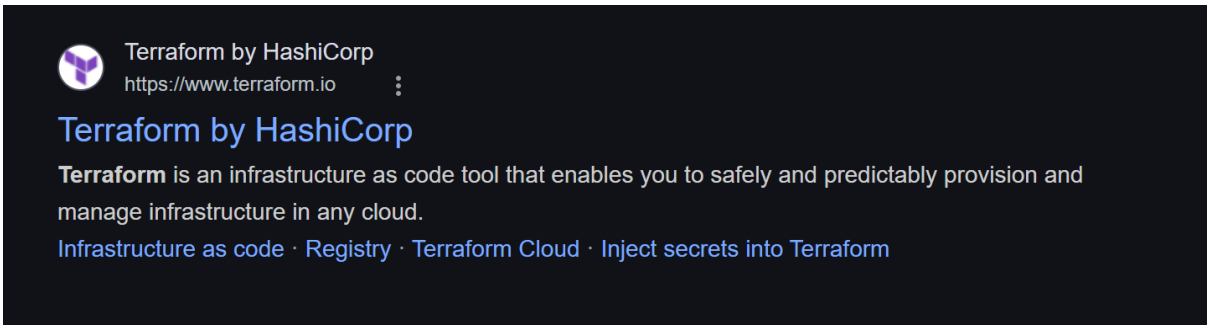
Host a Dynamic Ecommerce Website on AWS with Terraform

Course Overview: This project covers how to provision an AWS cloud infrastructure for a dynamic e-commerce web application using Terraform. We will use Terraform (Infrastructure as Code) to automate resource creation, integrate with GitHub for version control, and leverage AWS services like VPC, RDS, ALB, etc. Each step below corresponds to a phase in the project, complete with explanations and screenshots.

1. Course Introduction

Before diving in, ensure you understand the project goals and required tools. We will use **Terraform** by HashiCorp for infrastructure automation, AWS for cloud services, GitHub for code hosting, and supporting tools like Git and the AWS CLI.

- **Terraform:** HashiCorp Terraform is an infrastructure-as-code tool that lets you define and provision infrastructure in a predictable manner using configuration files. We will write Terraform configuration files (.tf files) to set up AWS resources for our e-commerce site.



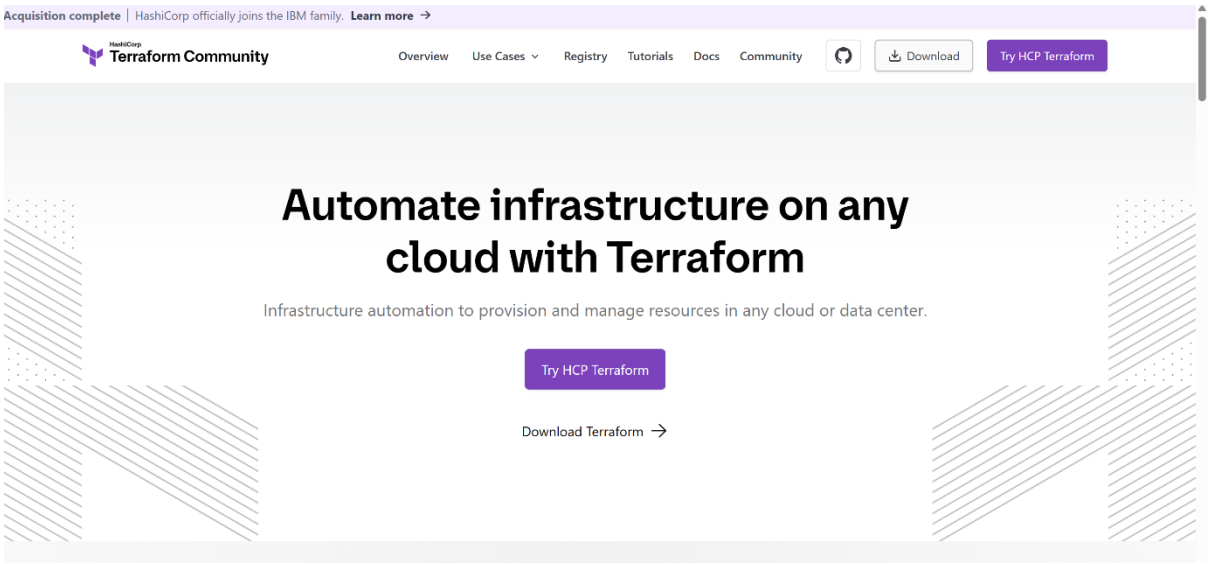
- **AWS Infrastructure:** Our site will run on AWS. Key resources include a Virtual Private Cloud (VPC), subnets, an internet gateway, NAT gateways, an RDS database, security groups, an Application Load Balancer (ALB), an Auto Scaling Group, and a Route 53 DNS record. Terraform will manage these.
- **GitHub & Git:** We'll keep our Terraform code in a GitHub repository (for version control and collaboration). Git is the version control software used to track code changes and push code to GitHub.

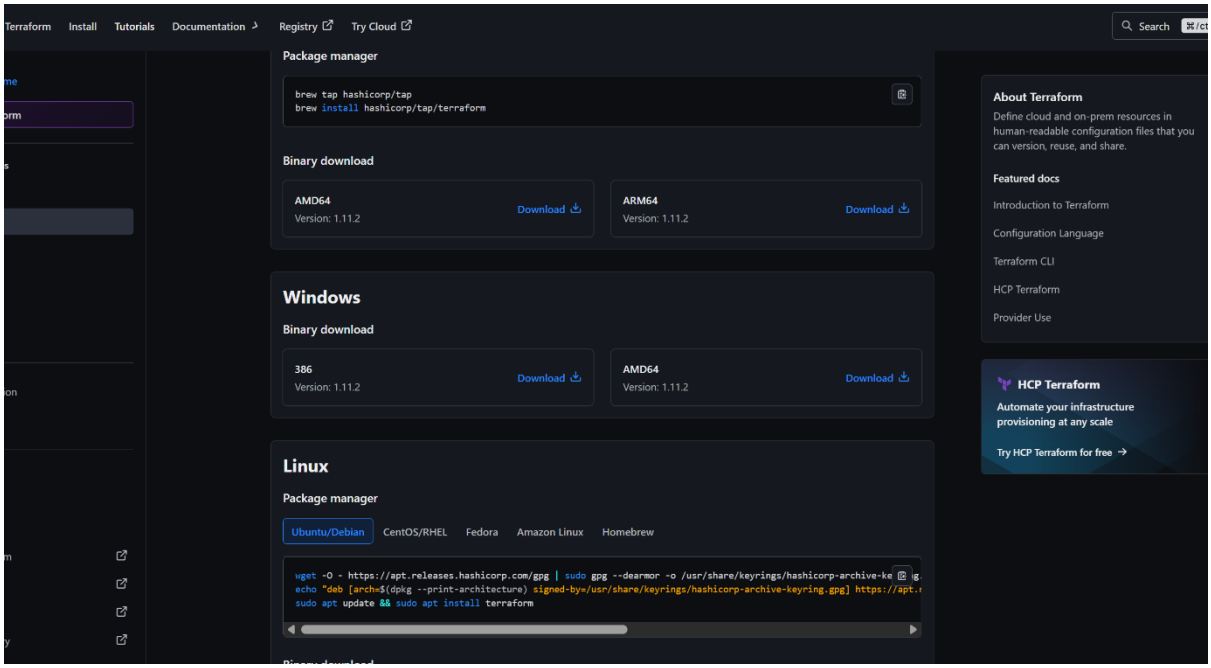
- **AWS CLI:** The AWS Command Line Interface will let us authenticate and interact with AWS from our local machine. We'll use it to configure credentials and verify connectivity.

2. Install Terraform (Windows)

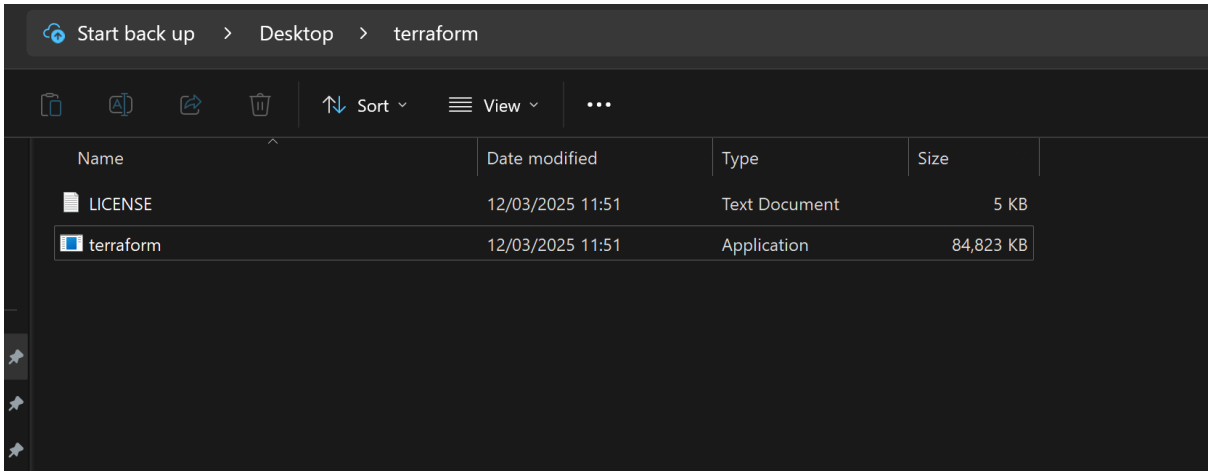
To use Terraform, first download and install the Terraform binary for Windows:

1. **Download Terraform:** Visit the official Terraform website and download the appropriate Windows ZIP archive (32-bit or 64-bit depending on your system). *Screenshot below shows Terraform's website with download options.*



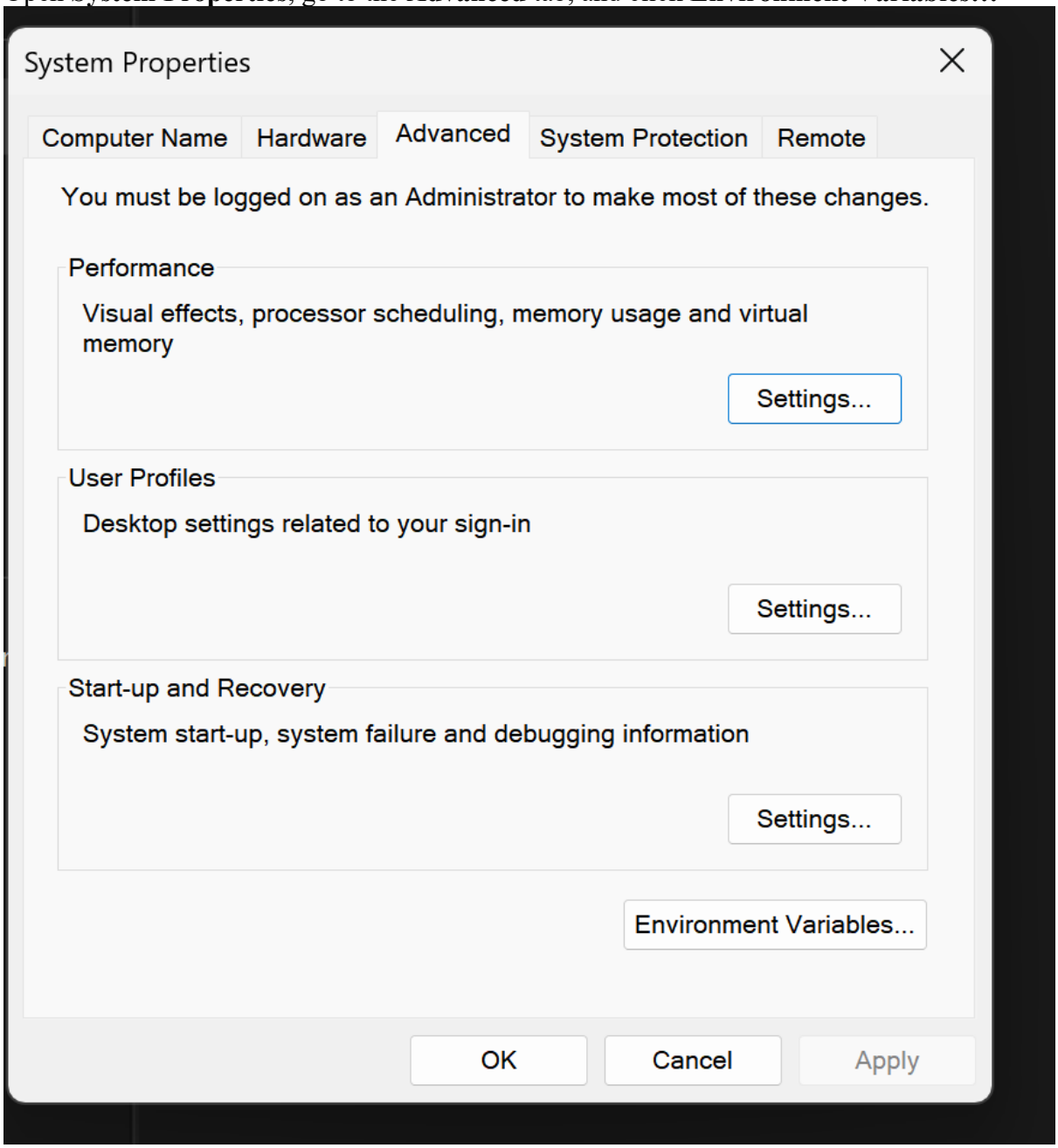


2. **Extract the ZIP:** Extract the downloaded terraform.zip to a folder (e.g., C:\Users\<YourName>\Desktop\terraform). You should see the terraform.exe binary in this folder.
   In the screenshot, the Terraform binary and LICENSE file are extracted to the Desktop.

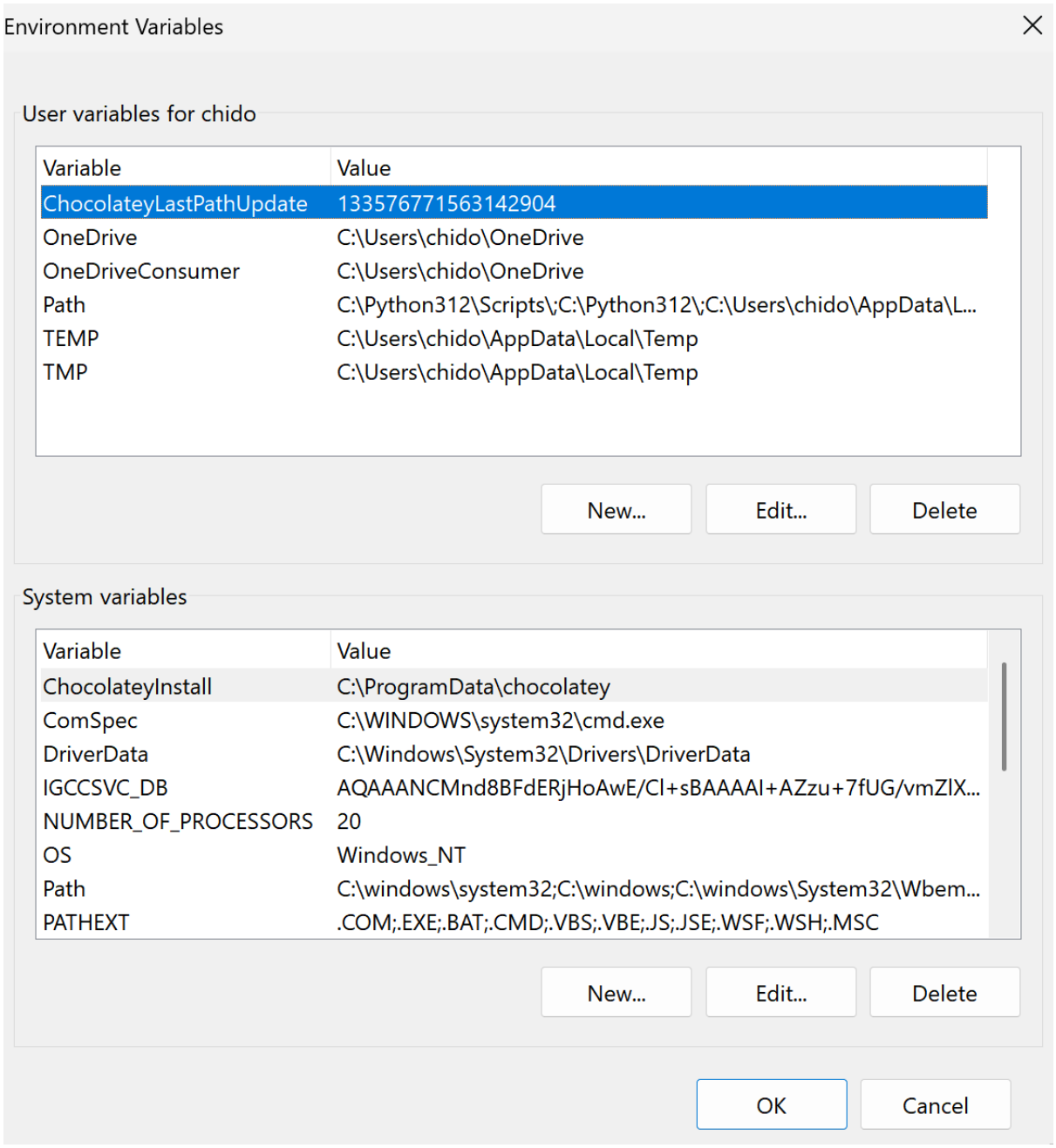


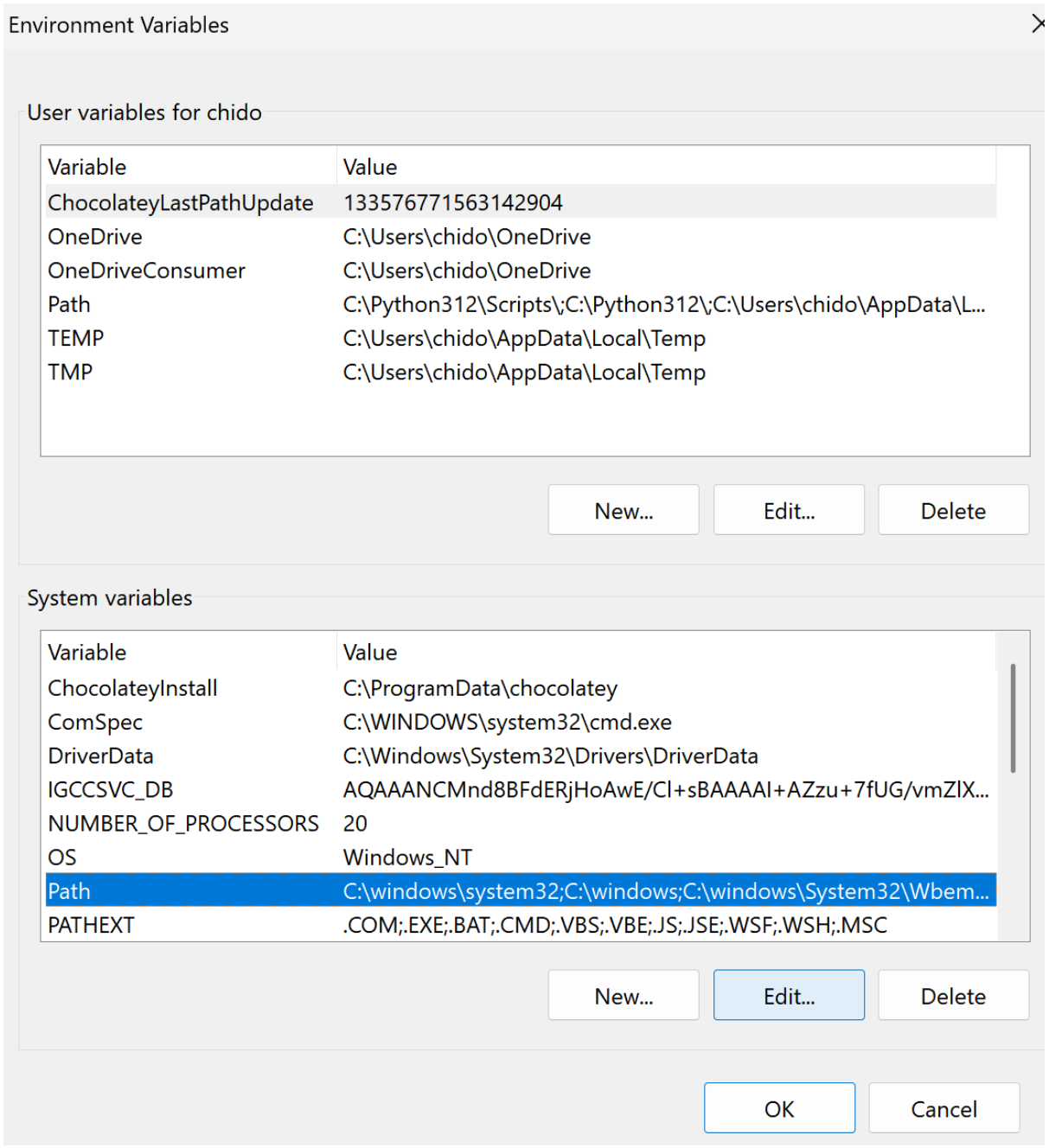
3. **Add to PATH:** Add the folder path (e.g., C:\Users\<YourName>\Desktop\terraform) to your Windows **Environment Variables** so that Terraform can be invoked from any directory:

Open **System Properties**, go to the **Advanced** tab, and click **Environment Variables…**

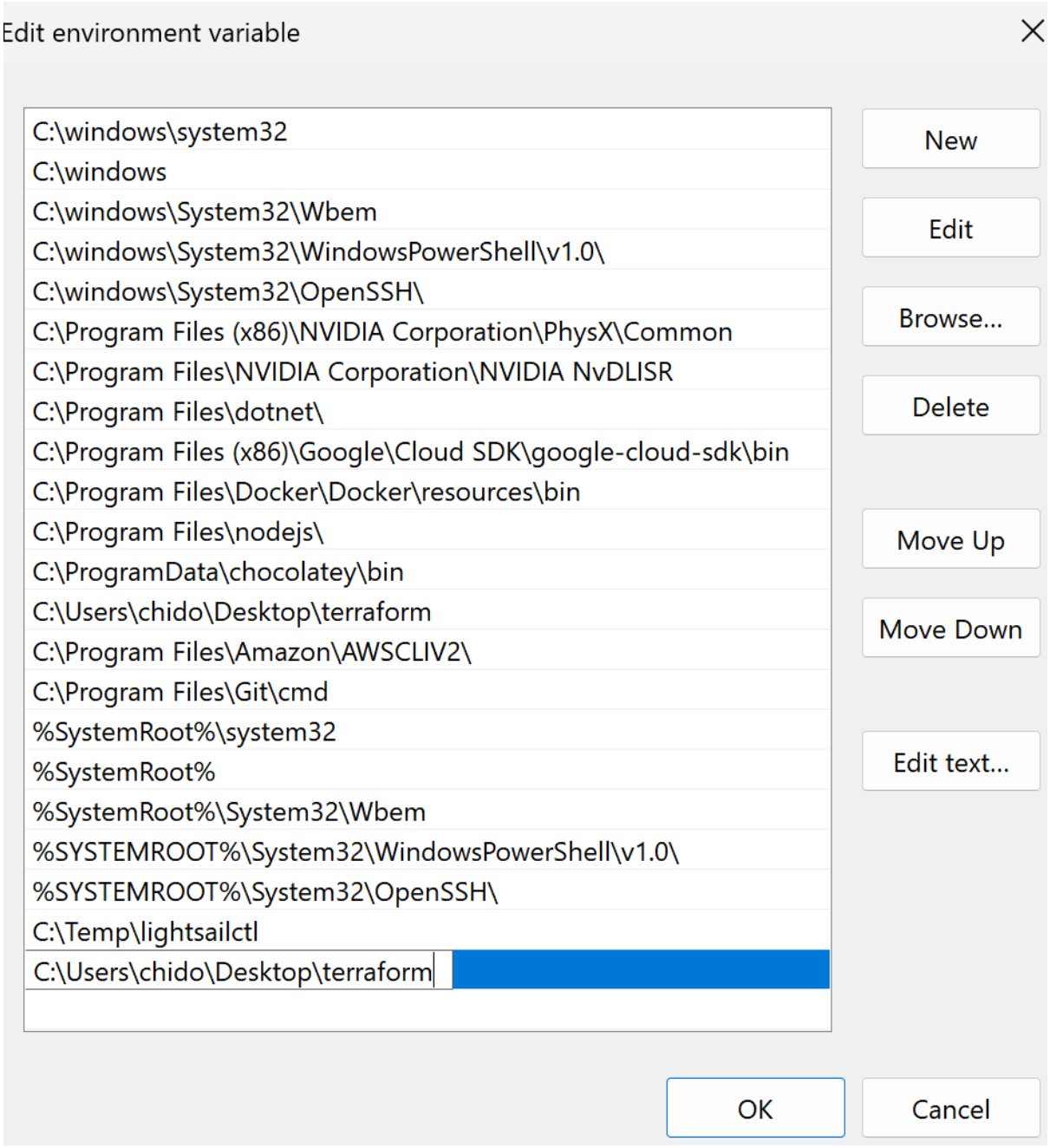


- Under “System variables,” select **Path** and click **Edit**

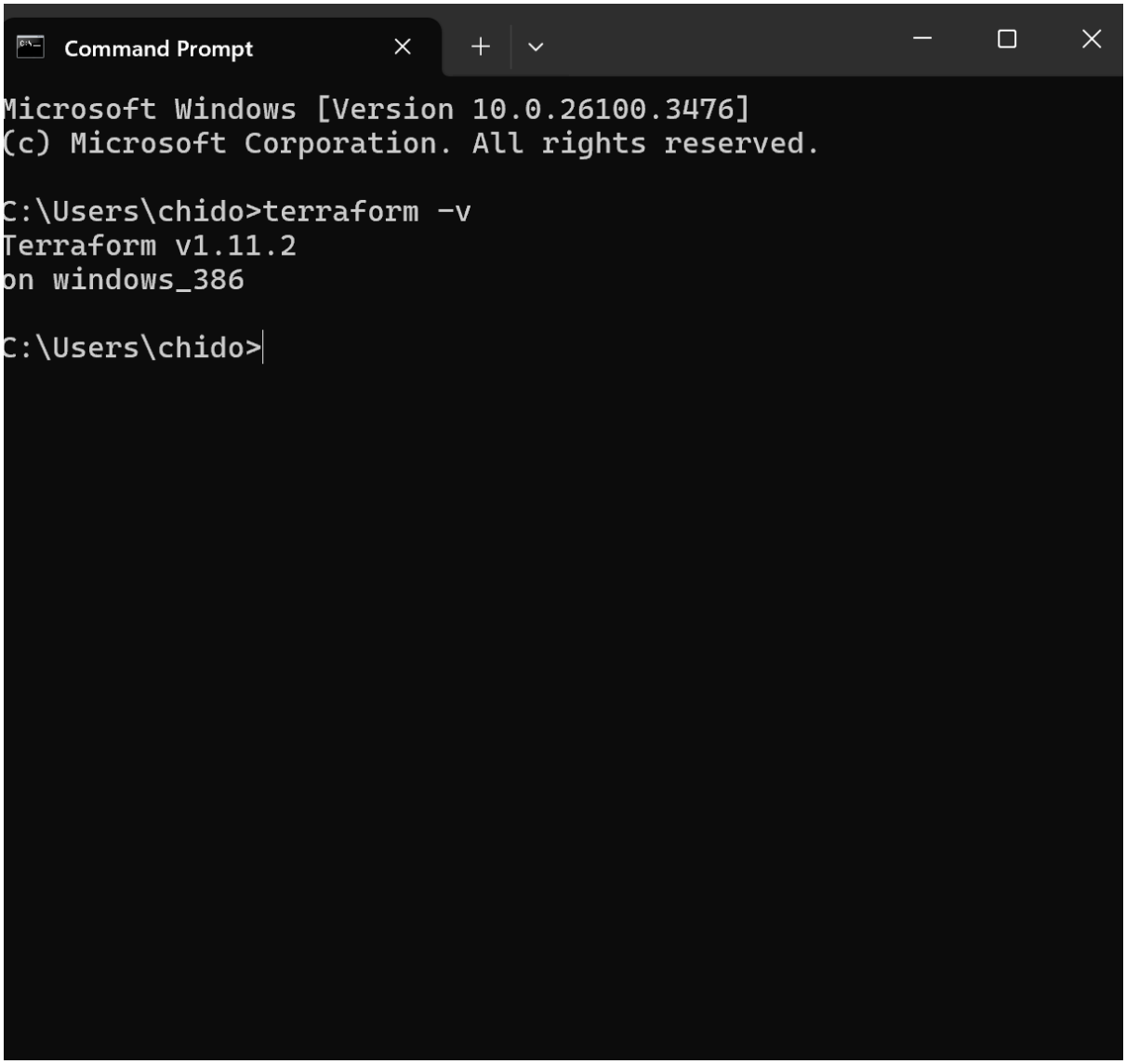

Environment Variables
User variables for chido
Variable
Value
ChocolateyLastPathUpdate
133576771563142904
OneDrive
C:\Users\chido\OneDrive
OneDriveConsumer
C:\Users\chido\OneDrive
Path
C:\Python312\Scripts\;C:\Python312\;C:\Users\chido\AppData\L...
TEMP
C:\Users\chido\AppData\Local\Temp
TMP
C:\Users\chido\AppData\Local\Temp
New...
Edit...
Delete
System variables
Variable
Value
ChocolateyInstall
C:\ProgramData\chocolatey
ComSpec
C:\WINDOWS\system32\cmd.exe
DriverData
C:\Windows\System32\Drivers\DriverData
IGCCSVC_DB
AQAAANCMnd8BFdERjHoAwE/Cl+sBAAAAI+AZzu+7fUG/vmZlX...
NUMBER_OF_PROCESSORS
20
OS
Windows_NT
Path
C:\windows\system32;C:\windows;C:\windows\System32\Wbem...
PATHEXT
.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
New...
Edit...
Delete
OK
Cancel



- Add a new entry with your Terraform folder path and click **OK**



4. **Verify installation:** Open a new Command Prompt and run terraform -v to verify that Terraform is installed and accessible. You should see the Terraform version output (e.g., *Terraform v1.1.2 on windows_386*).
   The screenshot shows a successful terraform -v command output in the Windows CMD.



Why these steps? Terraform is distributed as a single binary. We install it by placing the binary on our system and updating PATH so our shell (Command Prompt) can find it. Verification ensures everything is set up correctly.

