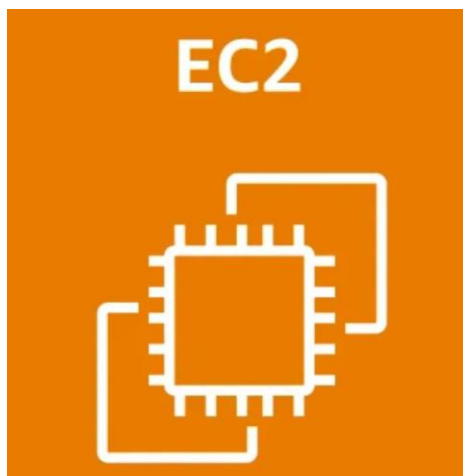


# **Provision an EC2 Instance on Amazon Web Services (AWS) Using Terraform**



## Contents

Introduction .....	3
Prerequisites .....	3
Step-by-Step Guide .....	4
Step 1: Configure AWS Credentials .....	4
Step 3: Initialize Terraform .....	6
Step 4: Format and Validate the Configuration .....	7
Step 5: Apply the Configuration .....	8
Step 6: Verify the Instance .....	8
Step 7: Destroy the created resources .....	9
Conclusion .....	11

## Table of Figures

Figure 1: Configure AWS Credentials.....	6
Figure 2: Initialize Terraform .....	7
Figure 3: EC2 Instance running in the AWS .....	8
Figure 4: Terraform destroy .....	9
Figure 5: Terraform destroy .....	10

## 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"  
  
  }  
  
}
```

```
rytek_ork_pace@DESKTOP-TU x + v
Welcome to Ubuntu 24.04.2 LTS (GNU/Linux 5.15.167.4-microsoft-standard-WSL2 x86_64)

* Documentation:  https://help.ubuntu.com
* Management:    https://landscape.canonical.com
* Support:       https://ubuntu.com/pro

System information as of Sun May  4 10:11:55 +0530 2025

System load:  0.0      Processes:      60
Usage of /:   0.2% of 1006.85GB   Users logged in:  0
Memory usage: 11%      IPv4 address for eth0: 172.25.73.5
Swap usage:   0%

* Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
  just raised the bar for easy, resilient and secure K8s cluster deployment.

https://ubuntu.com/engage/secure-kubernetes-at-the-edge

This message is shown once a day. To disable it please create the
/home/rytek_ork_pace/.hushlogin file.
rytek_ork_pace@DESKTOP-TULQ2RE:~$ export AWS_ACCESS_KEY_ID=your-access-key-id
export AWS_SECRET_ACCESS_KEY=your-secret-key
rytek_ork_pace@DESKTOP-TULQ2RE:~$ export AWS_ACCESS_KEY_ID=AKIAI44QH8DHBEXAMPLE
rytek_ork_pace@DESKTOP-TULQ2RE:~$ export AWS_SECRET_ACCESS_KEY=ajsdflsjdfkj
rytek_ork_pace@DESKTOP-TULQ2RE:~$ mkdir learn-terraform-aws-instance
rytek_ork_pace@DESKTOP-TULQ2RE:~$ cd learn-terraform-aws-instance
rytek_ork_pace@DESKTOP-TULQ2RE:~/learn-terraform-aws-instance$ touch main.tf
rytek_ork_pace@DESKTOP-TULQ2RE:~/learn-terraform-aws-instance$ nano main.tf
rytek_ork_pace@DESKTOP-TULQ2RE:~/learn-terraform-aws-instance$ terraform init
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 repository
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!
```

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 init
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 repository
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 see
any changes that are required for your infrastructure. All Terraform commands
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, other
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 validate
Success! The configuration is valid.

rytek_ork_pace@DESKTOP-TULQ2RE:~/learn-terraform-aws-instance$ terraform apply

```

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.

