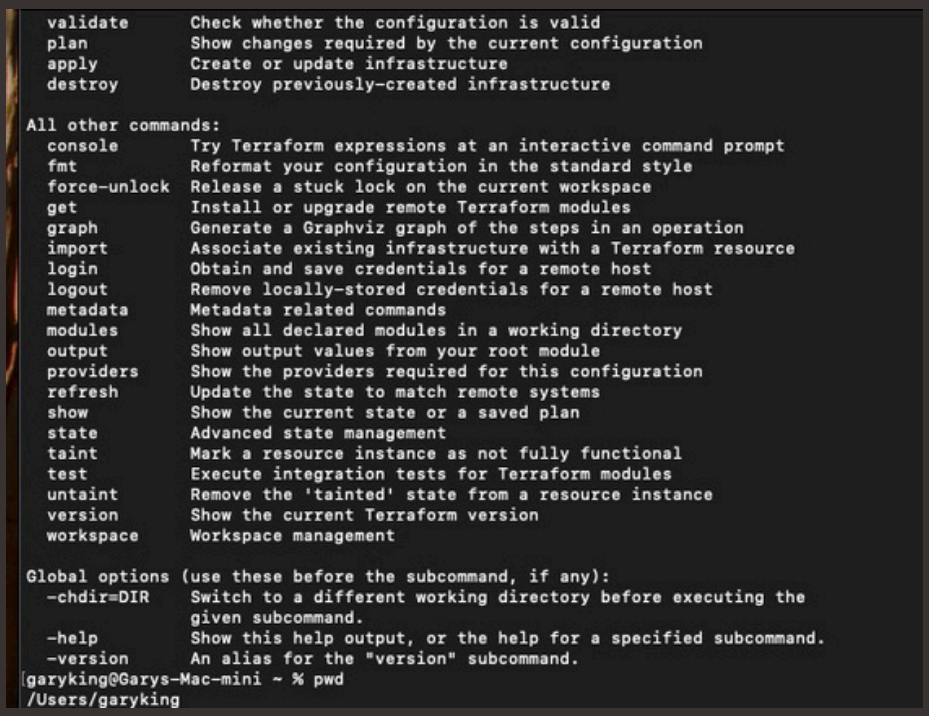
nextwork.org

Create S3 Buckets with Terraform

GA gazzok@gmail.com



GA
gazzok@gmail.com
NextWork Student

Introducing Today's Project!

In this project, I will demomstrate using Terraform to deploy infrastructure in AWS. The goal is to configure my AWS credentials in the terminal and create and manage S3 buckets with Terraform. Finally, I will upload files to S3 using Terraform.

Tools and concepts

Services I used were: Terraform Amazon S3 AWS CLI AWS IAM

Project reflection

The project took me approximately 1.5 hours. The most challenging part was setting up Terraform for the first time. It was most rewarding to use Terraform to launch resources in AWS using the AWS CLI. It truly showcased the power of Terraform and illustrated its usecases.

I completed the project today to improve my Terrafrom knowledge and learn more about Infrastructure as code (IaC). This project definitely met my goals adn has given me a solid base to continue my exploring and learning of Terraform.

GA
gazzok@gmail.com
NextWork Student

Introducing Terraform

Terraform is a tool that helps you build and manage your cloud infrastructure using code. Instead of setting up resources manually in the AWS console or CLI, you write a script that tells Terraform exactly what you want in your cloud infrastructure, like servers, databases, and networks. Terraform then automatically builds all of this for you, using your script as a blueprint.

Infrastructure as Code (IaC) is a way to manage your IT infrastructure. Instead of manually setting up your resources e.g. creating resources one by one in the Console, you're writing configuration files in code. 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.


validate Check whether the configuration is valid
plan Show changes required by the current configuration
apply Create or update infrastructure
destroy Destroy previously-created infrastructure
All other commands:
console Try Terraform expressions at an interactive command prompt
fmt Reformat your configuration in the standard style
force-unlock Release a stuck lock on the current workspace
get Install or upgrade remote Terraform modules
graph Generate a Graphviz graph of the steps in an operation
import Associate existing infrastructure with a Terraform resource
login Obtain and save credentials for a remote host
logout Remove locally-stored credentials for a remote host
metadata Metadata related commands
modules Show all declared modules in a working directory
output Show output values from your root module
providers Show the providers required for this configuration
refresh Update the state to match remote systems
show Show the current state or a saved plan
state Advanced state management
taint Mark a resource instance as not fully functional
test Execute integration tests for Terraform modules
untaint Remove the 'tainted' state from a resource instance
version Show the current Terraform version
workspace Workspace management
Global options (use these before the subcommand, if any):
-chdir=DIR Switch to a different working directory before executing the given subcommand.
-help Show this help output, or the help for a specified subcommand.
-version An alias for the "version" subcommand.
garyking@Garys-Mac-mini ~ % pwd
/Users/garyking

