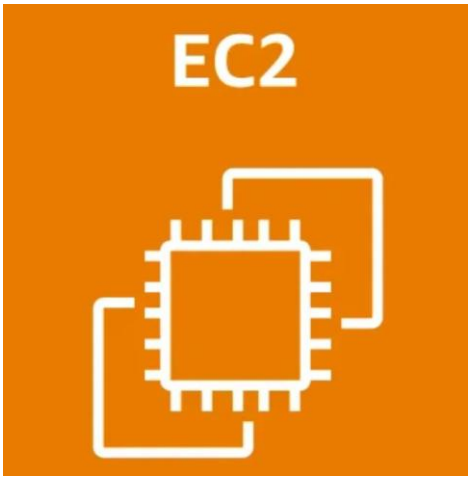
# Provision an EC2 Instance on Amazon Web Services (AWS) Using Terraform



# Contents



# Table of Figures

## Introduction

This tutorial is a hands-on project I created to demonstrate how to provision an EC2 instance on Amazon Web Services (AWS) using Terraform. As a student passionate about cloud computing and DevOps, I wanted to practically explore Infrastructure as Code (IaC) and share my learning experience with others who are on a similar path.

Cloud infrastructure is the backbone of today's digital services, and managing it efficiently is crucial. Terraform is a powerful tool that allows developers and IT professionals to define and provision infrastructure consistently and repeatably. In this tutorial, I document every step I followed, from setting up AWS credentials to writing and executing Terraform code to launch a virtual machine (EC2 instance).

This guide is designed to be beginner-friendly and easy to follow. Whether you're a student, an aspiring DevOps engineer, or simply exploring automation tools, this project will give you a clear, practical introduction to provisioning resources on AWS using Terraform.

By the end, you'll understand how to launch your first EC2 instance and gain insight into writing Terraform scripts, using the command line interface (CLI), and following best practices for cloud automation.

## Prerequisites

- AWS Account with administrative access
- IAM user with programmatic access (Access Key ID and Secret Access Key)
- Terraform is installed on your local machine
- A basic understanding of AWS and EC2 concepts

## Step-by-Step Guide

### Step 1: Configure AWS Credentials

Ensure you have your AWS Access Key ID and Secret Access Key ready. Configure them using the AWS CLI:

***export AWS_ACCESS_KEY_ID=***

***export AWS_SECRET_ACCESS_KEY=***

Create a directory for your configuration.

***mkdir learn-terraform-aws-instance***

Change into the directory.

***cd learn-terraform-aws-instance***

Create a file to define infrastructure.

***touch main.tf***

Open main.tf in your text editor, paste in the configuration below, and save the file.

***nano main.tf***

**Configuration File**

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.16"
    }
  }
 required_version = ">= 1.2.0"
}
provider "aws" {
  region  = "us-west-2"
}
resource "aws_instance" "app_server" {
  ami           = "ami-830c94e3"
  instance_type = "t2.micro"

  tags = {
    Name = "ExampleAppServerInstance"
  }
}
```

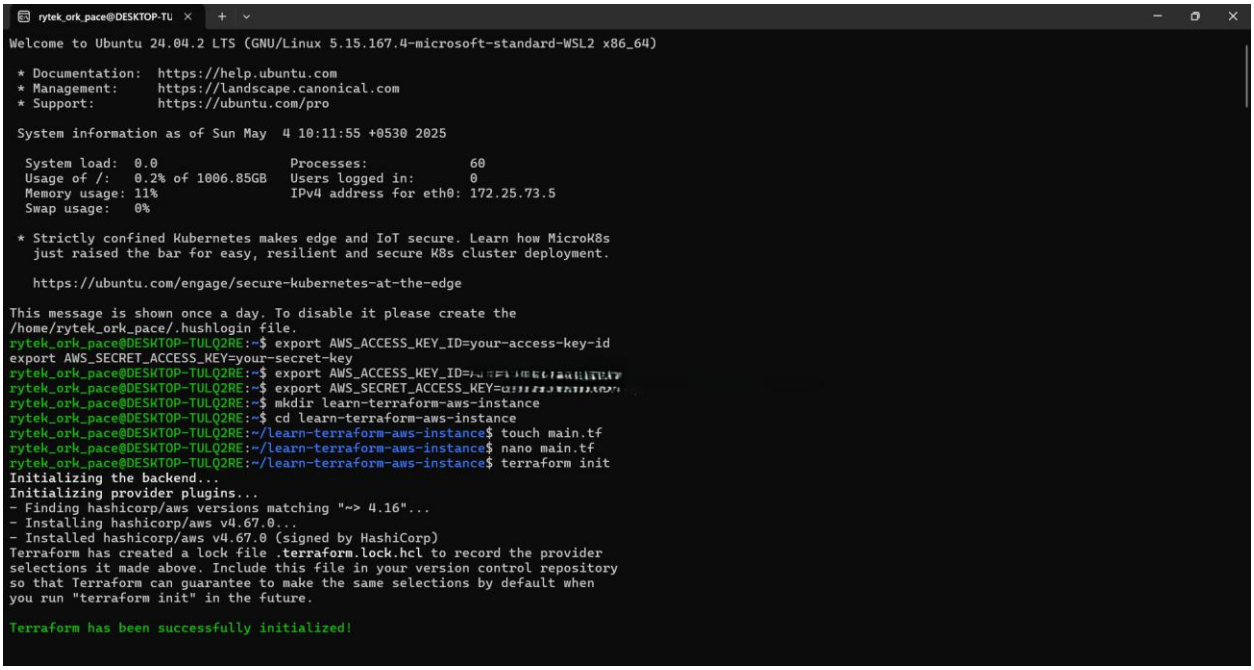



**Figure 1: Configure AWS Credentials**

## Step 3: Initialize Terraform

In your terminal, navigate to the project directory and initialize Terraform:

***terraform init***

