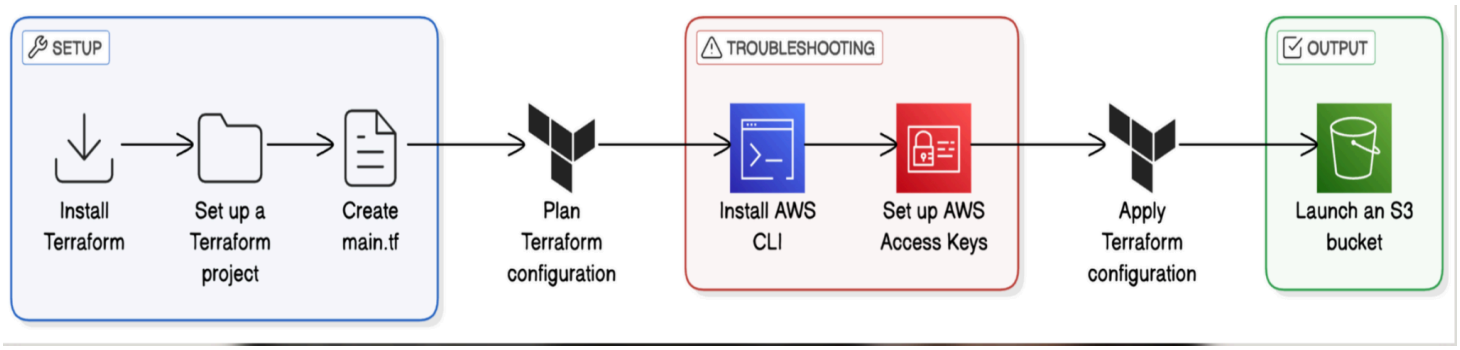
Create S3 Buckets with Terraform

Erik Gonzalez



Introducing Today's Project!

In this project, I will demonstrate installing and configuring Terraform. -Configure your AWS credentials in the terminal. -Create and manage S3 buckets with Terraform. -Upload files to S3 using Terraform. The goal is to gain hands-on experience and learn about Terraform.

Tools and concepts

Services I used were Terraform, S3, CLI, and TextEdit. Key concepts I learnt include infrastructure as code, creating configuration files, modularity of code, using provider plugin state files, lock files, Terraform concepts.

Project reflection

This project took me approximately two hours or learning and hands-on practice. The most challenging part was fixing the errors and coding part. It was most rewarding to see my bucket appear on my AWS!

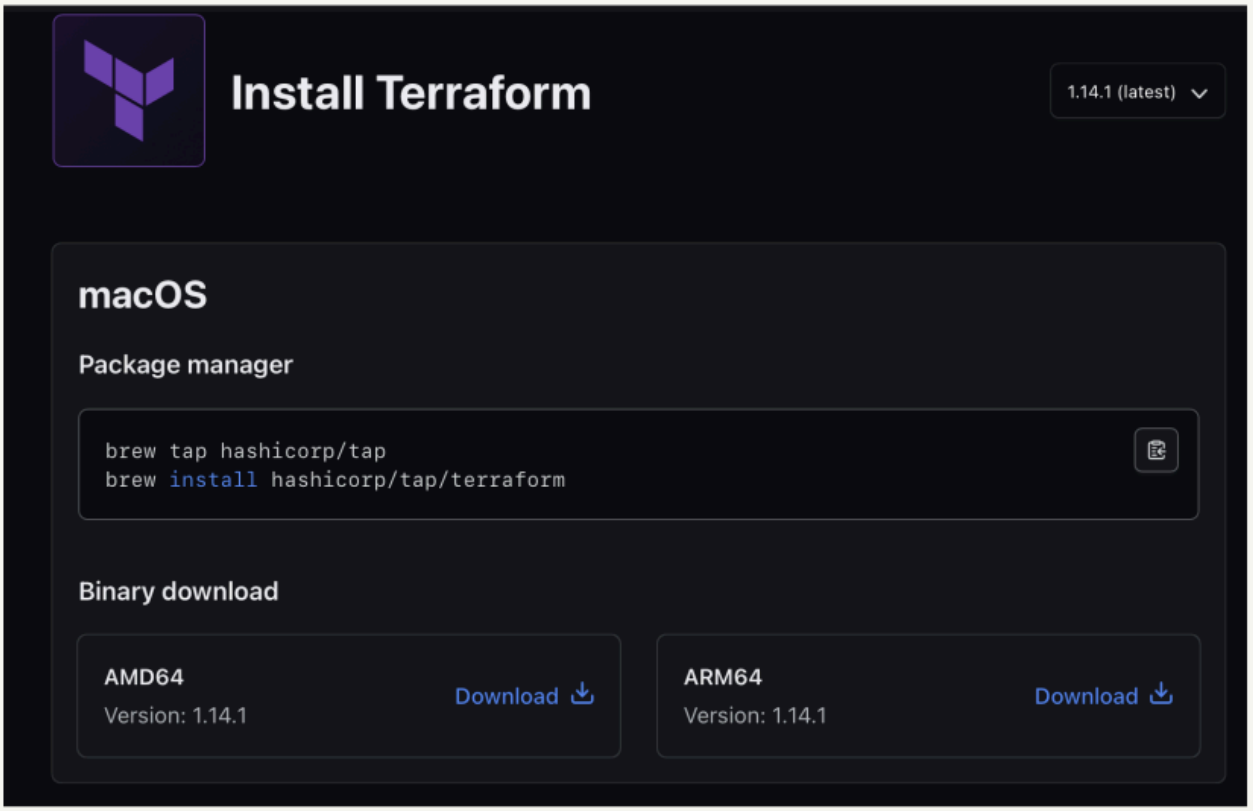
I chose to do this project today because I wanted to learn how to use Terraform as it is a very vital skillset to have in Cloud. Something that would make learning with NextWork even better are more projects!

