Advanced Terraform Variables: A Hands-On Tutorial with AWS

Introduction

In this advanced tutorial, we'll dive deeper into Terraform's parameterization capabilities. We'll cover advanced features like variable validation, maps, lists, conditionals, and using data sources. We'll continue working within the AWS ecosystem.

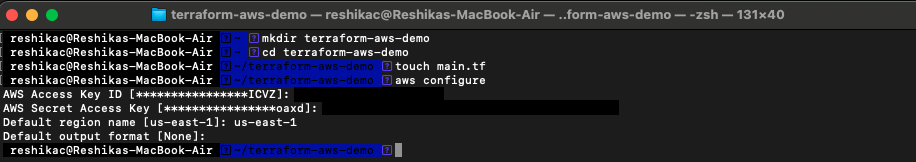
Prerequisites

Ensure you have:

1. AWS Account : Sign up if you don’t already have one.

2. Terraform Installed : Install Terraform on your machine using the [official documentation](https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli).

3. AWS CLI Configured : Install and configure the AWS CLI with your credentials.



Step 1: Set Up Your Project Directory

1. Create a new directory for your project:

```bash

mkdir terraform-aws-advanced

cd terraform-aws-advanced

```

2. Create the main configuration file :

```bash

touch main.tf

```

Step 2: Define Advanced Variables

We will work with maps, lists, conditionals, and validation.

1. Create a variables file (`variables.tf`):

```bash

touch variables.tf

```

2. Define advanced variables in `variables.tf`:

```hcl