GA
gazzok@gmail.com
NextWork Student

Configuration files

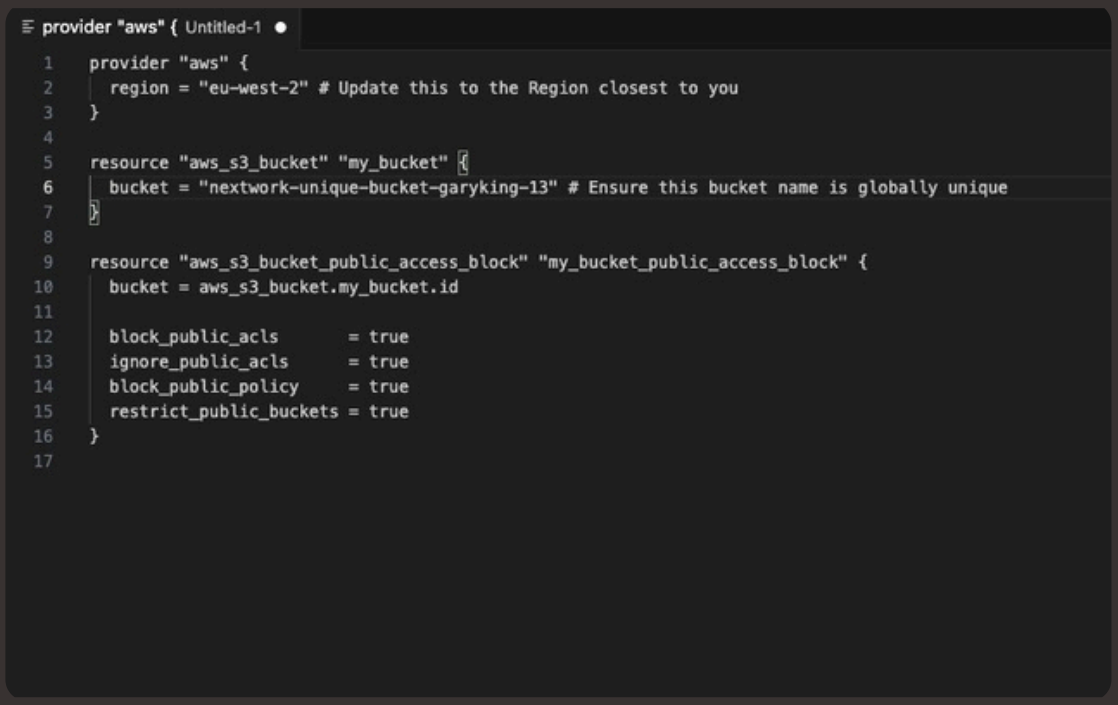
The configuration is structured in blocks to help organize the code better. The style of organizing code into individual blocks is called modularity. Each block handles a specific piece of your setup, like a resource or a configuration, which makes it easier to read, manage, and adjust things separately without affecting the rest of your infrastructure. These are the basic building blocks of any Terraform configuration: Provider blocks specify which plugins (like AWS, Google Cloud, or Azure) Terraform uses to deploy and manage resources in those services. Resource blocks define what resources to manage, such as networks, virtual machines, or storage buckets. Each resource is individually managed and can be customized with parameters defined by the provider.

GA
gazzok@gmail.com
NextWork Student

My main.tf configuration has three blocks

Thefirst block indicates the provider"aws": Indicates that AWS is theprovider. A provider is a plugin (a software add-on) that allows Terraform to interact with a specific cloud service, API, or on-premises resource. The provider translates your configurations into API calls that manage your infrastructure. The second block provisions the resource "aws_s3_bucket" "my_bucket": This block provisions an S3 bucket. "my_bucket" is an internal alias used in your code to reference this resource. The third block manages the resource "aws_s3_bucket_public_access_block" "my_bucket_public_access_block": Manages public access policies for the S3 bucket. In our case, options like block_public_acls, ignore_public_acls, block_public_policy, and restrict_public_buckets, are all set to true to stop public access to your bucket and its contents.

