Terraform Variables: A Hands-On Tutorial with AWS

Introduction

Terraform is an open-source infrastructure-as-code software tool that provides a consistent CLI workflow to manage hundreds of cloud services. In this tutorial, we'll explore how to use Terraform parameters (variables and outputs) to manage resources on AWS effectively.

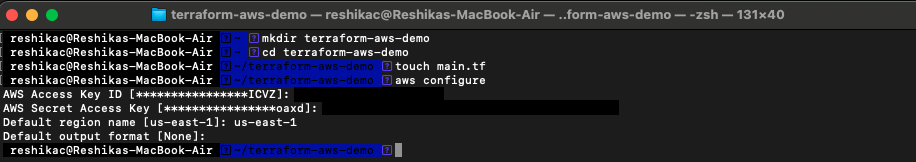
Prerequisites

Before we dive in, make sure you have the following:

1. \*\*AWS Account\*\*: Sign up for an AWS account if you don't already have one.

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

3. \*\*AWS CLI Configured\*\*: Install and configure AWS CLI with your credentials. You can follow the [AWS CLI documentation](https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html) for setup.



Step 1: Set Up Your Project Directory

1. \*\*Create a new directory\*\* for your Terraform project:

```bash

mkdir terraform-aws-demo

cd terraform-aws-demo

```

2. \*\*Create the main configuration file\*\* (`main.tf`):

```bash

touch main.tf

```

Step 2: Define Variables in Terraform

Terraform variables are a way to pass dynamic values to your configuration. This allows for more reusable and modular code.

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

```bash

touch variables.tf

```

2. \*\*Define the variables\*\* in `variables.tf`:

```hcl

variable "region" {

description = "The AWS region where resources will be created"

default = "us-east-1"

}

variable "instance\_type" {

description = "The type of instance to use for the EC2 instance"

default = "t2.micro"

}

variable "instance\_name" {

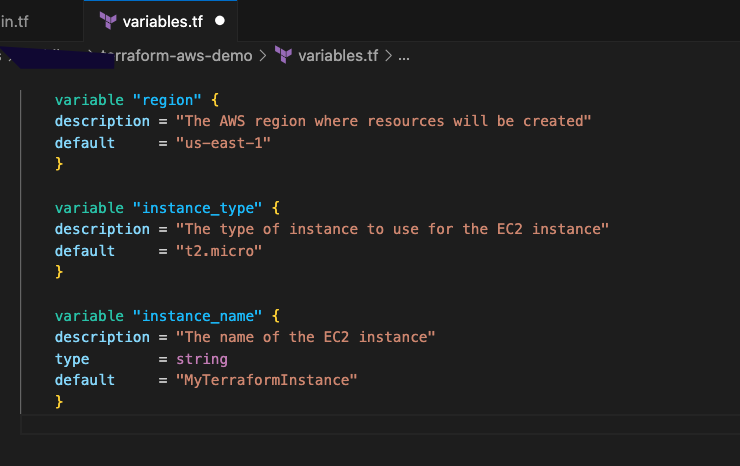
description = "The name of the EC2 instance"

type = string

default = "MyTerraformInstance"

}

```



Step 3: Create AWS Resources Using Variables

1. \*\*Define your AWS provider\*\* in `main.tf`:

```hcl

provider "aws" {

region = var.region

}

```

2. \*\*Create an EC2 instance resource\*\* using the defined variables:

```hcl

resource "aws\_instance" "example" {

ami = "ami-0c55b159cbfafe1f0" Amazon Linux 2 AMI ID for us-east-1

instance\_type = var.instance\_type

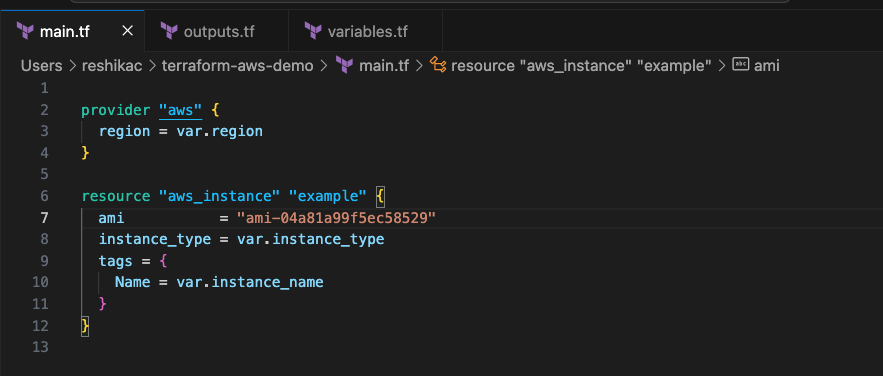
tags = {

Name = var.instance\_name

}

}

```



Step 4: Output the Instance Details

Terraform outputs allow you to extract and use information from your resources.

