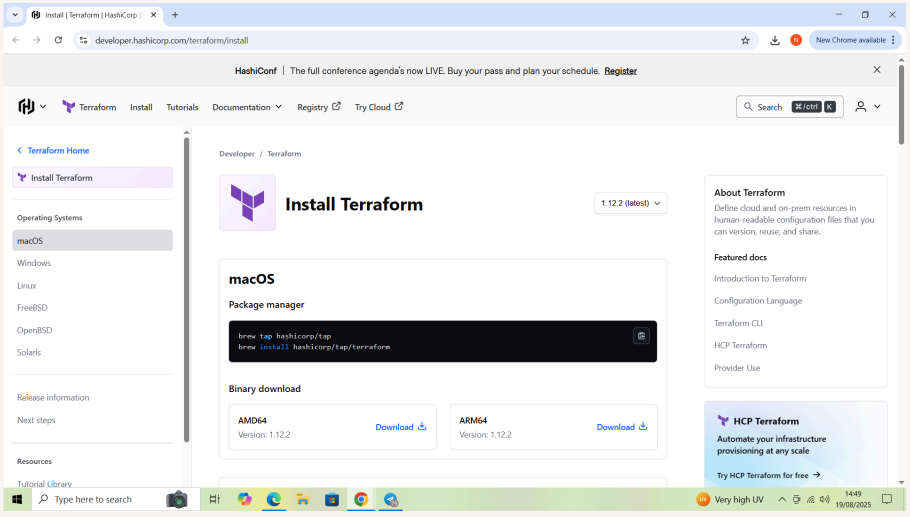
nextwork.org

Create S3 Buckets with Terraform

Nchindo Boris



Introducing Today's Project!

In this project, I will demonstrate how to use Terraform to automate the creation and management of an S3 bucket on AWS. The goal is to learn how Infrastructure as Code (IaC) can simplify cloud resource provisioning. By the end of this project, I will have a fully functioning Terraform setup that can create an S3 bucket, upload files to it, and manage it all through code.

Tools and concepts

Services I used were AWS S3, IAM, and Terraform. Key concepts I learnt include infrastructure as code (IaC), which involved tasks like; - Installing and configuring Terraform. - Configuring AWS credentials in the terminal. - Creating and managing S3 buckets with Terraform. - Uploading files to S3 bucket with Terraform.

Project reflection

This project took me approximately 45 minutes to complete. The most challenging part was setting up and troubleshooting AWS credentials to make sure Terraform could authenticate properly with my account. It was most rewarding to see the S3 bucket and the uploaded image in my AWS console

Nchindo Boris
NextWork Student

I did this project today to build experience with Terraform and AWS, and to better understand how infrastructure as code works in a real-world workflow. It was a practical and rewarding way to connect concepts to cloud infrastructure.

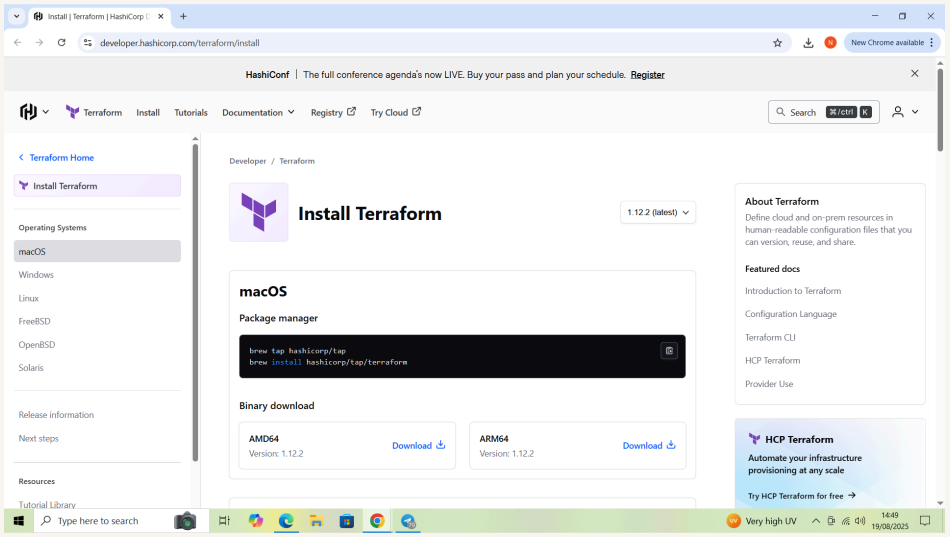
Nchindo Boris
NextWork Student

Introducing Terraform

Terraform is a tool that lets you build and manage infrastructure efficiently using code