Introducing Terraform

Terraform is a tool for managing IT resources using code. You use Terraform to process a configuration file you've prepared that details the desired state of an infrastructure. Terraform then creates/updates/deletes to set up that desired state.

Terraform is one of the most popular tools used for infrastructure as code (IaC), which is a way to manage IT manually managing resources using the console, or text commands in CLI. IaC automates the process with code.

Terraform uses configuration files to understand the desired state of an infrastructure. Main.tf is the main configuration file that is used in Terraform to describe the desired state.


Install Terraform
1.14.1 (latest)
macOS
Package manager
brew tap hashicorp/tap
brew install hashicorp/tap/terraform
Binary download
AMD64
Version: 1.14.1
Download
ARM64
Version: 1.14.1
Download

Configuration files

The configuration is structured in blocks instead of a single block of code. The advantage of doing this is (i.e. modularity) is the ability to work on/update different parts of main.tf without affecting other parts. Great for team collaboration too.

My main.tf configuration has three blocks

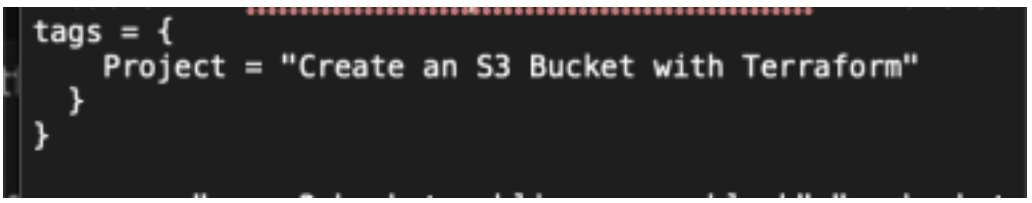
The first block indicates that I am using AWS as the provider for this infrastructure. The second block provides an Amazon S3 bucket and gives it a unique name. The third block manages the bucket's permission which means blocking all public access.

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

resource "aws_s3_bucket" "my_bucket" {
  bucket = "nextwork-unique-bucket-erik-1234" # Make sure this bucket name is globally unique by
typing a long random number
}

resource "aws_s3_bucket_public_access_block" "my_bucket_public_access_block" {
  bucket = aws_s3_bucket.my_bucket.id