1. \*\*Create an output file\*\* (`outputs.tf`):

```bash

touch outputs.tf

```

2. \*\*Define outputs\*\* in `outputs.tf` to capture important information about the created resources:

```hcl

output "instance\_id" {

description = "The ID of the EC2 instance"

value = aws\_instance.example.id

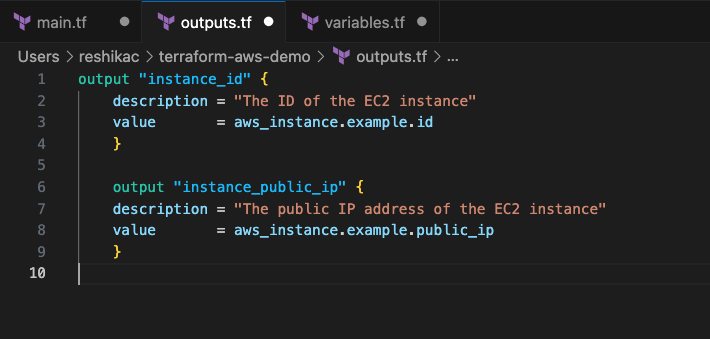
}

output "instance\_public\_ip" {

description = "The public IP address of the EC2 instance"

value = aws\_instance.example.public\_ip

}



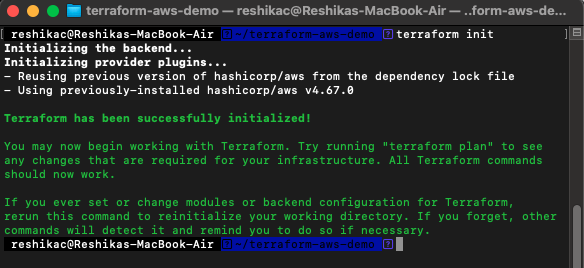
Step 5: Initialize and Apply Terraform Configuration

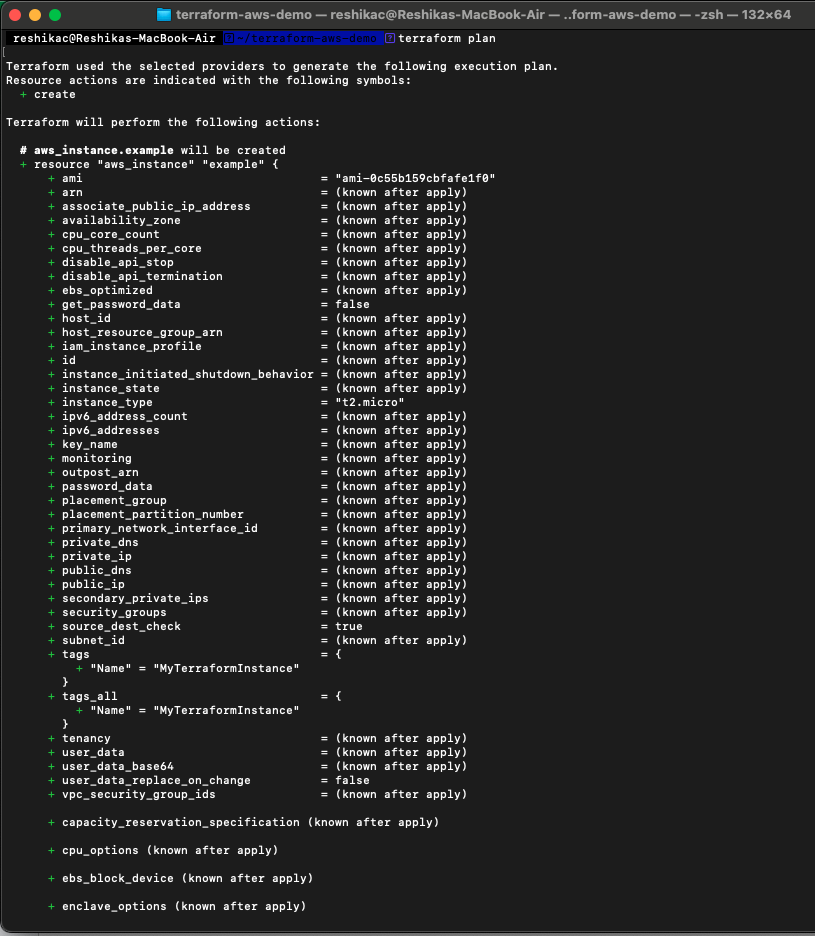
1. \*\*Initialize Terraform\*\* to download the required providers:

```bash

terraform init

```



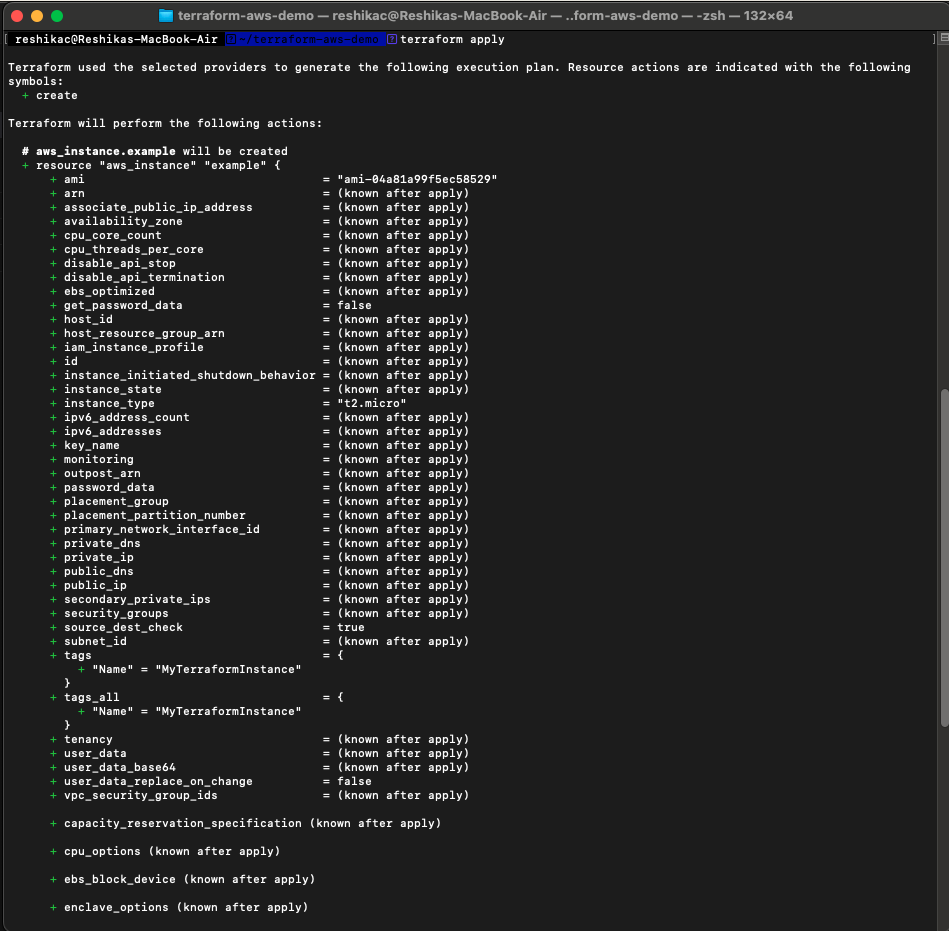


2. \*\*Apply the Terraform configuration\*\* to create the AWS resources:

```bash

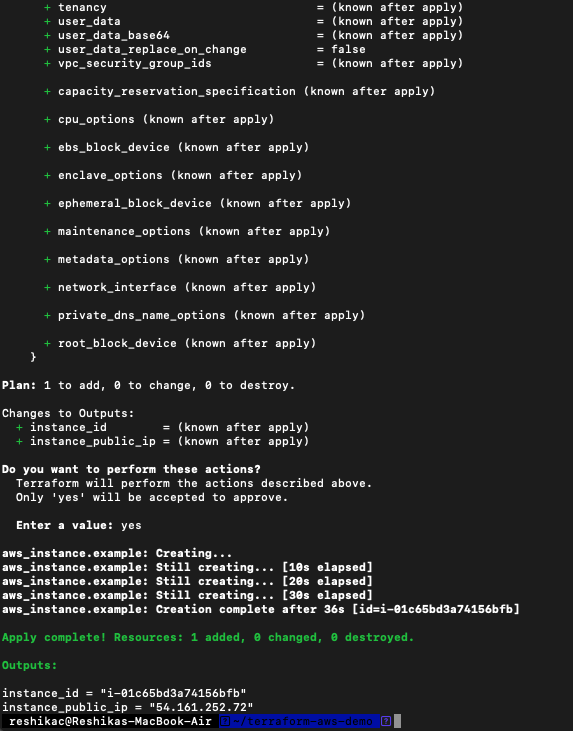
terraform apply

```



- Terraform will show you a plan of what it will create. Type `yes` to confirm and apply the changes.

Step 6: Review the Outputs



Once the apply is complete, Terraform will display the output values. These are the values we defined in the `outputs.tf` file:

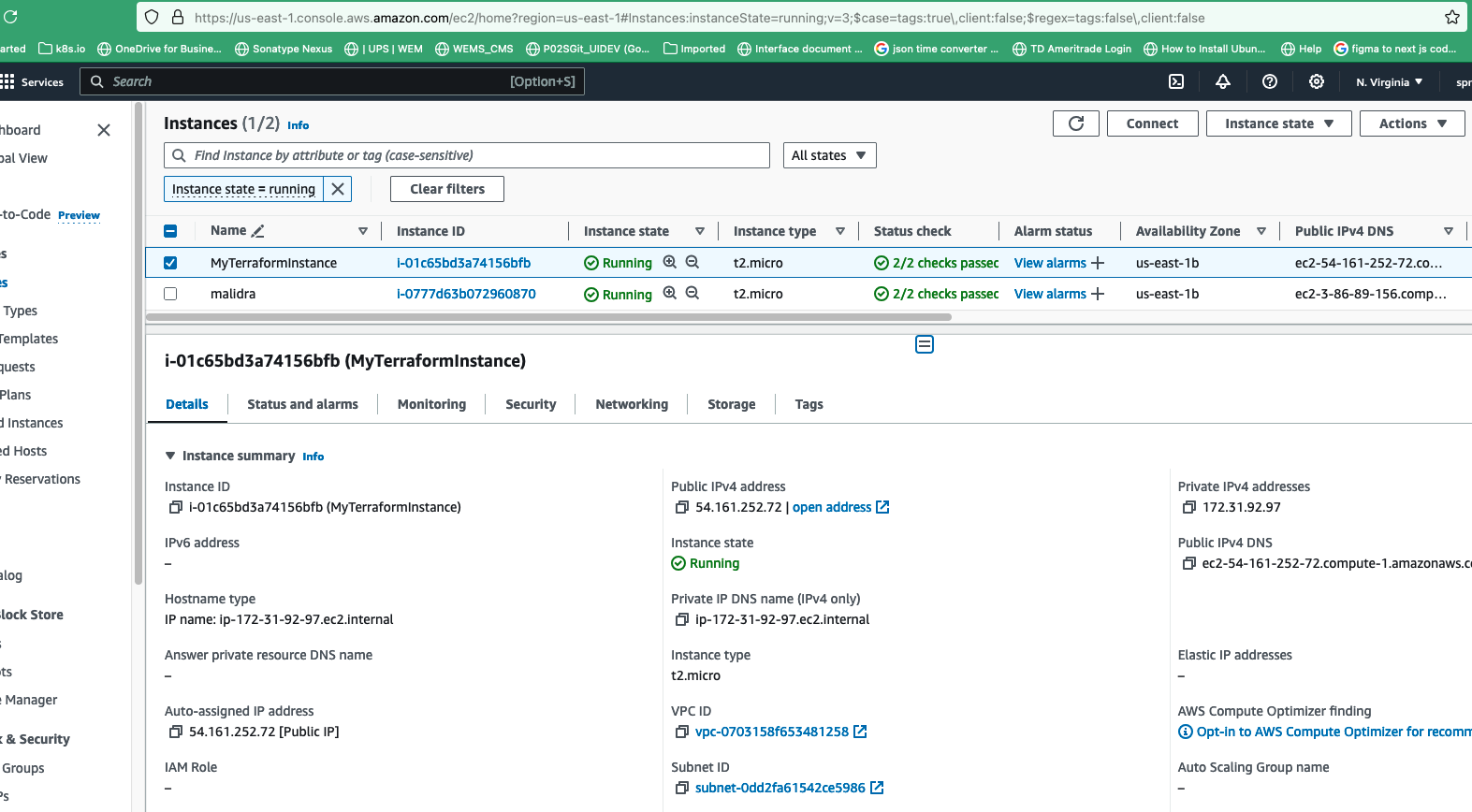
```bash

