🧾 Terraform + AWS Nginx Website Deployment Guide

🌐 Project Summary

Deployed a basic static HTML website on AWS using:

- **Terraform** for Infrastructure as Code (IaC)
- **AWS EC2** for hosting Nginx
- **S3 Bucket** for storing HTML files)
- **Custom VPC**, subnets, gateways, and routing
- **Remote backend using S3 and DynamoDB** for state management

---

⚙️ Backend Configuration in Terraform

```
terraform {
  backend "s3" {
    bucket         = "my-terraform-state-bucket-prince"
    key            = "ec2/nginx-instance.tfstate"
    region         = "us-east-1"
    encrypt        = true
    dynamodb_table = "terraform-lock-table-prince"
  }
}
```

🔍 Explanation (Line-by-Line)

- `terraform`: Start of Terraform block
- `backend "s3"`: Specifies S3 as the remote backend
- `bucket`: S3 bucket name where the `.tfstate` file will be saved
- `key`: File path and name for the state file inside the bucket
- `region`: AWS region for S3 bucket and DynamoDB table
- `encrypt`: Enables server-side encryption
- `dynamodb_table`: Table for state locking (prevents concurrent runs)

🧠 What is a Terraform Backend?

A **backend** is where Terraform stores its state file. It is responsible for:

- Keeping track of resources Terraform creates
- Allowing multiple people to work on the same infrastructure
- Managing state file locking to avoid corruption

✅ Benefits of Remote Backend (S3 + DynamoDB)

Feature	Local State	Remote Backend
Collaboration	❌ Risky	✅ Safe
Locking	❌ None	✅ With DynamoDB
Persistence	❌ Local only	✅ Cloud Storage
Version History	❌ Manual	✅ S3 Versioning

🐣 Bootstrap: Creating the Backend

You cannot use a backend that Terraform depends on unless it already exists. So we **bootstrap** the backend.

Step 1: Create S3 & DynamoDB via Terraform

Do this **without using a backend block**:

```
provider "aws" {
  region = "us-east-1"
}

resource "aws_s3_bucket" "terraform_state" {
  bucket = "my-terraform-state-bucket-prince"

  versioning {
    enabled = true
  }

  server_side_encryption_configuration {
    rule {
      apply_server_side_encryption_by_default {
        sse_algorithm = "AES256"
      }
    }
  }
}

resource "aws_dynamodb_table" "terraform_lock" {
  name         = "terraform-lock-table-prince"
  billing_mode = "PAY_PER_REQUEST"
  hash_key     = "LockID"

  attribute {
    name = "LockID"
    type = "S"
  }
}
```

Then run:

```
terraform init
terraform apply
```

🏗️ Main Infrastructure Deployment

Once the backend exists:

1. Add the backend block in a new project
2. Run `terraform init` to migrate state
3. Add VPC, subnet, EC2, security groups, and Nginx provisioning

EC2 Setup + Nginx

```
sudo dnf update -y
sudo dnf install nginx -y
sudo systemctl start nginx
sudo systemctl enable nginx
```

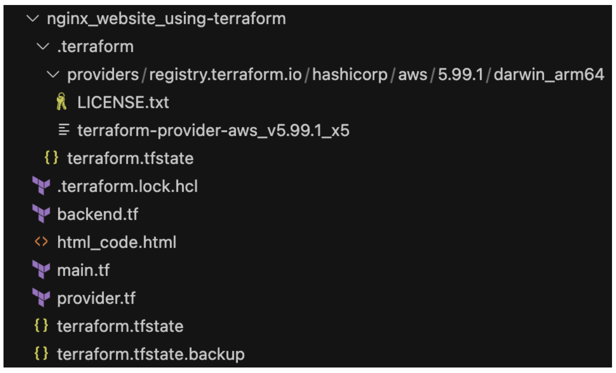
Deploy HTML:

```
echo "<h1>Hello from Nginx on EC2!</h1>" | sudo tee /usr/share/nginx/html/index.html
```