  block_public_acls       = true
  ignore_public_acls      = true
  block_public_policy     = true
  restrict_public_buckets = true
}
```

Customizing my S3 Bucket

For my project extension, I visited the official Terraform documentation to look for ways I can customize my bucket's configuration. The documentation shows example configurations, all the available parameters for resource and usage rules.

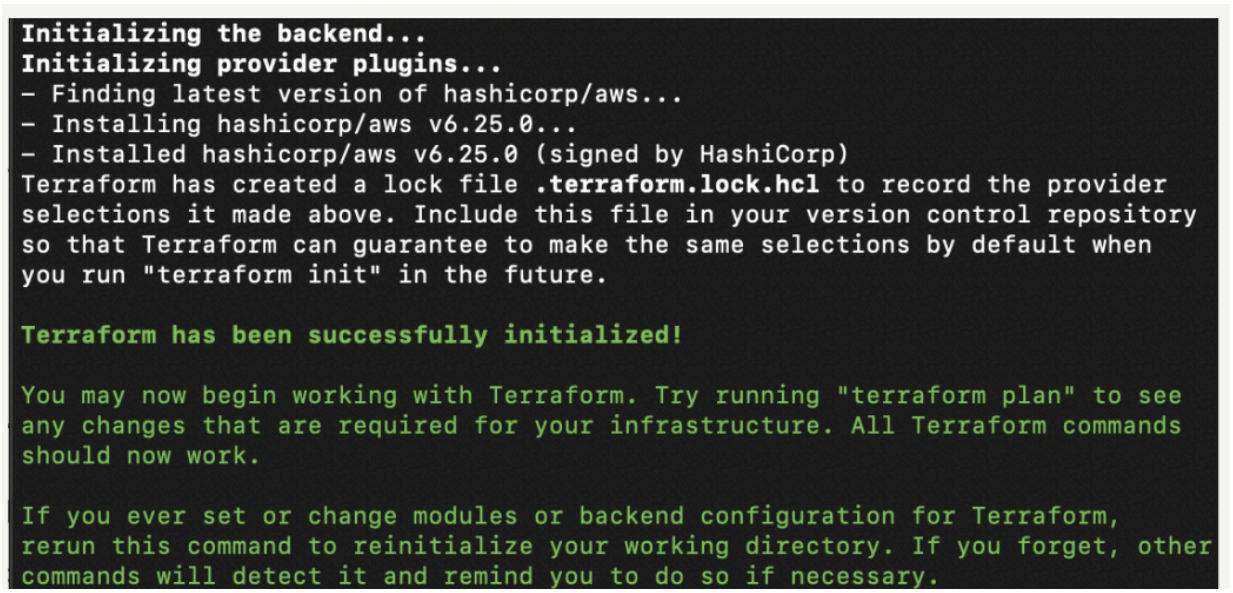
I chose to customize my bucket by adding tags because that allows me to identify the project thatI launch it for later on. When I launch my bucket, I can verify my customization by visiting the tag panel in the S3 console for this bucket.



Terraform commands

I ran 'terraform init' to initialize Terraform. This means getting it to set up the backend to store state files and install the necessary plugins i.e. AWS provider plugins.

Next, I ran 'terraform plan' to create an execution plan, showing you what changes Terraform will make to your infrastructure based on your configuration files (like main.tf). It shows what will be created, updated, or destroyed, so you can review and confirm the configuration before any real changes are made.

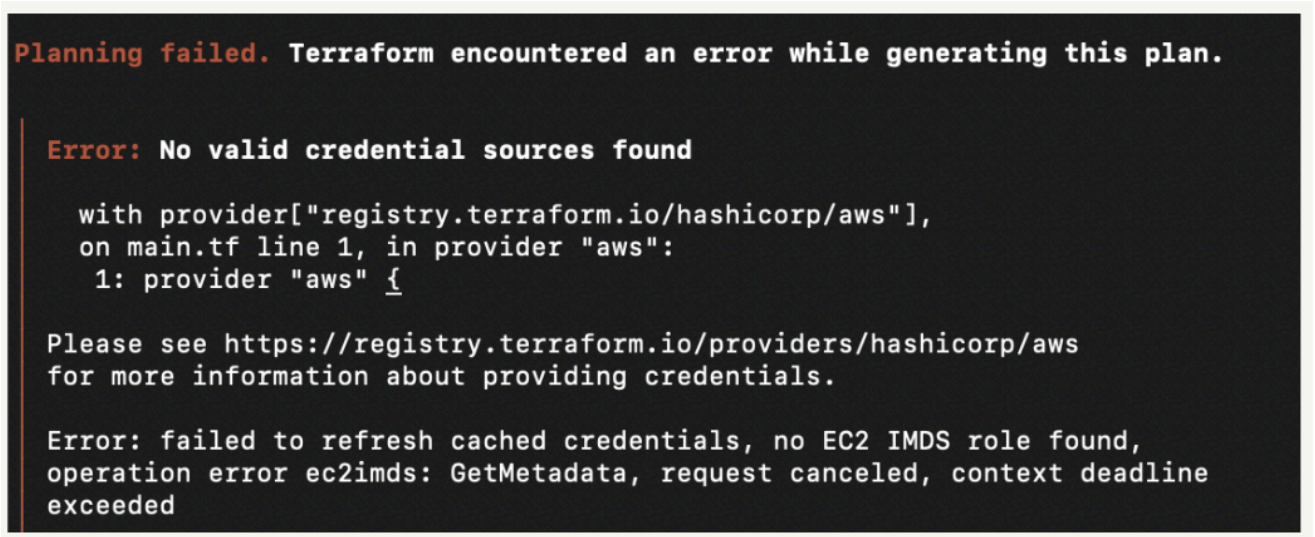


AWS CLI and Access Keys

When I tried to plan my Terraform configuration, I received an error message that says no valid credentials were found because my local terminal wasn't set up with AWS credentials yet. This needs to be set up to use the AWS provider plugin.

