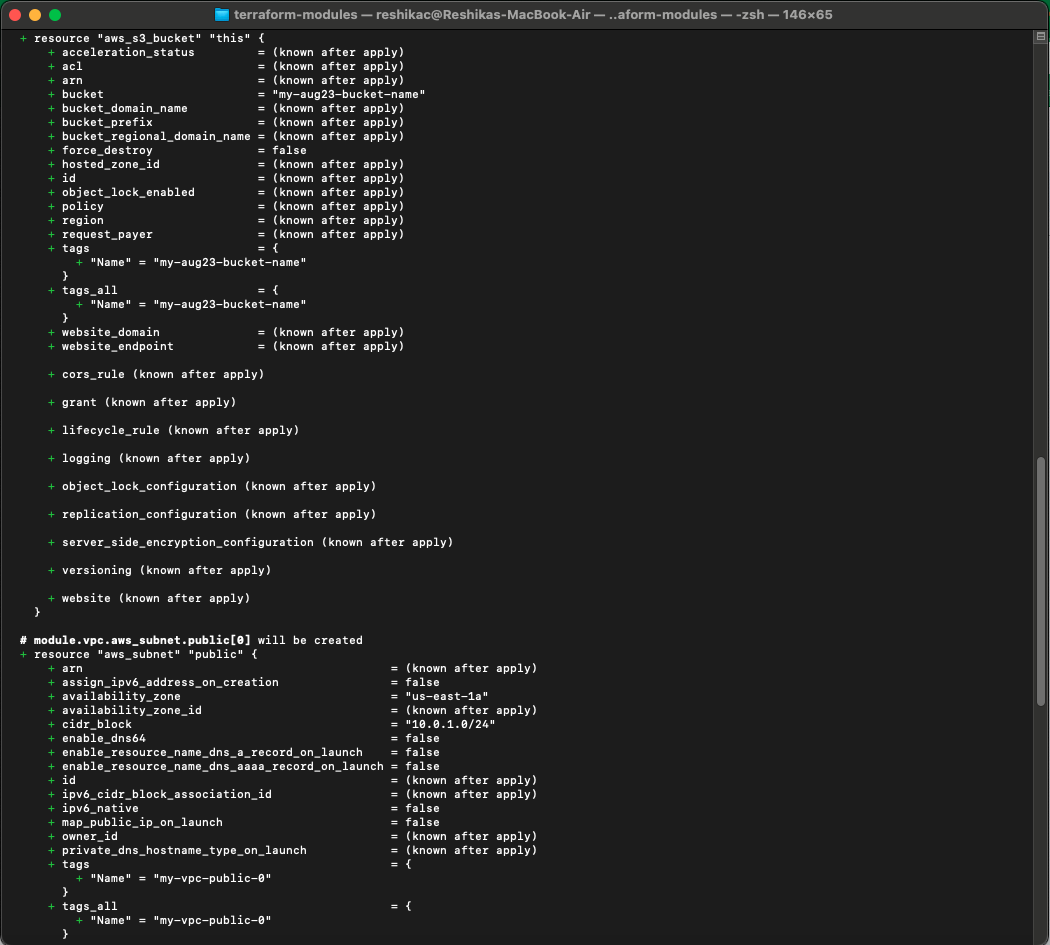
```bash

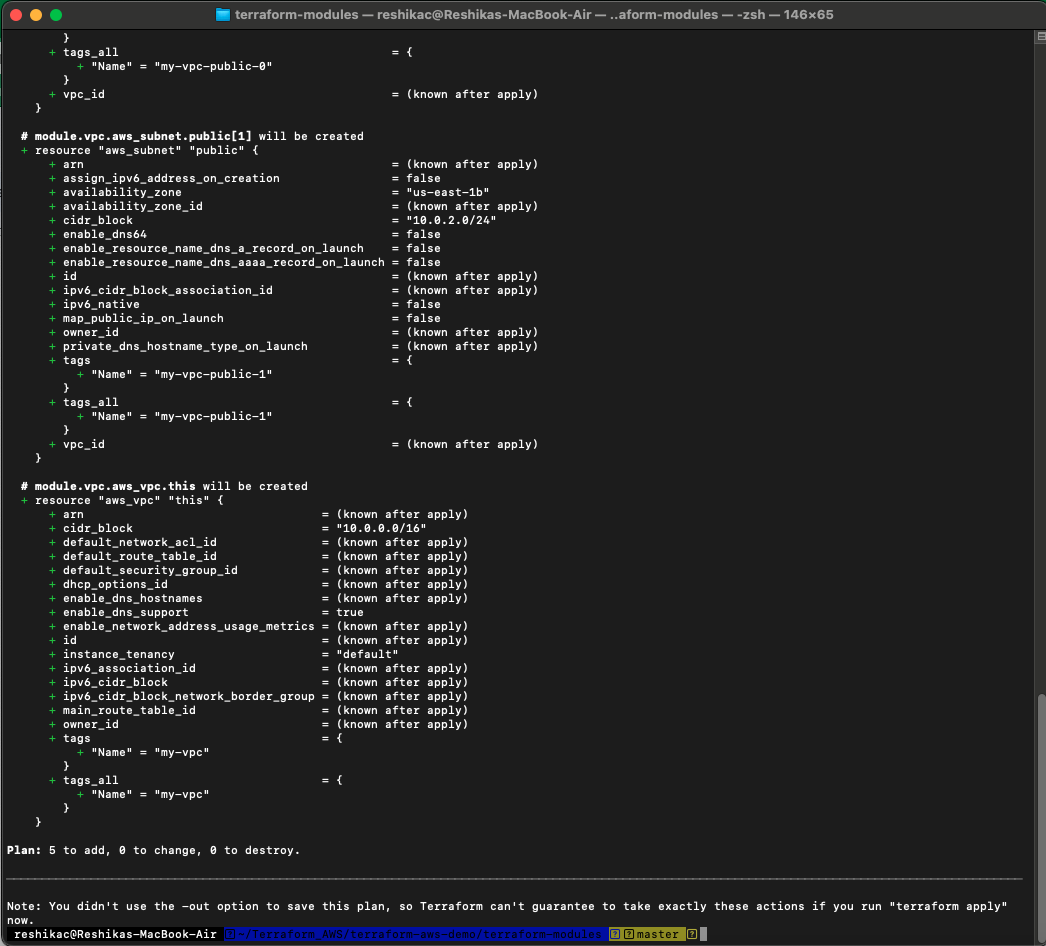
terraform init

terraform apply