GA
gazzok@gmail.com
NextWork Student



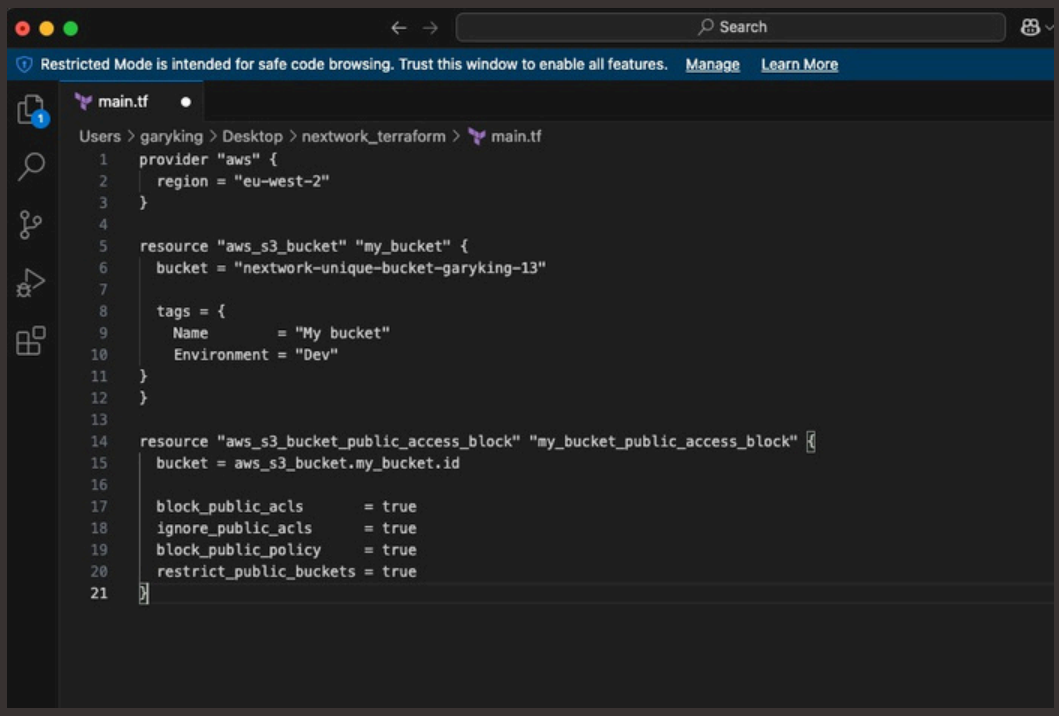
GA
gazzok@gmail.com
NextWork Student

Customizing my S3 Bucket

For my project extension, I visited the official Terraform documentation to read documentation on the aws provider, where we can find setup instructions, descriptions of each available resource, best practice tips and examples, and parameters for customization. The aws_s3_bucket in the Terraform documentation is all about how to use Terraform to create and manage AWS S3 buckets. It covers how to configure these buckets, set their properties, and manage interactions with other AWS services.

I chose to customise my bucket by addig tags because I wanted to be able to differentiate if this file was for a development environment or production environment.