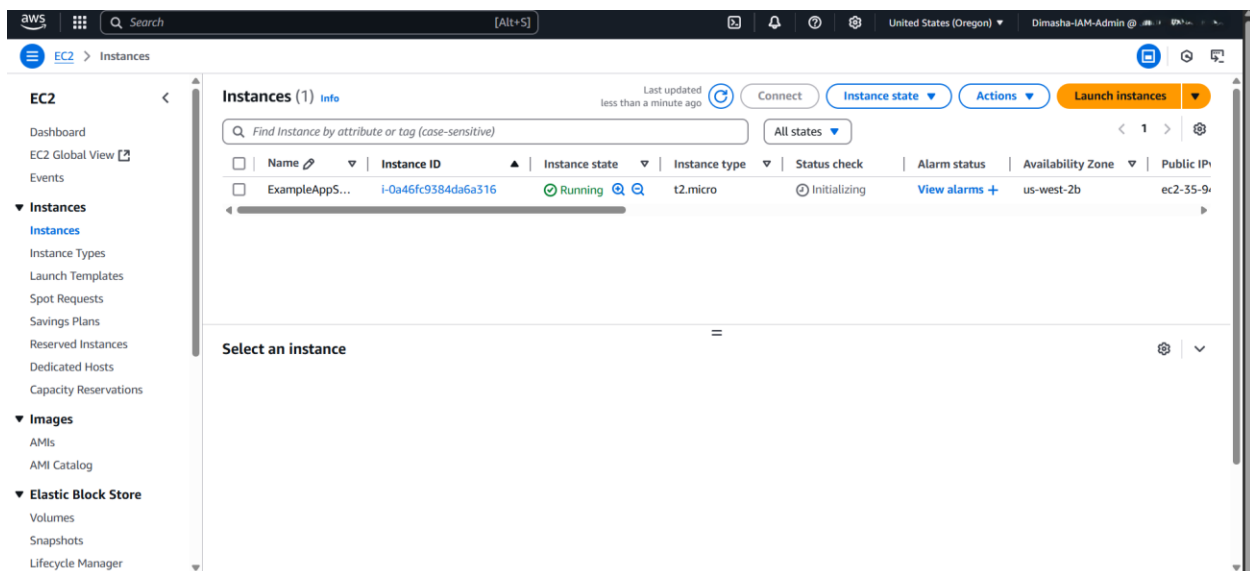


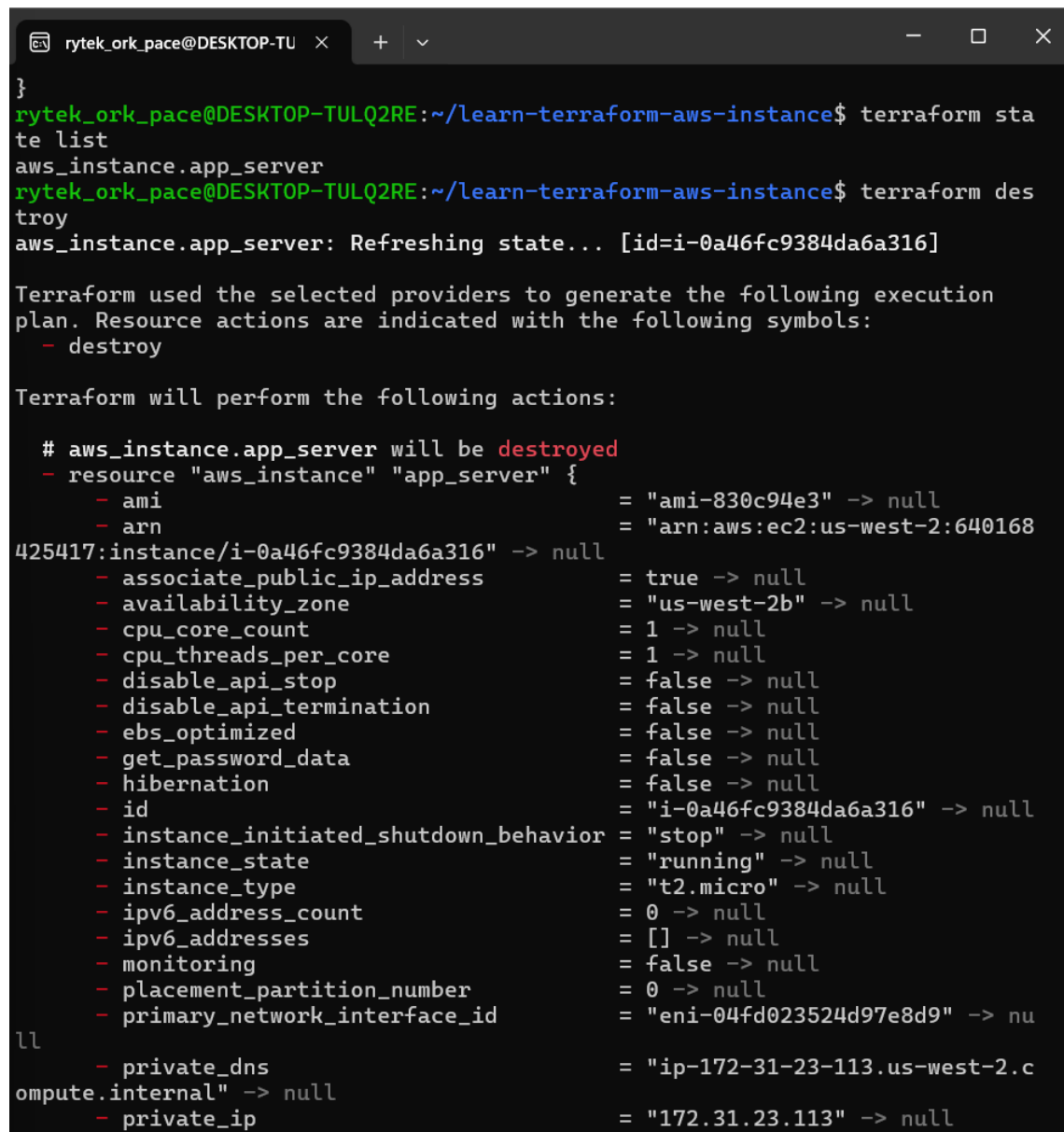
Figure 3: EC2 Instance running in the AWS