#A list of instance types to choose from

variable "allowed\_instance\_types" {

type = list(string)

description = "Allowed EC2 instance types"

default = ["t2.micro", "t3.micro", "t3.small"]

}

#Map for AMIs based on region

variable "ami\_map" {

type = map(string)

default = {

"us-east-1" = "ami-0c55b159cbfafe1f0"

"us-west-2" = "ami-0a38c1c38a15fed74"

}

}

#A conditional variable for environment

variable "environment" {

description = "Environment to deploy to"

type = string

default = "dev"

validation {

condition = contains(["dev", "staging", "production"], var.environment)

error\_message = "Environment must be one of 'dev', 'staging', or 'production'."

}

}

#Variable for instance count, validated against production environment

variable "instance\_count" {

description = "Number of instances to create"

type = number

default = 1

validation {

condition = var.environment != "production" || var.instance\_count <= 2

error\_message = "You can only deploy up to 2 instances in production."

}

}

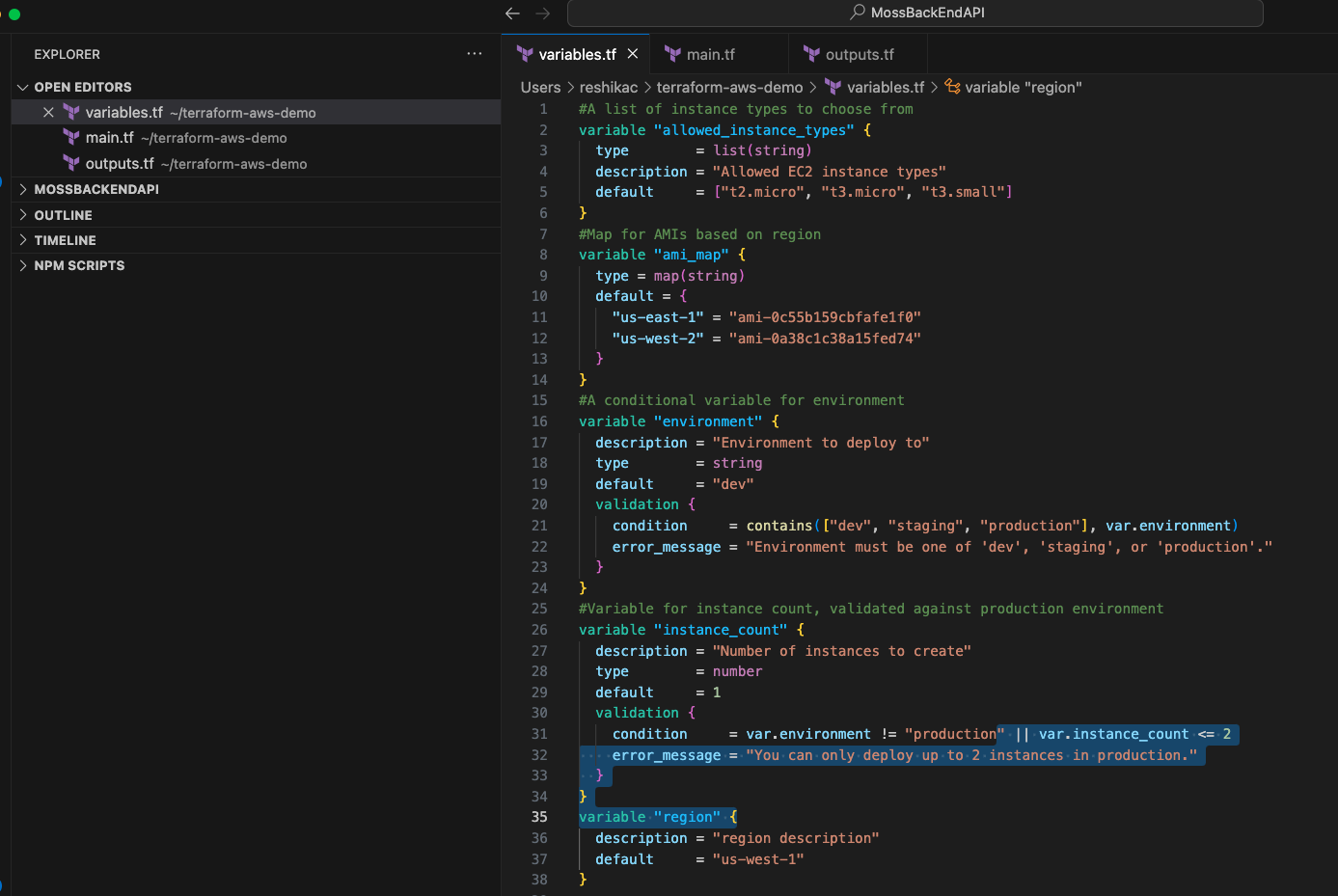
variable "region" {

description = "region description"

default = "us-west-1"

}

