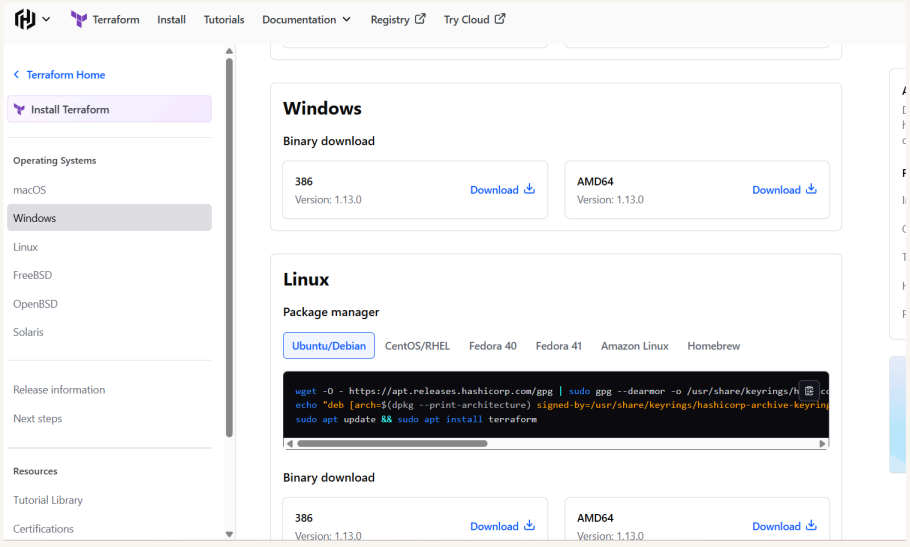
nextwork.org

Create S3 Buckets with Terraform

Louis Moyo



L
Louis Moyo
NextWork Student

Introducing Today's Project!

In this project, I will demonstrate how to use Terraform to automate the deployment of AWS resources. The goal is to install and configure Terraform, set up AWS credentials, and use Terraform scripts to create and manage S3 buckets. This helps eliminate manual setup, reduce errors, and show how infrastructure as code can streamline cloud operations.

Tools and concepts

Services I used were Amazon S3 and Terraform. Key concepts I learnt include how Infrastructure as Code (IaC) allows us to manage cloud resources declaratively, how to configure an S3 bucket with versioning and encryption, how to block public access for security, and how to upload and manage objects programmatically. I also learnt the importance of matching provider regions with bucket regions, and how Terraform ensures infrastructure state matches the configuration through plan and apply.

Project reflection

This project took me approximately a couple of hours. The most challenging part was troubleshooting errors such as file naming issues, hidden file extensions in Windows, and mismatched S3 regions which caused PermanentRedirect errors. It was most rewarding to see the object successfully uploaded and to gain confidence that Terraform can reliably automate these steps once the configuration is correct.

L

Louis Moyo
NextWork Student

I chose to do this project today because I wanted to strengthen my understanding of Infrastructure as Code and practice using Terraform with AWS. Something that would make learning with NextWork even better is adding more real-world troubleshooting examples and tips (like dealing with hidden file extensions or common AWS error codes), since those situations gave me the most hands-on learning.

L
Louis Moyo
NextWork Student

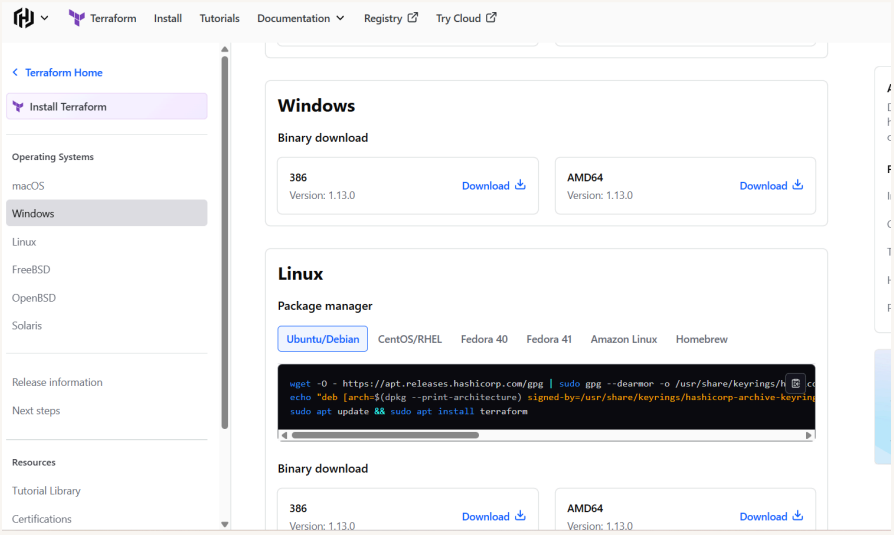
Introducing Terraform

Terraform is an open-source tool created by HashiCorp that lets you build and manage cloud infrastructure using code. Instead of manually clicking around in the AWS (or Azure, GCP, etc.) console, you write configuration files that describe exactly what resources you want-such as servers, databases, or networks. Terraform then automatically provisions those resources for you, making infrastructure setup consistent and repeatable.