3. Sign Up for a Free GitHub Account

If you don't have a GitHub account, create one to host your code:

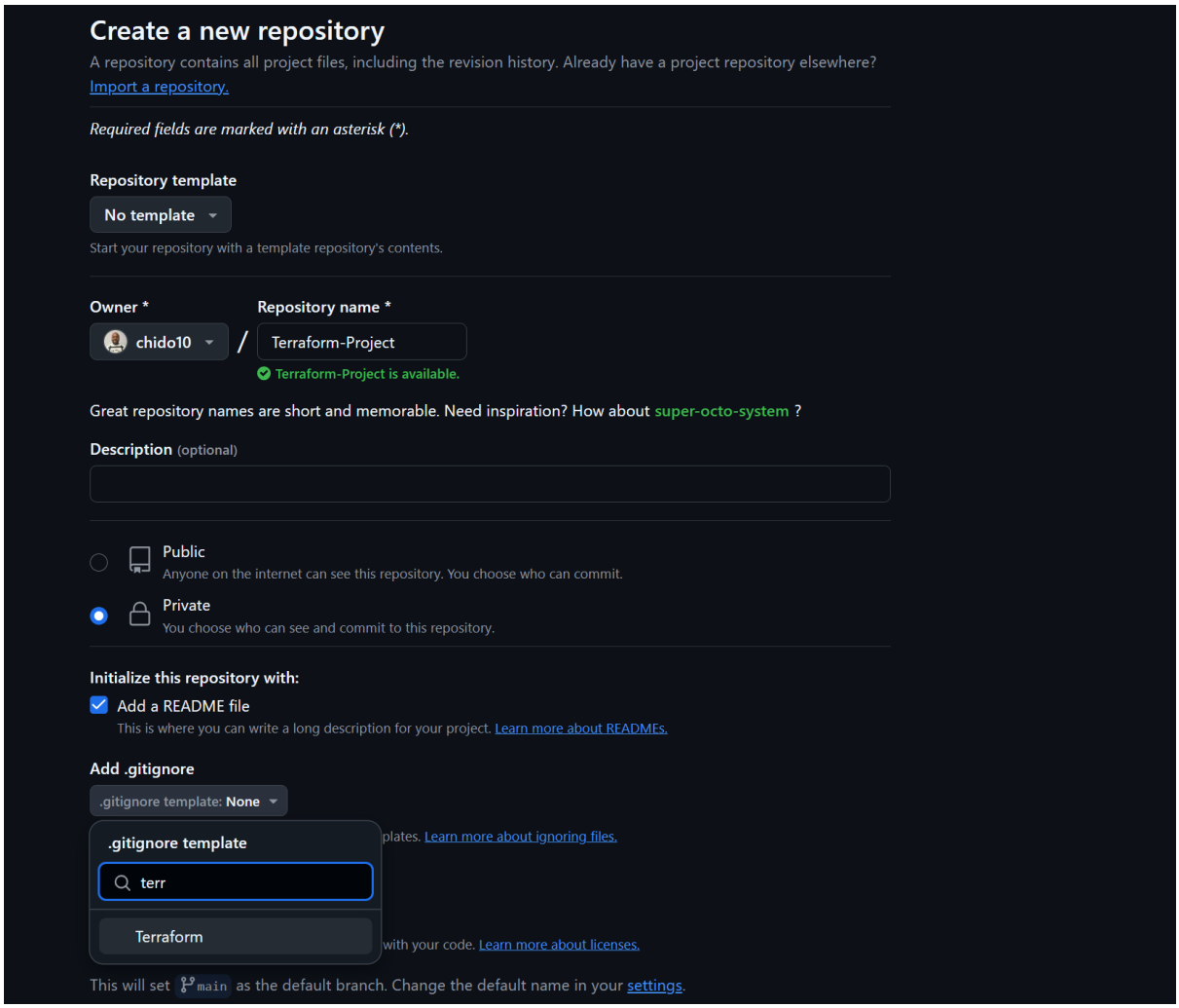
- **Create GitHub Account:** Go to github.com and sign up for a free account. Provide a username, email, and password, then follow the onboarding steps. (If you already have an account, skip this step.)
- **Confirm Email:** Verify your email address via the confirmation link sent by GitHub.
- **Set up Profile (Optional):** You can add a profile picture and bio. This is optional, but an example profile is shown in the screenshot.

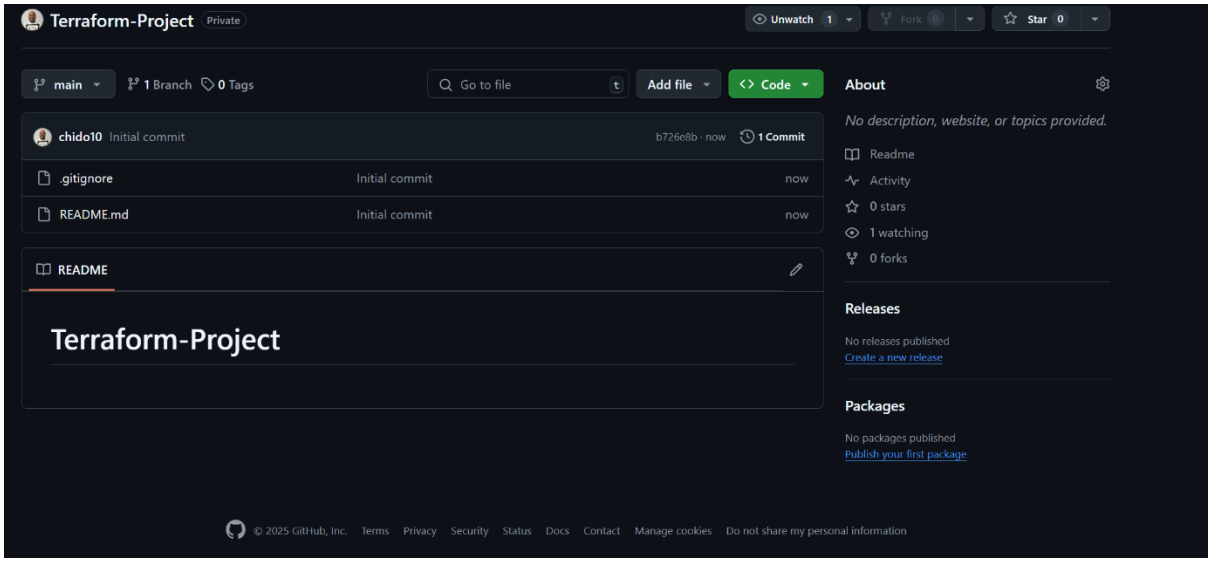
Having a GitHub account is essential for storing code and collaborating. With your account ready, we can create a repository for our Terraform code.

4. Create a GitHub Repository

Next, create a new repository on GitHub to store your Terraform configuration:

1. **New Repository:** In GitHub, click the + (plus) icon and select **"New repository."** Provide a repository name (e.g., **Terraform-Project**) and description (optional). Choose **Private** (since this project's code might contain sensitive info like backend configs) and initialize with a README and .gitignore (for Terraform).
2. **Repository Setup:** After creation, GitHub will display your new repo's main page. It should show the README and .gitignore from the initial commit (by GitHub).
   *The screenshot shows a new private repo **Terraform-Project** with initial commit files.*



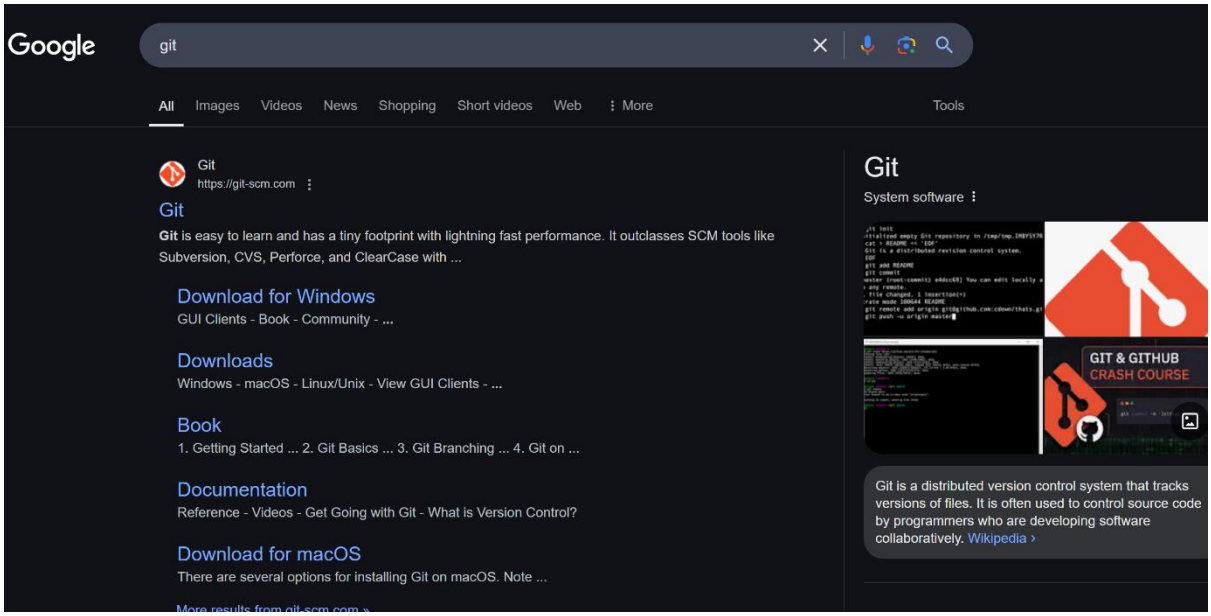


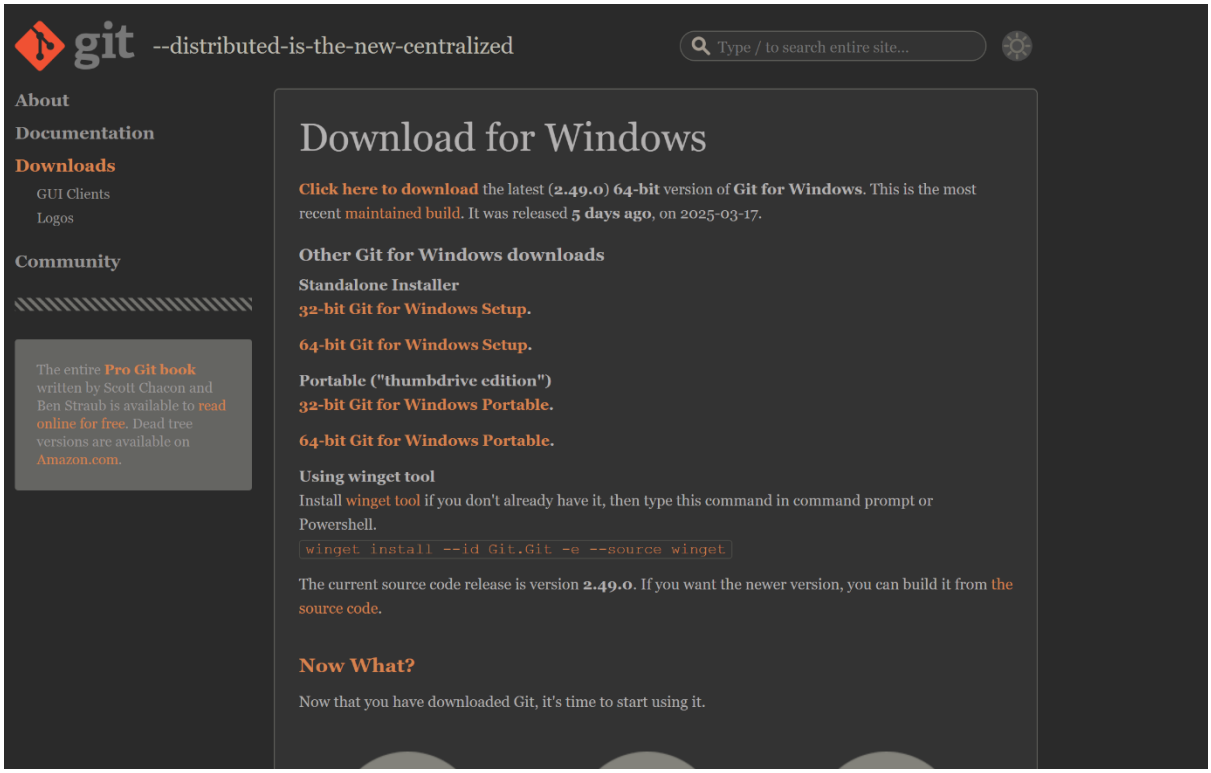
We'll use this repository to push our Terraform code. A private repo keeps your infrastructure code secure (**only you can access it unless you invite collaborators**).

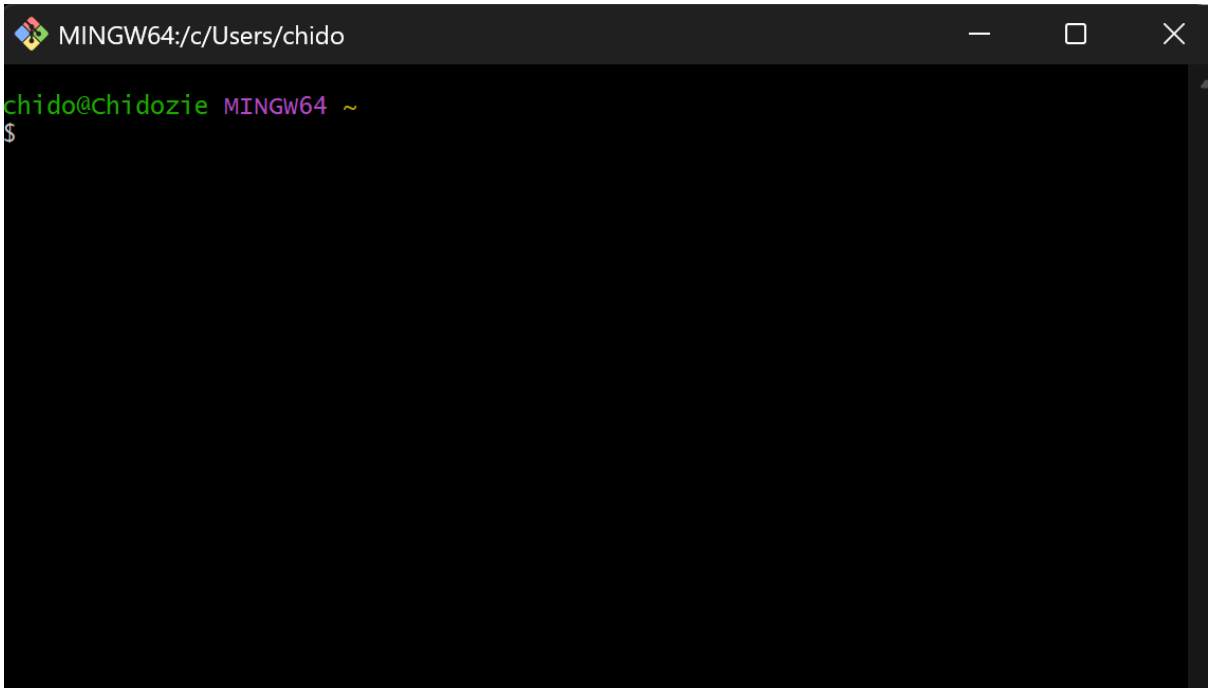
5. Install Git (Windows)

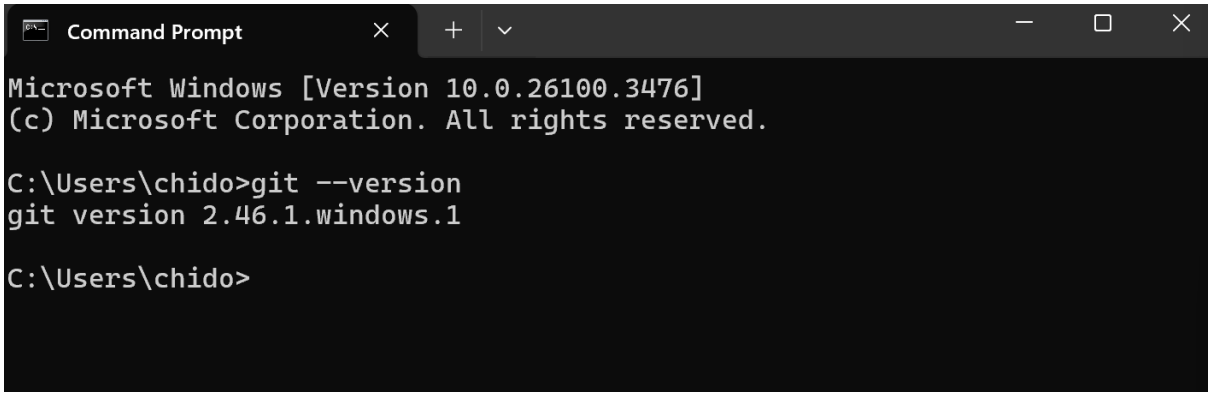
Git is required to clone the repository and push code changes. Here's how to install Git on Windows:

- **Download Git:** Visit the Git for Windows website and download the latest installer (an .exe file).
- **Run Installer:** Launch the installer and follow the prompts. You can accept the default settings (which include adding Git to your PATH, enabling Git Bash, etc.). The default options are usually fine for most users.
- **Complete Installation:** Finish the installer. You can verify Git is installed by opening **Command Prompt** or **PowerShell** and running git --version. You should see a version string (e.g., *git version 2.x.x*).


Google
git
All
Images
Videos
News
Shopping
Short videos
Web
More
Tools
Git
https://git-scm.com
Git
Git is easy to learn and has a tiny footprint with lightning fast performance. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with ...
Download for Windows
GUI Clients - Book - Community - ...
Downloads
Windows - macOS - Linux/Unix - View GUI Clients - ...
Book
1. Getting Started ... 2. Git Basics ... 3. Git Branching ... 4. Git on ...
Documentation
Reference - Videos - Get Going with Git - What is Version Control?
Download for macOS
There are several options for installing Git on macOS. Note ...
Git
System software
GIT & GITHUB
CRASH COURSE
Git is a distributed version control system that tracks versions of files. It is often used to control source code by programmers who are developing software collaboratively. Wikipedia ›


git
--distributed-is-the-new-centralized
Type / to search entire site...
About
Documentation
Downloads
GUI Clients
Logos
Community
The entire Pro Git book written by Scott Chacon and Ben Straub is available to read online for free. Dead tree versions are available on Amazon.com.
Download for Windows
Click here to download the latest (2.49.0) 64-bit version of Git for Windows. This is the most recent maintained build. It was released 5 days ago, on 2025-03-17.
Other Git for Windows downloads
Standalone Installer
32-bit Git for Windows Setup.
64-bit Git for Windows Setup.
Portable ("thumbdrive edition")
32-bit Git for Windows Portable.
64-bit Git for Windows Portable.
Using winget tool
Install winget tool if you don't already have it, then type this command in command prompt or Powershell.
winget install --id Git.Git -e --source winget
The current source code release is version 2.49.0. If you want the newer version, you can build it from the source code.
Now What?
Now that you have downloaded Git, it's time to start using it.





Git installation ensures you have the git command available to manage version control. Now you can interact with GitHub from your computer.

(There was also a note about Mac users in step 6, but since this is Windows-focused, you can skip Mac installation details unless you switch to a Mac.)

6. Install Git (Mac Users Only)

If you are on a Mac, installing Git might not require a separate step since macOS often has Git (via Xcode Command Line Tools). If not, you can install Homebrew and run brew install git or download Git for Mac. Since our primary environment is Windows, this step is labeled "ONLY watch if you are using a Mac." (No content needed for Windows users.)

In summary, Mac users ensure Git is installed (check with git --version). Windows users should focus on step 5.

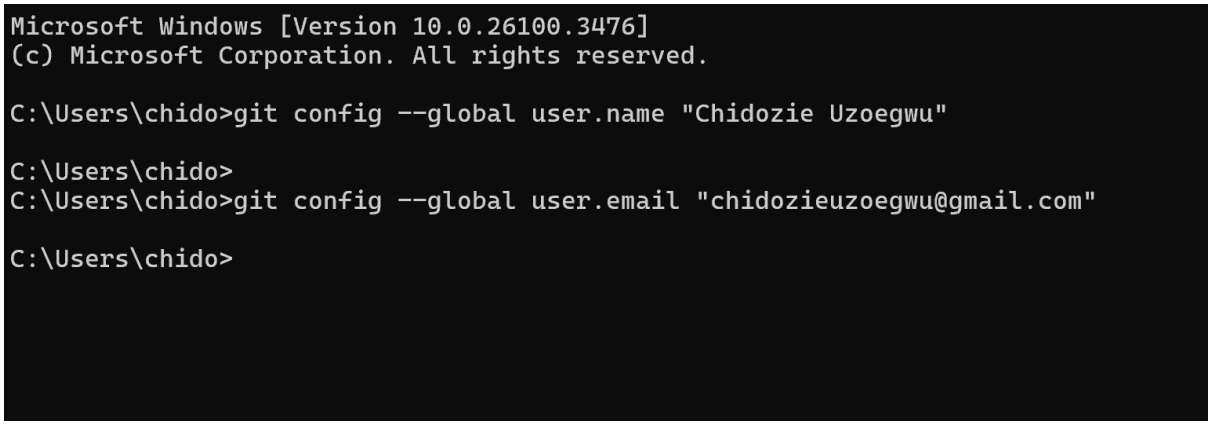
7. Create Key Pairs (SSH Keys)

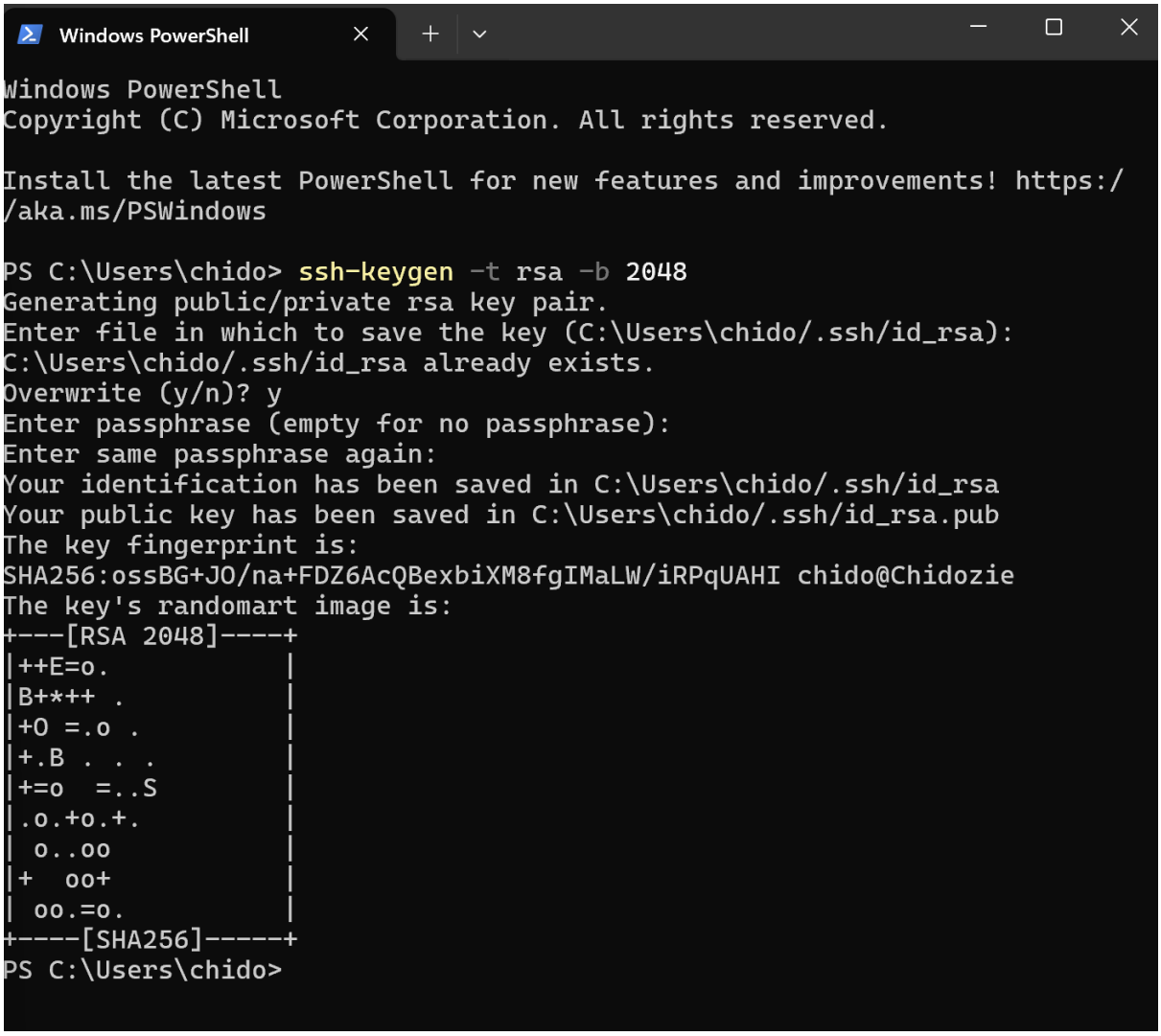
To securely interact with GitHub without typing your password each time, create an SSH key pair and add the public key to GitHub:

- **Generate SSH Key:** On Windows, open **Git Bash** (installed with Git) or PowerShell and run:

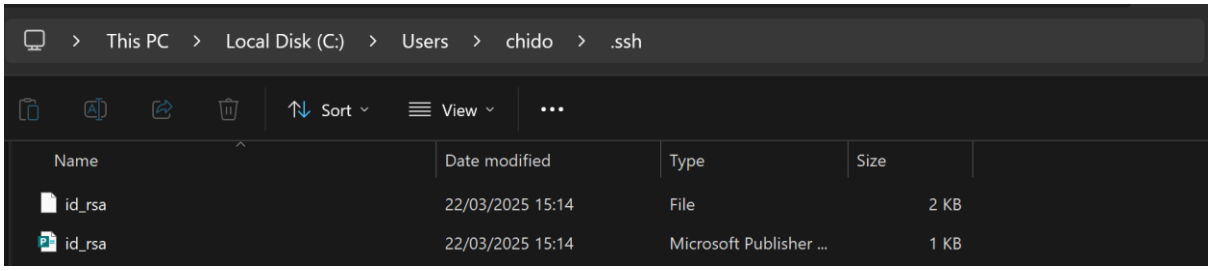
  ssh-keygen -t rsa -b 4096 -C "your_email@example.com"

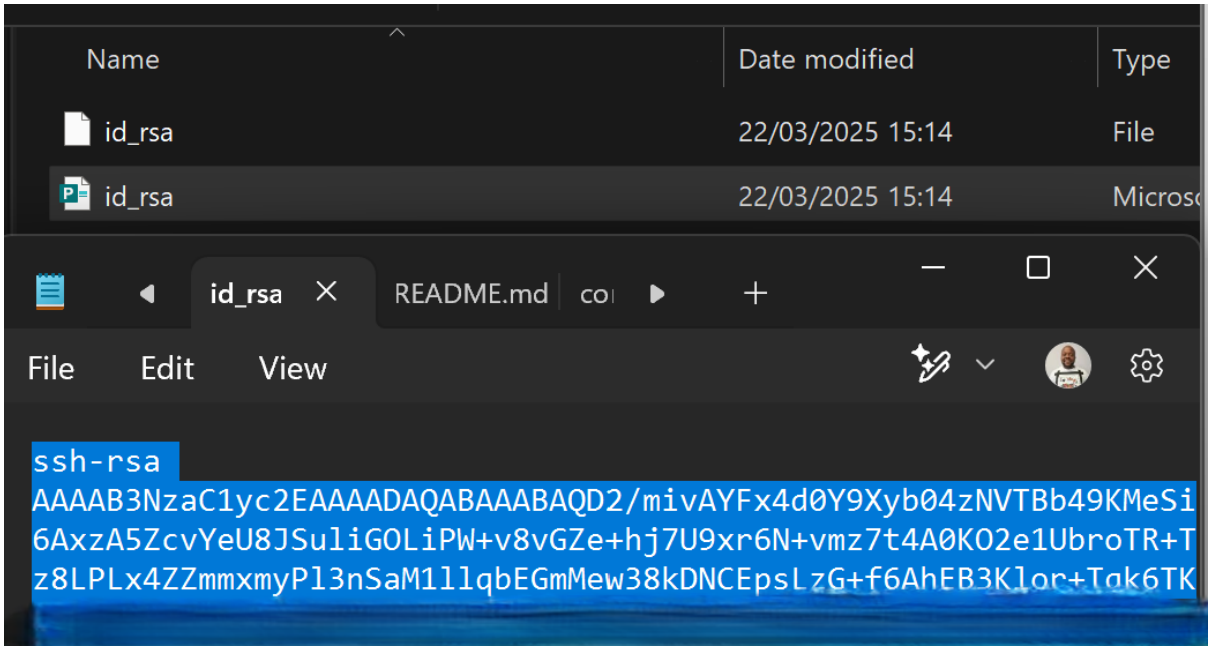
  Press enter to accept the default key file location (e.g., C:\Users\<YourName>\.ssh\id_rsa). Choose a passphrase or press enter for none.





- **Locate Keys:** After generation, you'll have two files in ~/.ssh/ directory: **id_rsa** (private key) and **id_rsa.pub** (public key). Keep the private key secure; we will use the public key.





The screenshot shows the .ssh folder containing id_rsa and id_rsa.pub (the latter is shown with a different icon or extension, but the content is the public key).