## Step 7: Destroy the created resources

To delete the resources created:

*terraform destroy*



```
rytek_ork_pace@DESKTOP-TU x + v
}
rytek_ork_pace@DESKTOP-TULQ2RE:~/learn-terraform-aws-instance$ terraform sta
te list
aws_instance.app_server
rytek_ork_pace@DESKTOP-TULQ2RE:~/learn-terraform-aws-instance$ terraform des
troy
aws_instance.app_server: Refreshing state... [id=i-0a46fc9384da6a316]

Terraform used the selected providers to generate the following execution
plan. Resource actions are indicated with the following symbols:
  - destroy

Terraform will perform the following actions:

# aws_instance.app_server will be destroyed
- resource "aws_instance" "app_server" {
  - ami                    = "ami-830c94e3" -> null
  - arn                    = "arn:aws:ec2:us-west-2:640168
425417:instance/i-0a46fc9384da6a316" -> null
  - associate_public_ip_address = true -> null
  - availability_zone         = "us-west-2b" -> null
  - cpu_core_count            = 1 -> null
  - cpu_threads_per_core      = 1 -> null
  - disable_api_stop          = false -> null
  - disable_api_termination   = false -> null
  - ebs_optimized             = false -> null
  - get_password_data         = false -> null
  - hibernation                = false -> null
  - id                        = "i-0a46fc9384da6a316" -> null
  - instance_initiated_shutdown_behavior = "stop" -> null
  - instance_state            = "running" -> null
  - instance_type             = "t2.micro" -> null
  - ipv6_address_count         = 0 -> null
  - ipv6_addresses            = [] -> null
  - monitoring                = false -> null
  - placement_partition_number = 0 -> null
  - primary_network_interface_id = "eni-04fd023524d97e8d9" -> nu
ll
  - private_dns              = "ip-172-31-23-113.us-west-2.c
ompute.internal" -> null
  - private_ip               = "172.31.23.113" -> null
```

Figure 4: Terraform destroy

Type `yes` to confirm.

```
rytek_ork_pace@DESKTOP-TU x + v - □ ×

- enable_resource_name_dns_aaaa_record = false -> null
- hostname_type                         = "ip-name" -> null
}

- root_block_device {
- delete_on_termination = true -> null
- device_name           = "/dev/sda1" -> null
- encrypted             = false -> null
- iops                  = 0 -> null
- tags                  = {} -> null
- throughput            = 0 -> null
- volume_id             = "vol-07c7493fec9838677" -> null
- volume_size           = 8 -> null
- volume_type           = "standard" -> null
  # (1 unchanged attribute hidden)
}
}

Plan: 0 to add, 0 to change, 1 to destroy.

Do you really want to destroy all resources?
Terraform will destroy all your managed infrastructure, as shown above.
There is no undo. Only 'yes' will be accepted to confirm.

Enter a value: yes

aws_instance.app_server: Destroying... [id=i-0a46fc9384da6a316]
aws_instance.app_server: Still destroying... [id=i-0a46fc9384da6a316, 10s elapsed]
aws_instance.app_server: Still destroying... [id=i-0a46fc9384da6a316, 20s elapsed]
aws_instance.app_server: Still destroying... [id=i-0a46fc9384da6a316, 31s elapsed]
aws_instance.app_server: Still destroying... [id=i-0a46fc9384da6a316, 41s elapsed]
aws_instance.app_server: Still destroying... [id=i-0a46fc9384da6a316, 51s elapsed]
aws_instance.app_server: Destruction complete after 56s

Destroy complete! Resources: 1 destroyed.
rytek_ork_pace@DESKTOP-TULQ2RE:~/learn-terraform-aws-instance$
```

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.