Terraform is one of the most popular tools used for infrastructure as code (IaC), which is a practice that involves managing and provisioning cloud infrastructure using code instead of manually through a web console. Things built from code are built the same way every time. You can repair and rebuild things quickly, and other people can build identical instances of the same thing. This makes managing large-scale systems more efficient, less error-prone, and way faster than doing everything manually. Terraform is a top IaC tool because it supports multiple cloud providers, so you can set up multi-cloud infrastructures that use AWS, Azure, GCP and more. It also uses a simple configuration language.

Terraform uses configuration files to define and manage infrastructure. These files describe the desired state of your infrastructure, and Terraform figures out how to achieve that state. main.tf is a central file in a Terraform project.

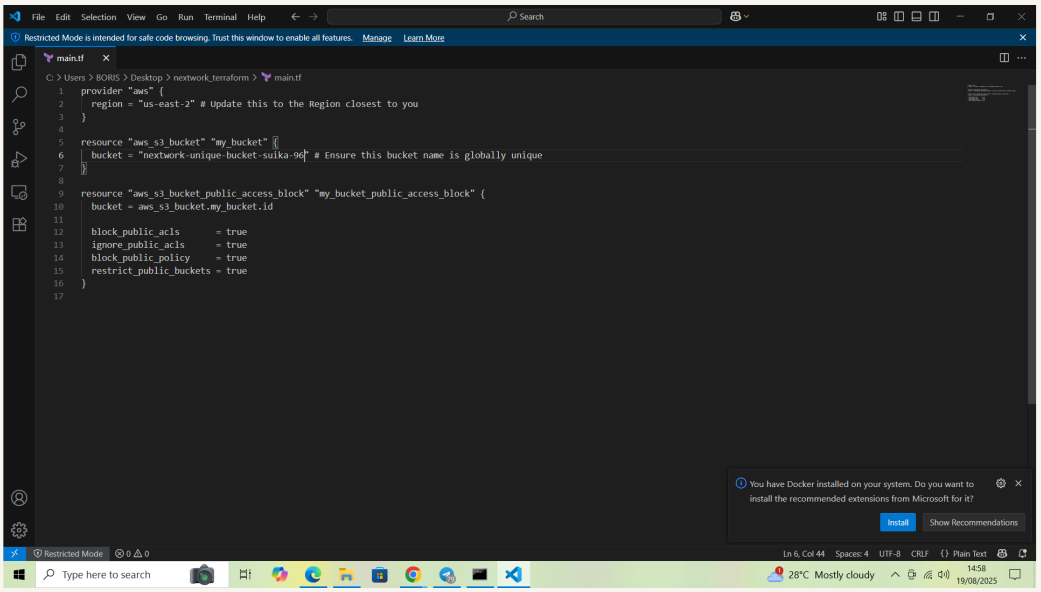

developer.hashicorp.com/terraform/install
HashiConf | The full conference agenda's now LIVE. Buy your pass and plan your schedule. Register
Terraform
Install
Tutorials
Documentation
Registry
Try Cloud
Search
Terraform Home
Install Terraform
Operating Systems
macOS
Windows
Linux
FreeBSD
OpenBSD
Solaris
Release information
Next steps
Resources
Tutorial Library
Developer / Terraform
Install Terraform
1.12.2 (latest)
macOS
Package manager
brew tap hashicorp/tap
brew install hashicorp/tap/terraform
Binary download
AMD64
Version: 1.12.2
Download
ARM64
Version: 1.12.2
Download
About Terraform
Define cloud and on-prem resources in human-readable configuration files that you can version, reuse, and share.
Featured docs
Introduction to Terraform
Configuration Language
Terraform CLI
HCP Terraform
Provider Use
HCP Terraform
Automate your infrastructure provisioning at any scale
Try HCP Terraform for free
Type here to search
Very high UV
14:49
19/08/2025

Configuration files

The configuration is structured in blocks, such as provider and resource blocks. Each block has a specific role, like telling Terraform which cloud to use, what resources to create, or how to output useful information. The advantage of doing this is that it keeps your infrastructure organised, modular, and easy to maintain. You can update, reuse, or share parts of your setup without affecting unrelated components, making your infrastructure as code clean, scalable, and efficient.