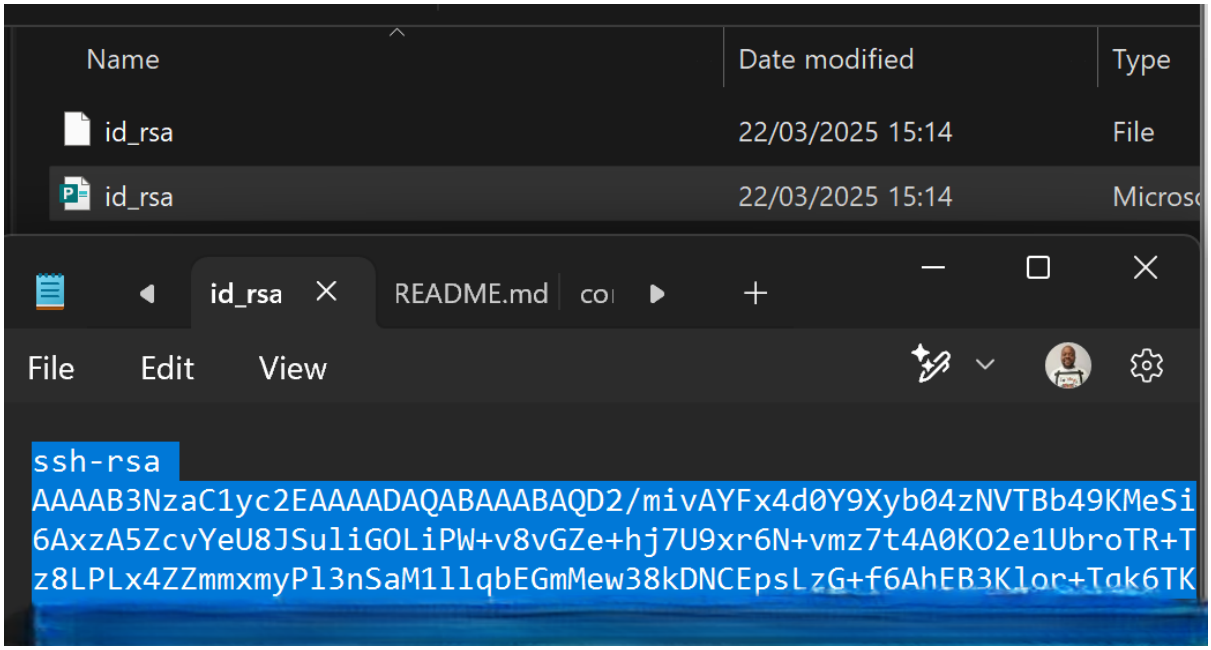
The SSH key pair will allow GitHub to recognize your computer. Next, we'll add the public key to your GitHub account.

8. Add the Public SSH Key to GitHub

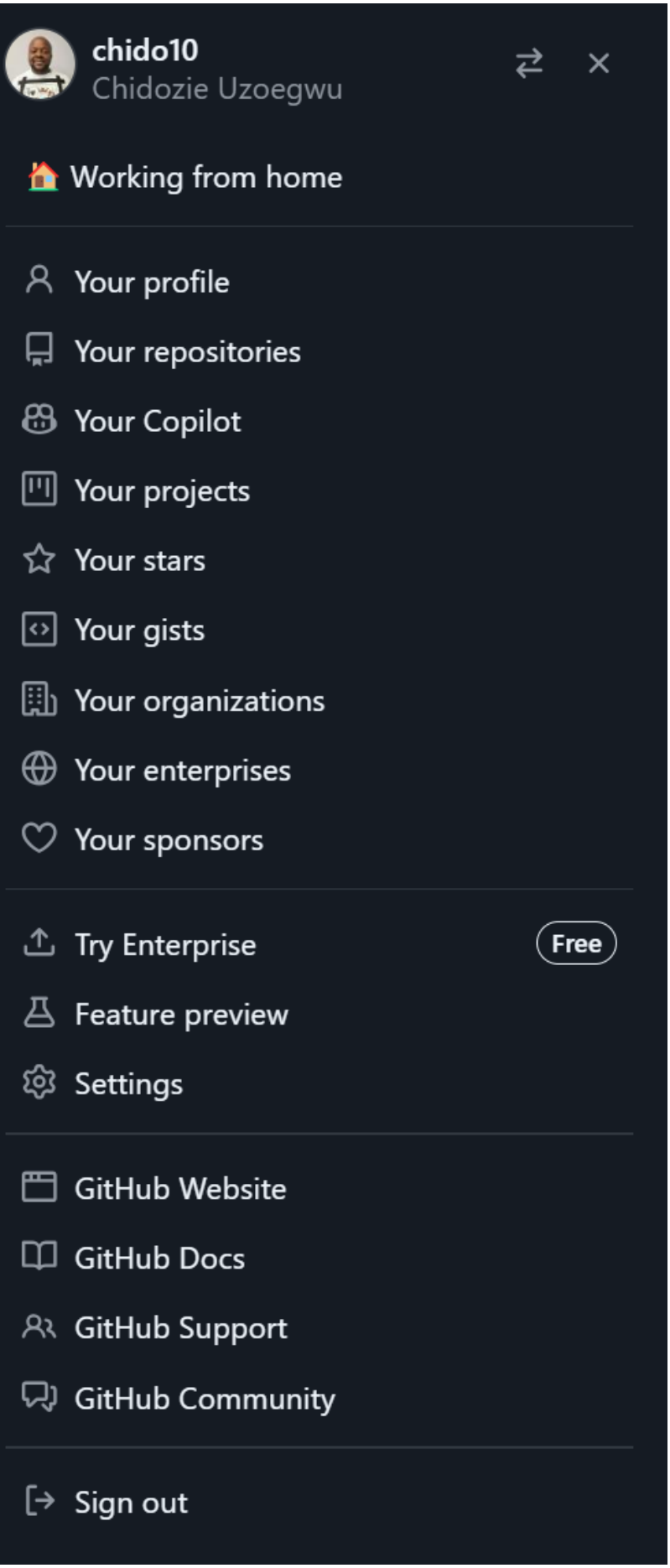
Now, configure GitHub to trust your computer's SSH key:

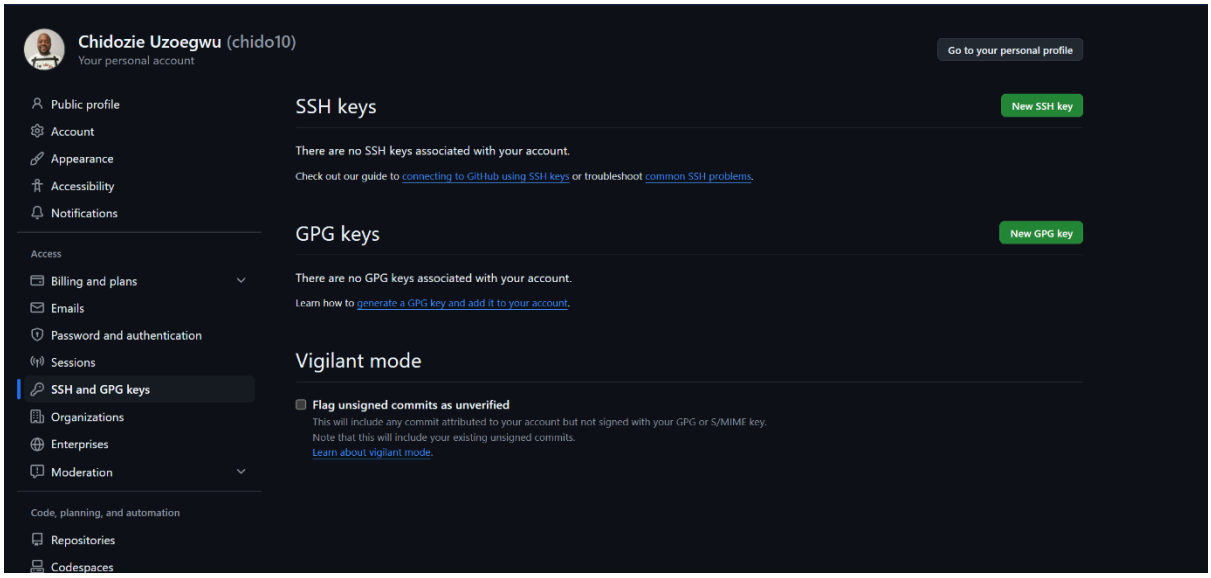
1. **Copy Public Key:** Open the id_rsa.pub file with a text editor (Notepad or VS Code). Copy the entire contents (it starts with ssh-rsa and ends with your email).

   The screenshot below shows the public key text highlighted in notepad

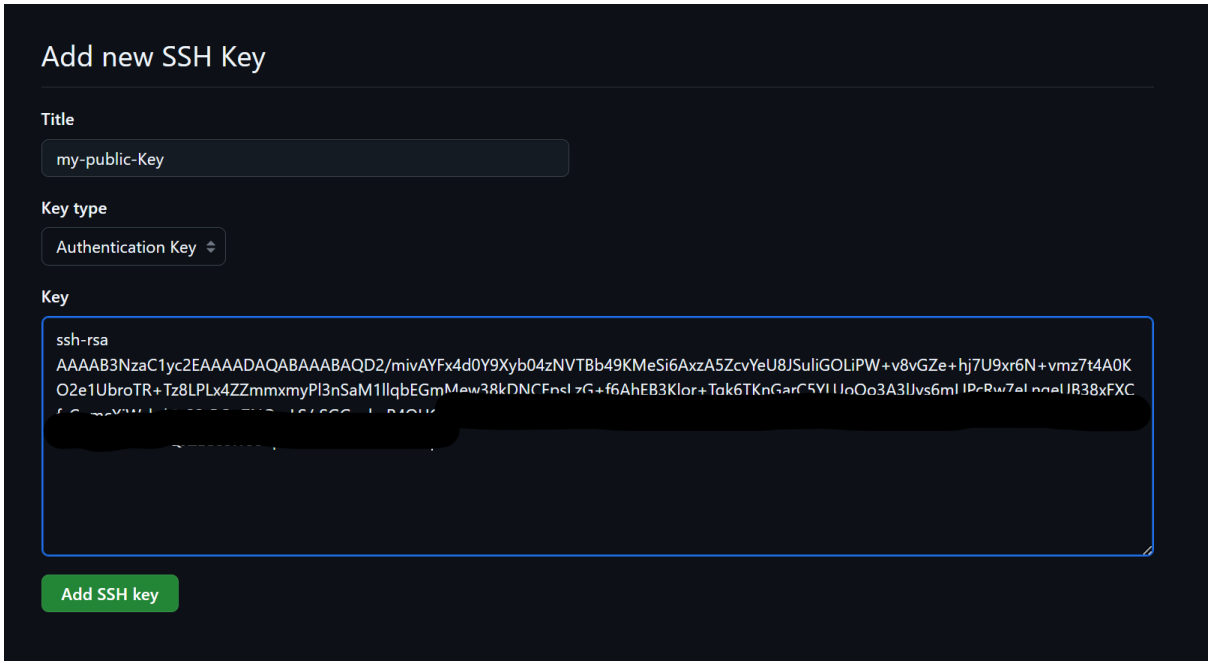


2. **GitHub Settings:** Go to GitHub, click your profile picture (top-right) and choose **"Settings."** On the left sidebar, click **"SSH and GPG keys."**


chido10
Chidozie Uzoegwu
Working from home
Your profile
Your repositories
Your Copilot
Your projects
Your stars
Your gists
Your organizations
Your enterprises
Your sponsors
Try Enterprise
Free
Feature preview
Settings
GitHub Website
GitHub Docs
GitHub Support
GitHub Community
Sign out

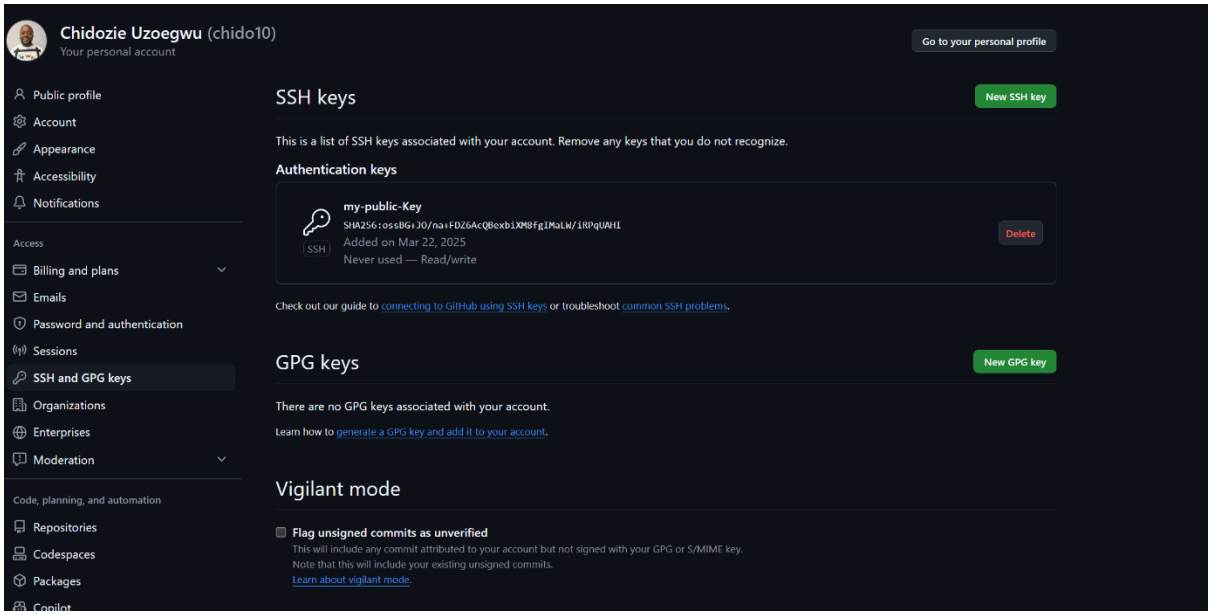


3. **New SSH Key:** Click **"New SSH key."** Give it a title like "my-public-Key" (or a description of the machine) and paste the public key into the Key field. Choose key type **Authentication Key** if prompted.



The screenshot shows adding a new SSH Key titled "my-public-Key" with the key pasted.

4. **Save Key:** Click **"Add SSH key."** You may be asked to confirm with your GitHub password. Once added, you'll see the key in your SSH keys list on GitHub.
   Below, GitHub now lists "my-public-Key" (added on Mar 22, 2025) under SSH keys.

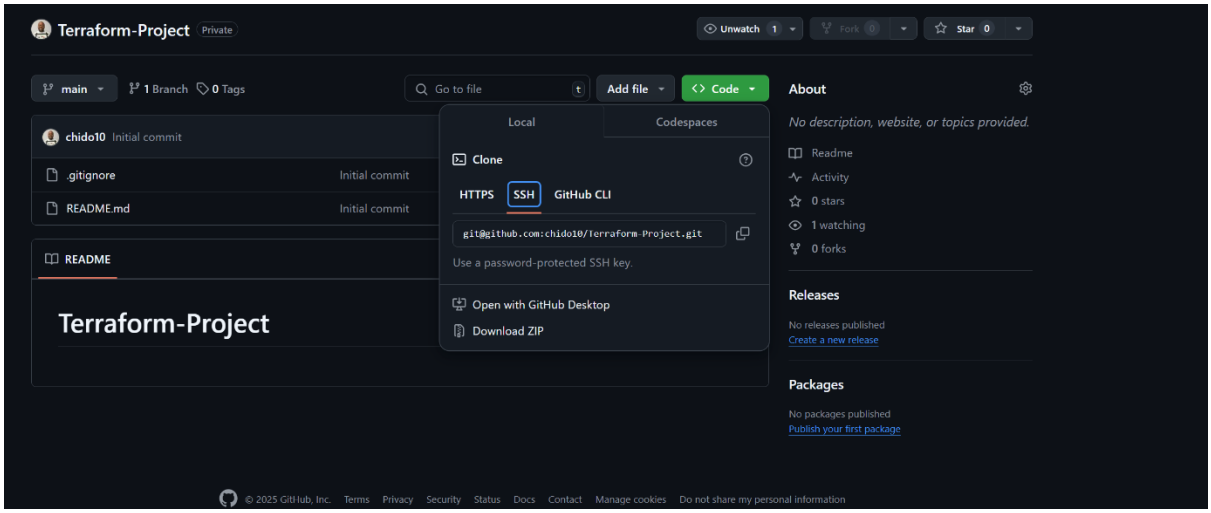


GitHub now knows your public key. This means you can authenticate via SSH for Git operations without a username/password.

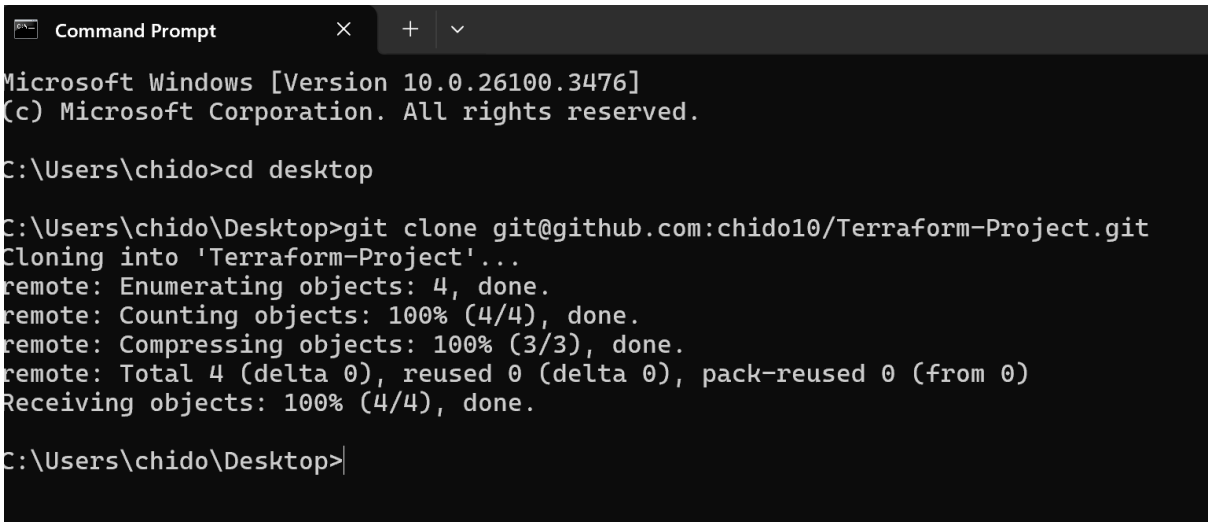
9. Clone Your Private Repository

With Git installed and SSH set up, clone the GitHub repository to your local machine:

1. **Get Clone URL:** Navigate to your GitHub repository page and click the green **"Code"** button. Select **SSH** to get the SSH clone URL (it will look like git@github.com:YourUsername/RepositoryName.git). Copy that URL.
   *Screenshot: The **Terraform-Project** repo's Code dropdown with the SSH URL selected.*

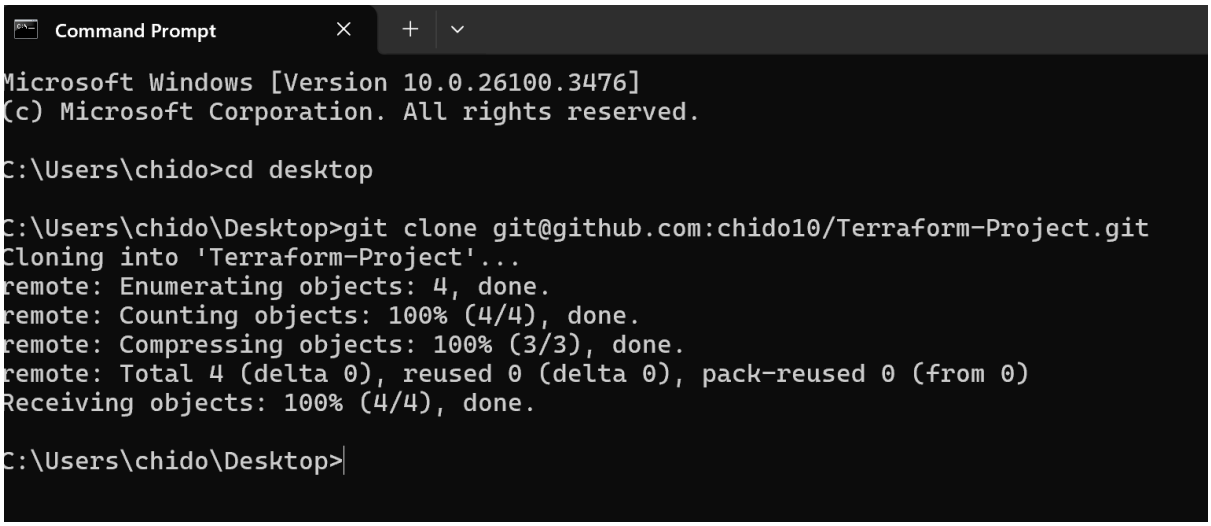


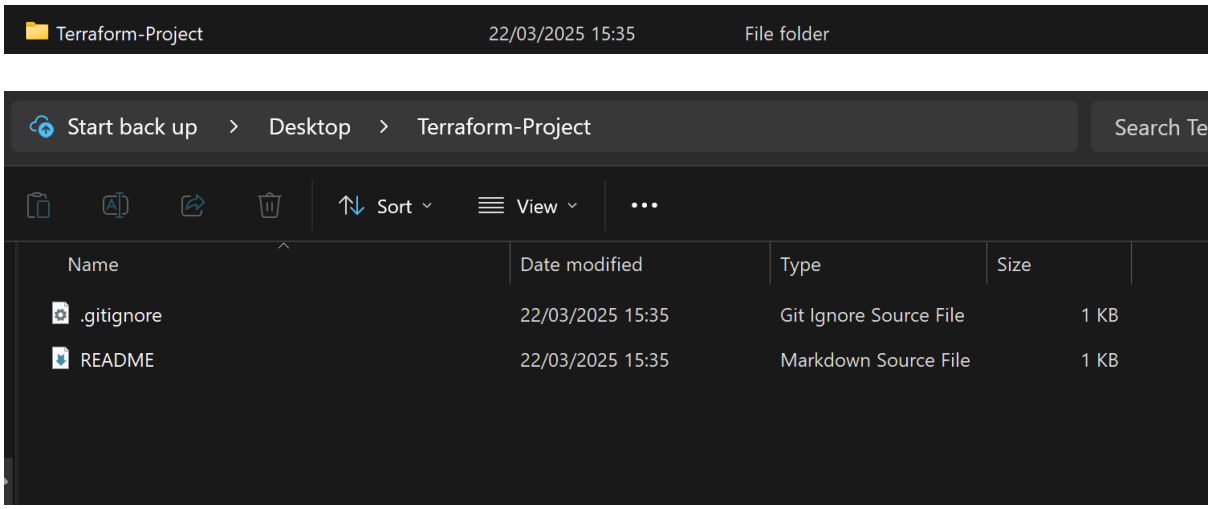
2. **Clone via Command Line:** Open Command Prompt (or Git Bash) in the directory where you want the project folder (e.g., your Desktop). Run:
   git clone git@github.com:<YourUsername>/Terraform-Project.git
   This will create a folder named **Terraform-Project** with the repository content.
   Screenshot below shows the clone command executed in CMD.



3. **Clone Confirmation:** Once done, navigate into the new folder (cd Terraform-Project) and list files. You should see the README.md and .gitignore that GitHub created.

Below you can see the cloning process and confirmation in the terminal output (4 objects received, etc.).





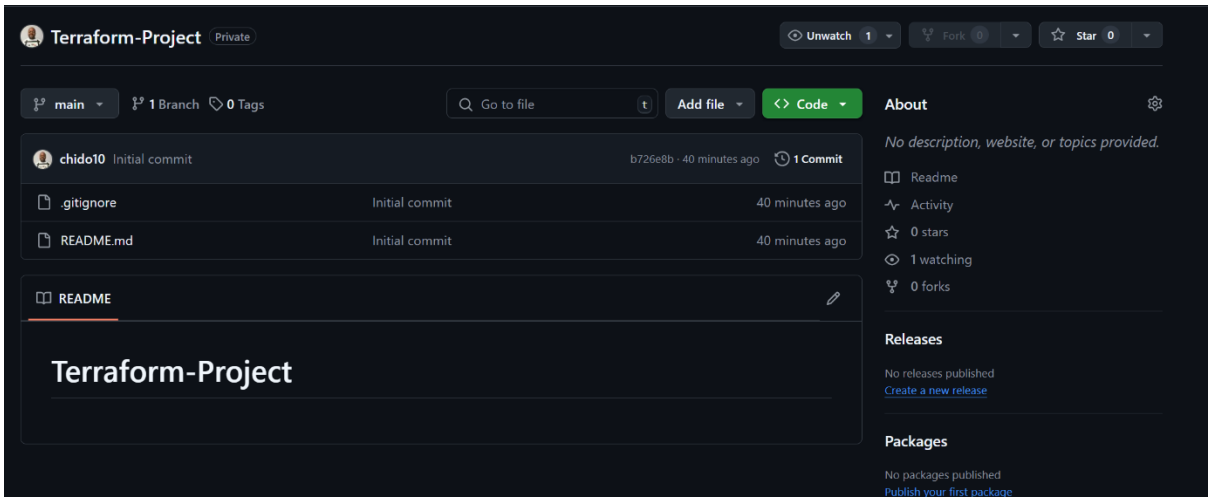
You now have the repository locally. You can edit files and push changes back to GitHub, which we'll do next.

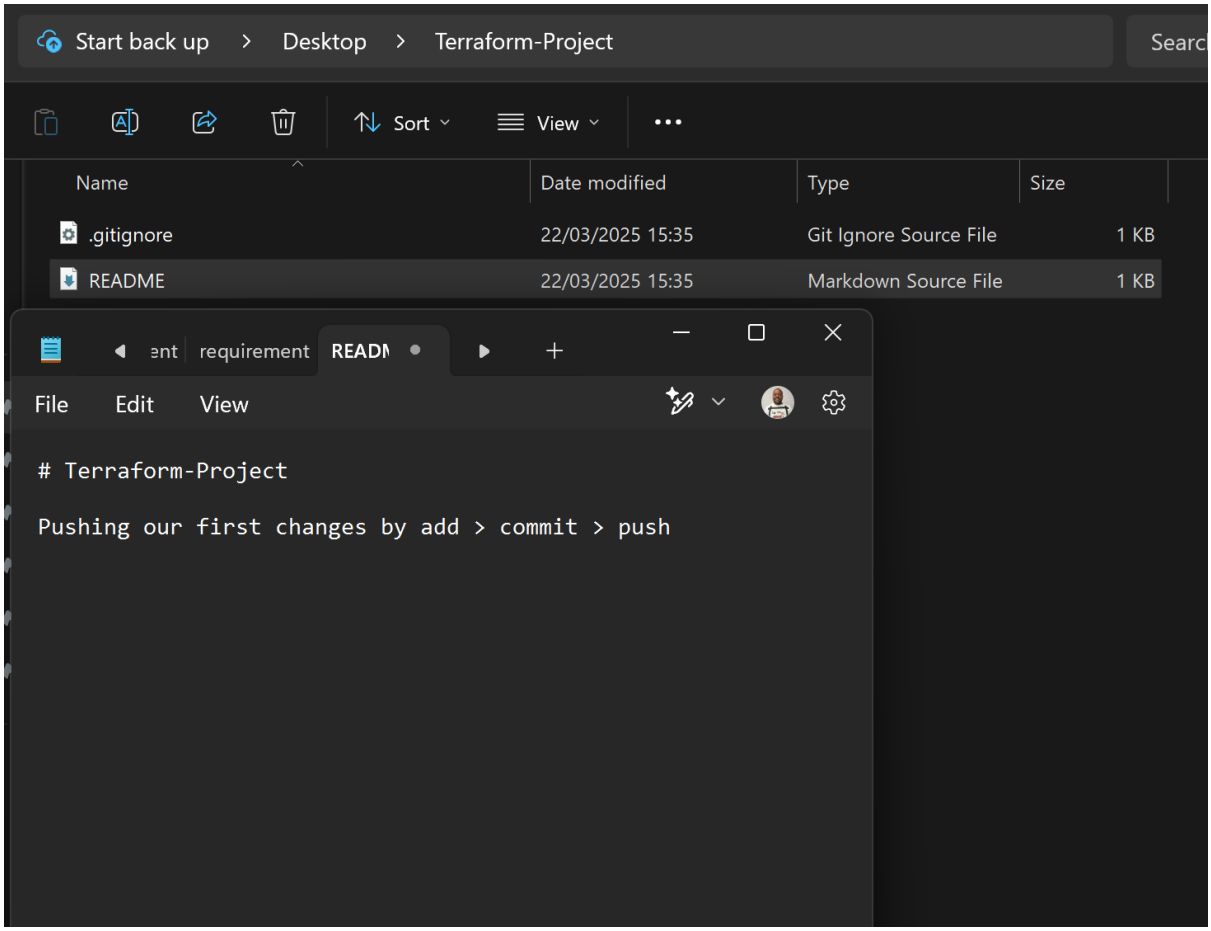
10. Push Changes from Local to Remote

Let's make a simple change to test pushing code to GitHub:

- **Edit README:** Open the README.md file from the **Terraform-Project** folder (you can use Visual Studio Code or any text editor). Add a line or two (for example, "Pushing our first changes by add > commit > push"). Save the file.

  In the screenshot, the README has a new line "Pushing our first changes by add > commit > push."