```
rytek_ork_pace@DESKTOP-TULQ2RE:~/learn-terraform-aws-instance$ terraform ini
t
Initializing the backend...
Initializing provider plugins...
- Finding hashicorp/aws versions matching "~> 4.16"...
- Installing hashicorp/aws v4.67.0...
- Installed hashicorp/aws v4.67.0 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control reposito
ry
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to se
e
any changes that are required for your infrastructure. All Terraform command
s
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, ot
her
commands will detect it and remind you to do so if necessary.
rytek_ork_pace@DESKTOP-TULQ2RE:~/learn-terraform-aws-instance$ terraform fmt
main.tf
rytek_ork_pace@DESKTOP-TULQ2RE:~/learn-terraform-aws-instance$ terraform val
idate
Success! The configuration is valid.

rytek_ork_pace@DESKTOP-TULQ2RE:~/learn-terraform-aws-instance$ terraform app
ly
```

Figure 2: Initialize Terraform

## Step 4: Format and Validate the Configuration

Format your configuration.

***terraform fmt***

Check the syntax and configuration with:

***terraform validate***

## Step 5: Apply the Configuration

Run the following command to create the EC2 instance:

***terraform apply***

Confirm the action by typing `yes` when prompted.

## Step 6: Verify the Instance

Log in to the AWS Management Console and go to the EC2 Dashboard to confirm the instance is running.

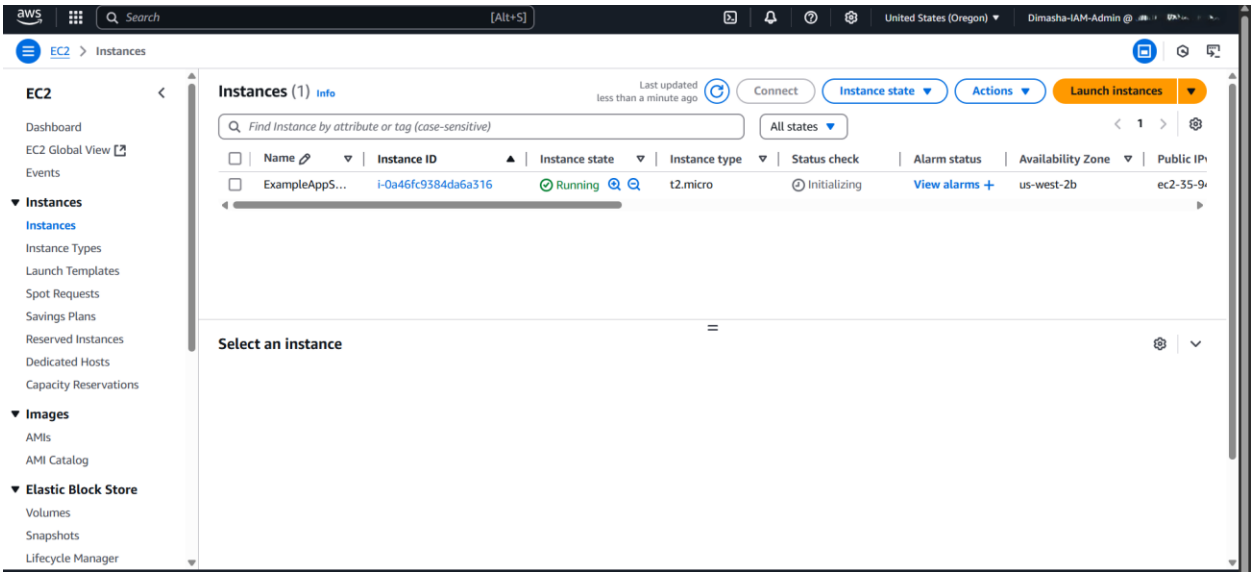



Figure 3: EC2 Instance running in the AWS

## Step 7: Destroy the created resources

To delete the resources created:

***terraform destroy***

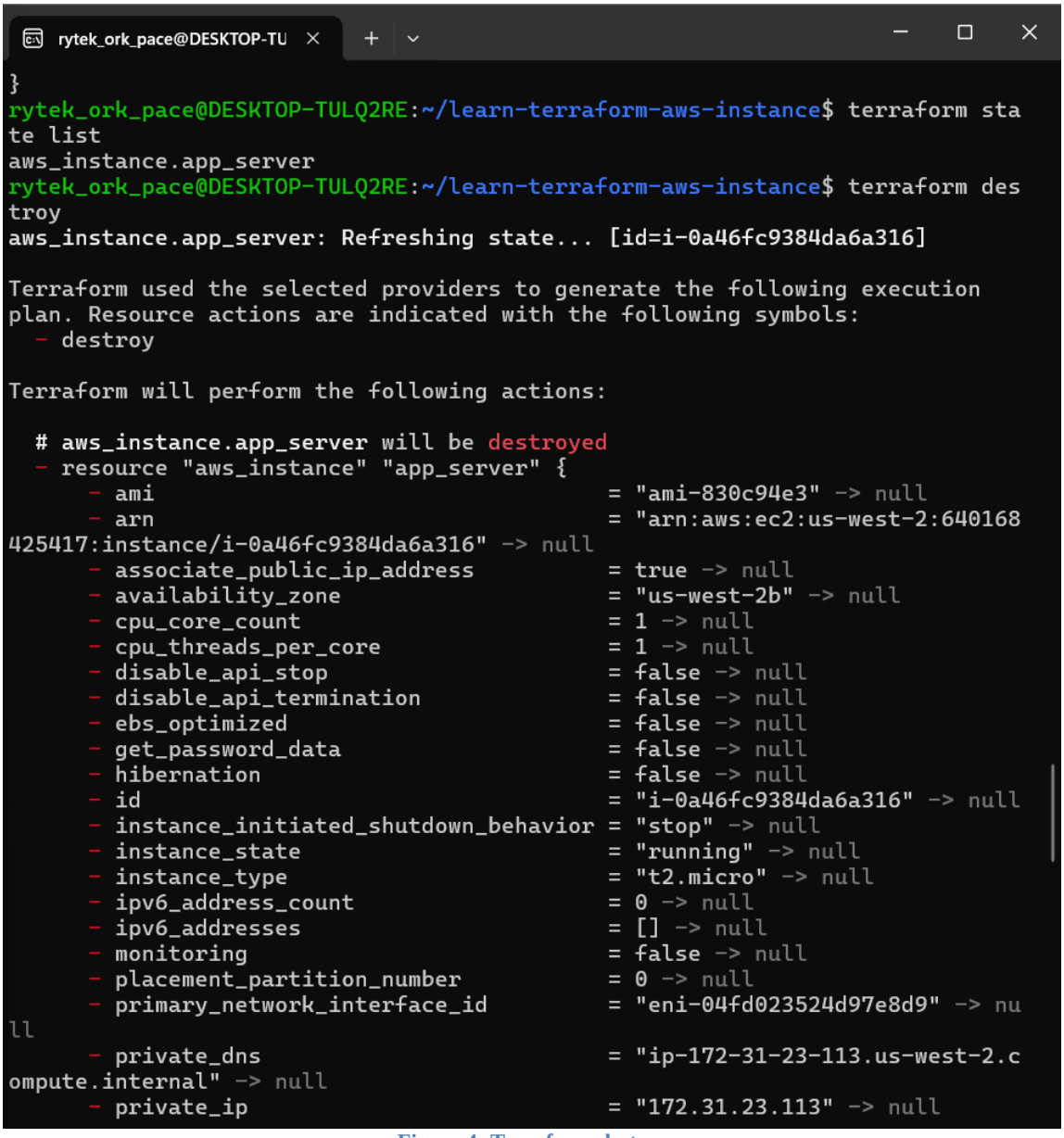



Figure 4: Terraform destroy

Type `yes` to confirm.

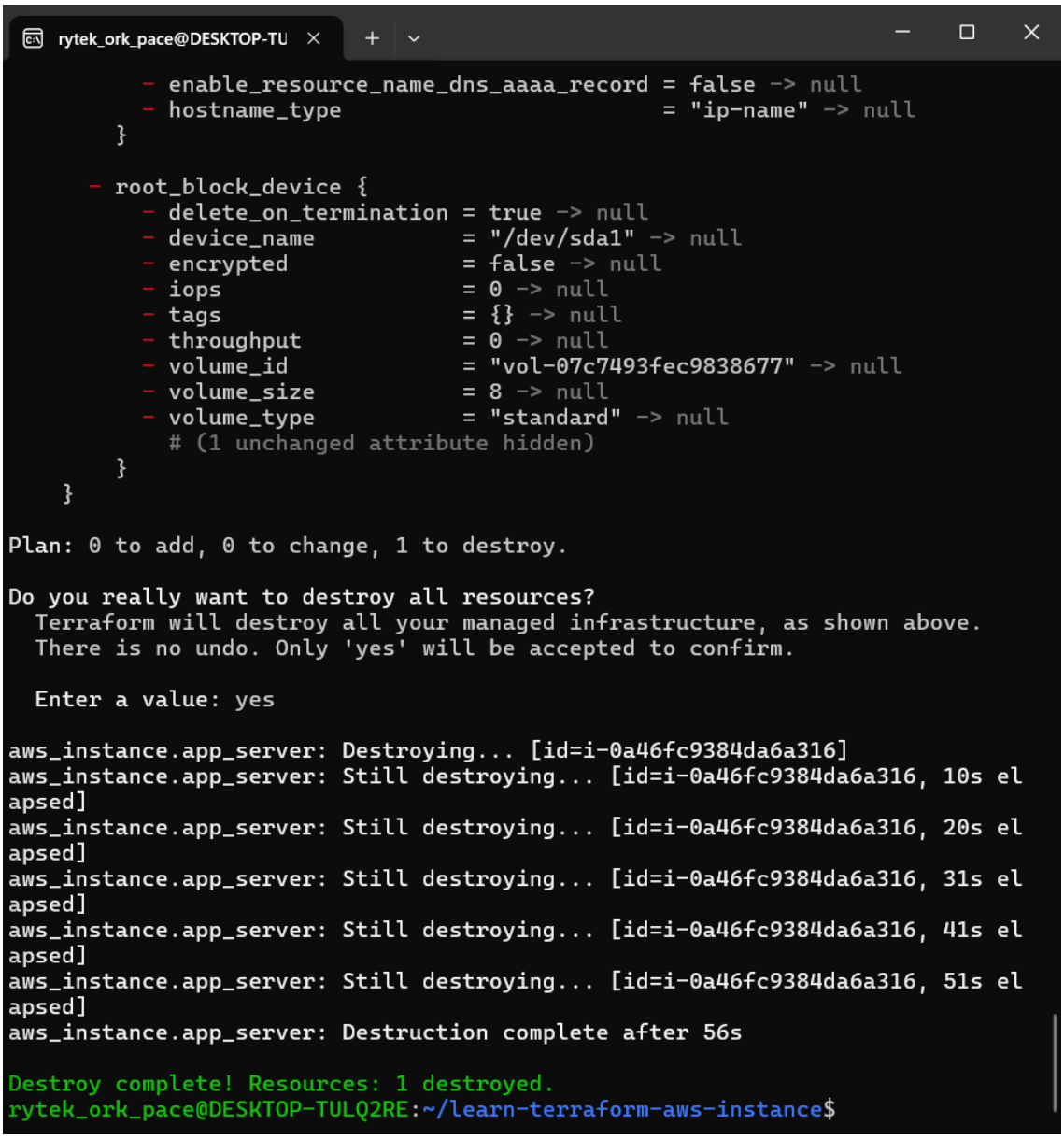



**Figure 5: Terraform destroy**

## Conclusion

This project taught me the importance of automation and repeatability in managing cloud infrastructure. Using Terraform, I was able to easily define, provision, and manage an EC2 instance on AWS — all through code. It demonstrated how powerful and scalable infrastructure as code can be, especially when combined with cloud platforms like AWS.

Through this experience, I've strengthened my skills in Terraform, understood the workflow of provisioning resources, and gained confidence in using the command-line interface for DevOps tasks. This is just the beginning of my cloud journey, and I'm excited to explore more advanced Terraform configurations, integrate automation with CI/CD pipelines, and continue building scalable and secure infrastructure solutions.

I hope this guide serves as a helpful resource for others stepping into the world of cloud computing and DevOps engineering.