My main.tf configuration has three blocks

The first block indicates that Terraform should use AWS as the cloud provider, specifying the region and credentials needed to interact with AWS services. The second block provisions an S3 bucket, telling AWS to create a new storage bucket and giving it a unique name. The third block manages the public access settings for the S3 bucket, ensuring that public access is fully blocked by setting all the appropriate security flags to true.


File Edit Selection View Go Run Terminal Help
Search
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
main.tf
C: > Users > BORIS > Desktop > nextwork_terraform > main.tf
provider "aws" {
region = "us-east-2" # Update this to the Region closest to you
}
resource "aws_s3_bucket" "my_bucket" {
bucket = "nextwork-unique-bucket-suika-96" # Ensure this bucket name is globally unique
}
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
}
You have Docker installed on your system. Do you want to install the recommended extensions from Microsoft for it?
Install
Show Recommendations
Restricted Mode
Ln 6, Col 44 Spaces: 4 UTF-8 CRLF Plain Text
Type here to search
28°C Mostly cloudy
14:58
19/08/2025

Customizing my S3 Bucket

For my project extension, I visited the official Terraform documentation to explore ways to customise the aws_s3_bucket resource further. The documentation shows detailed information on all the configurable properties for S3 buckets, such as bucket versioning, lifecycle rules, encryption options, logging, tags, and more. It provides examples and best practices on how to set these properties using Terraform code, enabling me to tailor the S3 bucket to meet specific needs beyond just creating a basic bucket.

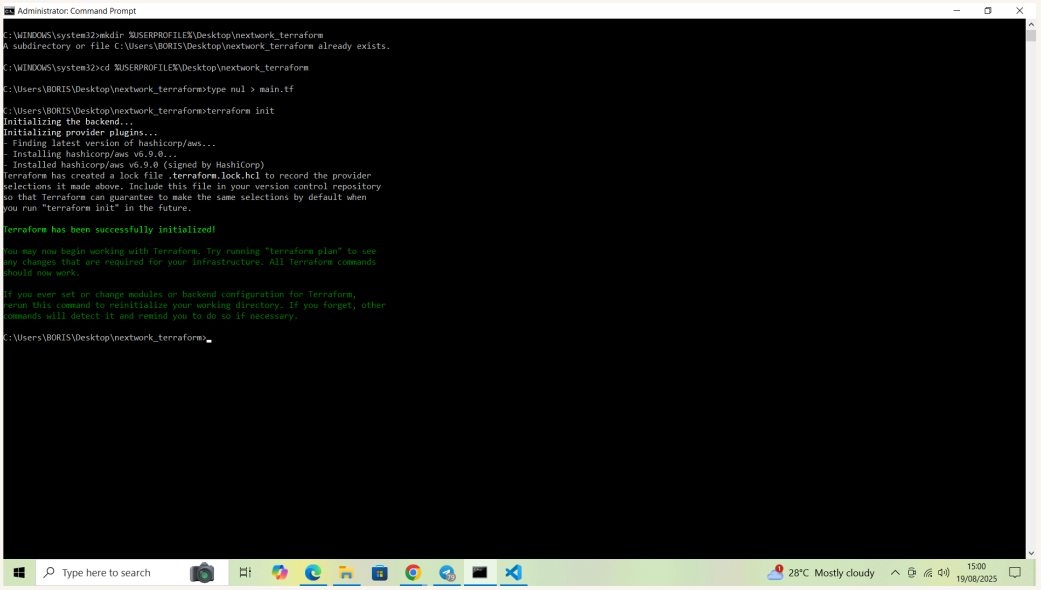
I chose to customise my bucket by enabling versioning because it allows me to keep multiple versions of the objects stored in the bucket, which is useful for recovering from accidental deletes or overwrites. When I launch my bucket, I can verify my customisation by checking the bucket properties in the AWS S3 console or by running AWS CLI commands to see that versioning is enabled on the bucket. Additionally, I can upload and modify files to confirm that previous versions are being retained.


main.tf
main.tf > resource "aws_s3_bucket_public_access_block" "my_bucket_public_access_block"
provider "aws" {
region = "your-region-code"
}
resource "aws_s3_bucket" "my_bucket" {
bucket = "nextwork-unique-bucket-johnnie-1699" # Make sure this bucket name is globally unique by typing a long random nu
tags = {
Name = "My bucket"
Environment = "Test"
}
}
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
}
resource "aws_s3_bucket_versioning" "version_1" {
bucket = aws_s3_bucket.my_bucket.id
versioning_configuration {
status = "Enabled"
}
}

Terraform commands

I ran terraform init to set up my Terraform project by downloading the necessary provider plugins, configuring the backend to keep track of my infrastructure state, preparing any modules used in the configuration, and creating a lock file to ensure consistent versions across environments.