Terraform-Project
Private
Unwatch 1
Fork 0
Star 0
main
1 Branch
0 Tags
Go to file
t
Add file
Code
About
No description, website, or topics provided.
chido10 Initial commit
b726e8b · 40 minutes ago
1 Commit
.gitignore
Initial commit
40 minutes ago
README.md
Initial commit
40 minutes ago
Readme
Activity
0 stars
1 watching
0 forks
README
Terraform-Project
Releases
No releases published
Create a new release
Packages
No packages published
Publish your first package

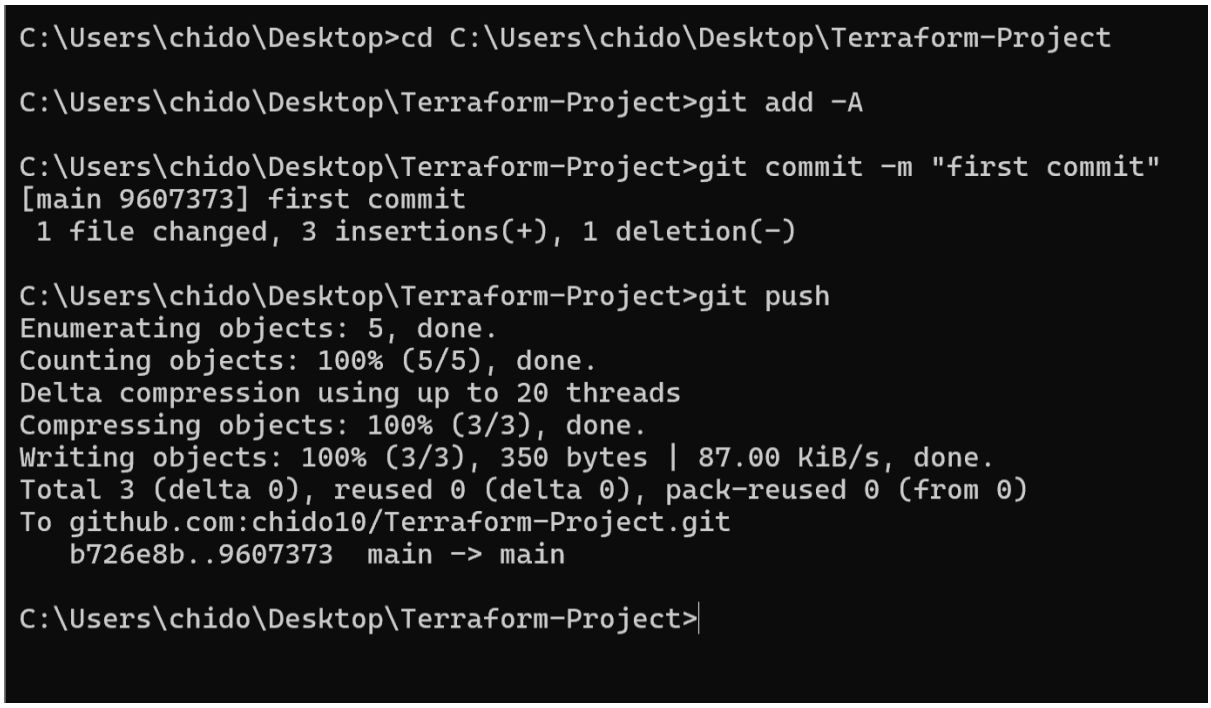

Start back up
Desktop
Terraform-Project
Sort
View
Name
Date modified
Type
Size
.gitignore
22/03/2025 15:35
Git Ignore Source File
1 KB
README
22/03/2025 15:35
Markdown Source File
1 KB
ent
requirement
READM
File
Edit
View
Terraform-Project
Pushing our first changes by add > commit > push

- **Git Add & Commit:** In Command Prompt, navigate to the Terraform-Project directory (cd C:\Users\<YourName>\Desktop\Terraform-Project). Run:
- git add -A
- git commit -m "first commit"

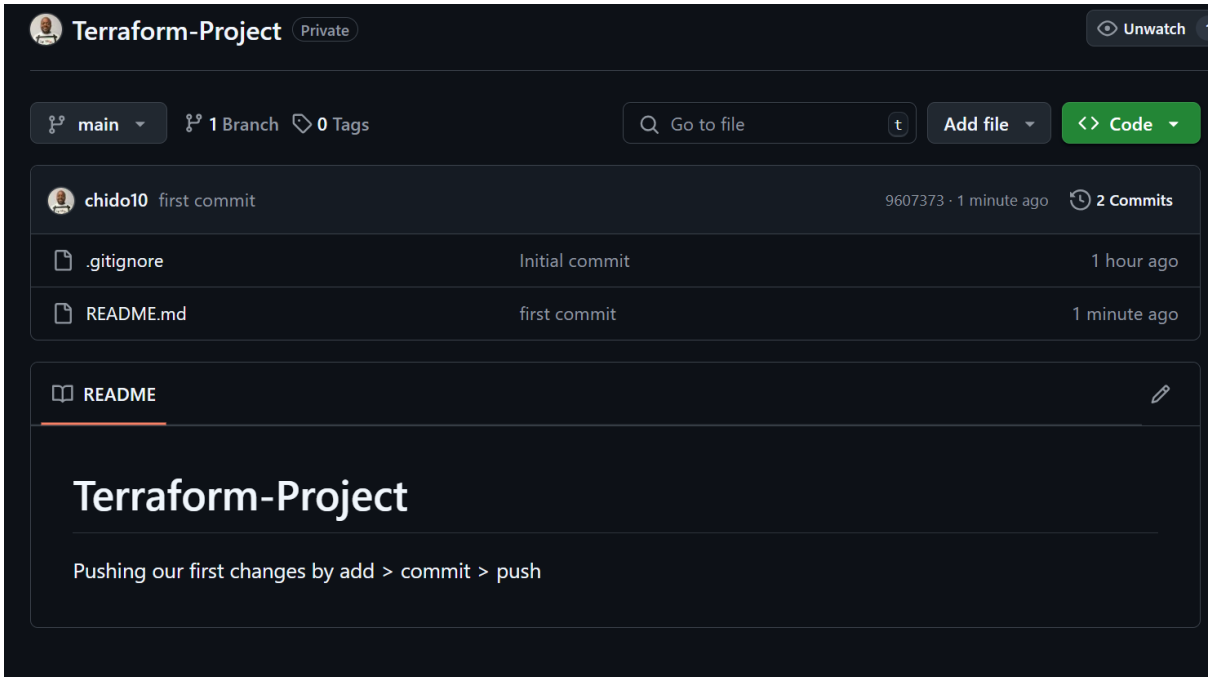
This stages all changes and commits them with a message.

Screenshot shows git add -A and git commit -m "first commit" executed, resulting in a commit that changed 1 file (README.md).

- **Git Push:** Push the commit to GitHub by running:
  git push
  This uses the SSH key for authentication and uploads your commit.
  Screenshot: The terminal output of git push indicates writing objects and confirms main -> main on GitHub.



- **Verify on GitHub:** Refresh the GitHub repository page. You should see the new commit and the updated README content on GitHub.
  The screenshot shows GitHub now listing 2 commits ("Initial commit" and "first commit"), and the README on GitHub includes the new text.



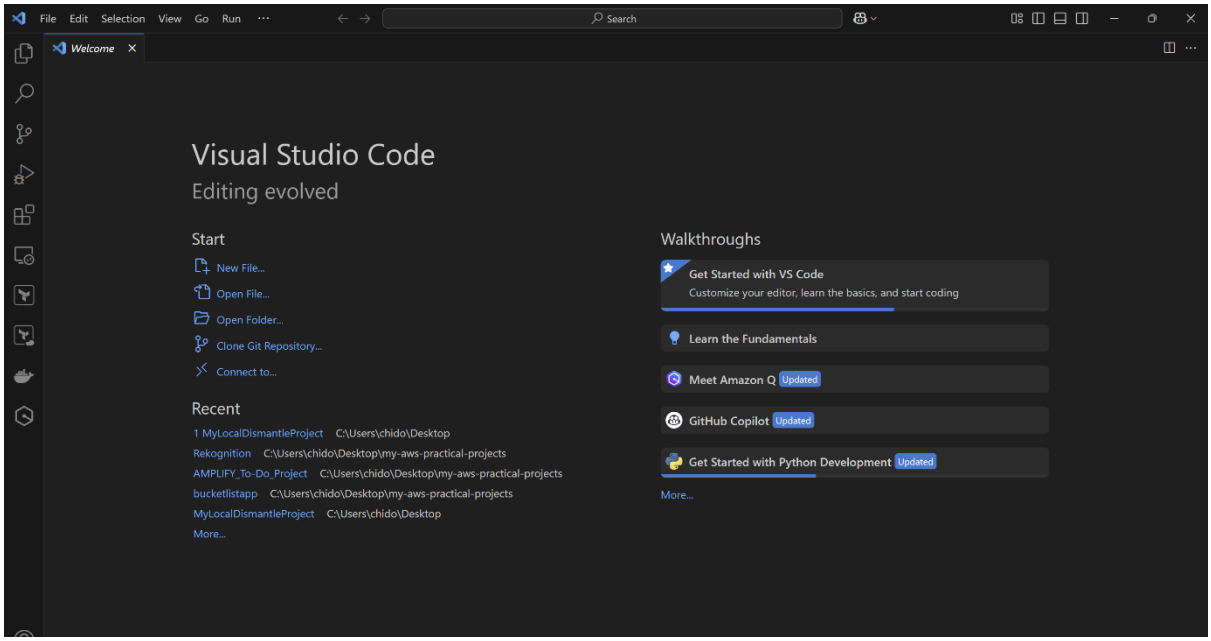
Now we have our Terraform code repository ready to go. Next, we'll set up a development environment and AWS credentials.

11. Install Visual Studio Code (VS Code)

We'll use VS Code to write and manage our Terraform code. It's a free, lightweight code editor with great Terraform support:

- **Download VS Code:** Go to the Visual Studio Code website and download the installer for Windows. Choose the correct installer (User Installer or System Installer, 64-bit for most modern systems).
  Screenshot shows the VS Code download page with options for Windows, Linux, and Mac.

- **Install VS Code:** Run the installer and follow the steps (accept the agreement, choose install location, etc.). Optionally, select “Add to PATH” during installation for convenience. After installation, launch VS Code.
- **Initial Screen:** When VS Code opens, you’ll see a Welcome page. From here, you can open the **Terraform-Project** folder.
  The screenshot shows the VS Code welcome tab (with recent projects) after installation.

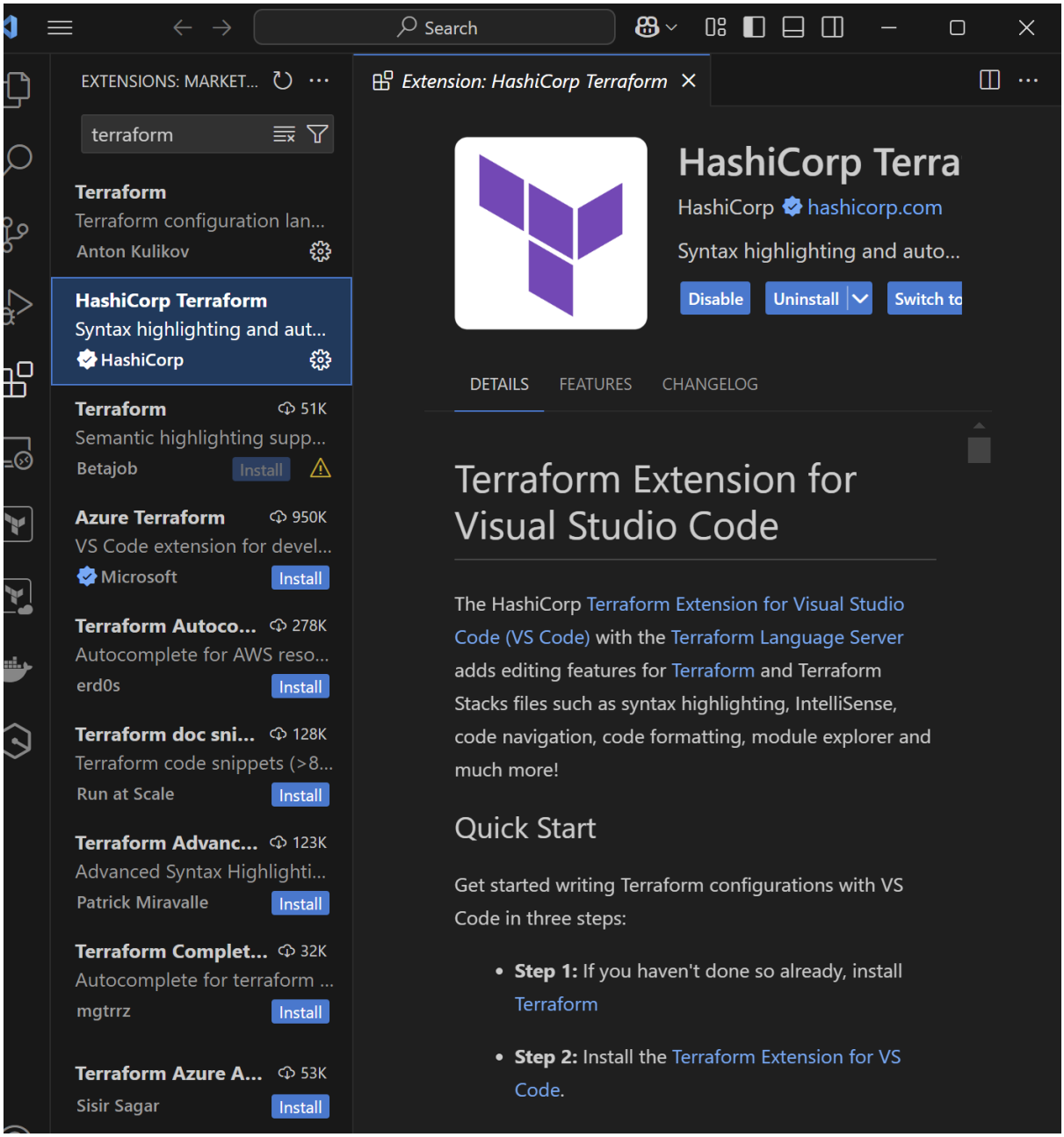


Having VS Code will make it easier to edit multiple Terraform files, and next we'll enhance it with Terraform-specific extensions.

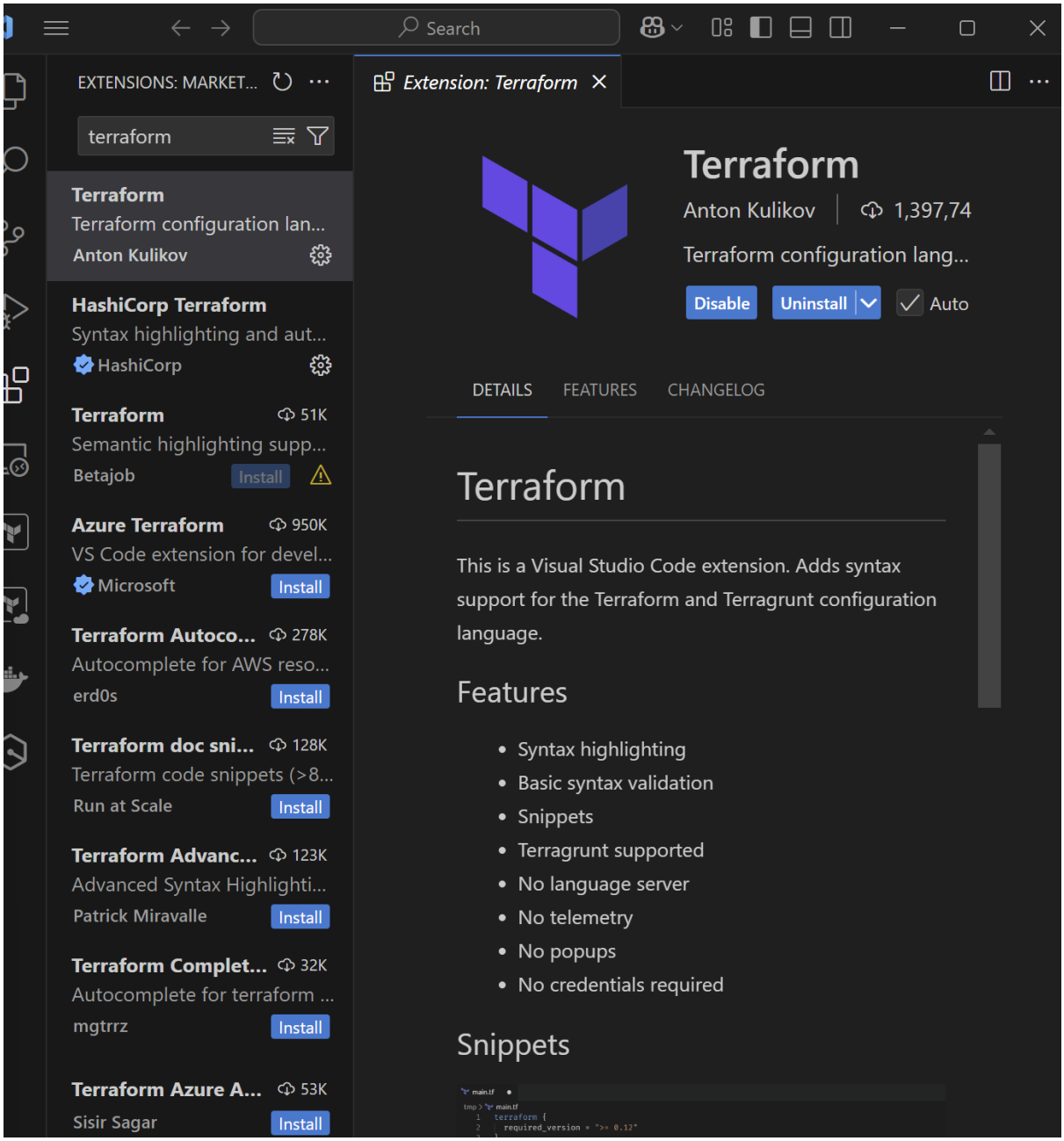
12. Install the Extensions for Terraform

To improve the Terraform development experience, we install VS Code extensions for Terraform:

- **HashiCorp Terraform Extension:** In VS Code, click the Extensions icon (on the left toolbar, looks like building blocks) and search for "HashiCorp Terraform". Install the official **HashiCorp Terraform** extension. This adds syntax highlighting, autocompletion, and integration with Terraform Language Server.
  Screenshot: The HashiCorp Terraform extension page in VS Code showing it is installed (Disable/Uninstall buttons visible).



- **Terraform by Anton Kulikov (optional):** There are other extensions like one by Anton Kulikov which provides Terraform configuration language support. If recommended by the course, you can install it as well. In the search results, you might see “Terraform” by Anton Kulikov and others.
  Screenshot: Anton Kulikov’s Terraform extension in the VS Code marketplace (over 1.3M downloads). It’s shown as installed in this example.



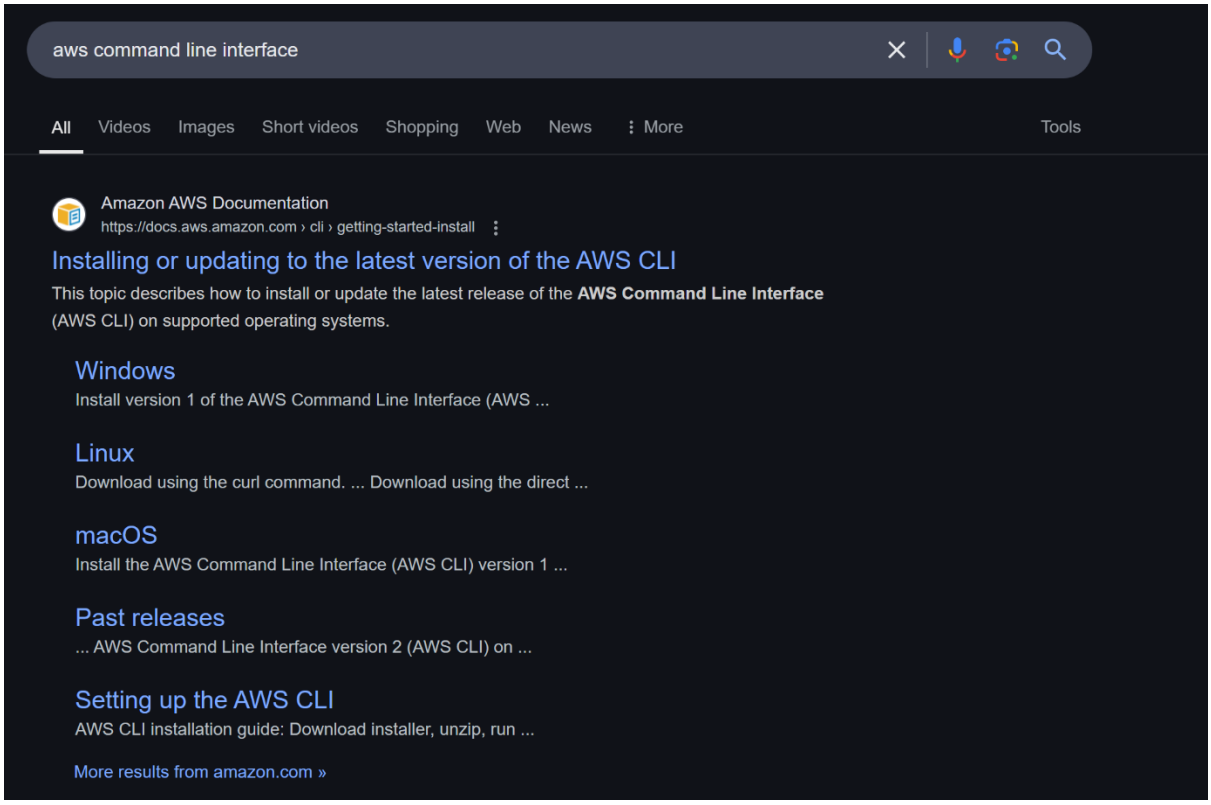
- **AWS Toolkit (optional):** For AWS integration, you could also install the AWS Toolkit for VS Code, but it's not required for Terraform usage.

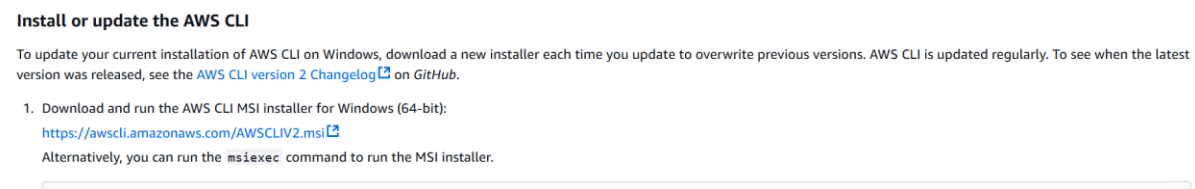
After installing, your Terraform files (*.tf) will have proper color coding and validation. This helps catch errors early and provides quick snippets and documentation while writing code.

13. Install the AWS CLI (Windows)

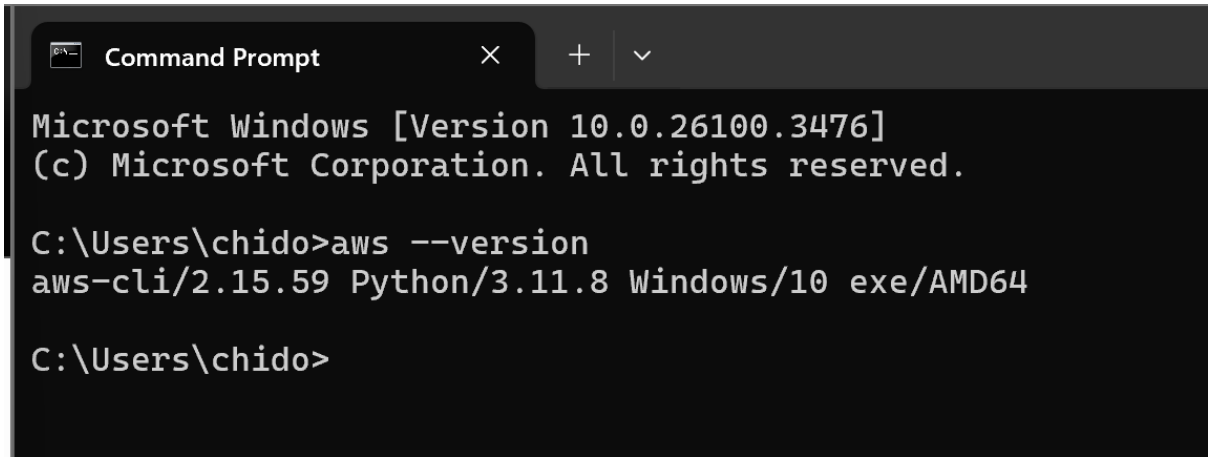
Terraform will need AWS credentials to provision resources. The AWS Command Line Interface (CLI) helps manage those credentials and lets you test connectivity:

- **Download AWS CLI:** Go to AWS documentation for CLI version 2 (the latest). Download the Windows 64-bit installer (MSI). For example, from AWS docs: **AWSCLIV2.msi** link





- **Run Installer:** Launch the downloaded MSI and proceed through the installation wizard (it's straightforward – accept terms, install). It will install AWS CLI to your Program Files.
- **Verify Installation:** Open a new Command Prompt and run aws --version. You should get output like aws-cli/2.15.59 Python/3.11.8 Windows/10 exe/AMD64 (version numbers may differ).
  Screenshot: Verifying AWS CLI installation with aws --version output in CMD.

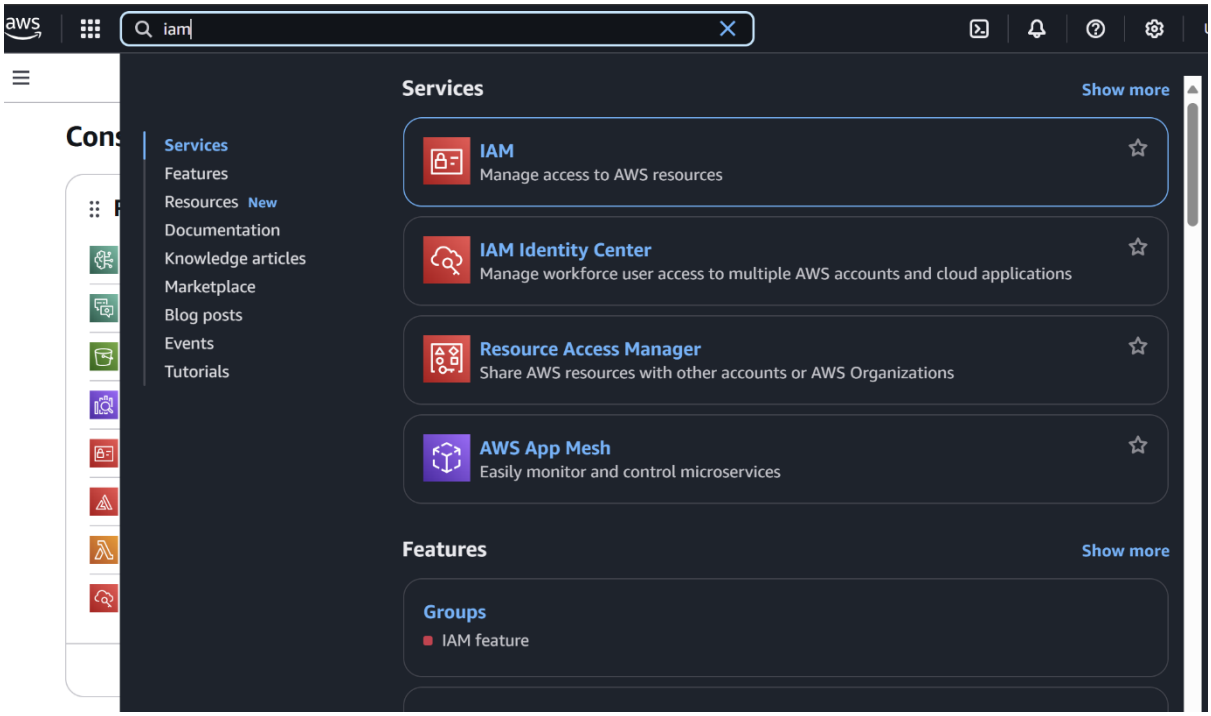


With AWS CLI installed, we can generate access keys and configure our credentials. The CLI will also be useful to quickly test AWS connectivity or perform AWS tasks outside Terraform if needed.

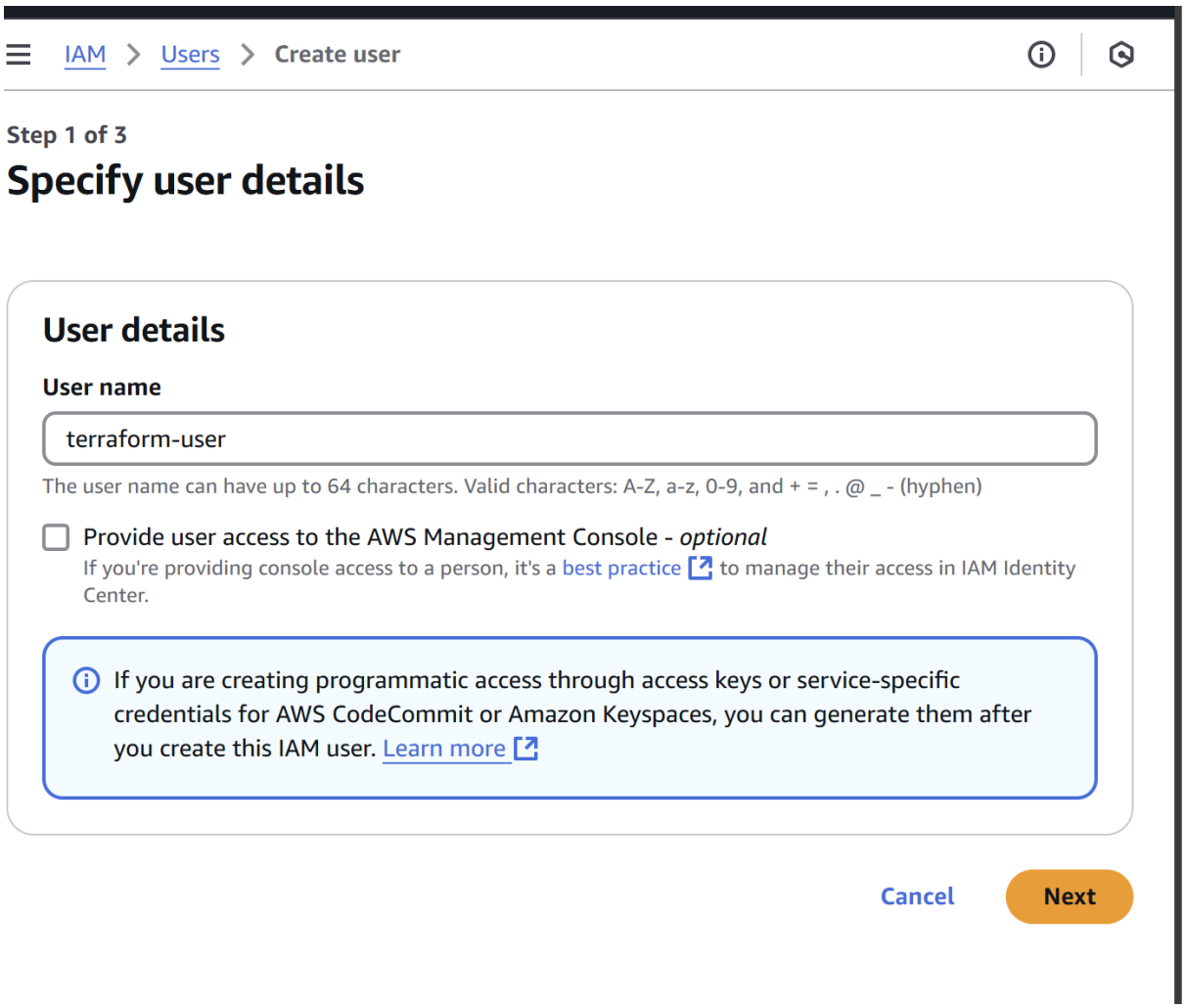
14. Creating an IAM User in AWS

It's a **best practice** to use a separate IAM user for Terraform rather than your root AWS account. We'll create a new IAM user with programmatic access:

1. **Open IAM Console:** Log in to the AWS Management Console, and search for "IAM". Click on the IAM service (Identity and Access Management).