Infrastructure as Code (IaC) is the practice of managing infrastructure (servers, storage, networking, etc.) using machine-readable configuration files instead of manual processes. With IaC, your infrastructure setup is version-controlled like software, easily shared across teams, and can be recreated anytime. Terraform is one of the most popular IaC tools because it works across many cloud providers (AWS, Azure, Google Cloud) and allows you to manage them all with a single workflow. Its configuration syntax is simple, human-readable, and powerful enough to handle complex environments.

Terraform uses configuration files to describe the infrastructure you want. main.tf is the primary configuration file where you define the resources Terraform should create and manage, like servers, storage, or networks.

Louis Moyo
NextWork Student



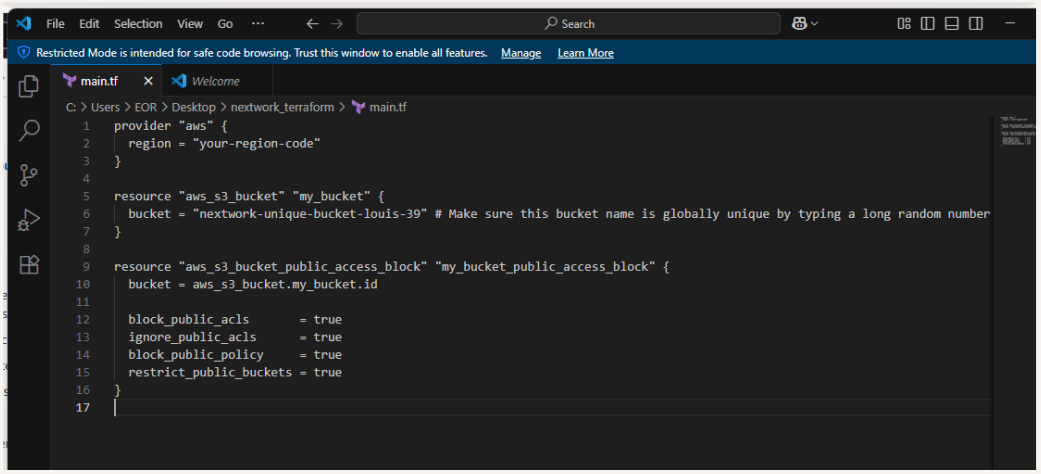
L
Louis Moyo
NextWork Student

Configuration files

The configuration is structured in blocks written in HashiCorp Configuration Language (HCL). Each block defines what resources or settings should exist. The advantage is consistency and automation - the same code can be reused, versioned, and deployed reliably across environments.

My main.tf configuration has three blocks

The first block indicates the provider (AWS) and region. The second block provisions the S3 bucket resource. The third block manages bucket settings like name, versioning, or tags. Together they fully describe the infrastructure Terraform should build.