Outputs:

instance\_id = "i-01c65bd3a74156bfb"

instance\_public\_ip = "54.161.252.72"

```



Step 7: Cleanup

When you're done with your infrastructure, it's good practice to clean up resources to avoid unnecessary costs.

1. \*\*Destroy the resources\*\* created by Terraform:

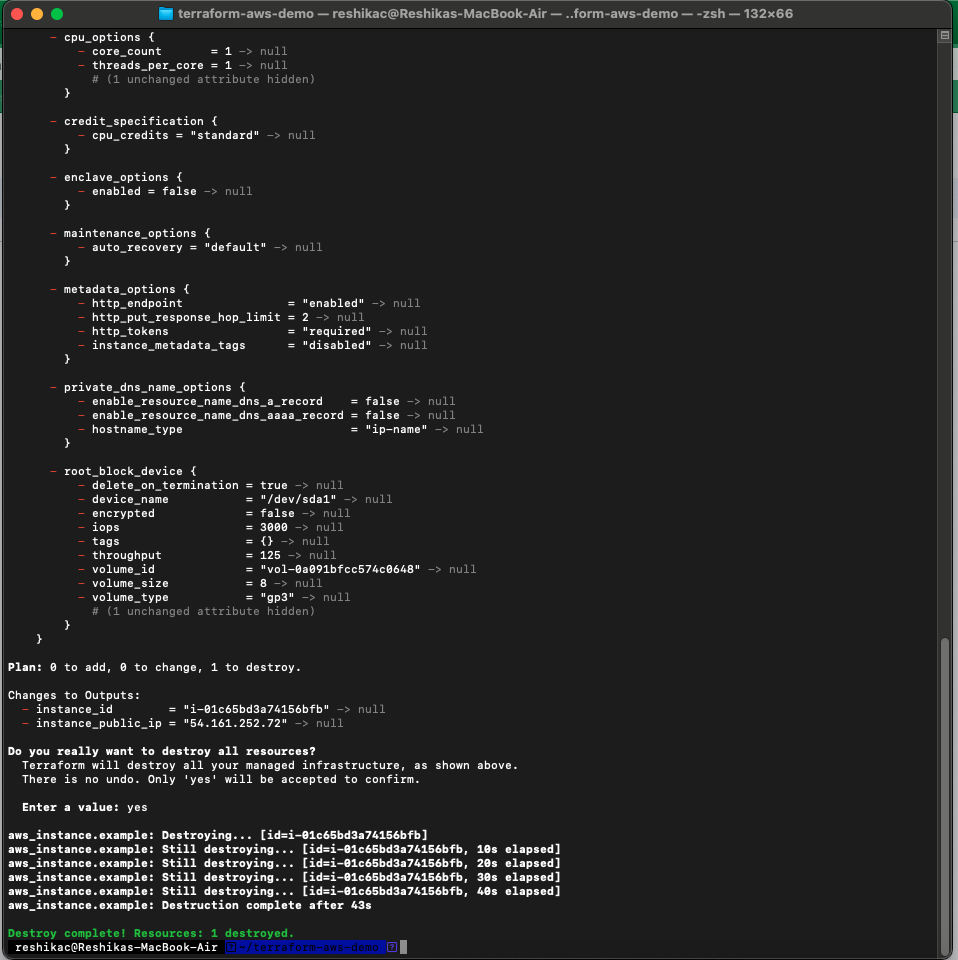
```bash

terraform destroy

```

- Terraform will show you what will be destroyed. Type `yes` to confirm.





Conclusion

In this tutorial, you learned how to define and use variables in Terraform to create and manage AWS resources. You also learned how to output important information about the resources you create. By using variables, you can make your Terraform configurations more flexible and reusable.