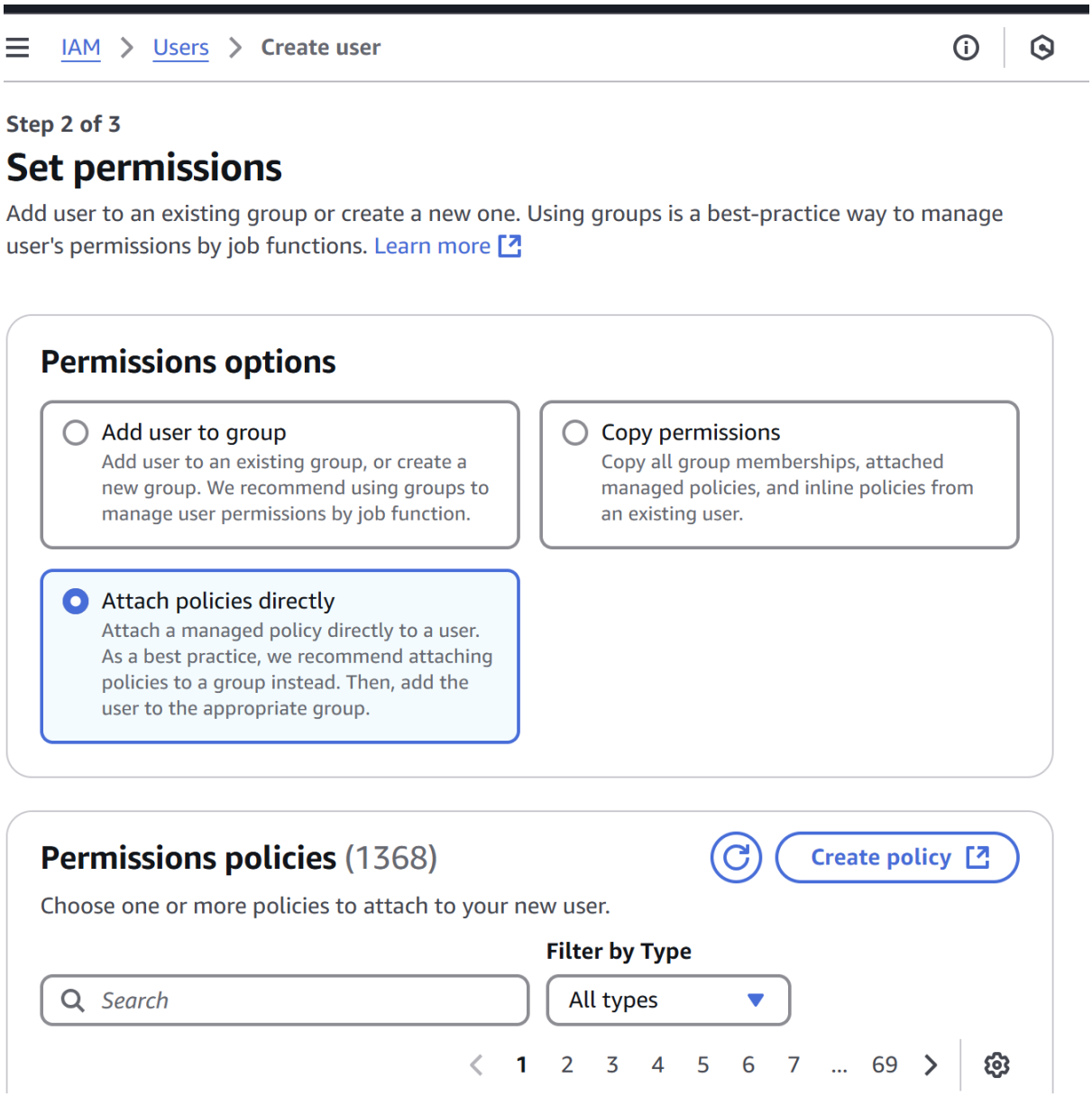


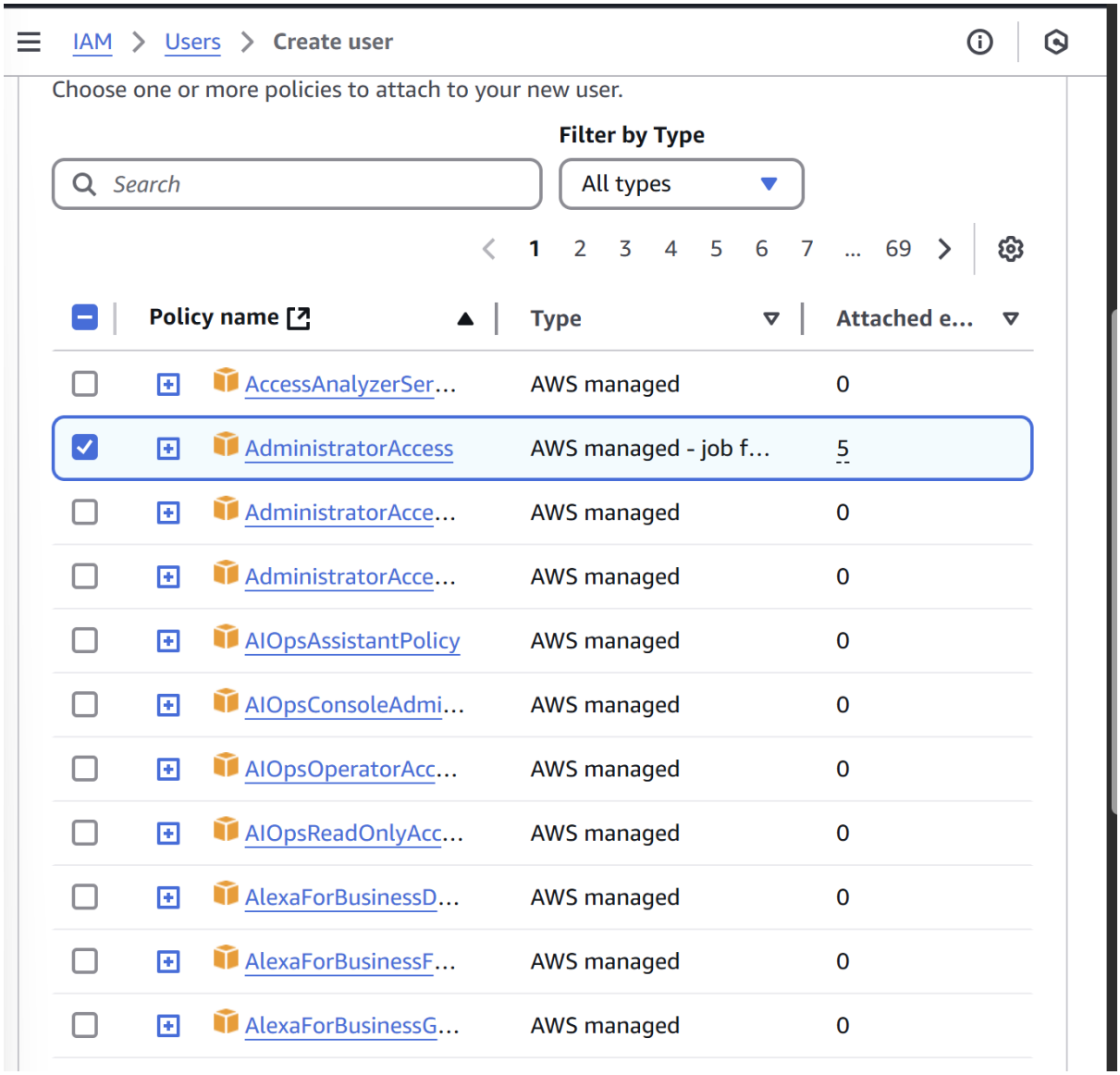
2. **Add User:** In the IAM dashboard, click **Users** in the sidebar, then **Add users**.
3. **User Details:** Provide a username, e.g., **terraform-user** (or terraform-user1 as per the screenshot). Ensure **"Provide user access to the AWS Management Console – optional"** is **unchecked** (we only need programmatic access).
   Screenshot: Creating a user terraform-user (console access not provided). Only programmatic access will be configured via access keys.



4. **Permissions:** For simplicity, attach an **AdministratorAccess** policy to this user (this gives full access to AWS resources, which is okay for learning purposes, but in a real

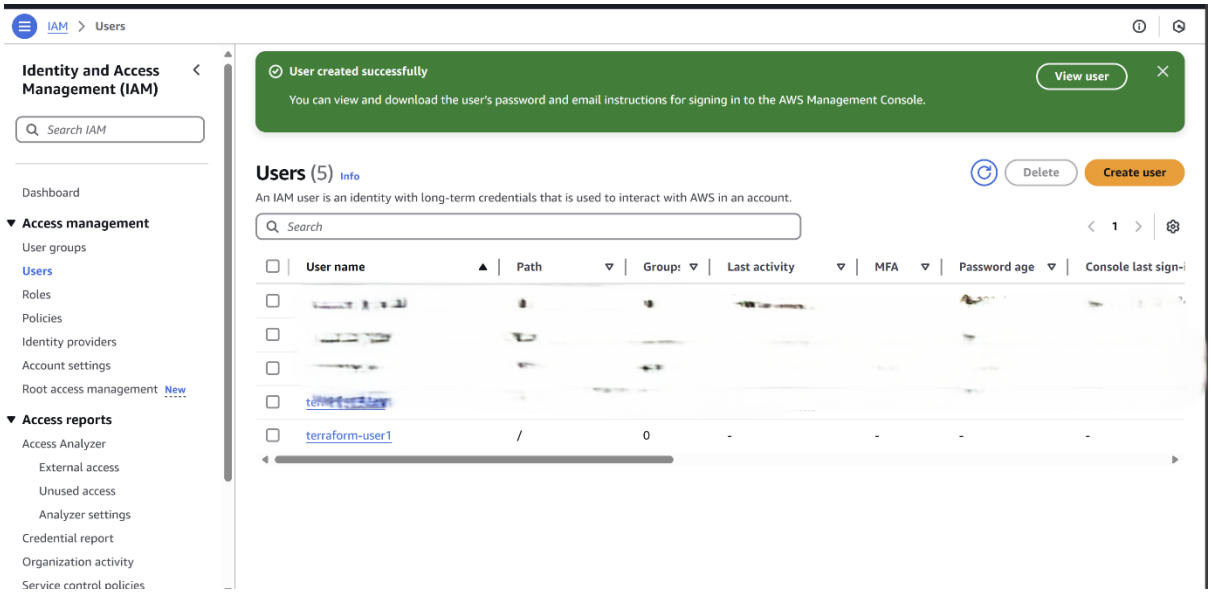
environment, you'd use least privilege). On the "Set permissions" step, choose **Attach policies directly**, search for **AdministratorAccess**, and check it.





5. **Skip Tags:** You can skip adding tags (optional). Proceed to **Review** and then **Create user**.
6. **User Created:** AWS will show a success message. You should see the new user in the list.

   Screenshot: IAM console showing "User created successfully" and listing terraform-user1 among users.



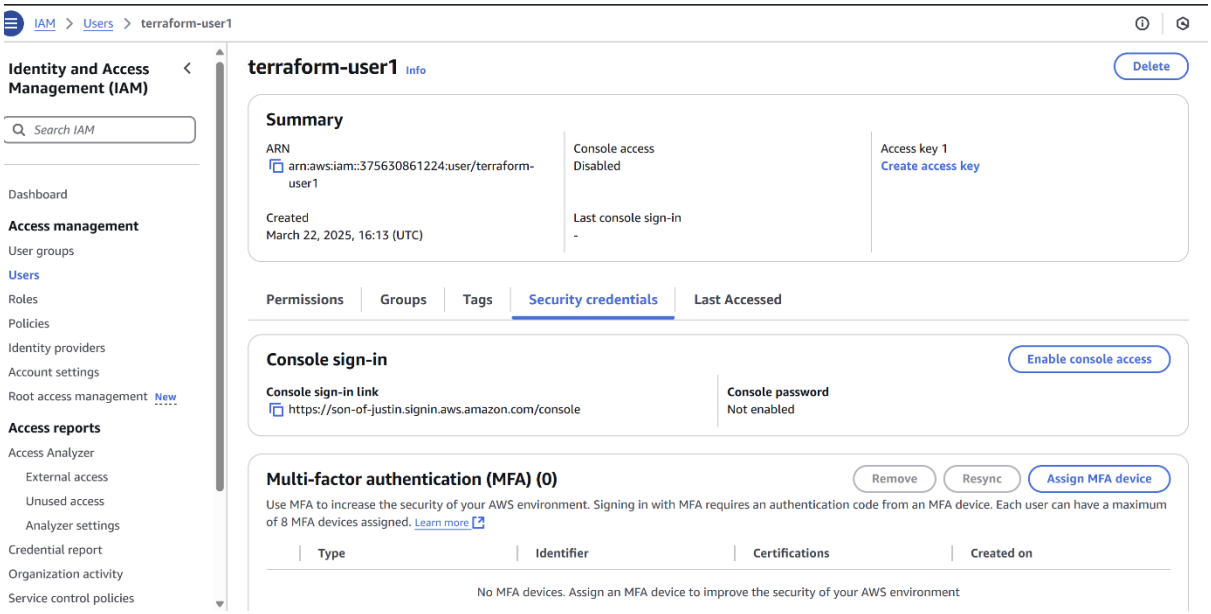
Now we have an IAM user for Terraform. Next, we'll create an access key (ID and secret) for this user to use with AWS CLI and Terraform.

15. Generating an Access Key for the IAM User

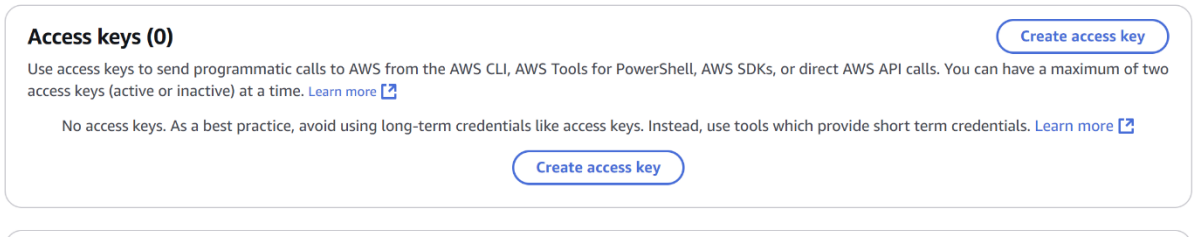
After creating the IAM user, generate an **Access Key ID and Secret Access Key** (these are the credentials Terraform will use):

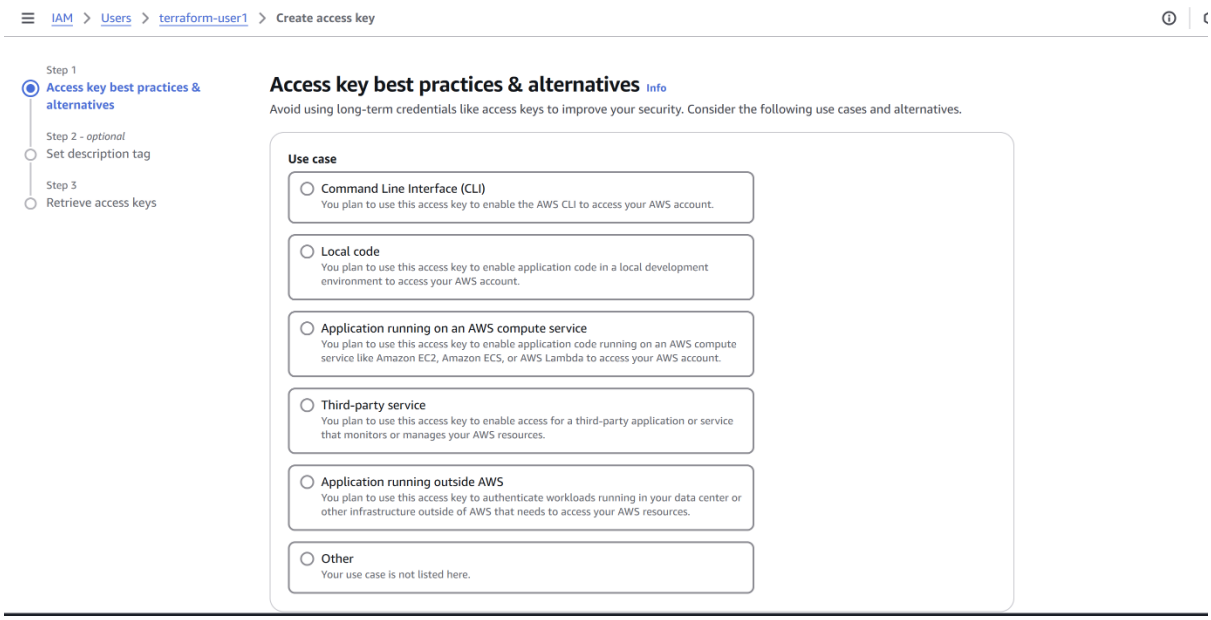
1. **Open User Security Settings:** Click on your newly created user in the IAM console to view its details. Navigate to the **Security credentials** tab.

   Screenshot: IAM user summary with a section for Access keys (none created yet). The "Create access key" option is visible.



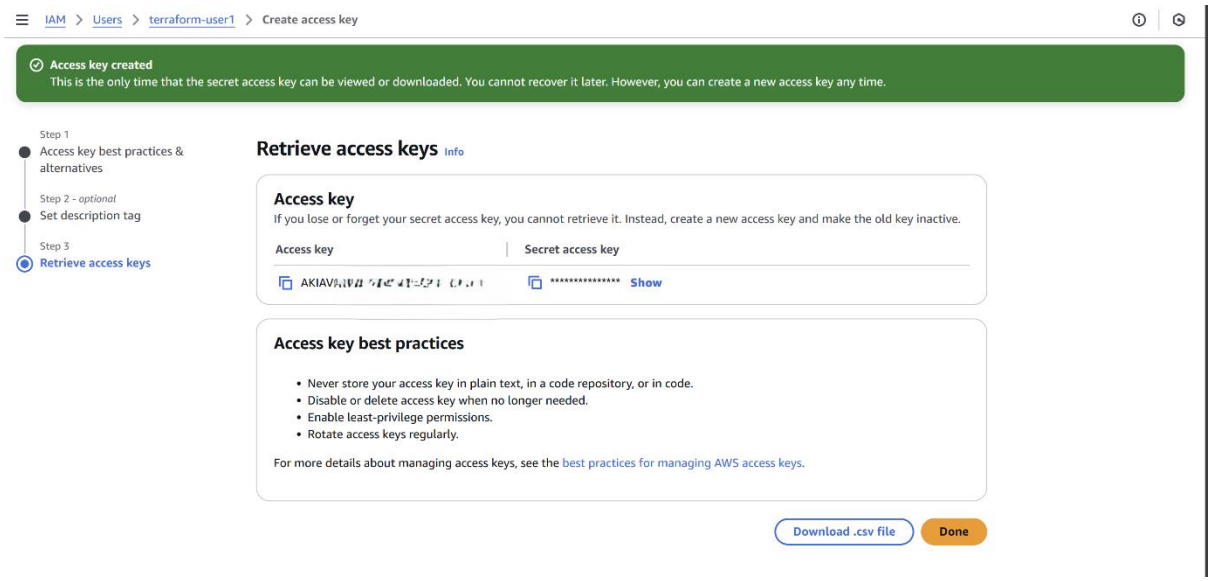
2. **Create Access Key:** In the **Access keys** section, click **Create access key**. AWS will prompt with a wizard about best practices:
    - **Select Use Case:** Choose **Command Line Interface (CLI)** as the use case (since Terraform will use CLI calls).

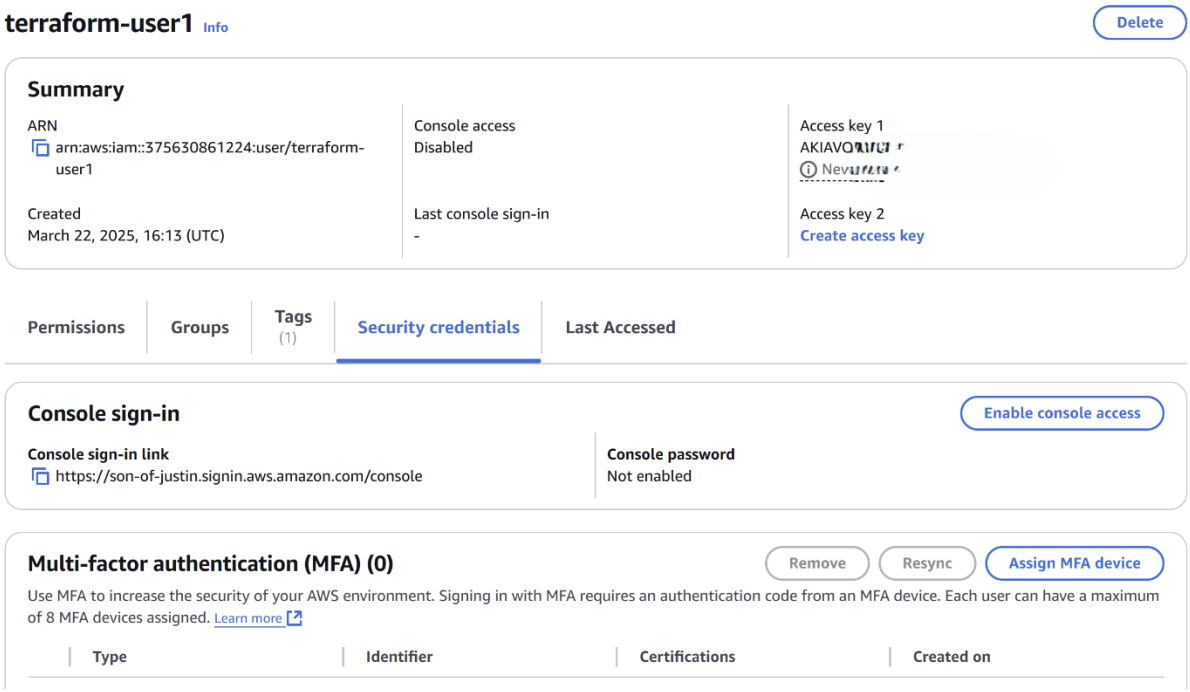




- **(Optional Tag)**: You can add a tag or description (or skip).
- **Retrieve Keys:** AWS will now show you the **Access Key ID** and **Secret Access Key** one time. **Copy these values** or download the CSV file. *This is the only time you can see the secret!*

  Screenshot: Access key ID (AKIA... format) is shown and secret key is masked (you could click "Show" to view it). A download .csv option is provided.





3. **Store the Keys Securely:** Save the Access Key ID and Secret Access Key to a safe place (you'll need them in the next step). Treat the secret key like a password – don't share it or commit it to code.

We now have programmatic credentials. Let's configure the AWS CLI (and Terraform) to use them.

16. Create a Named Profile (AWS CLI Configure)

Using the AWS CLI, configure a profile for Terraform with the new access keys. This lets Terraform (and you) use a profile name instead of embedding keys in code:

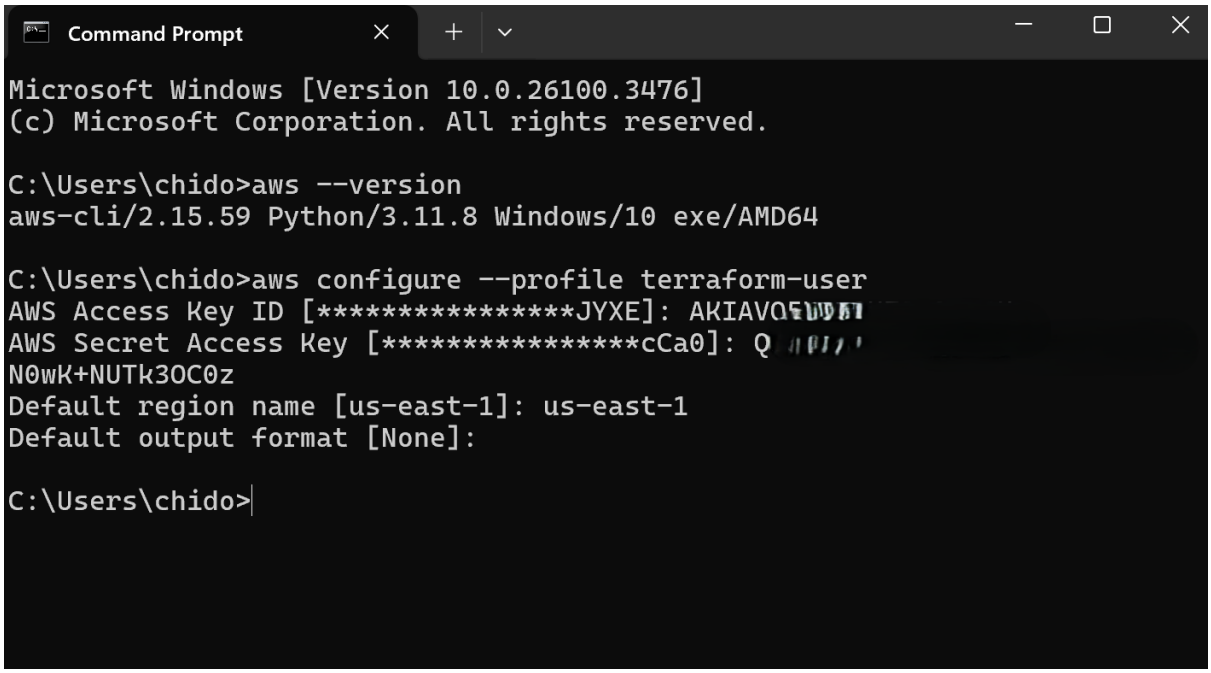
- **AWS Configure:** Open Command Prompt and run:

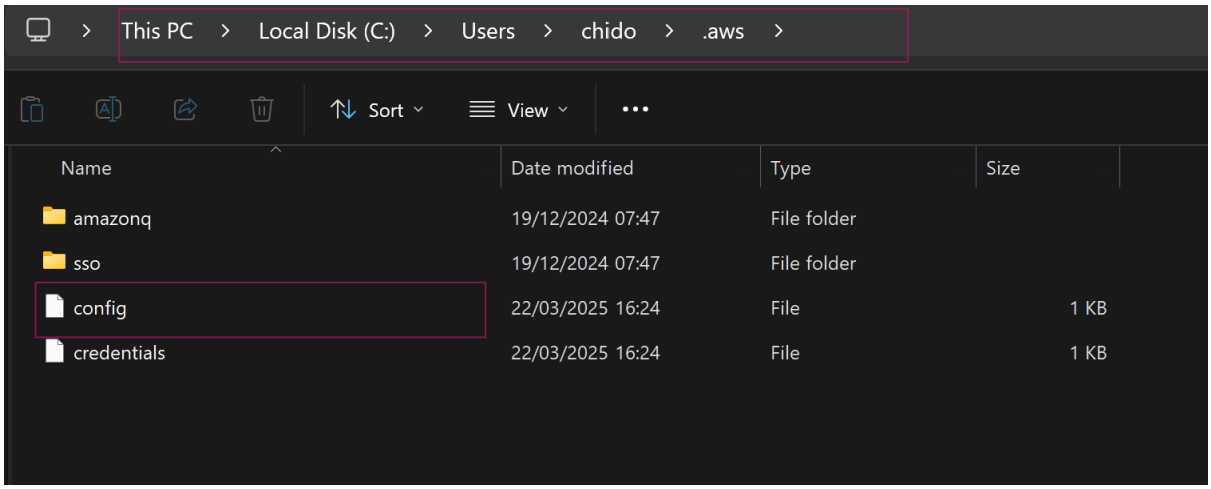
  aws configure --profile terraform-user

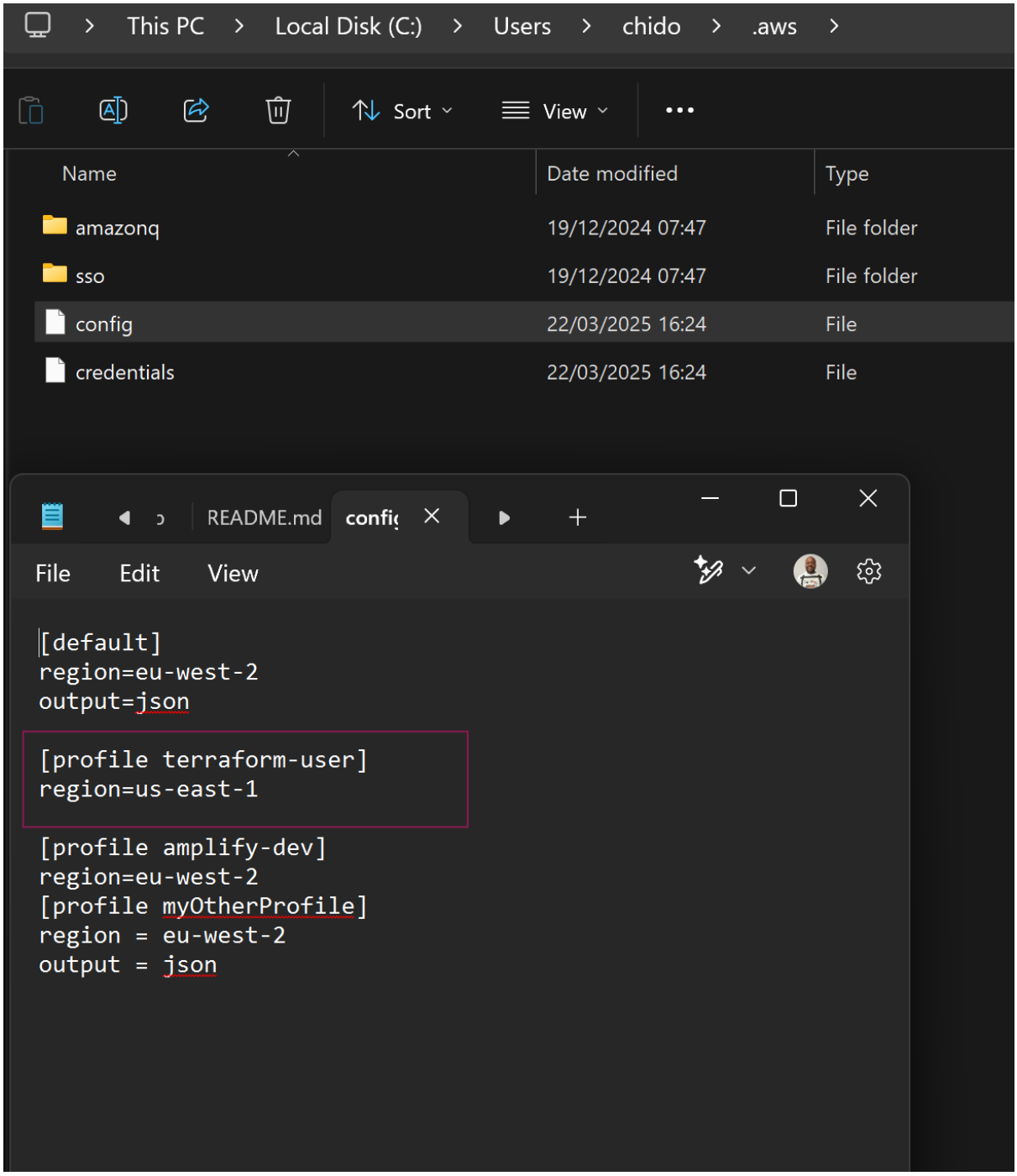
  When prompted:

  - Enter the **Access Key ID** (from previous step).
  - Enter the **Secret Access Key**.
  - Default region name: choose your desired region (e.g., **us-east-1** or the region you want to create resources in, such as **eu-west-1**). Make sure it matches where you intend to host the infrastructure.
  - Default output format: you can leave this blank or enter **json**.