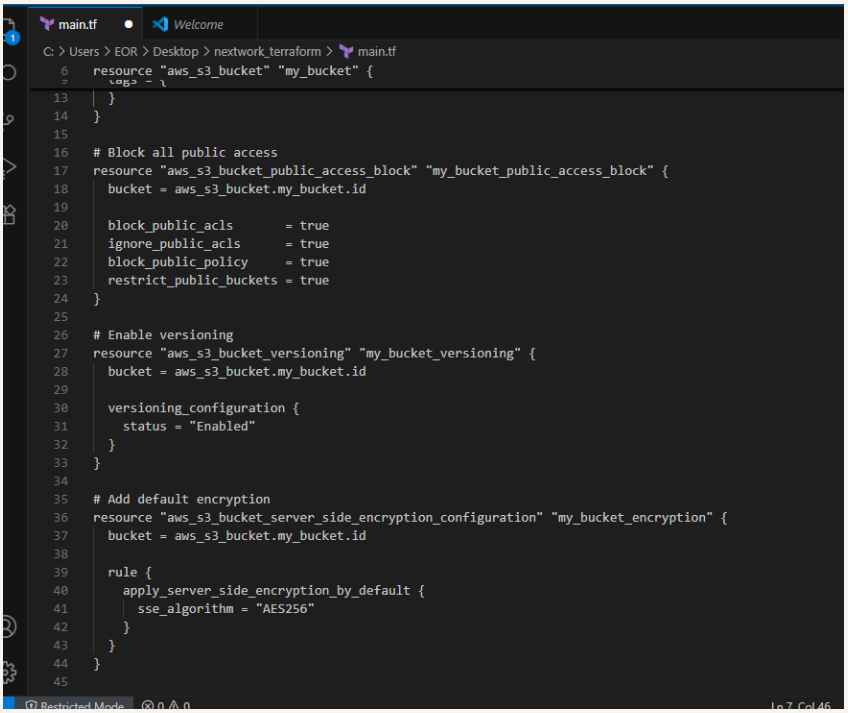

File Edit Selection View Go
Search
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
main.tf
Welcome
C: > Users > EOR > Desktop > nextwork_terraform > main.tf
provider "aws" {
region = "your-region-code"
}
resource "aws_s3_bucket" "my_bucket" {
bucket = "nextwork-unique-bucket-louis-39" # 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
}

L
Louis Moyo
NextWork Student

Customizing my S3 Bucket

For my project extension, I visited the official Terraform documentation to explore the aws_s3_bucket resource. The documentation shows examples of how to create an S3 bucket, configure options like versioning, encryption, and tags, and connect the bucket with other AWS services. It also explains parameters and best practices, helping me customise my bucket setup beyond just creating it.

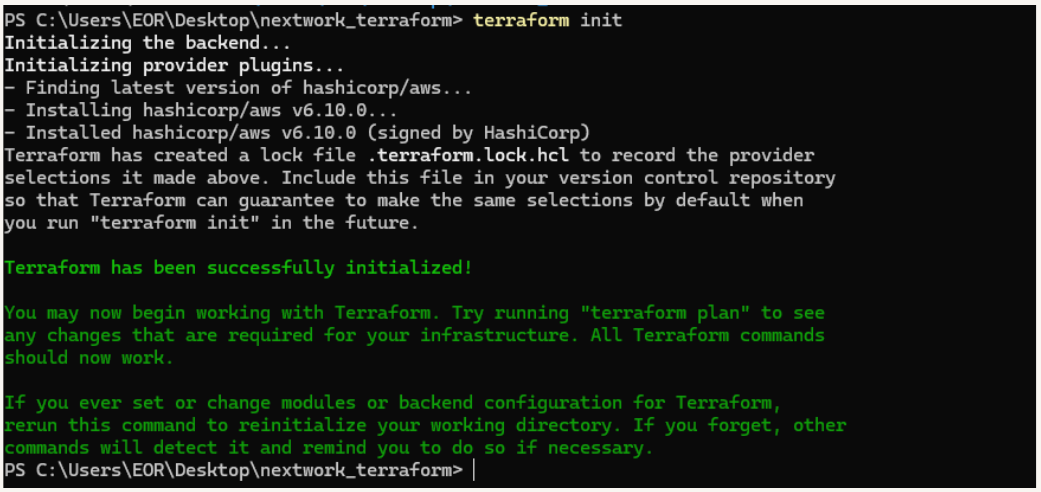
I chose to customise my bucket by enabling server-side encryption and adding tags. This improves security by ensuring all objects are encrypted at rest, and the tags help with organisation and cost tracking. When I launch my bucket, I can verify my customisation in the AWS S3 console by checking the bucket properties to confirm encryption is active and the tags are applied.


main.tf
Welcome
C: > Users > EOR > Desktop > nextwork_terraform > main.tf
resource "aws_s3_bucket" "my_bucket" {
}
}
# Block all public access
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
}
# Enable versioning
resource "aws_s3_bucket_versioning" "my_bucket_versioning" {
bucket = aws_s3_bucket.my_bucket.id
versioning_configuration {
status = "Enabled"
}
}
# Add default encryption
resource "aws_s3_bucket_server_side_encryption_configuration" "my_bucket_encryption" {
bucket = aws_s3_bucket.my_bucket.id
rule {
apply_server_side_encryption_by_default {
sse_algorithm = "AES256"
}
}
}

Terraform commands

I ran terraform init to set up my working directory for Terraform. This command downloaded the required provider plugins (like AWS), created a lock file to track dependencies, and prepared the environment so my configuration files could run. It's the first step that ensures Terraform knows how to talk to AWS and manage resources correctly.