Next, I ran terraform plan to generate an execution plan that shows exactly what changes Terraform will make to my AWS infrastructure based on my configuration. This lets me review what resources will be created, updated, or destroyed before actually applying any changes.

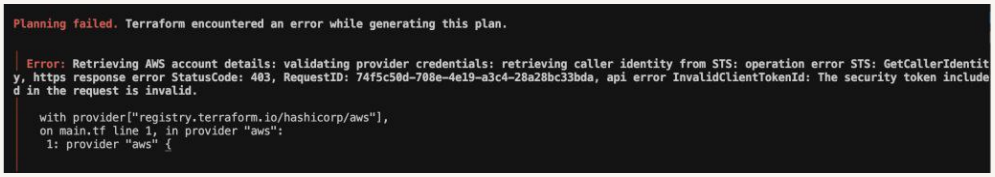


AWS CLI and Access Keys

When I tried to plan my Terraform configuration, I received an error message that says my AWS credentials were missing or not configured because Terraform couldn't authenticate with AWS without proper access keys set up through the AWS CLI

To resolve my error, first, I installed AWS CLI, which is a command-line tool that allows me to manage AWS services directly from my terminal. This means I can run commands to configure and control AWS resources without using the web console, making it easier for Terraform to authenticate and interact with my AWS account.

I set up AWS access keys to allow the AWS CLI to securely authenticate and communicate with my AWS account. This way, Terraform can use the CLI with the proper permissions to create and manage resources like the S3 bucket without needing me to manually log in through the web console each time.

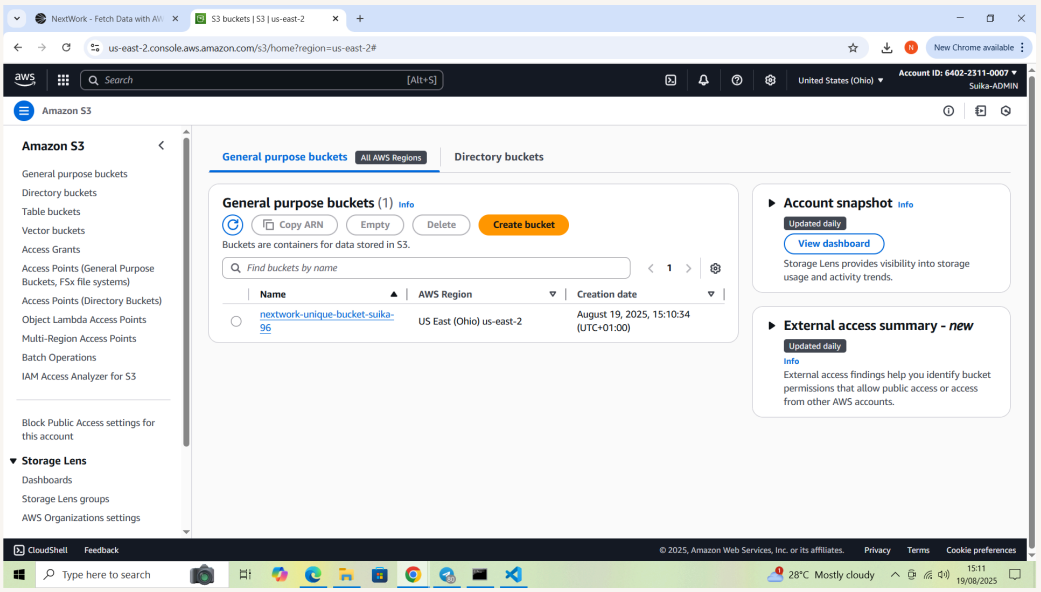

Planning failed. Terraform encountered an error while generating this plan.
Error: Retrieving AWS account details: validating provider credentials: retrieving caller identity from STS: operation error STS: GetCallerIdentity, https response error StatusCode: 403, RequestID: 74f5c50d-708e-4e19-a3c4-28a28bc33bda, api error InvalidClientTokenId: The security token included in the request is invalid.
with provider["registry.terraform.io/hashicorp/aws"],
on main.tf line 1, in provider "aws":
1: provider "aws" {