- **Confirmation:** The CLI doesn't show output on success, but it creates/updates a profile in the AWS credentials file. You can verify by running:
  aws configure list --profile terraform-user (it should list the details you entered, except the secret which is hidden).
  Screenshot: The aws configure --profile terraform-user command inputting keys and setting region to us-east-1.





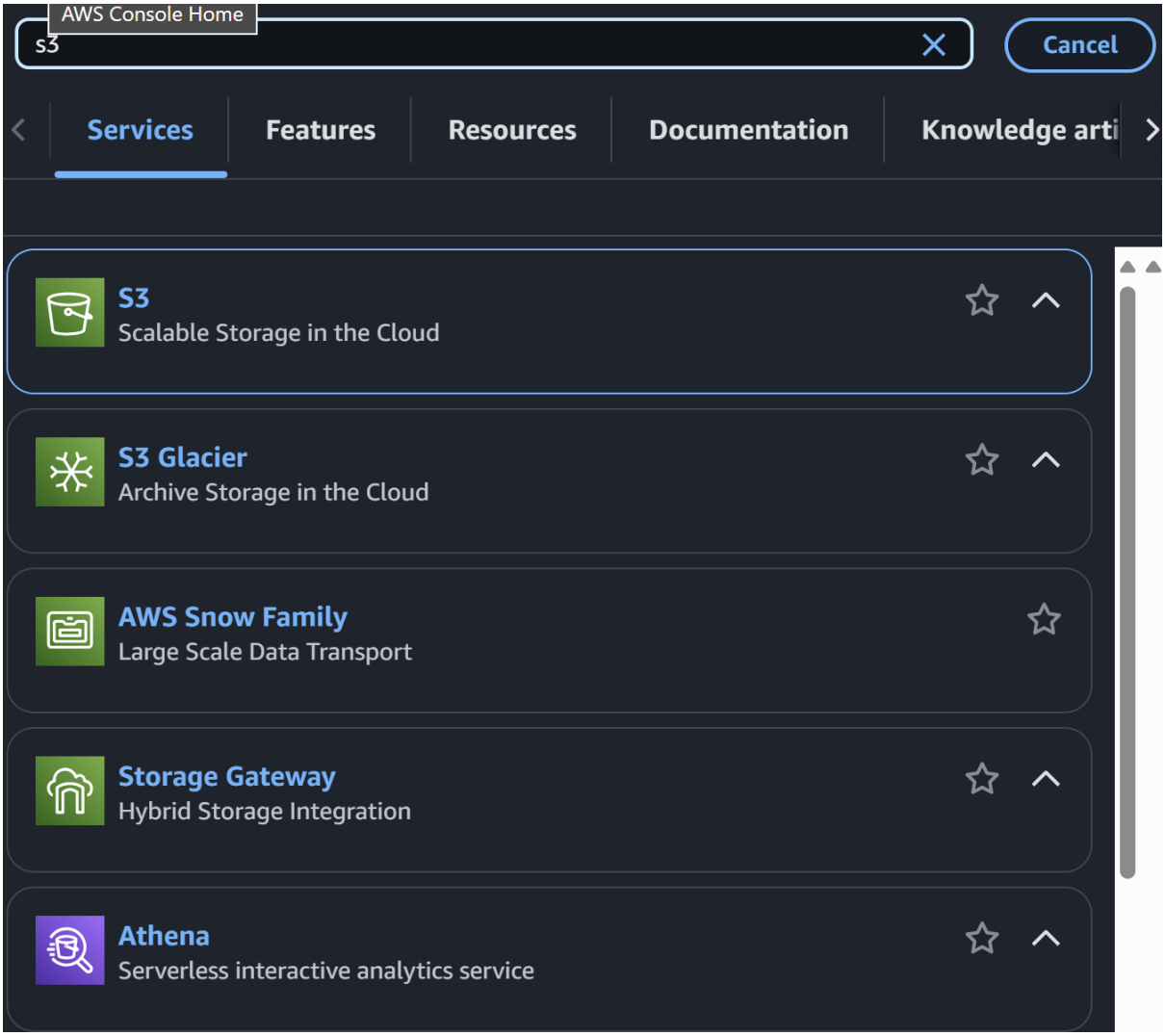


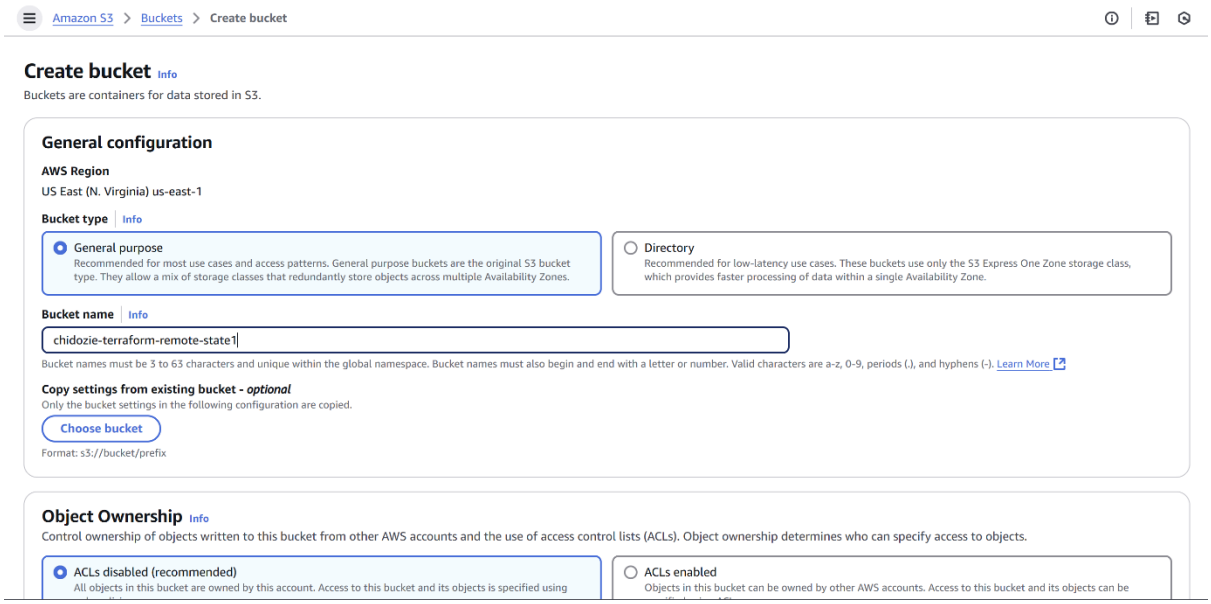
Now Terraform can reference the **terraform-user** profile when authenticating with AWS, instead of hardcoding credentials. We'll see this in the Terraform provider setup.

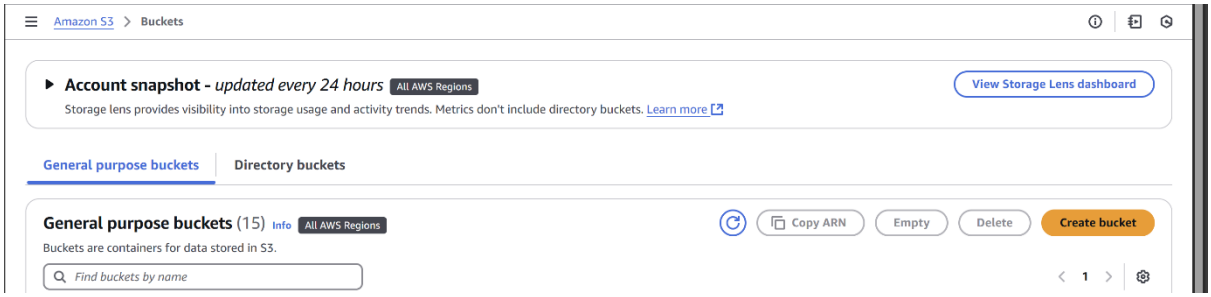
17. Create an S3 Bucket for Terraform State

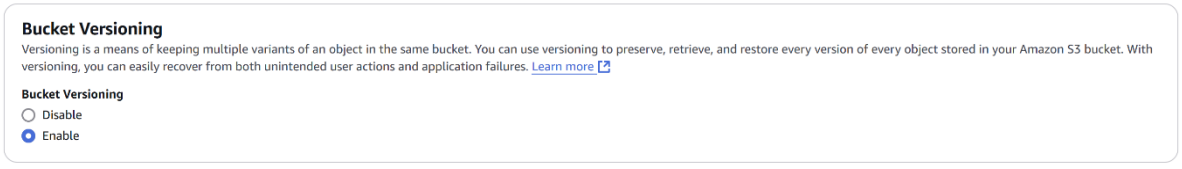
Terraform uses a **state file** to keep track of deployed resources. Storing state remotely (in an S3 bucket) is recommended for collaboration and back-up. We'll create an S3 bucket to hold the Terraform state:

- **Choose a Bucket Name:** Form a globally unique bucket name (e.g., *my-terraform-state-*). AWS bucket names must be unique across all AWS accounts, so add random text if needed.
- **Create Bucket (Console):** In AWS Console, go to the S3 service. Click **"Create bucket."** Enter the bucket name, choose the same region as your Terraform deployment region (e.g., us-east-1). You can leave other settings as default (Block Public Access should remain on). Create the bucket.
- **Bucket Ready:** Once created, you'll see it in the S3 list. No objects yet (we will let Terraform create the state file).



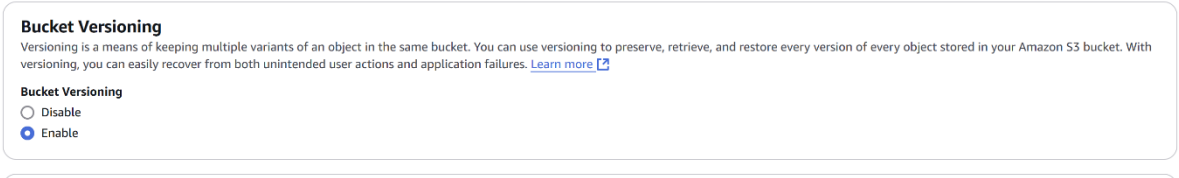






Alternatively, you could create the bucket via AWS CLI (aws s3 mb s3://<name>) or even Terraform itself. In this project, the bucket is for Terraform's own use, so it's often created beforehand.

- **Enable Versioning (Recommended):** For state file safety, enable versioning on the bucket (so old state files are preserved). In the bucket's **Properties**, turn Versioning **On**.



Now, note the bucket name and region. We will configure Terraform to use this bucket for storing state in the next step.

18. Writing Terraform Configurations for AWS Resources

Now the real Infrastructure as Code work begins. We will write Terraform syntax to create each needed AWS resource step by step. Each resource type (VPC, subnets, gateways, etc.) will be defined in a .tf file.

Key Terraform files in this project:

- **provider.tf / main.tf:** Configures the AWS provider and backend (state storage). We specify AWS profile (from step 16), region, and the S3 backend bucket for state.
- **vpc.tf:** Defines the VPC, subnets (public and private), internet gateway, and default route table for public subnets.
- **nat-gateway.tf:** Allocates Elastic IPs and creates NAT Gateways in the public subnets, plus corresponding route tables for private subnets to use the NAT.
- **security-group.tf:** Creates security groups (e.g., open web traffic on 80/443 for the web servers, allow DB access from app servers, etc.).
- **rds.tf:** Sets up an Amazon RDS MySQL database instance in a private subnet.
- **alb.tf:** Defines an Application Load Balancer (with target groups and listeners for our auto-scaled web servers).
- **autoscaling.tf:** Sets up an Auto Scaling Group with Launch Template to create EC2 instances (our web app servers) across subnets, and attaches them to the ALB target group.
- **sns.tf:** Creates an SNS topic (for notifications, perhaps to get alerts when instances scale).
- **route53.tf:** Creates a DNS record in Route 53 (assuming a hosted zone exists) to point a domain to the ALB.
- **outputs.tf:** Defines Terraform output values (like the website URL or ALB DNS, DB endpoint, etc., to easily see after apply).

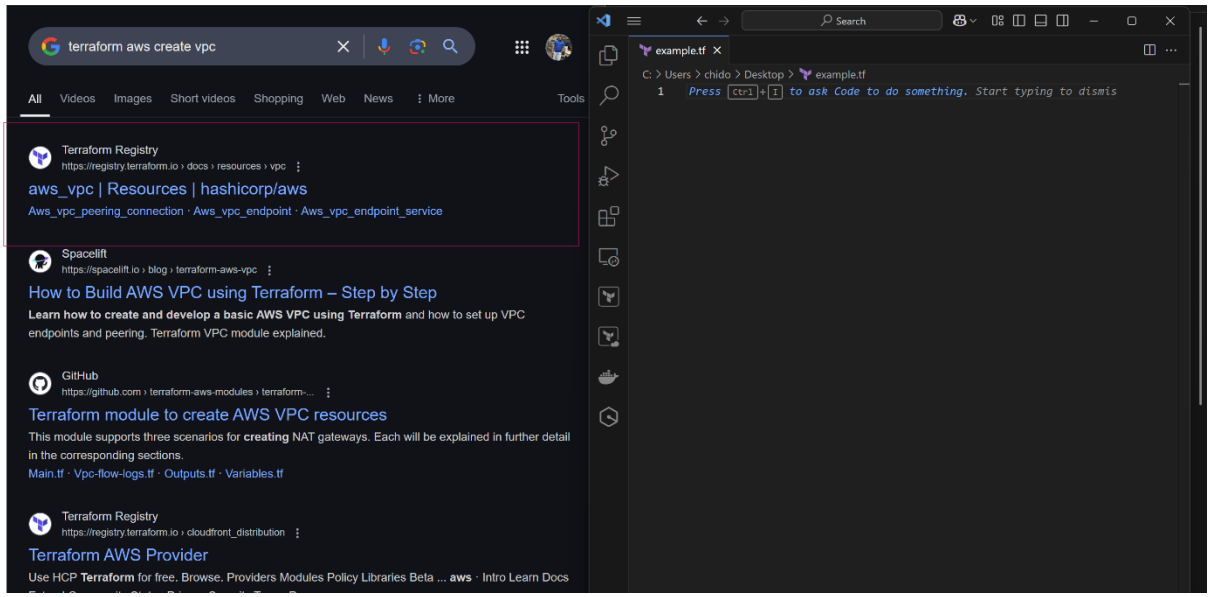
We also use a **variable.tf** to define input variables (like instance AMI ID, instance size, DB name/password, etc.) which can be customized without changing code.

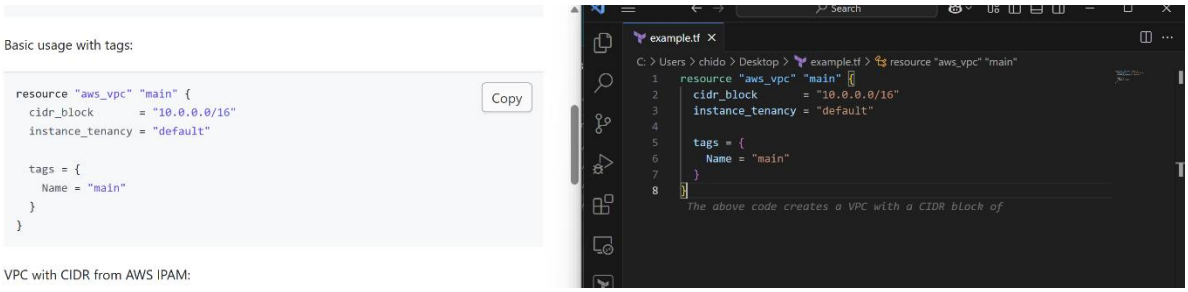
When writing these configurations, a common approach is:

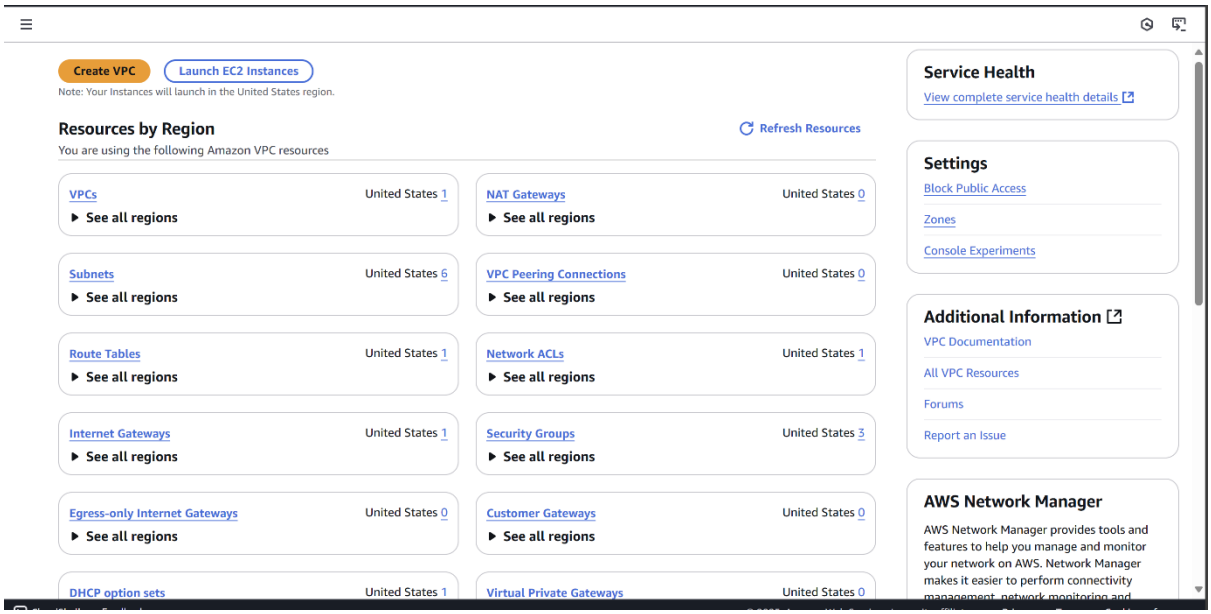
- Start with one resource (say, VPC), run terraform apply to create it, then continue adding resources incrementally.

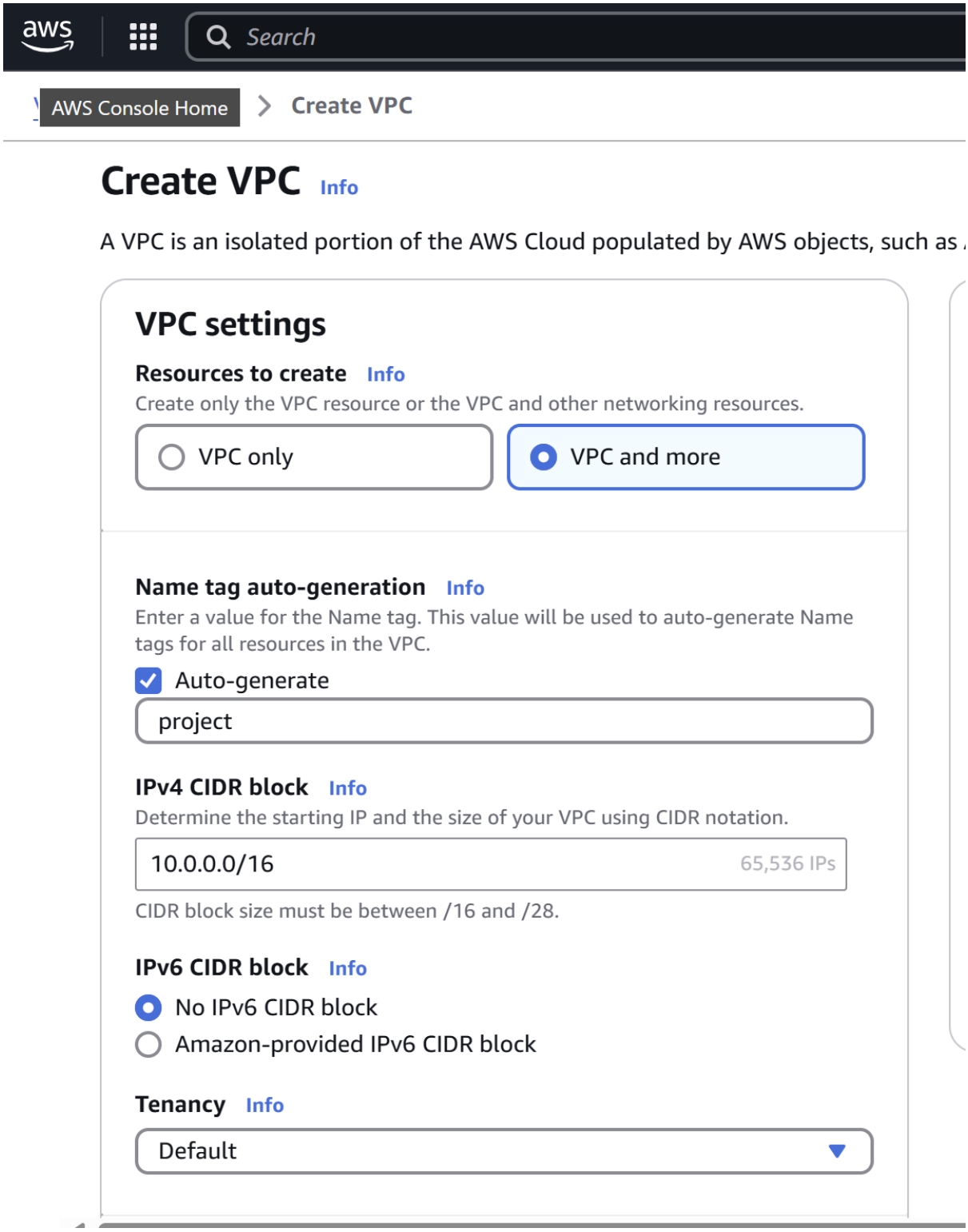
- Use Terraform documentation to find resource syntax: For example, aws_vpc requires a CIDR block, etc.

The example below shows how the terraform uses real structure of the aws service like vpc to write its infrastructure as code







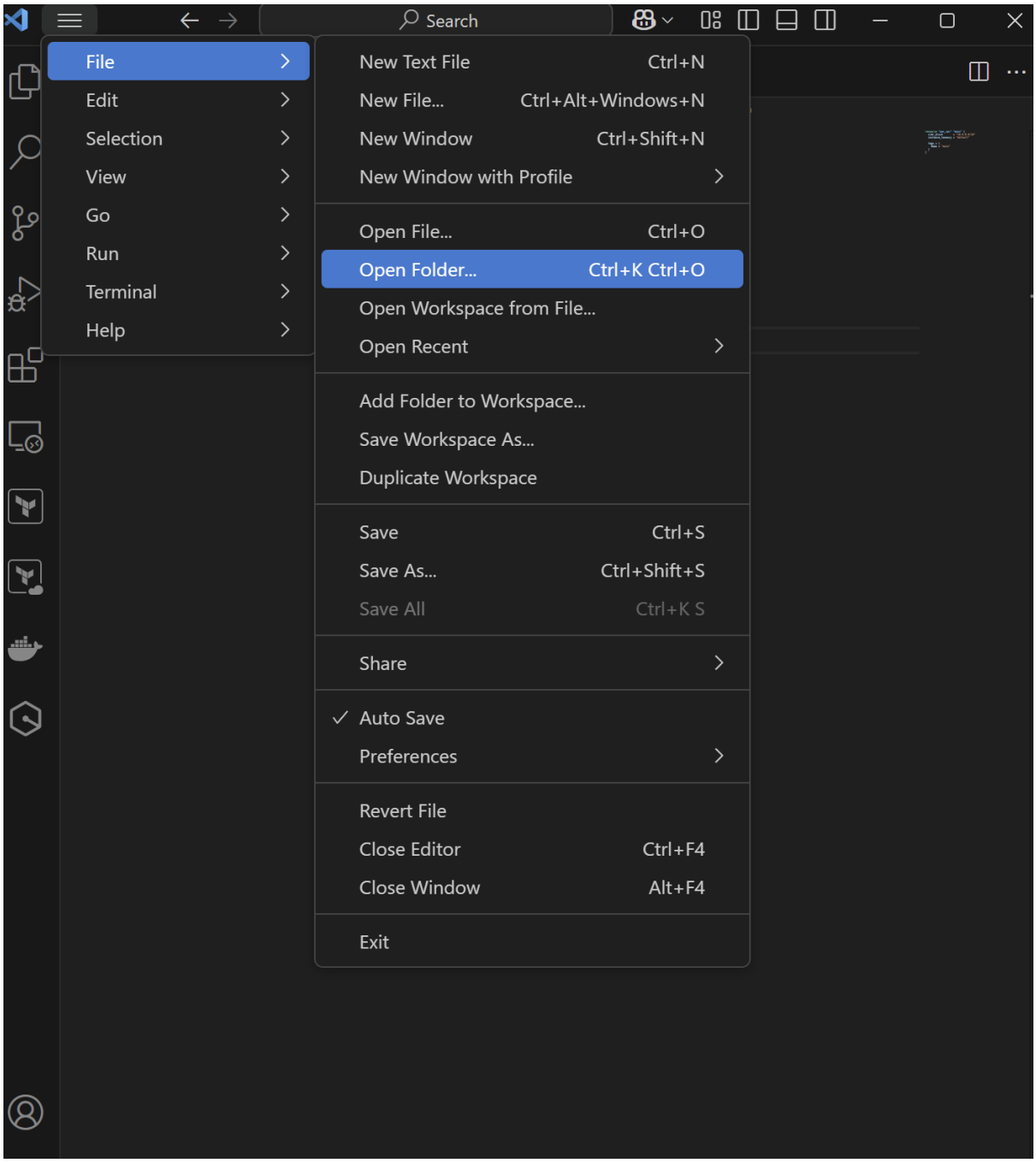


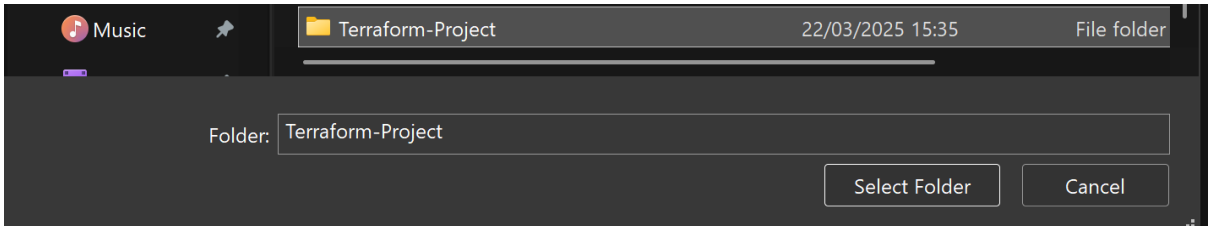
- Reference resources by name to connect them (e.g., subnet IDs in other resources).