```



Step 3: Utilize Advanced Variables in Your Configuration

1. Define the provider and AWS resources in `main.tf`:

```hcl

provider "aws" {

region = var.region

}

resource "aws\_instance" "example" {

count = var.instance\_count

ami = var.ami\_map[var.region]

instance\_type = var.allowed\_instance\_types[0] Selecting the first instance type

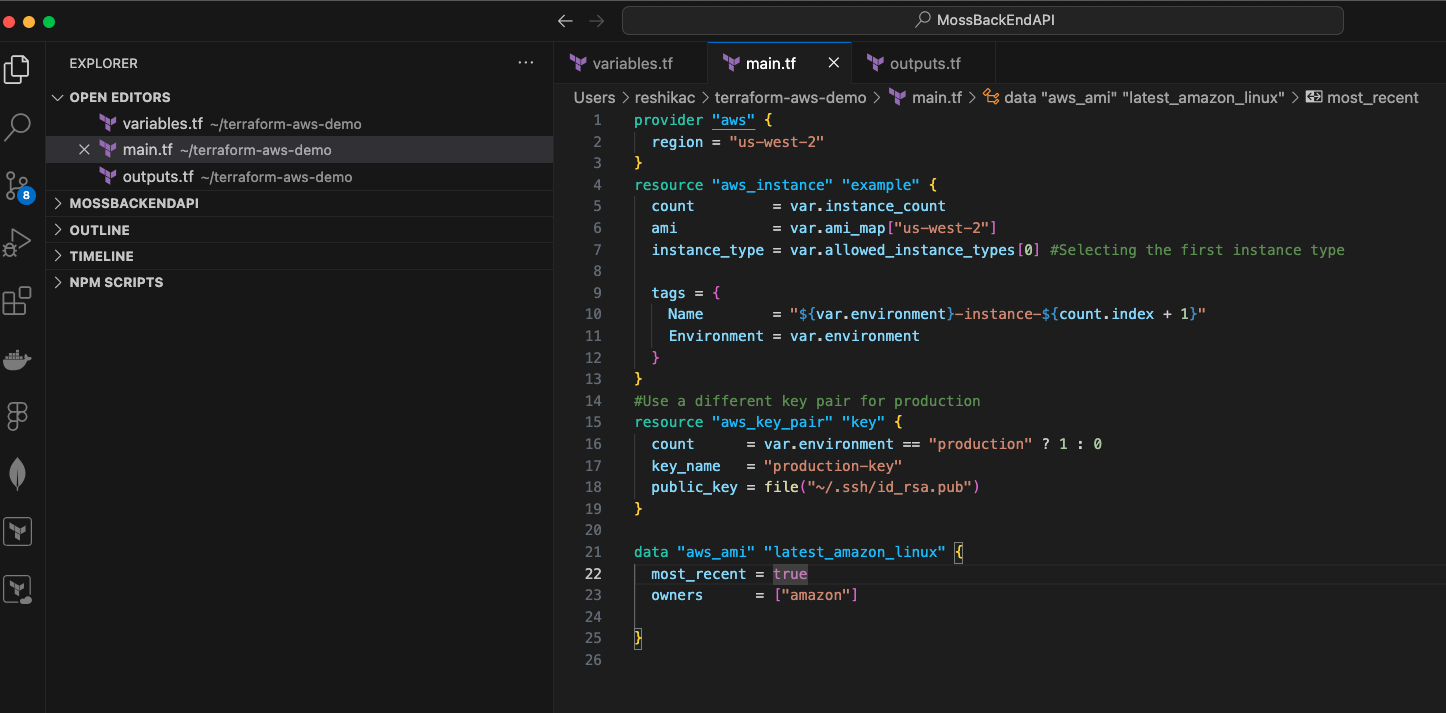
tags = {

Name = "${var.environment}-instance-${count.index + 1}"

Environment = var.environment

}

}

```

2. Use conditional logic to modify resource configurations:

```hcl

Use a different key pair for production

resource "aws\_key\_pair" "key" {

count = var.environment == "production" ? 1 : 0

key\_name = "production-key"

public\_key = file("~/.ssh/id\_rsa.pub")

}

Add a different tag for production instances

resource "aws\_instance" "example" {

count = var.instance\_count

ami = var.ami\_map[“us-west-2”]

instance\_type = var.allowed\_instance\_types[0]

tags = merge(

{

Name = "${var.environment}-instance-${count.index + 1}"

Environment = var.environment

},

var.environment == "production" ? { "SecurityLevel" = "High" } : {}

)

}

```

Step 4: Output Advanced Information

1. Define outputs in a new file `outputs.tf`:

```bash

touch outputs.tf

```

2. Capture and output advanced information :

```hcl

output "instance\_ids" {

description = "IDs of all instances"

value = aws\_instance.example.\*.id

}

output "production\_key\_name" {

description = "Name of the production key pair"

value = aws\_key\_pair.key[0].key\_name

condition = var.environment == "production"

}

```

Step 5: Data Sources in Terraform

Data sources allow Terraform to query information from AWS or other providers that can be used in your configuration.