Lanching the S3 Bucket

I ran terraform apply to apply the changes written in the Terraform configuration. It creates, modifies, or deletes resources in the infrastructure based on code. Running terraform apply will affect my AWS account by creating the S3 bucket in my AWS account

The sequence of running terraform init, plan, and apply is crucial because each command prepares and validates the steps for deploying infrastructure safely and correctly: terraform init sets up the project by downloading necessary provider plugins (like AWS), initialising the backend, and preparing the environment to use Terraform. terraform plan then analyses the current configuration and compares it with the existing infrastructure (if any), showing you a preview of what changes Terraform will make. This helps avoid mistakes. terraform apply takes that plan and executes it, creating, updating, or deleting resources in your cloud environment. Running them in this order ensures that you initialise properly, preview your changes, and only then apply them, reducing the risk of accidental misconfigurations or resource loss.

Nchindo Boris
NextWork Student

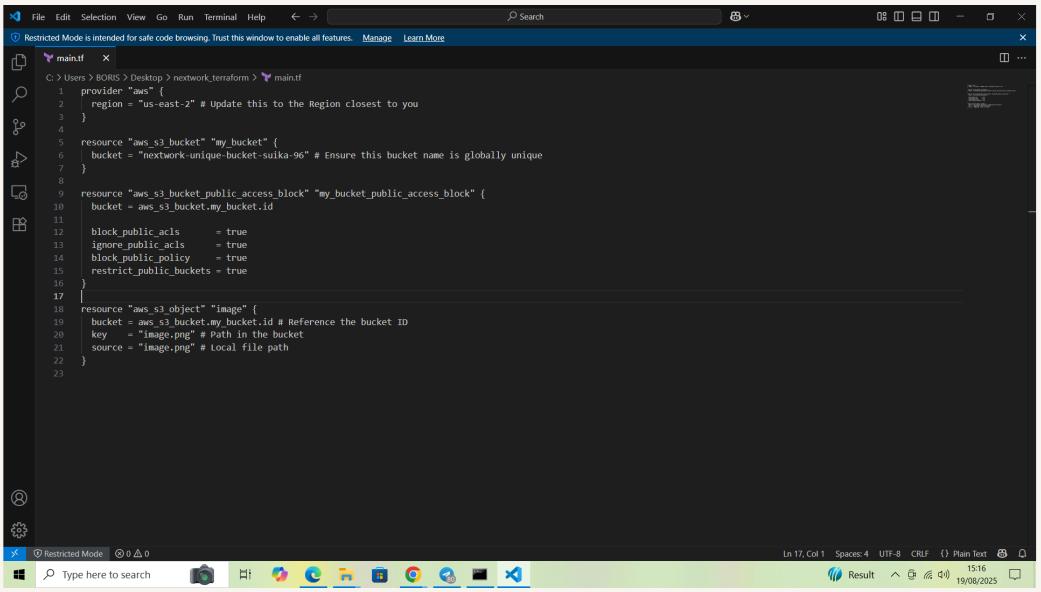


Uploading an S3 Object

I created a new resource block to upload an image file to my S3 bucket using Terraform. This block allows me to define the image as an object stored in the bucket, which means Terraform can manage not just the infrastructure but also the contents of that infrastructure. By including the file's name, location, and destination in the configuration, I ensure that every time I apply the Terraform plan, the object is reliably added or updated in the bucket.

We need to run terraform apply again because we made changes to our Terraform configuration; in this case, we added a new resource block to upload a file to our S3 bucket. Running terraform apply ensures those changes are applied to our AWS environment, keeping the actual infrastructure in sync with our updated configuration.

To validate that I've updated my configuration successfully, I checked my S3 bucket in the AWS Console and confirmed that the image file appeared there. Then, I downloaded the image from the bucket and verified that it matched the original file I uploaded, confirming that the upload worked.


File Edit Selection View Go Run Terminal Help
Search
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
main.tf
C: > Users > BORIS > Desktop > nextwork_terraform > main.tf
provider "aws" {
region = "us-east-2" # Update this to the Region closest to you
}
resource "aws_s3_bucket" "my_bucket" {
bucket = "nextwork-unique-bucket-suika-96" # Ensure this bucket name is globally unique
}
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
}
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
}
Restricted Mode
Ln 17, Col 1 Spaces: 4 UTF-8 CRLF Plain Text
Type here to search
Result
15:16
19/08/2025

The place to learn & showcase your skills

Check out nextwork.org for more projects