To resolve my error, first I installed AWS CLI, which is tool that lets you manage your AWS services from your terminal. Instead of having to use the AWS Management Console, you can now run text commands from your local machine.

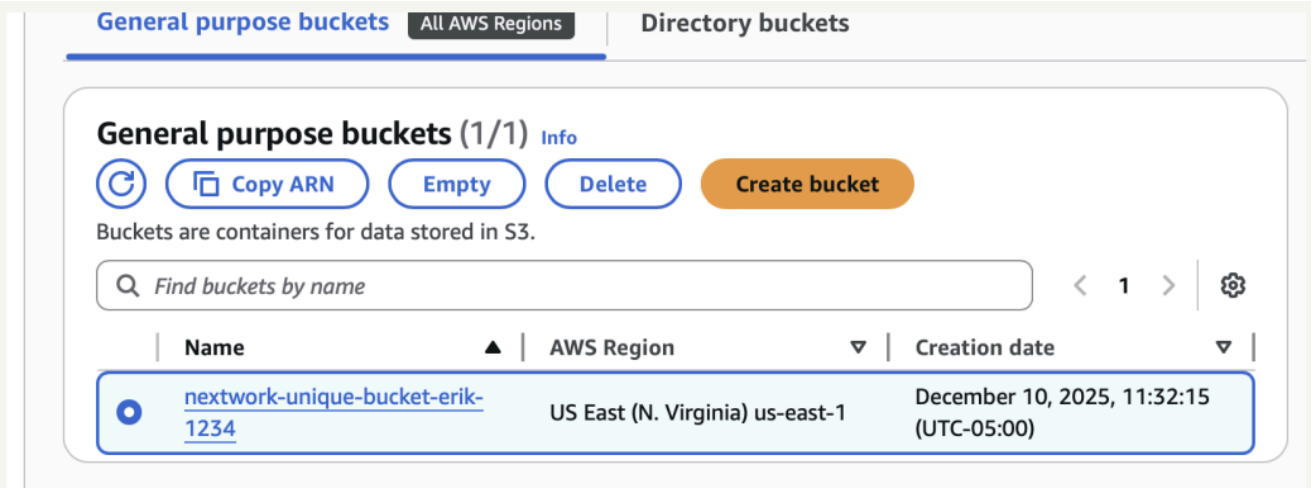
I set up AWS access keys so that I have a way to login into my AWS account over the CLI in my local computer. Once the access key and secret access key are passed through, my terminal now has the missing credentials.



Launching the S3 Bucket

I ran 'terraform apply' to apply the changes I've written in my Terraform configuration. Running 'terraform apply' will affect my AWS account by creating two resources: an S3 bucket, and a set of permission settings.

The sequence of running terraform init, plan, and apply is crucial because I need to initialize Terraform first before I get to make any comparison between main.tf and my current infrastructure, or to connect to AWS in the first place.



Uploading an S3 Object

I created a new resource block to upload an image called "image.png" from my local computer onto my S3 bucket.

We need to run the terraform application again because I modified my configuration file to accommodate for the image.

To validate that I've updated my configuration successfully, I checked the S3 bucket and confirmed the new image is inside, I also downloaded the image and confirmed it was the correct image that was initially uploaded.

```
resource "aws_s3_object" "image" {
  bucket = aws_s3_bucket.my_bucket.id # Reference the bucket ID
  key    = "image.png" # Path in the bucket
  source = "image.png" # Local file path
}
```