1. Add a data source in `main.tf`:

```hcl

data "aws\_ami" "latest\_amazon\_linux" {

most\_recent = true

owners = ["amazon"]

}

resource "aws\_instance" "latest\_amazon\_linux" {

ami = data.aws\_ami.latest\_amazon\_linux.id

instance\_type = var.allowed\_instance\_types[0]

tags = {

Name = "AmazonLinuxInstance"

}

}

```

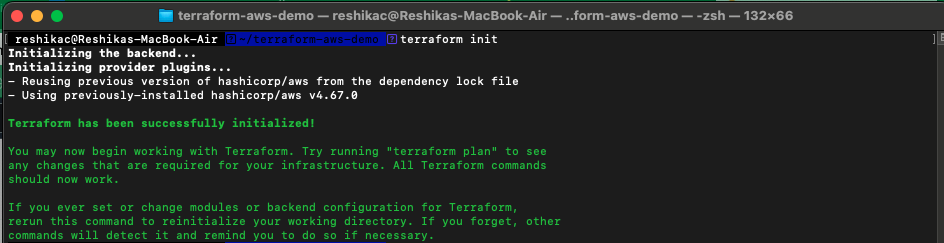
Step 6: Initialize and Apply the Terraform Configuration

1. Initialize Terraform :

```bash

terraform init

```

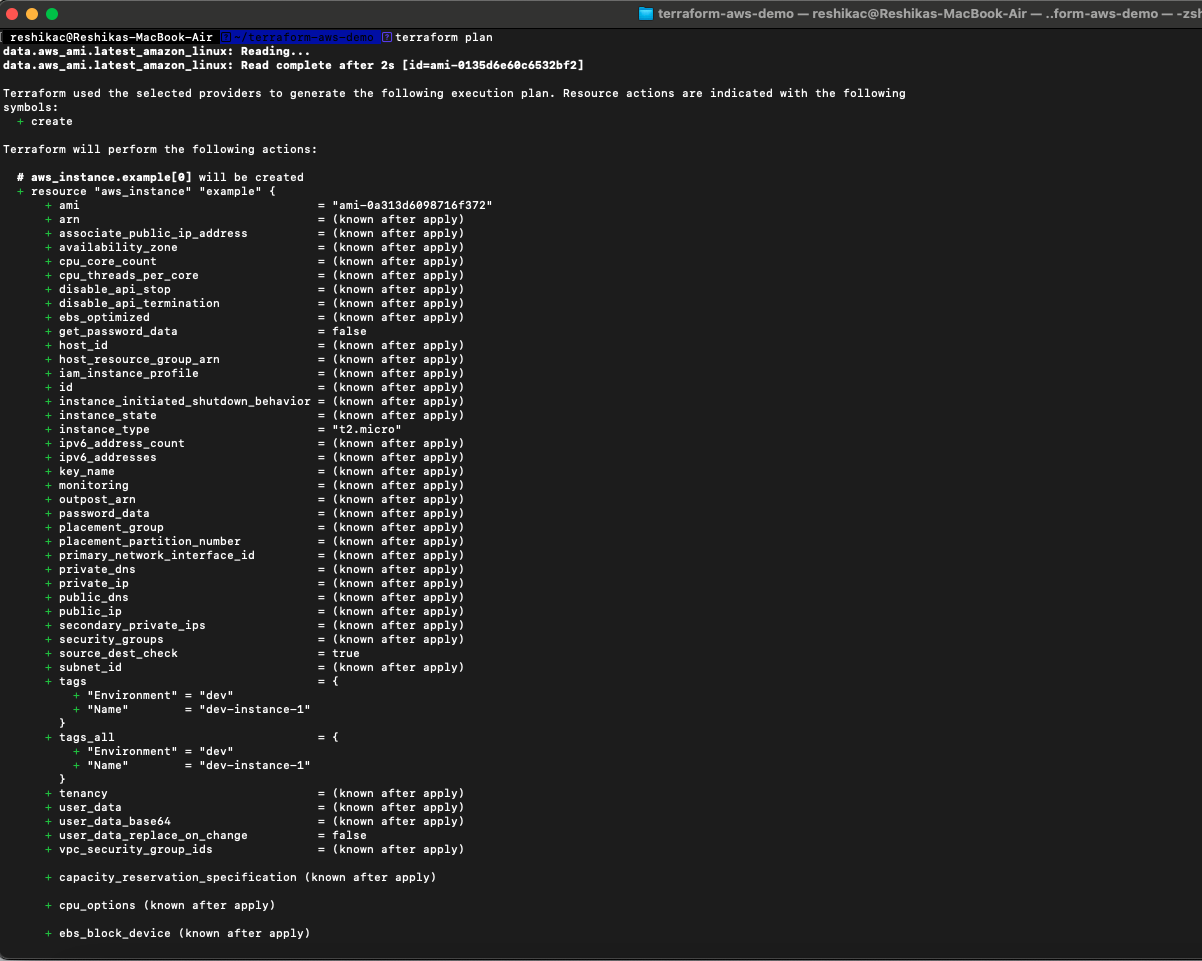