Access in browser: `http://<EC2-PUBLIC-IP>`

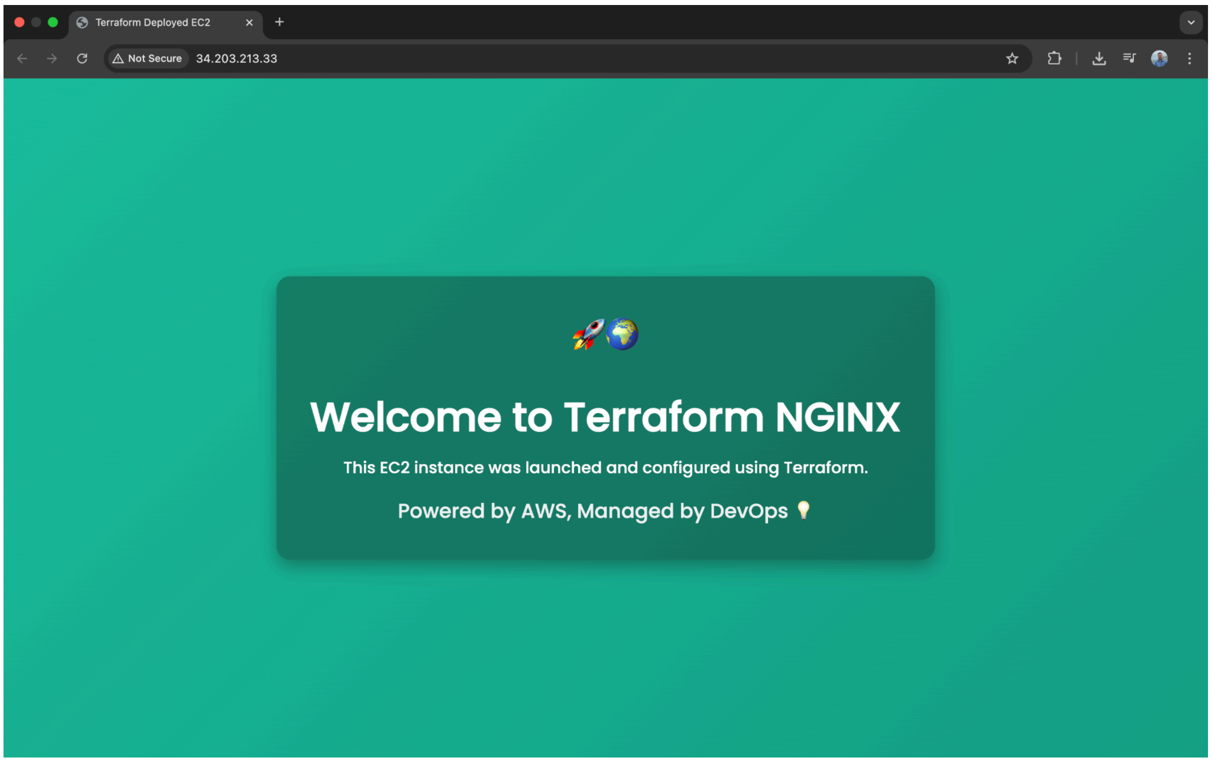
📂 Project Folder Structure

```
nginx_website_using-terraform/
├── main.tf
├── backend.tf
├── provider.tf
├── files/
    └── index.html
```



```
nginx_website_using-terraform
  .terraform
    providers / registry.terraform.io / hashicorp / aws / 5.99.1 / darwin_arm64
      LICENSE.txt
      terraform-provider-aws_v5.99.1_x5
    terraform.tfstate
  .terraform.lock.hcl
  backend.tf
  html_code.html
  main.tf
  provider.tf
  terraform.tfstate
  terraform.tfstate.backup
```

👨‍💻 Final Output



🚀🌍

Welcome to Terraform NGINX

This EC2 instance was launched and configured using Terraform.

Powered by AWS, Managed by DevOps 💡