```

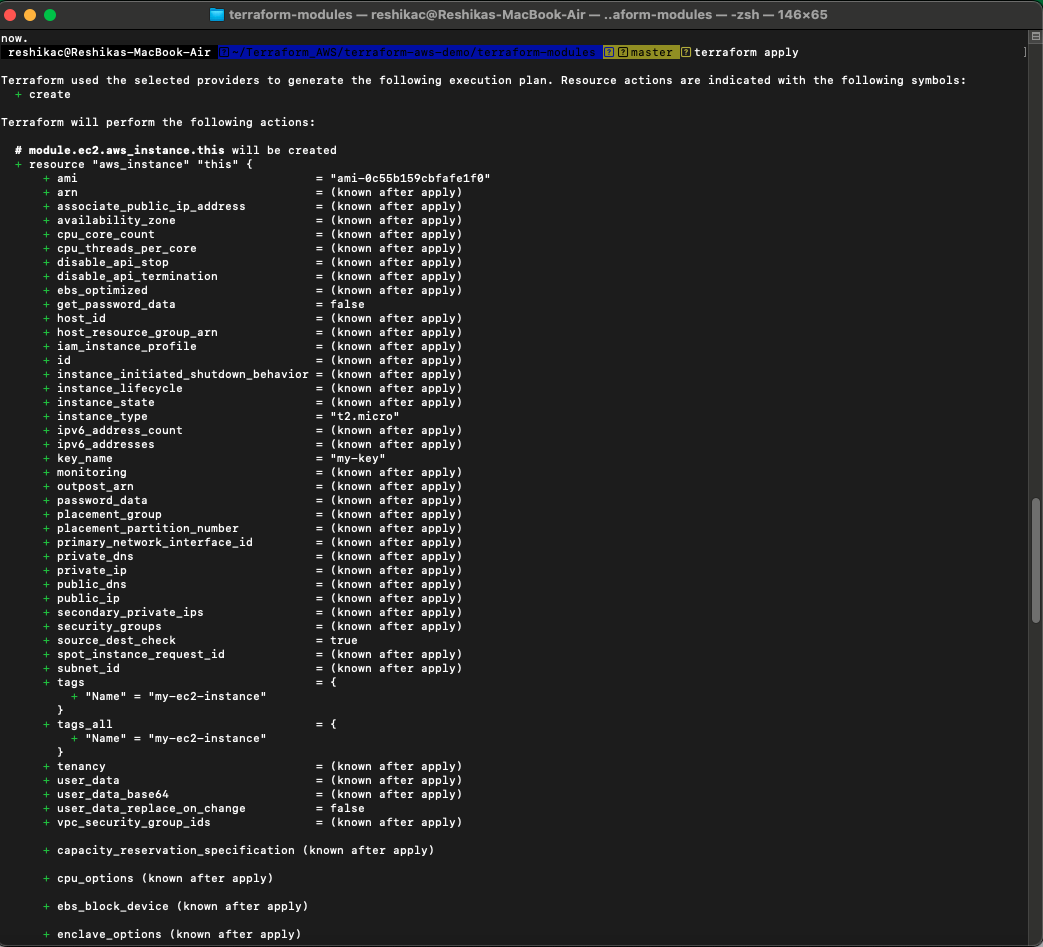


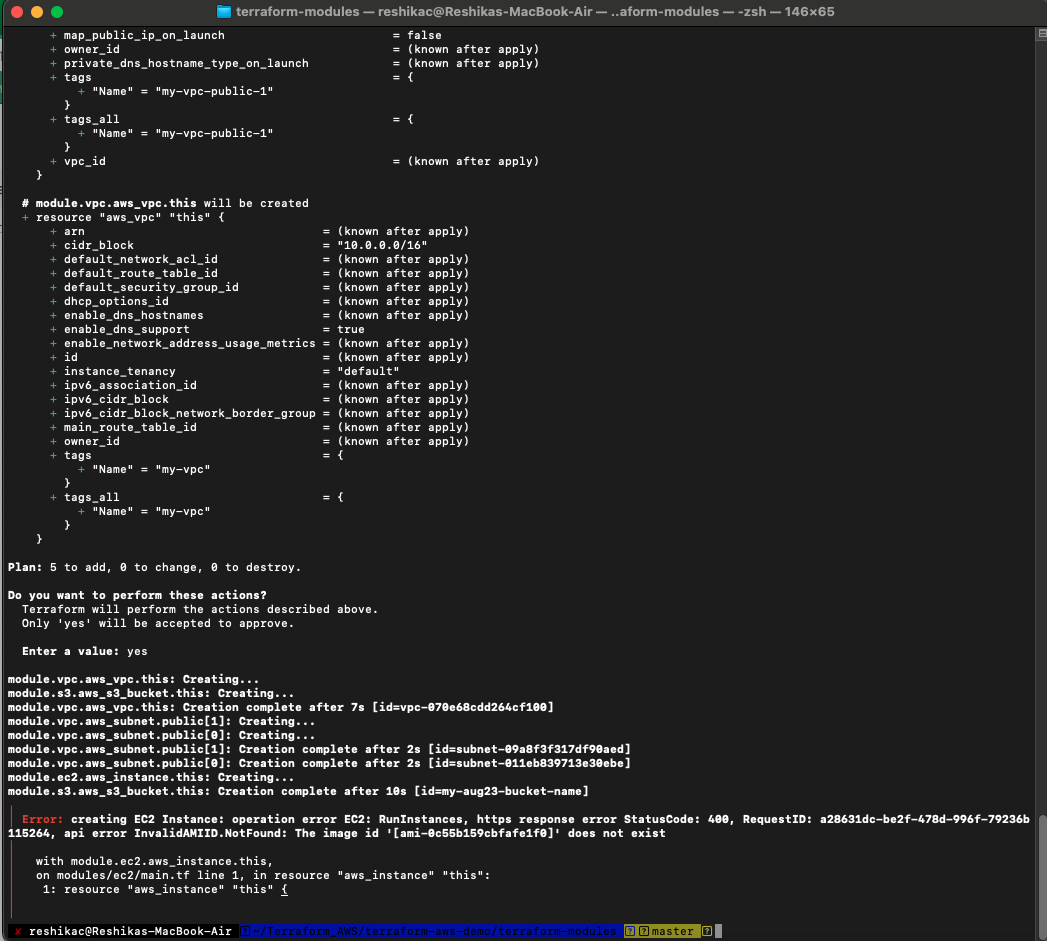




Terraform will prompt you to confirm the actions it plans to take. Type `yes` to proceed.

Step 7: Outputs





After applying, Terraform will output the resources it created. You can see the VPC ID, Subnet IDs, EC2 Instance ID, and S3 Bucket ARN from the respective module outputs.

Conclusion

You now have a Terraform setup using modules for AWS resources. This structure is scalable and maintainable, allowing you to add more resources and reuse modules as needed.