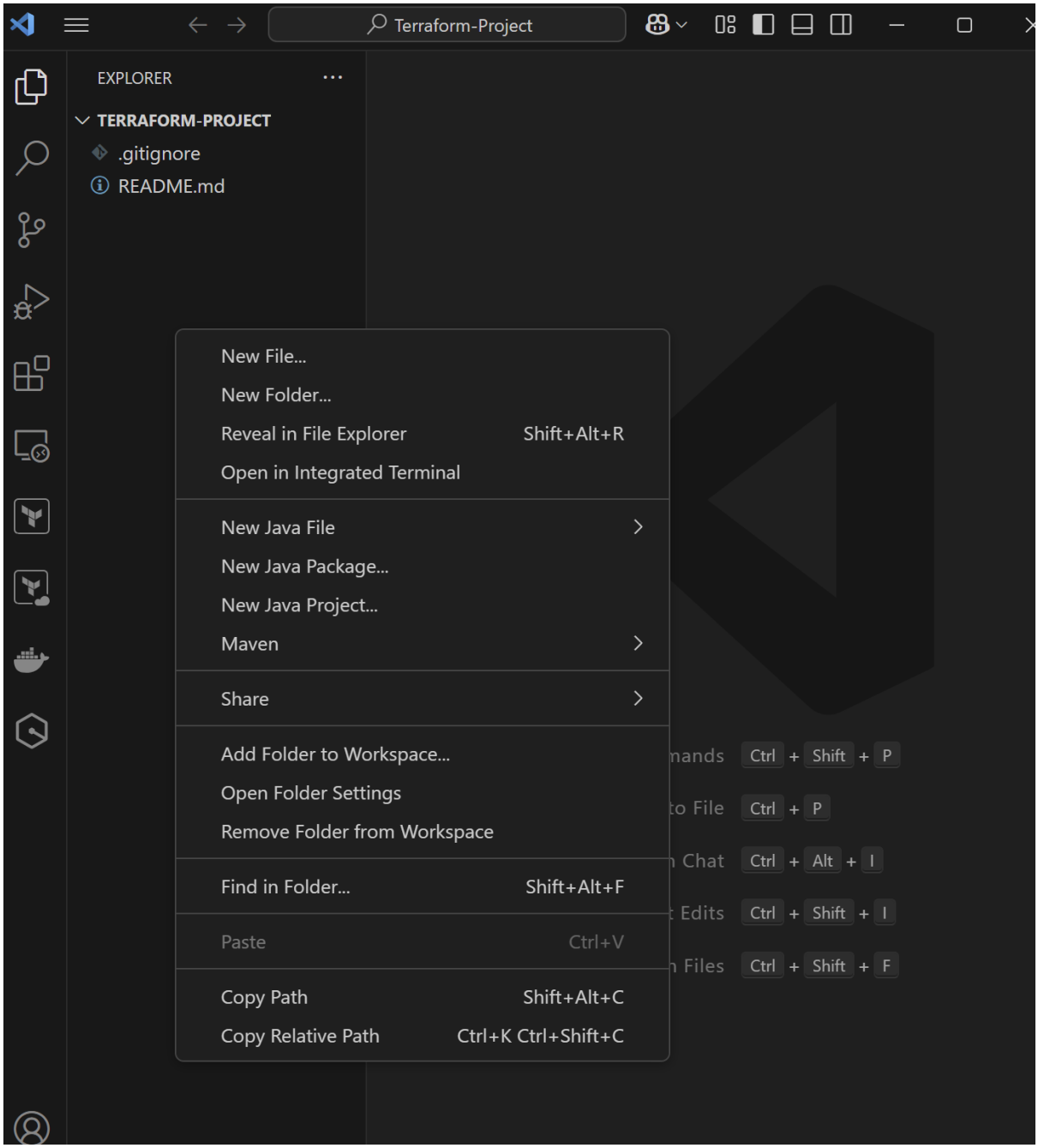
In our project, the code is already written, but conceptually, each block in the .tf files corresponds to an AWS resource:

19. Authenticate Terraform with AWS

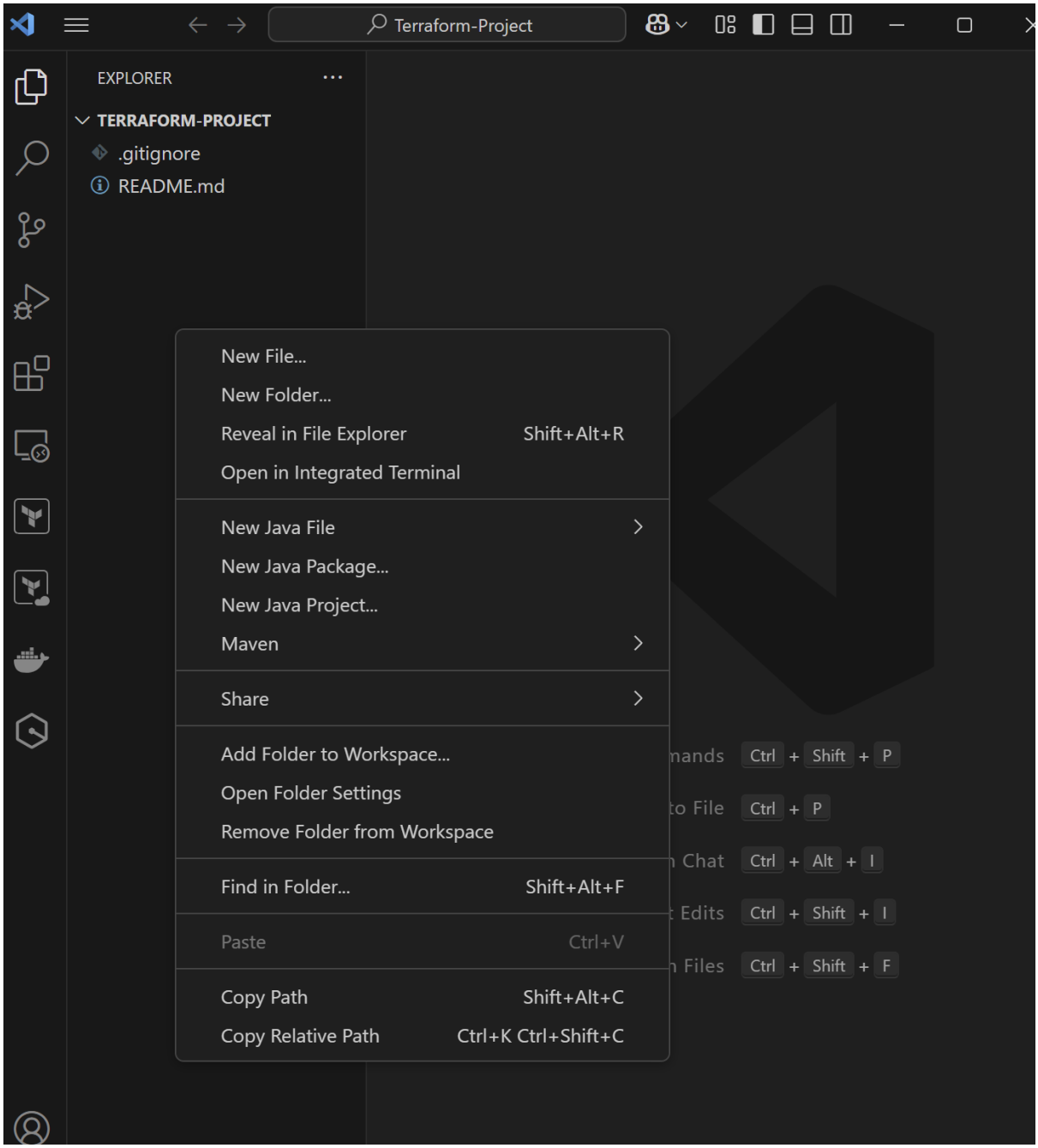
Before provisioning, we will first open our terraform folder in VS code.

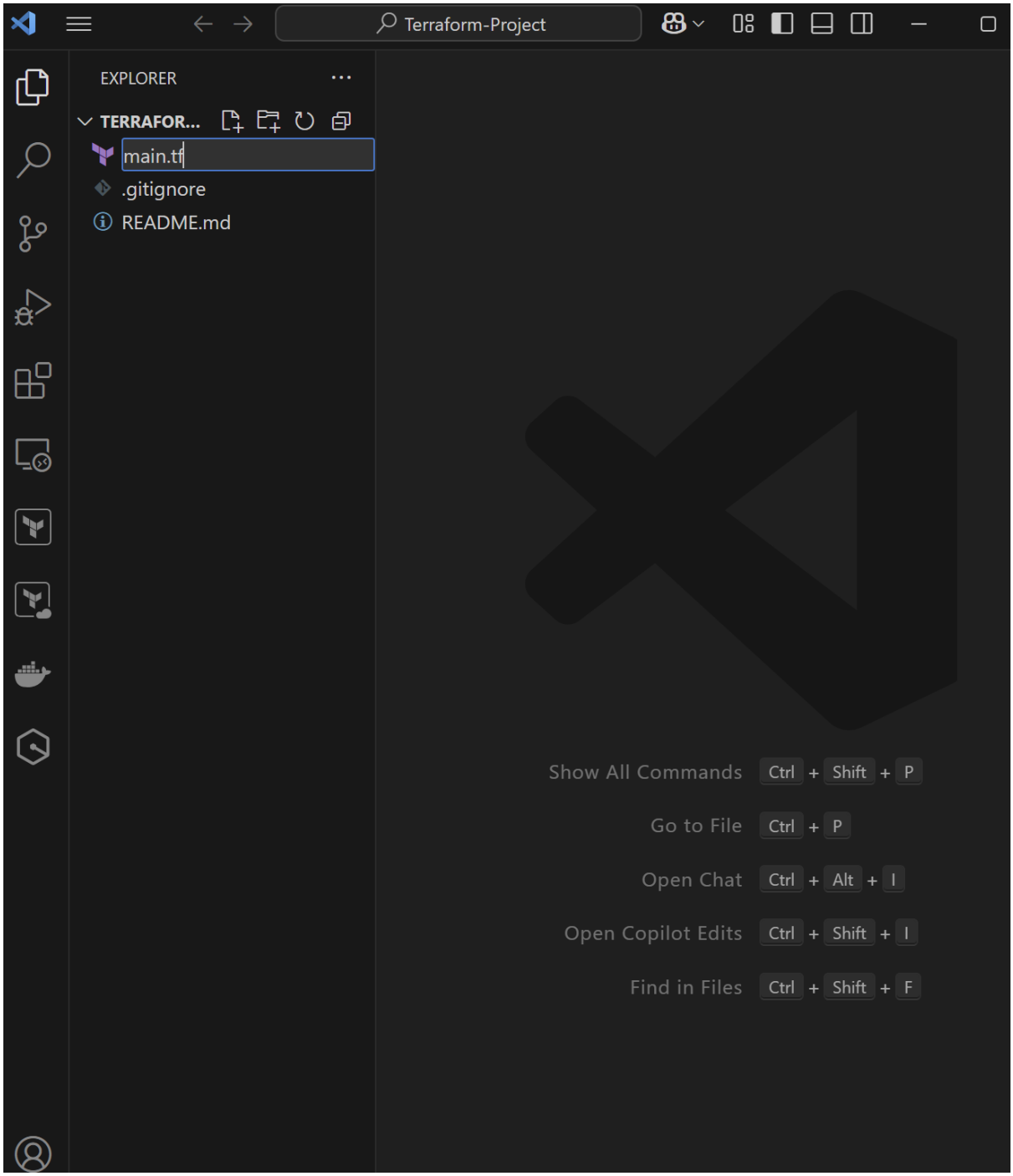




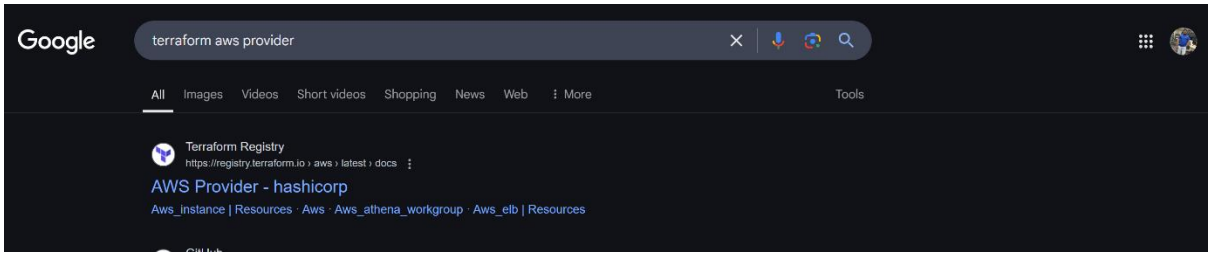


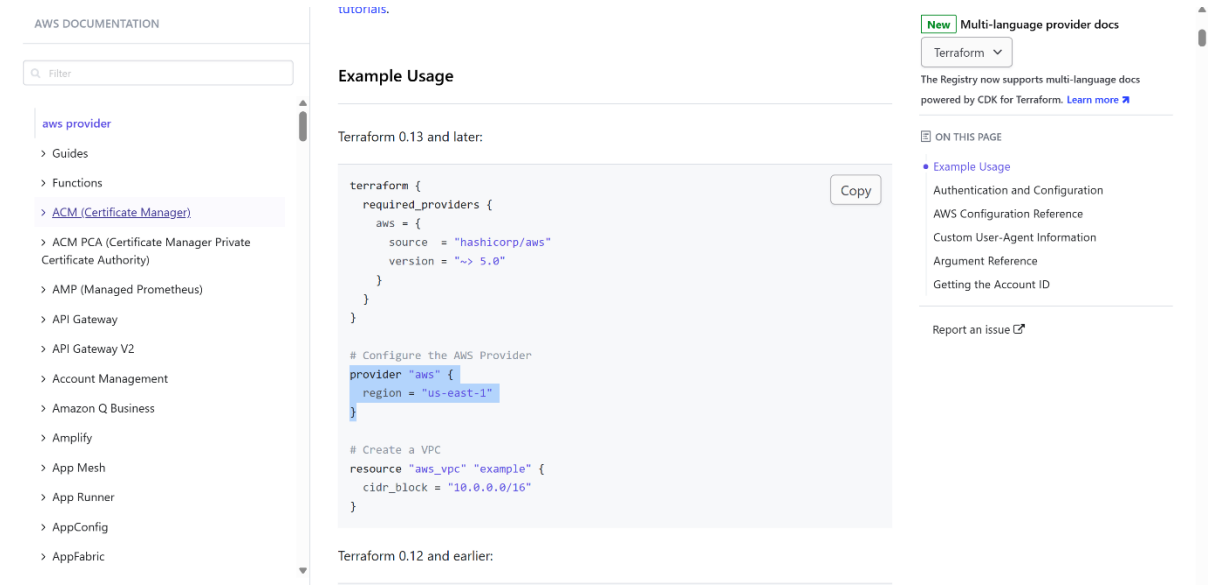
Create a main.tf file inside our folder.

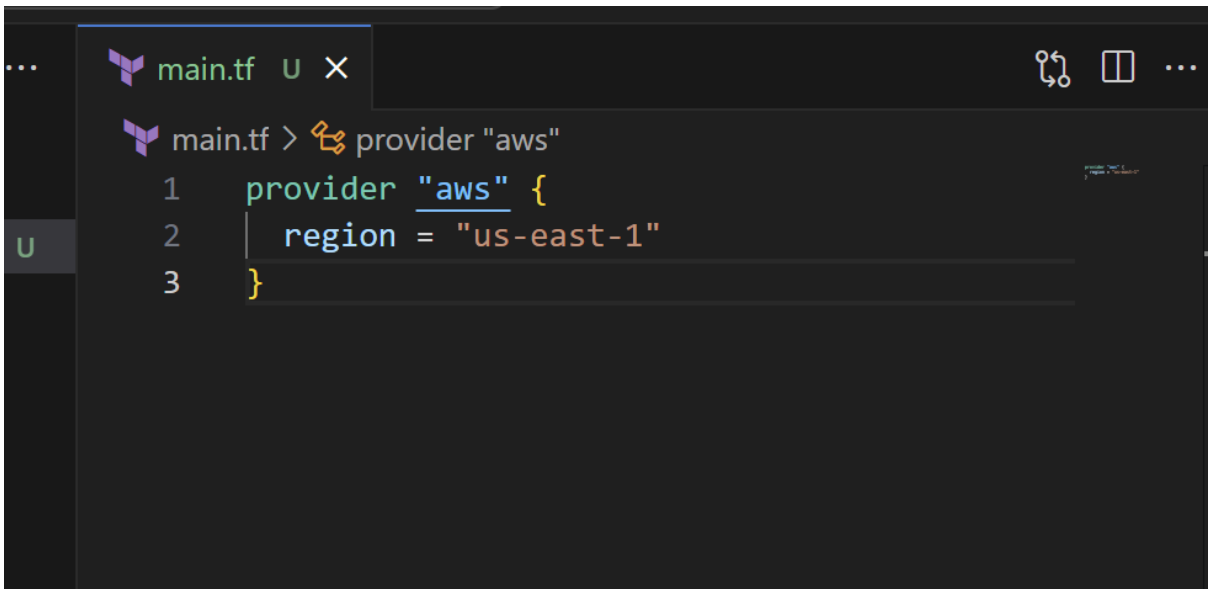

Terraform-Project
EXPLORER
TERRAFORM-PROJECT
.gitignore
README.md
New File...
New Folder...
Reveal in File Explorer Shift+Alt+R
Open in Integrated Terminal
New Java File
New Java Package...
New Java Project...
Maven
Share
Add Folder to Workspace...
Open Folder Settings
Remove Folder from Workspace
Find in Folder... Shift+Alt+F
Paste Ctrl+V
Copy Path Shift+Alt+C
Copy Relative Path Ctrl+K Ctrl+Shift+C
mands Ctrl + Shift + P
to File Ctrl + P
Chat Ctrl + Alt + I
Edits Ctrl + Shift + I
Files Ctrl + Shift + F

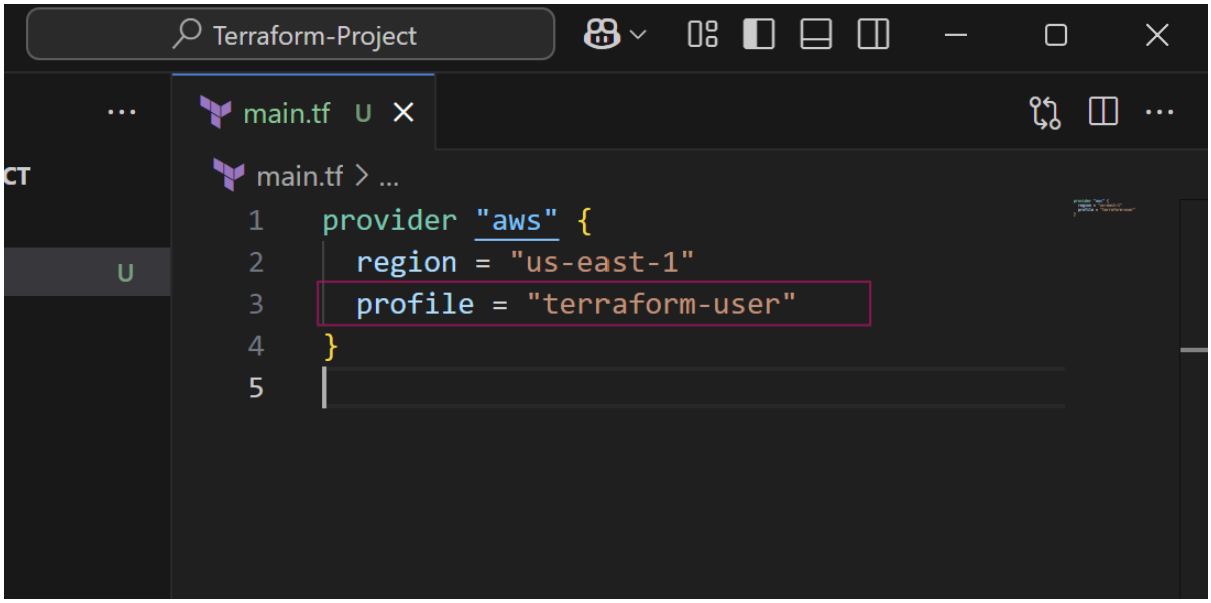


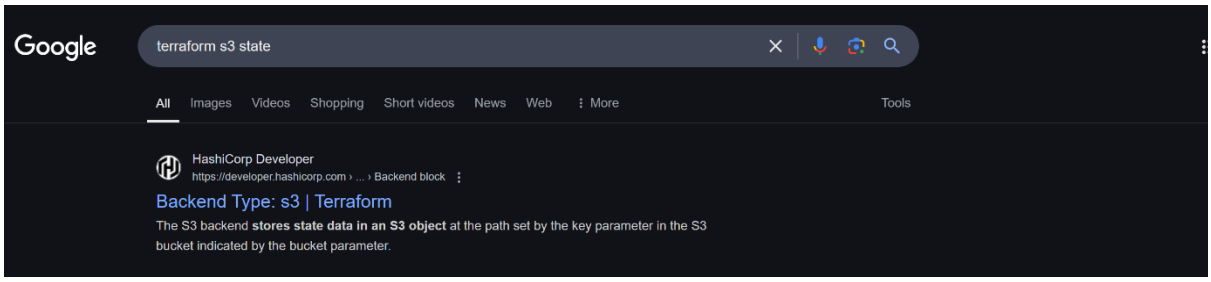
- **Backend Configuration:** In main.tf (or a dedicated backend.tf), configure the backend to set our provider which is AWS, our profile (which we have set up already) and S3. We will do this using the terraform documentation as shown below:

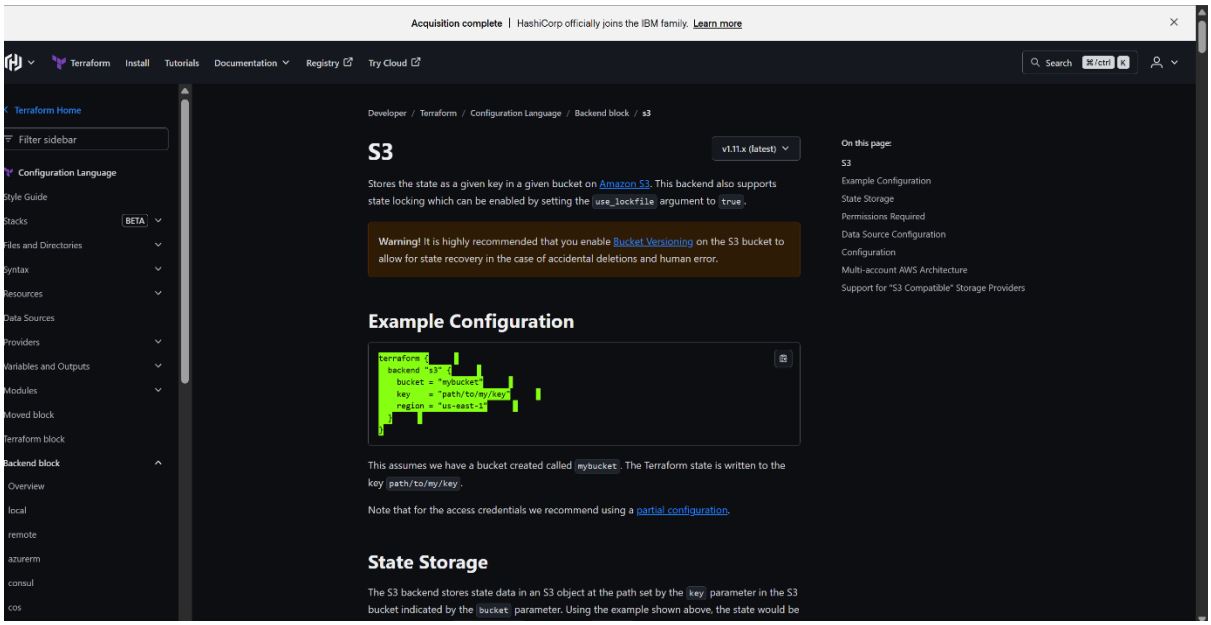

Google
terraform aws provider
All
Images
Videos
Short videos
Shopping
News
Web
More
Tools
Terraform Registry
https://registry.terraform.io › aws › latest › docs
AWS Provider - hashicorp
Aws_instance | Resources · Aws · Aws_athena_workgroup · Aws_elb | Resources


AWS DOCUMENTATION
Filter
aws provider
Guides
Functions
ACM (Certificate Manager)
ACM PCA (Certificate Manager Private Certificate Authority)
AMP (Managed Prometheus)
API Gateway
API Gateway V2
Account Management
Amazon Q Business
Amplify
App Mesh
App Runner
AppConfig
AppFabric
tutorials.
Example Usage
Terraform 0.13 and later:
Copy
terraform {
required_providers {
aws = {
source = "hashicorp/aws"
version = "~> 5.0"
}
}
}
# Configure the AWS Provider
provider "aws" {
region = "us-east-1"
}
# Create a VPC
resource "aws_vpc" "example" {
cidr_block = "10.0.0.0/16"
}
Terraform 0.12 and earlier:
New Multi-language provider docs
Terraform
The Registry now supports multi-language docs powered by CDK for Terraform. Learn more
ON THIS PAGE
Example Usage
Authentication and Configuration
AWS Configuration Reference
Custom User-Agent Information
Argument Reference
Getting the Account ID
Report an issue


main.tf U
main.tf > provider "aws"
1 provider "aws" {
2 region = "us-east-1"
3 }
U


Terraform-Project
main.tf U
main.tf > ...
1 provider "aws" {
2 region = "us-east-1"
3 profile = "terraform-user"
4 }
5


Google
terraform s3 state
All Images Videos Shopping Short videos News Web More Tools
HashiCorp Developer
https://developer.hashicorp.com › ... › Backend block
Backend Type: s3 | Terraform
The S3 backend stores state data in an S3 object at the path set by the key parameter in the S3 bucket indicated by the bucket parameter.

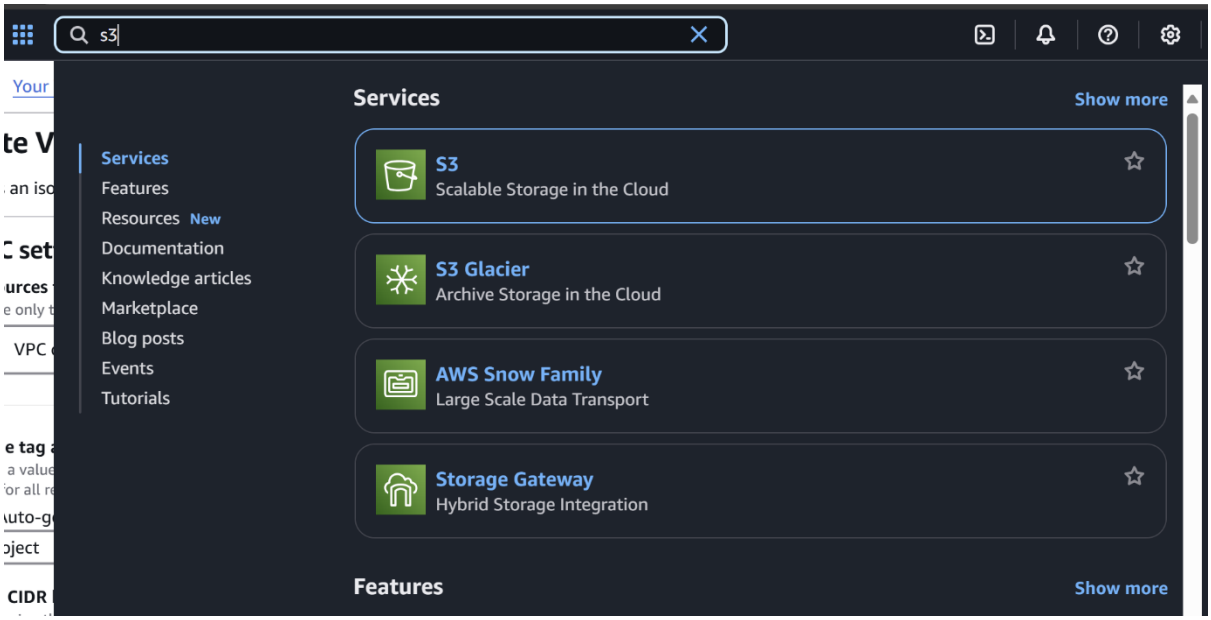

Acquisition complete | HashiCorp officially joins the IBM family. Learn more
Terraform Install Tutorials Documentation Registry Try Cloud
Search
Terraform Home
Filter sidebar
Configuration Language
Backend block
Overview
local
remote
azurerm
consul
cos
Developer / Terraform / Configuration Language / Backend block / s3
S3
v1.11.x (latest)
Stores the state as a given key in a given bucket on Amazon S3. This backend also supports state locking which can be enabled by setting the use_lockfile argument to true.
Warning! It is highly recommended that you enable Bucket Versioning on the S3 bucket to allow for state recovery in the case of accidental deletions and human error.
Example Configuration
terraform {
backend "s3" {
bucket = "mybucket"
key = "path/to/my/key"
region = "us-east-1"
}
}
This assumes we have a bucket created called mybucket. The Terraform state is written to the key path/to/my/key.
Note that for the access credentials we recommend using a partial configuration.
State Storage
The S3 backend stores state data in an S3 object at the path set by the key parameter in the S3 bucket indicated by the bucket parameter. Using the example shown above, the state would be
On this page:
S3
Example Configuration
State Storage
Permissions Required
Data Source Configuration
Configuration
Multi-account AWS Architecture
Support for "S3 Compatible" Storage Providers

main.tf U

C:\Users\chido\Desktop\Terraform-Project\main.tf • Untracked

```
1   provider "aws" {
2     region = "us-east-1"
3     profile = "terraform-user"
4   }
5
6   terraform {
7     backend "s3" {
8       bucket = "mybucket"
9       key    = "path/to/my/key"
10      region = "us-east-1"
11    }
12  }
13
```

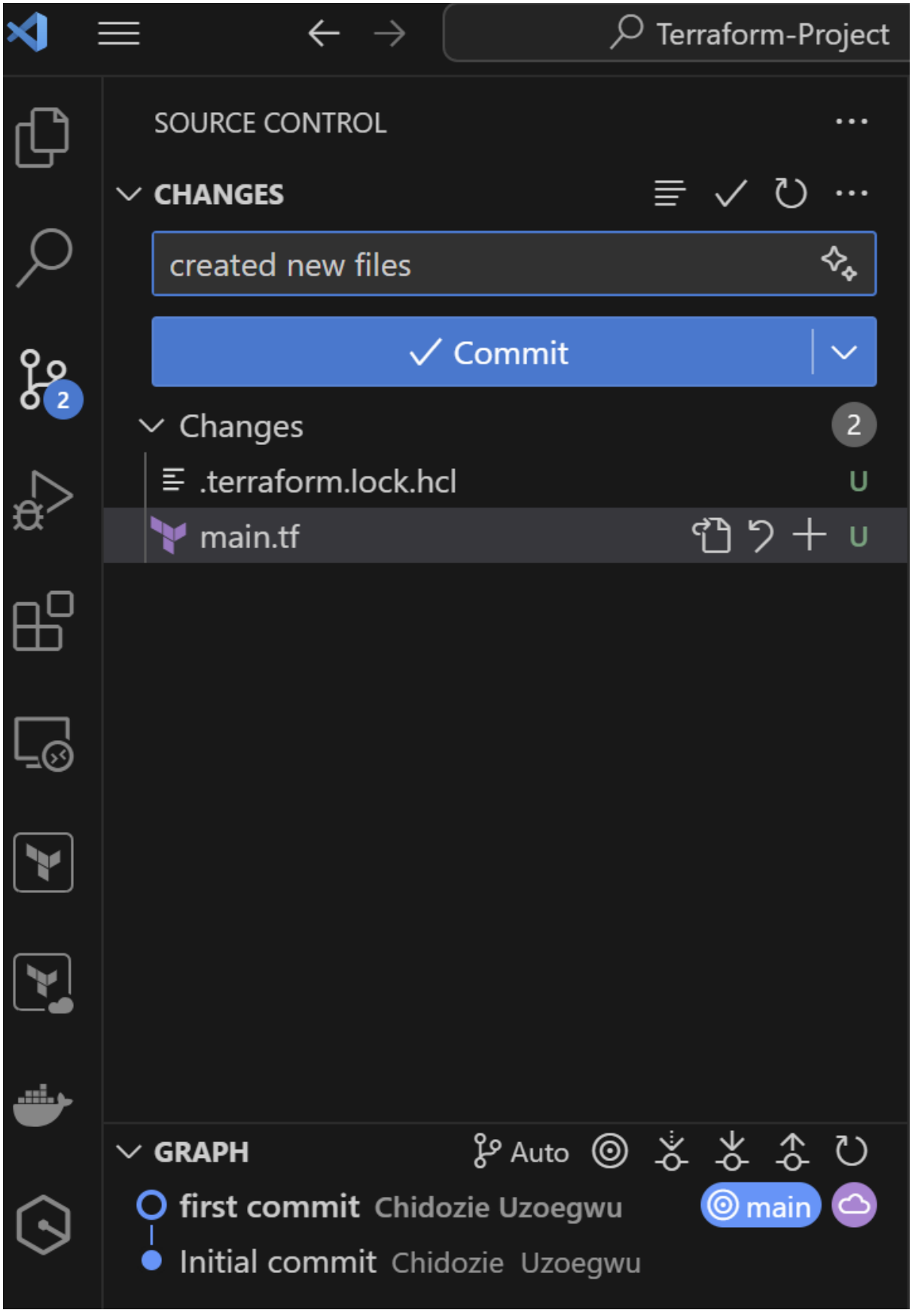
We will use the s3 bucket name and other required details from it to fill up out our terraform code/values.