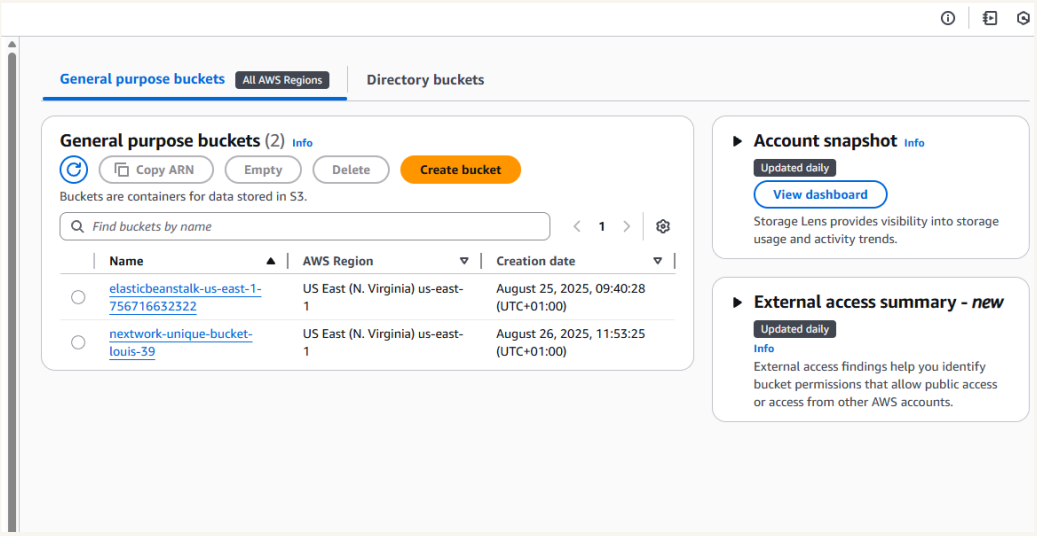
Next, I ran terraform plan to preview the changes Terraform would make in my AWS environment. This command compared my current infrastructure with the configuration defined in main.tf and showed me which resources would be created, updated, or destroyed. It's like a dry run that validates my configuration and helps avoid unexpected changes before actually applying them.


PS C:\Users\EOR\Desktop\nextwork_terraform> terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v6.10.0...
- Installed hashicorp/aws v6.10.0 (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.
PS C:\Users\EOR\Desktop\nextwork_terraform>

Lanching the S3 Bucket

I ran terraform apply to create and configure the AWS resources defined in my Terraform file. Running terraform apply affects my AWS account by provisioning real infrastructure (in this case, an S3 bucket) exactly as described in my configuration. It turns my planned configuration into actual resources, so I can see my bucket in AWS after the command finishes.

The sequence of running terraform init, terraform plan, and terraform apply is crucial because each step builds on the previous one. Terraform init sets up the working directory and downloads the required provider plugins. terraform plan previews what changes will be made to my AWS account, so I can confirm everything looks correct. Finally, terraform apply executes those changes and deploys the resources. This order ensures I don’t accidentally create or destroy resources without understanding the impact first.


General purpose buckets
All AWS Regions
Directory buckets
General purpose buckets (2) Info
Copy ARN
Empty
Delete
Create bucket
Buckets are containers for data stored in S3.
Find buckets by name
1
Name
AWS Region
Creation date
elasticbeanstalk-us-east-1-756716632322
US East (N. Virginia) us-east-1
August 25, 2025, 09:40:28 (UTC+01:00)
nextwork-unique-bucket-louis-39
US East (N. Virginia) us-east-1
August 26, 2025, 11:53:25 (UTC+01:00)
Account snapshot Info
Updated daily
View dashboard
Storage Lens provides visibility into storage usage and activity trends.
External access summary - new
Updated daily
Info
External access findings help you identify bucket permissions that allow public access or access from other AWS accounts.

Uploading an S3 Object

I created a new resource block to define an additional piece of infrastructure (an S3 object) separately from the bucket. Each resource block in Terraform manages one type of resource, so I added a new block to upload my image file into the existing S3 bucket.

We need to run terraform apply again because whenever you change your Terraform configuration (for example, updating the region, adding a new resource, or modifying a property), Terraform has to reconcile the desired state (your .tf file) with the actual state (your AWS resources). Running apply ensures that the new changes are applied and your infrastructure matches what's defined in your code.

To validate that I updated my configuration successfully, I first ran terraform apply and confirmed that it completed without errors. Then I logged into the AWS Management Console, opened the S3 service, navigated to my bucket, and verified that the image.png file was present. Finally, I downloaded the file from the bucket and confirmed it matched the original file I uploaded from my local machine.

```
46  resource "aws_s3_object" "image" {
47    bucket       = aws_s3_bucket.my_bucket.id  # Reference the bucket ID
48    key          = "image.png"                 # Path in the bucket
49    source       = "image.png"                 # Local file path
50    content_type = "image/png"
```

The place to learn & showcase your skills

Check out nextwork.org for more projects