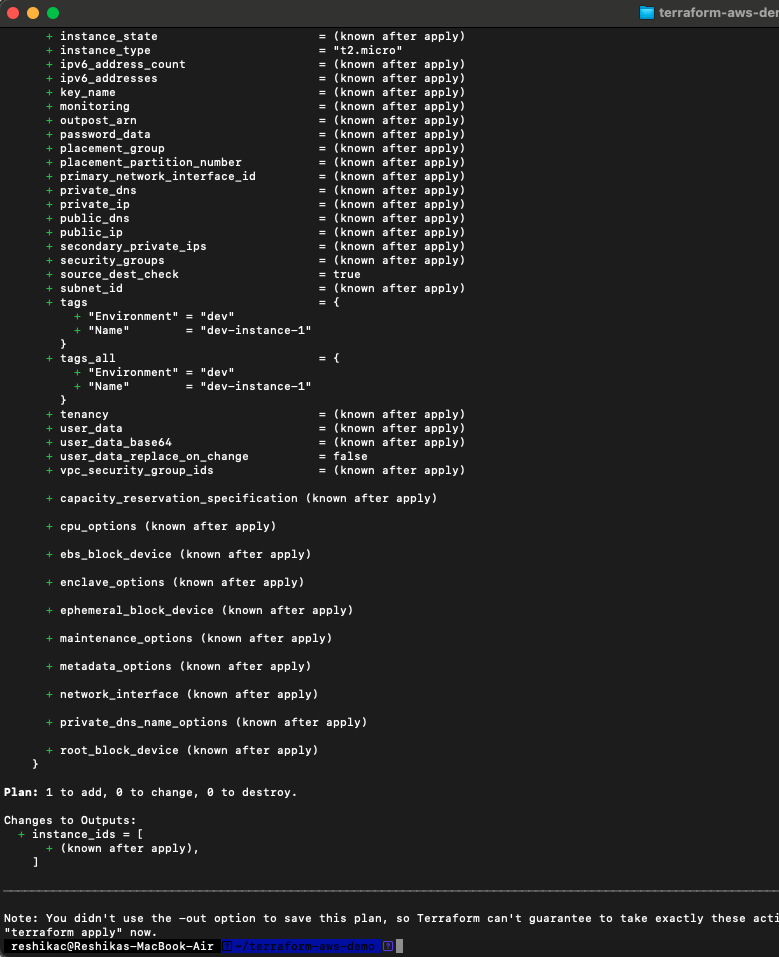
2. Apply the Terraform configuration :

```bash

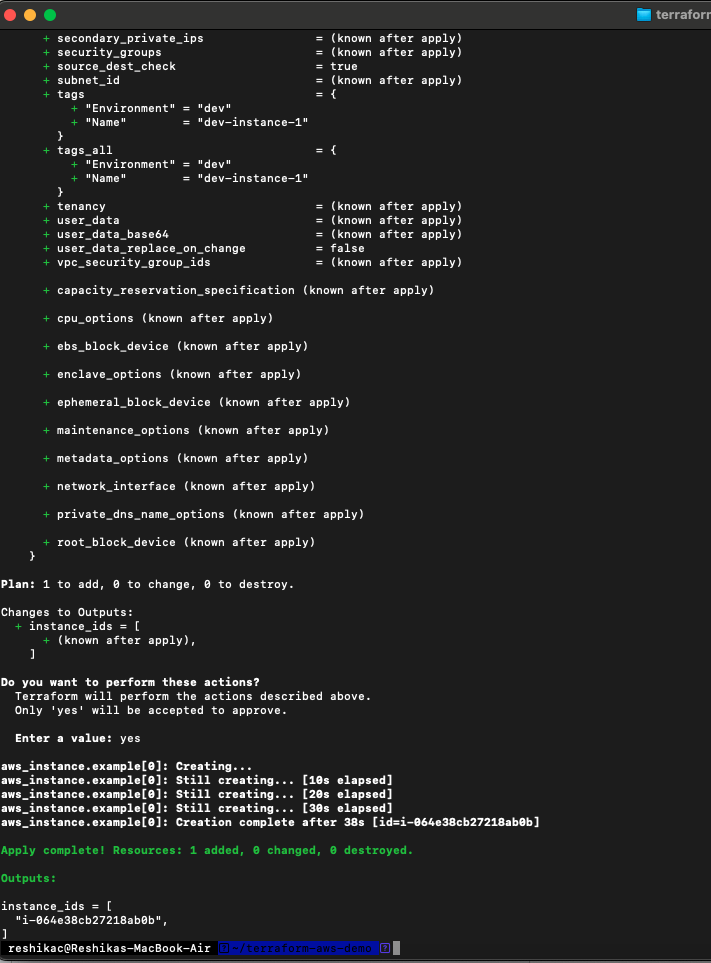
terraform apply

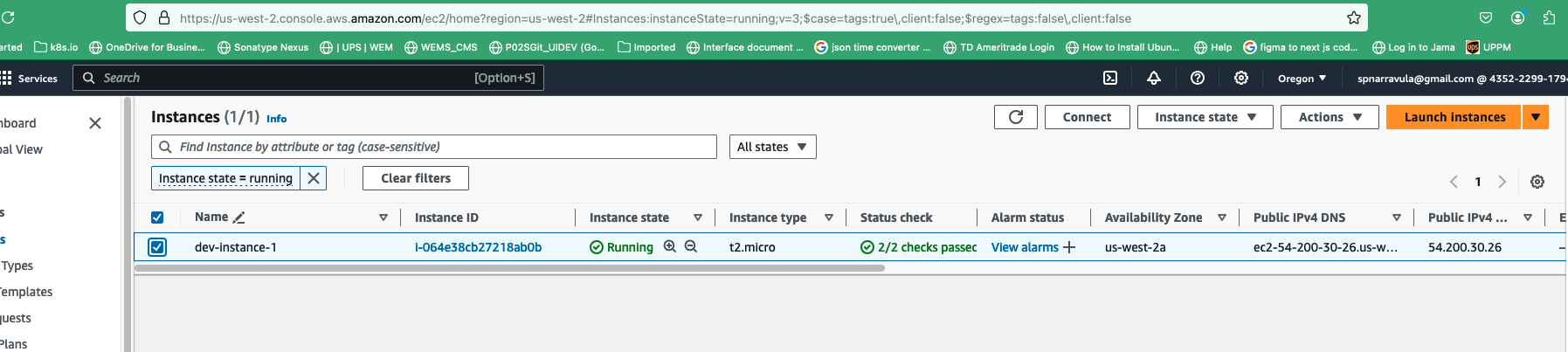
```





- Terraform will display a plan. Review it and type `yes` to confirm and deploy.