GA
gazzok@gmail.com
NextWork Student

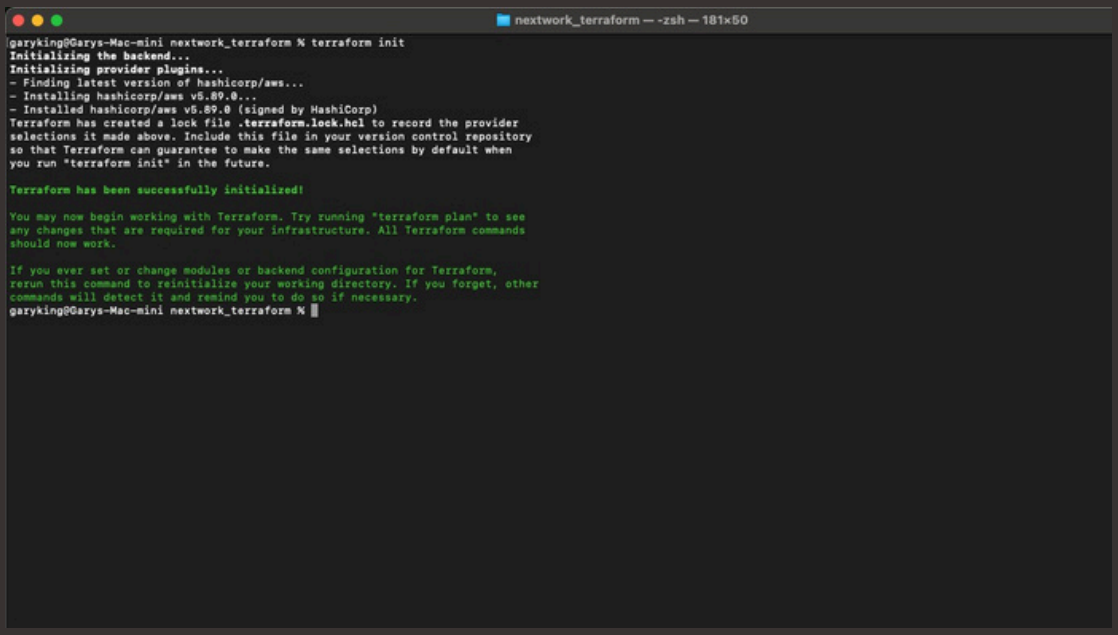


GA
gazzok@gmail.com
NextWork Student

Terraform commands

I ran 'terraform init' to Terraform init is the first command you run for a new Terraform project. It sets up your project by: Downloading necessary plugins: It grabs the specific providers Terraform needs to manage your cloud services, like AWS. Setting up the backend: Prepares where Terraform will store its record of your setup, so it knows what your cloud setup looks like and can manage it efficiently. Preparing modules: If your setup uses reusable code components, called modules, this is where the code is downloaded from their source to replace placeholders in your code. Creating a lock file: This file keeps track of the versions of everything Terraform is using, making sure that everyone on your team is using the same setup to avoid inconsistencies.

Next, I ran 'terraform plan' to create an execution plan, showing you what changes Terraform will mkae to my infrastructure based on my configuration file. It shows what will be created, updated, or destroyed so we can review and confirm the configuration before any real changes are made.


nextwork_terraform — -zsh — 181×50
garyking@Garys-Mac-mini nextwork_terraform % terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.89.0...
- Installed hashicorp/aws v5.89.0 (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.
garyking@Garys-Mac-mini nextwork_terraform %