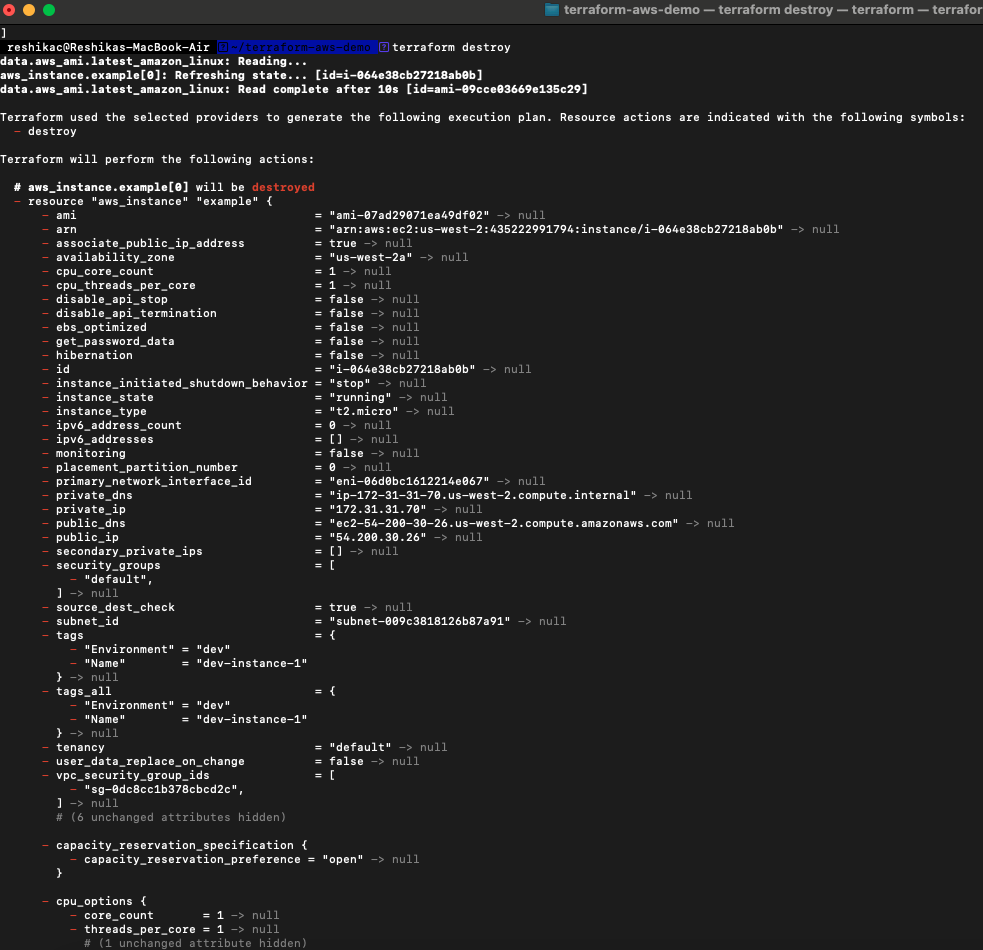
Step 7: Cleanup

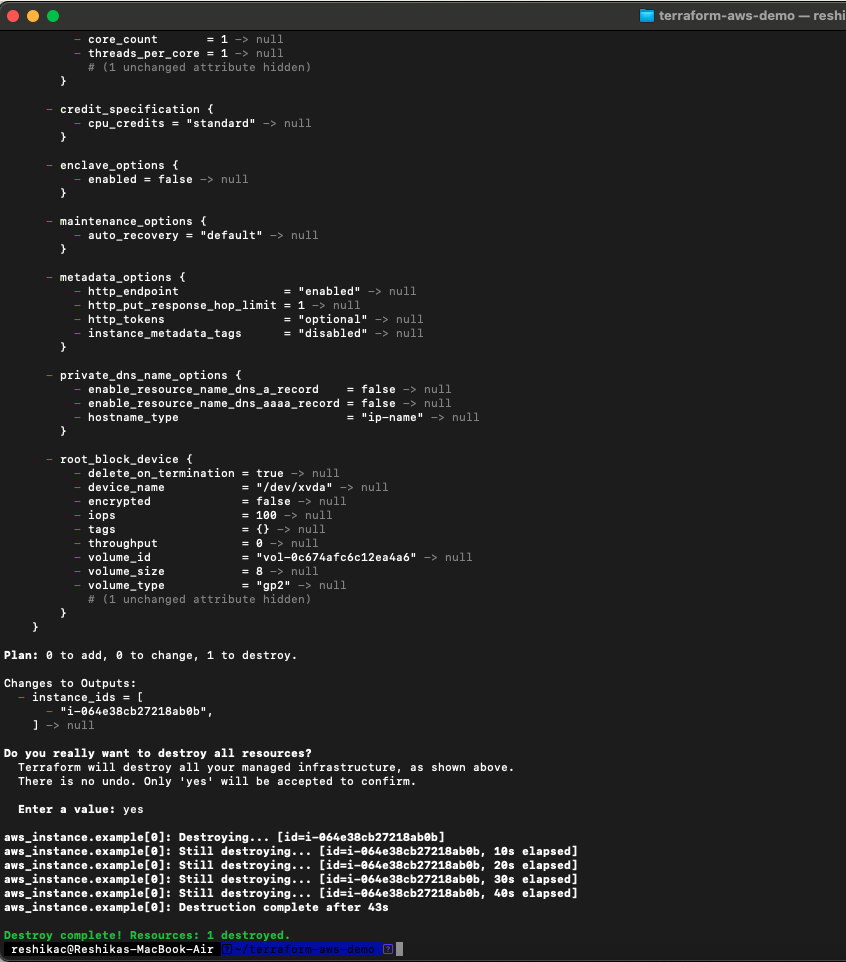
1. Destroy the resources once you're done:

```bash

terraform destroy

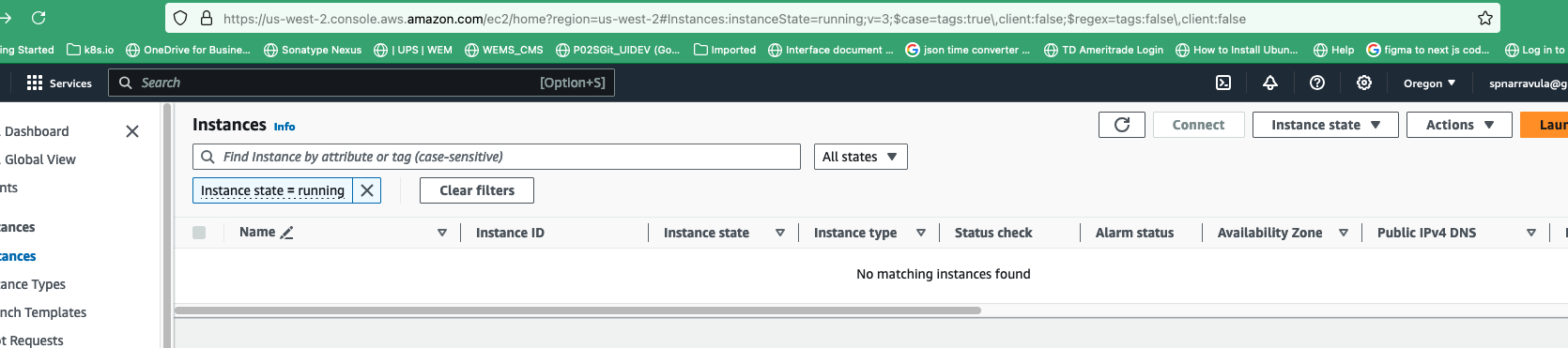
```





- Review the resources that will be destroyed, then type `yes`.

Conclusion



In this advanced tutorial, you learned how to use more complex variable types like lists and maps, apply conditional logic, and validate variables in Terraform. You also explored using data sources to dynamically query information from AWS. These features enhance the flexibility and power of your Terraform configurations, making them more robust and adaptable to various scenarios.