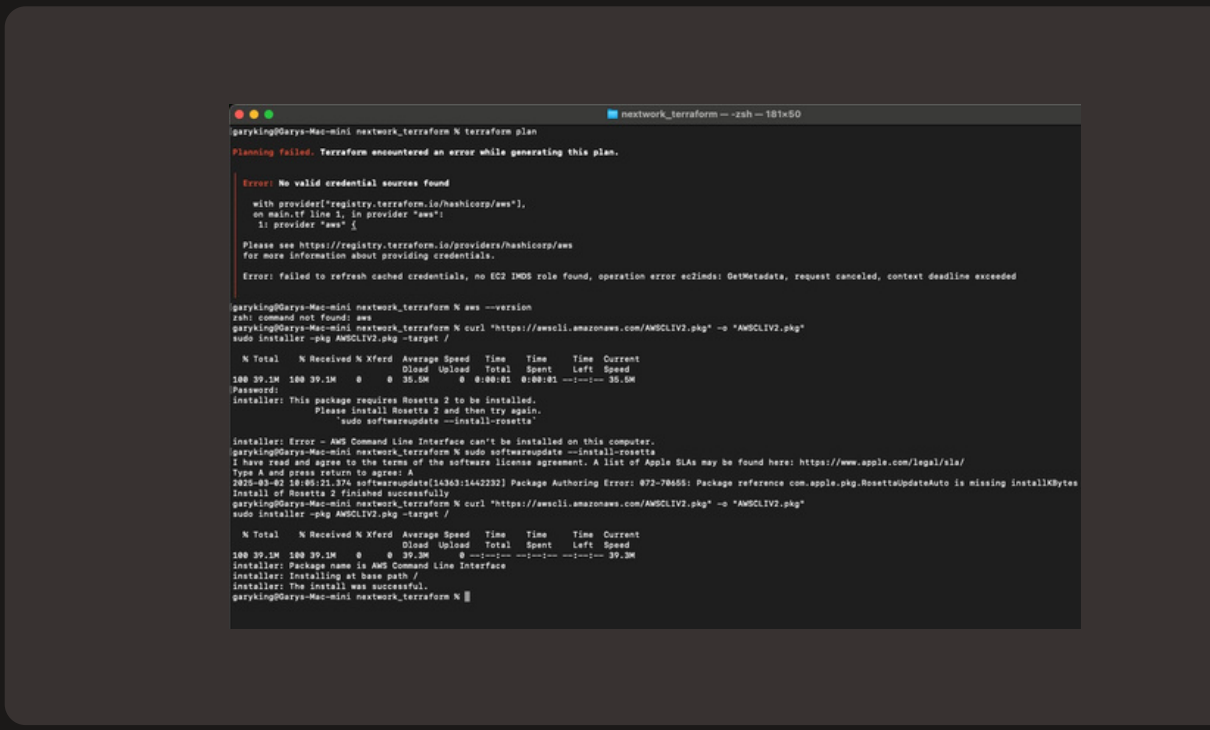
GA
gazzok@gmail.com
NextWork Student

AWS CLI and Access Keys

When I tried to plan my Terraform configuration, I received an error message that says 'Planning Failed' because I did not have AWS CLI installed.

To resolve my error, first I installed AWS CLI, which is The AWS CLI (Command Line Interface). This is a powerful 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 to connect the CLI to the management console as the CLI needs another way of authenticating you.

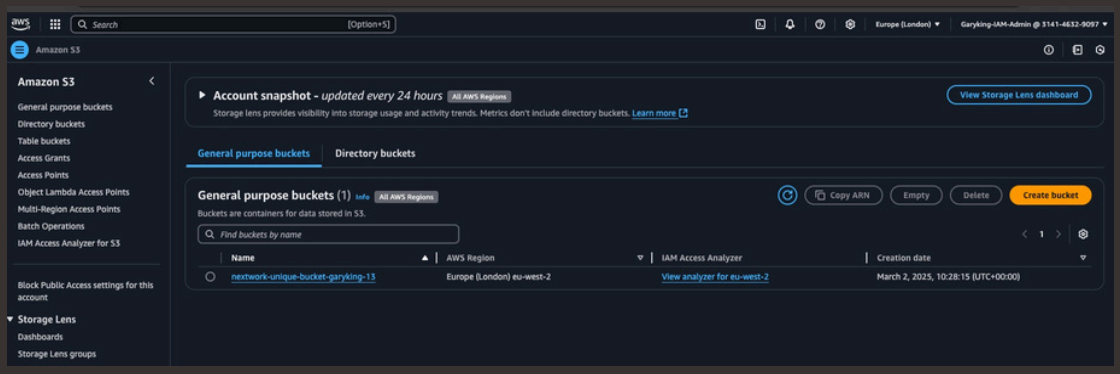


GA
gazzok@gmail.com
NextWork Student

Lanching the S3 Bucket

I ran 'terraform apply' to apply the changes I'd written in my Terraform configuration. It creates, modifies, or deletes resources in your infrastructure based on your code. In my case, this command creates the S3 bucket in my AWS account.

If you run terraform apply before terraform init, you would run into an error because terraform init needs to set up your project first by downloading necessary plugins and setting up the state file, which is a file Terraform uses to track the current state of your infrastructure. There actually aren't any errors if you skip terraform plan. This is because terraform apply will automatically run a plan and ask for confirmation before making any changes. But, skipping the explicit terraform plan means you might miss reviewing these changes in detail. It's best practice to run a plan to catch any potential errors or unexpected results before they affect your live infrastructure.


Search
[Option+S]
Europe (London)
Amazon S3
Amazon S3
General purpose buckets
Directory buckets
Table buckets
Access Grants
Access Points
Object Lambda Access Points
Multi-Region Access Points
Batch Operations
IAM Access Analyzer for S3
Block Public Access settings for this account
Storage Lens
Dashboards
Storage Lens groups
Account snapshot - updated every 24 hours
All AWS Regions
Storage lens provides visibility into storage usage and activity trends. Metrics don't include directory buckets. Learn more
View Storage Lens dashboard
General purpose buckets
Directory buckets
General purpose buckets (1) Info
All AWS Regions
Buckets are containers for data stored in S3.
Copy ARN
Empty
Delete
Create bucket
Find buckets by name
Name
AWS Region
IAM Access Analyzer
Creation date
nextwork-unique-bucket-garyking-13
Europe (London) eu-west-2
View analyzer for eu-west-2
March 2, 2025, 10:28:15 (UTC+00:00)

GA
gazzok@gmail.com
NextWork Student

Uploading an S3 Object

I created a new resource block to add an image to my S3 bucket using Terraform.

We need to run terraform apply again because you need to run terraform apply anytime you change your Terraform configuration. This will make sure the changes are correctly applied to your infrastructure.

To validate that I've updated my configuration successfully, I logged back into my management console and into my S3 bucket. I then was able to see my image.png which I downloaded and opened to verify it worked - which it did!