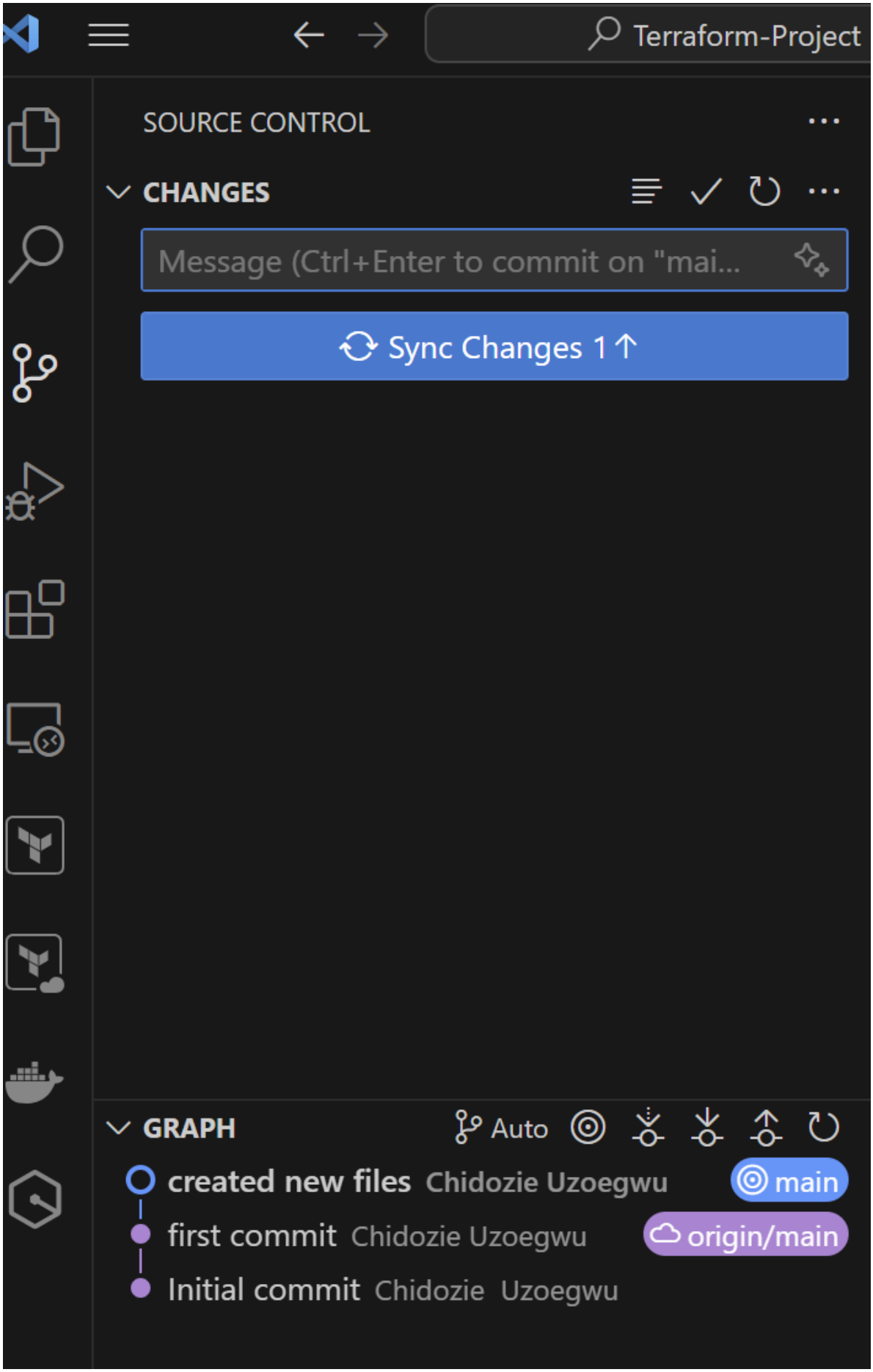


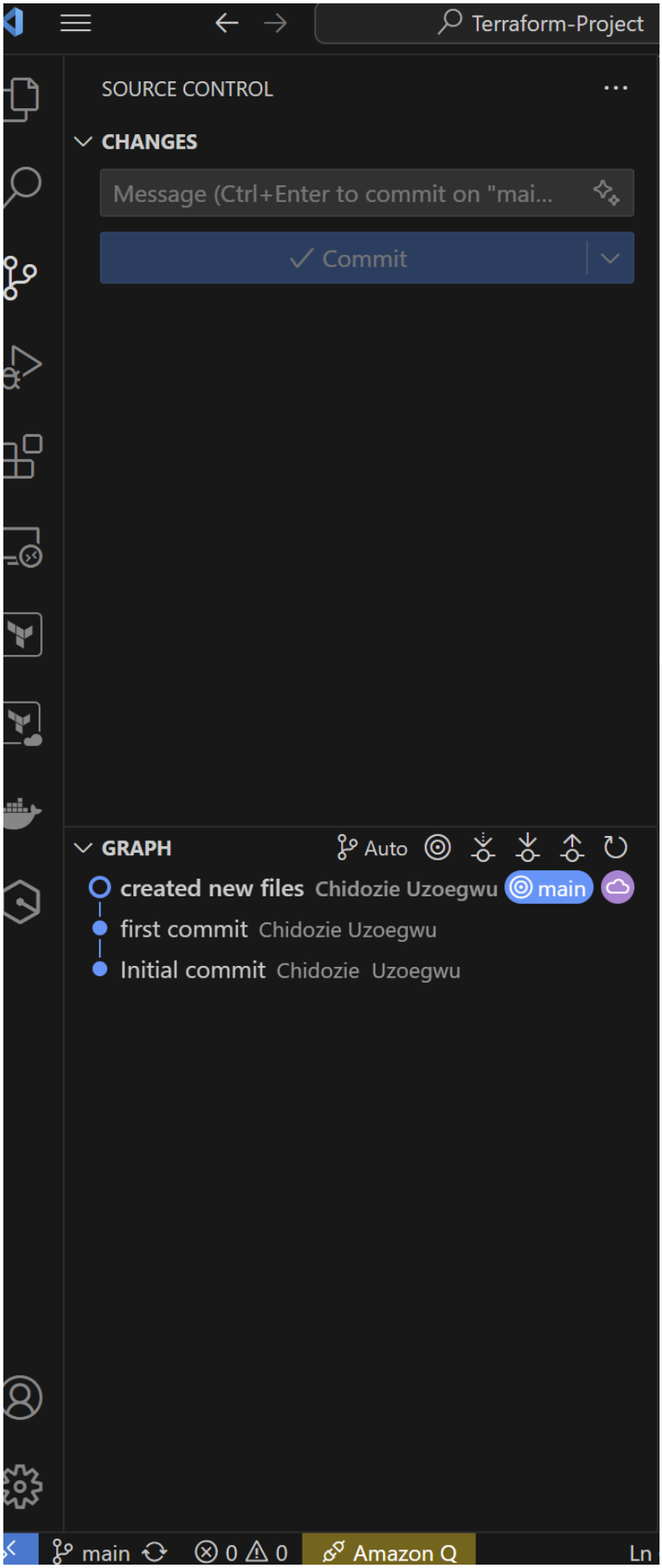
○	chidozie-terraform-remote-state1	US East (N. Virginia) us-east-1	View analyzer for us-east-1	March 22, 2025, 16:48:10 (UTC+00:00)

- This tells Terraform to store state in S3 and use our AWS CLI profile for authentication. The profile attribute ensures Terraform uses the credentials we set up in step 16.
- **Initialize Terraform:** Run terraform init in the project directory. This will:
    - Download the AWS provider plugin.
    - Initialize the backend (it will check access to your S3 bucket). If configured correctly, you should see a success message. If there's an error (like wrong bucket or permissions), troubleshoot before proceeding.

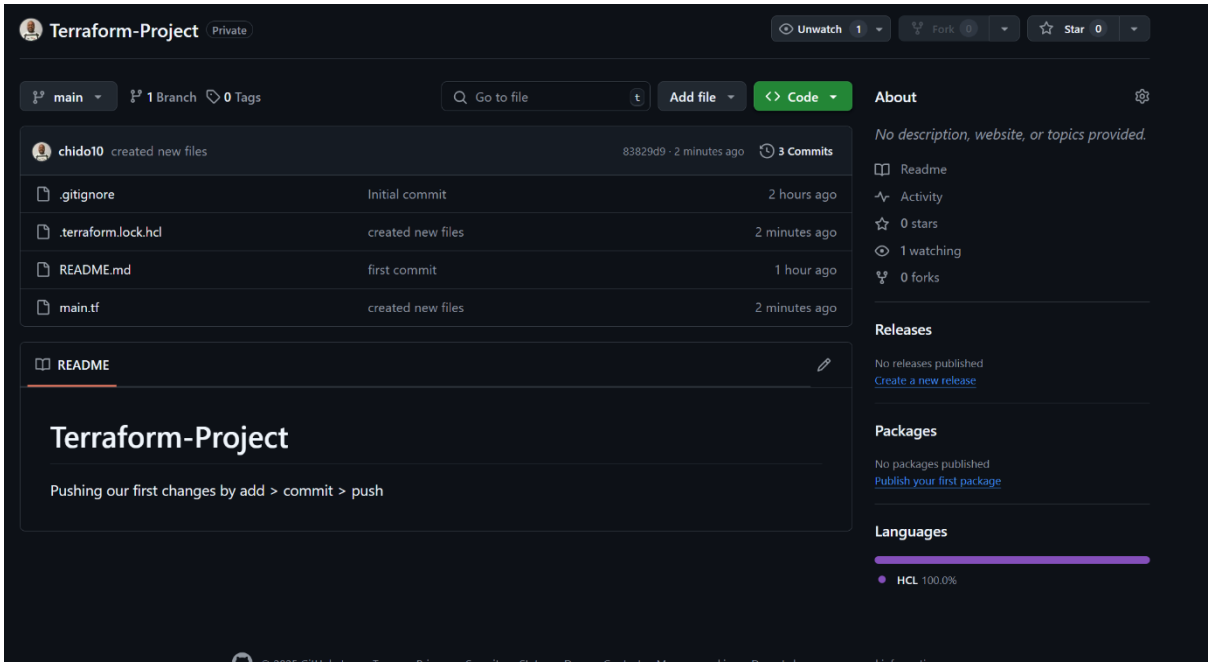
Next, we need to commit this as part of CI/CD procedure, we will be committing and pushing/Sync Changes any file we created to our GitHub. We will use the git function in our VS code to do this swiftly and then confirm in our GITHUB if this update reflects init. For any commit it is a good practice to add a note to describe what we are committing. See below

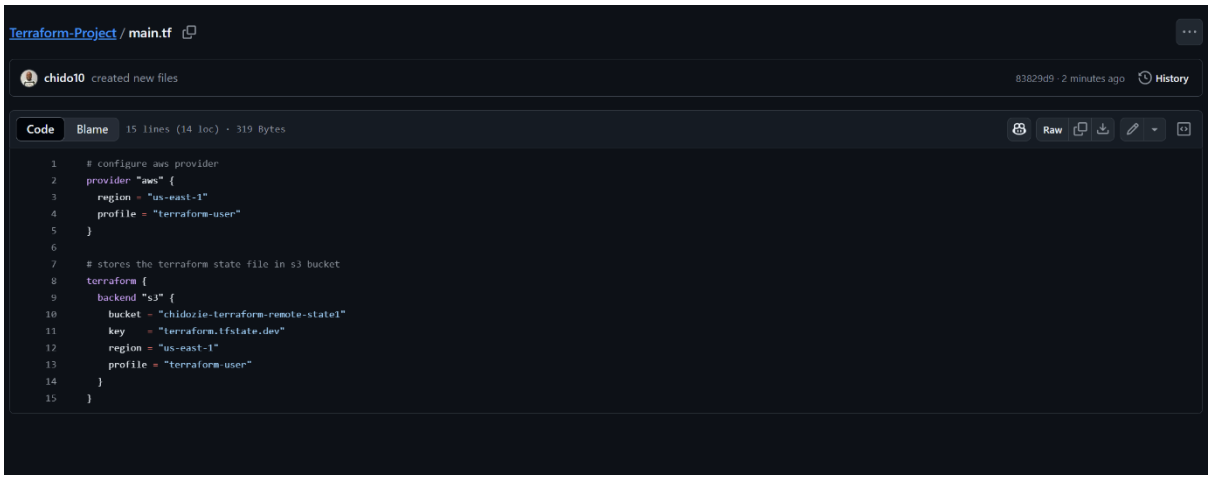

Terraform-Project
SOURCE CONTROL
CHANGES
created new files
Commit
Changes
2
.terraform.lock.hcl
U
main.tf
U
GRAPH
Auto
first commit Chidozie Uzoegwu
main
Initial commit Chidozie Uzoegwu


Terraform-Project
SOURCE CONTROL
CHANGES
Message (Ctrl+Enter to commit on "mai...
Sync Changes 1↑
GRAPH
Auto
created new files Chidozie Uzoegwu
main
first commit Chidozie Uzoegwu
origin/main
Initial commit Chidozie Uzoegwu


Terraform-Project
SOURCE CONTROL
CHANGES
Message (Ctrl+Enter to commit on "mai...
Commit
GRAPH
Auto
created new files Chidozie Uzoegwu main
first commit Chidozie Uzoegwu
Initial commit Chidozie Uzoegwu
main
0 0
Amazon Q
Ln

We can go ahead and confirm if this update is pushed to out GitHub. See below

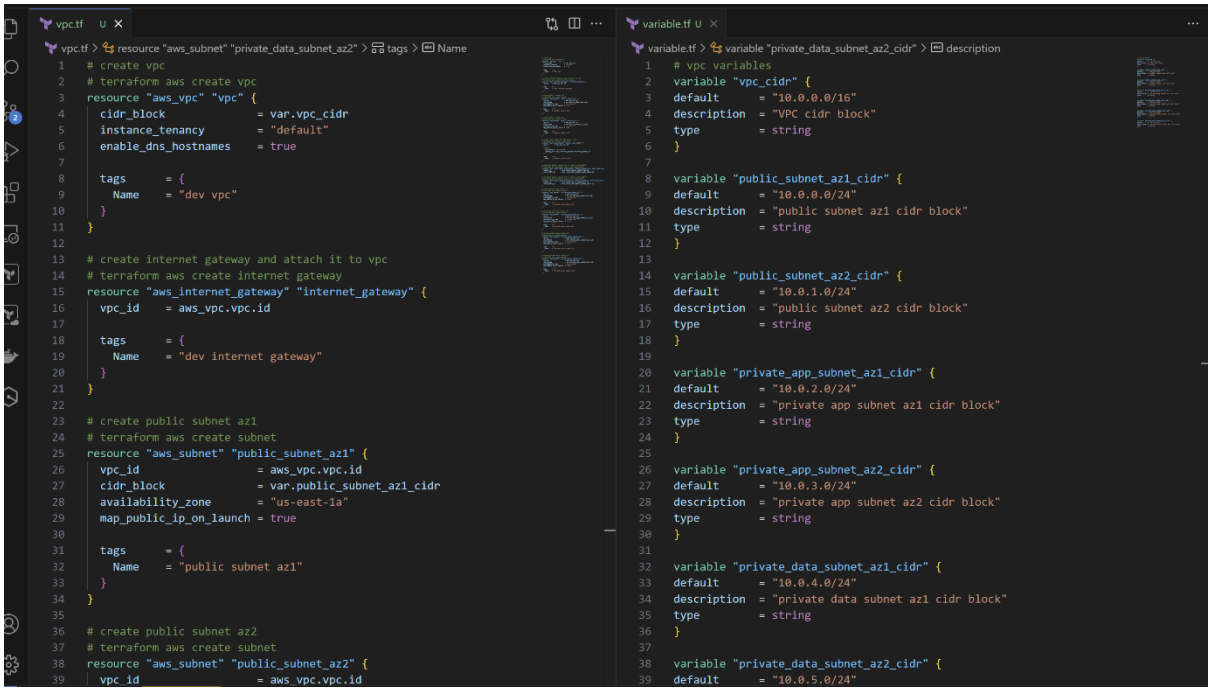




Now Terraform is authenticated and ready to create resources in AWS.

20. Create VPC

Next is to create the networking foundation – the VPC (vpc.tf) and subnets:
We also need the variable file (variables.tf), which defines input parameters that make our configuration flexible and reusable. These variables are referenced in vpc.tf to dynamically assign values like CIDR blocks, availability zones, and naming conventions. See below



vpc.tf U

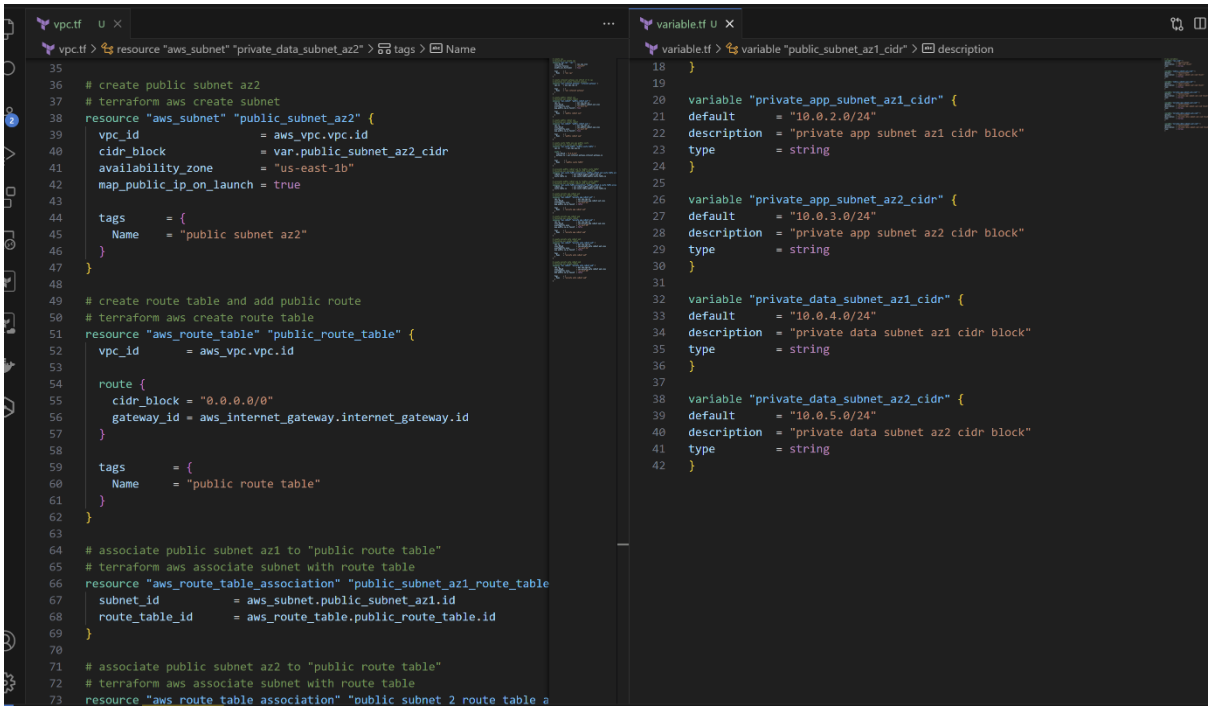
vpc.tf > resource "aws_subnet" "private_data_subnet_az2" > tags > Name

```
# create vpc
# terraform aws create vpc
resource "aws_vpc" "vpc" {
  cidr_block              = var.vpc_cidr
  instance_tenancy        = "default"
  enable_dns_hostnames    = true

  tags      = {
    Name    = "dev vpc"
  }
}

# create internet gateway and attach it to vpc
# terraform aws create internet gateway
resource "aws_internet_gateway" "internet_gateway" {
  vpc_id    = aws_vpc.vpc.id

  tags      = {
    Name    = "dev internet gateway"
  }
}

# create public subnet az1
# terraform aws create subnet
resource "aws_subnet" "public_subnet_az1" {
  vpc_id                  = aws_vpc.vpc.id
  cidr_block              = var.public_subnet_az1_cidr
  availability_zone       = "us-east-1a"
  map_public_ip_on_launch = true

  tags      = {
    Name    = "public subnet az1"
  }
}

# create public subnet az2
# terraform aws create subnet
resource "aws_subnet" "public_subnet_az2" {
  vpc_id
```

variable.tf U

variable.tf > variable "private_data_subnet_az2_cidr" > description

```
# vpc variables
variable "vpc_cidr" {
default      = "10.0.0.0/16"
description  = "VPC cidr block"
type         = string
}

variable "public_subnet_az1_cidr" {
default      = "10.0.0.0/24"
description  = "public subnet az1 cidr block"
type         = string
}

variable "public_subnet_az2_cidr" {
default      = "10.0.1.0/24"
description  = "public subnet az2 cidr block"
type         = string
}

variable "private_app_subnet_az1_cidr" {
default      = "10.0.2.0/24"
description  = "private app subnet az1 cidr block"
type         = string
}

variable "private_app_subnet_az2_cidr" {
default      = "10.0.3.0/24"
description  = "private app subnet az2 cidr block"
type         = string
}

variable "private_data_subnet_az1_cidr" {
default      = "10.0.4.0/24"
description  = "private data subnet az1 cidr block"
type         = string
}

variable "private_data_subnet_az2_cidr" {
default      = "10.0.5.0/24"
```



vpc.tf U

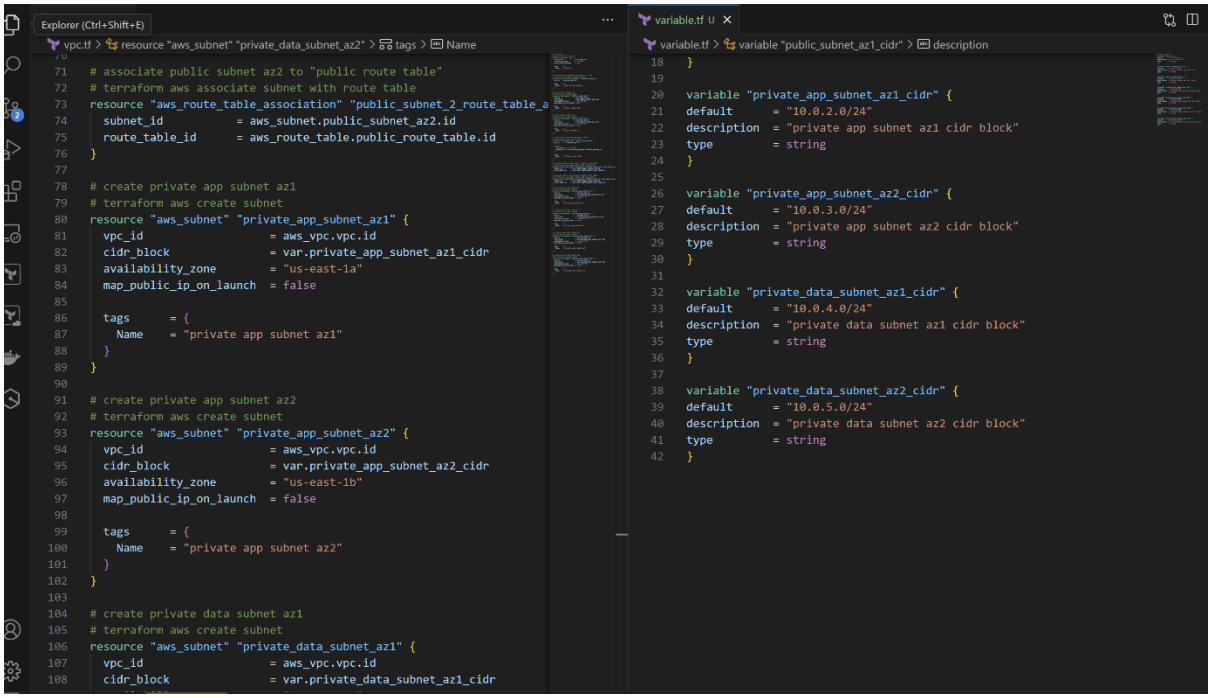
vpc.tf > resource "aws_subnet" "private_data_subnet_az2" > tags > Name

```
# create public subnet az2
# terraform aws create subnet
resource "aws_subnet" "public_subnet_az2" {
  vpc_id                  = aws_vpc.vpc.id
  cidr_block              = var.public_subnet_az2_cidr
  availability_zone       = "us-east-1b"
  map_public_ip_on_launch = true

  tags      = {
    Name    = "public subnet az2"
  }
}

# create route table and add public route
# terraform aws create route table
resource "aws_route_table" "public_route_table" {
  vpc_id       = aws_vpc.vpc.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.internet_gateway.id
  }

  tags       = {
    Name     = "public route table"
  }
}

# associate public subnet az1 to "public route table"
# terraform aws associate subnet with route table
resource "aws_route_table_association" "public_subnet_az1_route_table
  subnet_id           = aws_subnet.public_subnet_az1.id
  route_table_id      = aws_route_table.public_route_table.id
}

# associate public subnet az2 to "public route table"
# terraform aws associate subnet with route table
resource "aws_route_table_association" "public_subnet_2_route_table_a
```

variable.tf U

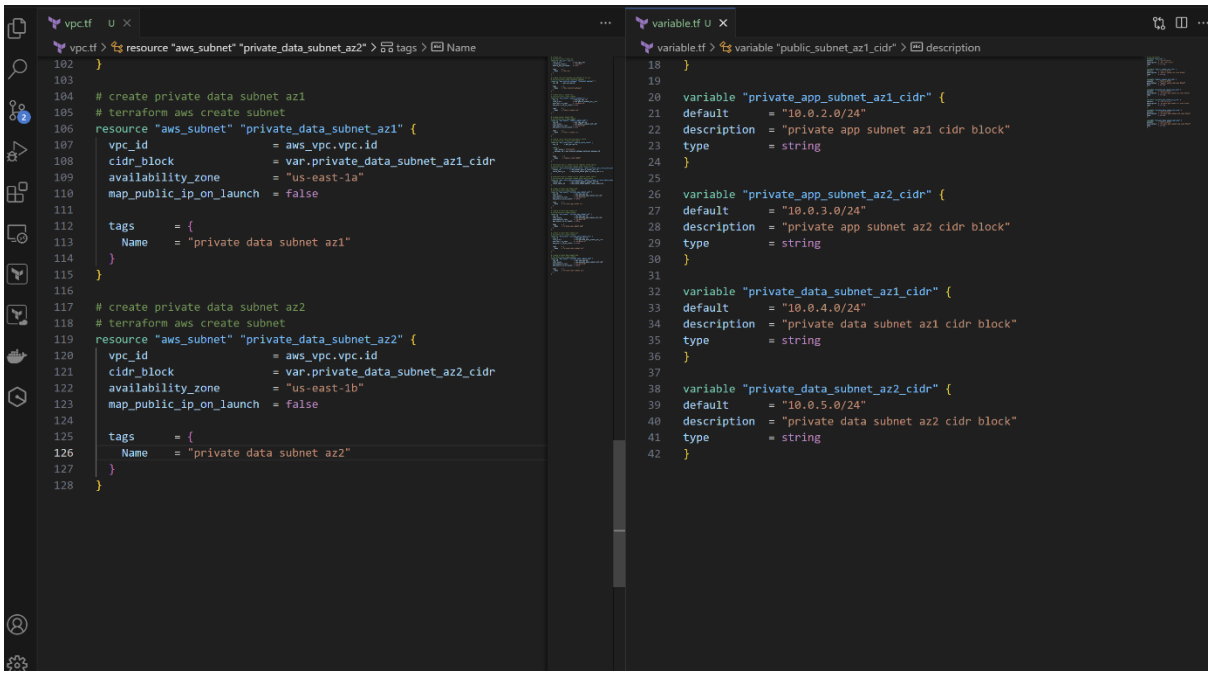
variable.tf > variable "public_subnet_az1_cidr" > description

```
}

variable "private_app_subnet_az1_cidr" {
default      = "10.0.2.0/24"
description  = "private app subnet az1 cidr block"
type         = string
}

variable "private_app_subnet_az2_cidr" {
default      = "10.0.3.0/24"
description  = "private app subnet az2 cidr block"
type         = string
}

variable "private_data_subnet_az1_cidr" {
default      = "10.0.4.0/24"
description  = "private data subnet az1 cidr block"
type         = string
}

variable "private_data_subnet_az2_cidr" {
default      = "10.0.5.0/24"
description  = "private data subnet az2 cidr block"
type         = string
}
```



```
# associate public subnet az2 to "public route table"
# terraform aws associate subnet with route table
resource "aws_route_table_association" "public_subnet_2_route_table_a
  subnet_id           = aws_subnet.public_subnet_az2.id
  route_table_id      = aws_route_table.public_route_table.id
}

# create private app subnet az1
# terraform aws create subnet
resource "aws_subnet" "private_app_subnet_az1" {
  vpc_id                  = aws_vpc.vpc.id
  cidr_block              = var.private_app_subnet_az1_cidr
  availability_zone       = "us-east-1a"
  map_public_ip_on_launch = false

  tags      = {
    Name    = "private app subnet az1"
  }
}

# create private app subnet az2
# terraform aws create subnet
resource "aws_subnet" "private_app_subnet_az2" {
  vpc_id                  = aws_vpc.vpc.id
  cidr_block              = var.private_app_subnet_az2_cidr
  availability_zone       = "us-east-1b"
  map_public_ip_on_launch = false

  tags      = {
    Name    = "private app subnet az2"
  }
}

# create private data subnet az1
# terraform aws create subnet
resource "aws_subnet" "private_data_subnet_az1" {
  vpc_id                  = aws_vpc.vpc.id
  cidr_block              = var.private_data_subnet_az1_cidr
```

```
}

variable "private_app_subnet_az1_cidr" {
default      = "10.0.2.0/24"
description  = "private app subnet az1 cidr block"
type         = string
}

variable "private_app_subnet_az2_cidr" {
default      = "10.0.3.0/24"
description  = "private app subnet az2 cidr block"
type         = string
}

variable "private_data_subnet_az1_cidr" {
default      = "10.0.4.0/24"
description  = "private data subnet az1 cidr block"
type         = string
}

variable "private_data_subnet_az2_cidr" {
default      = "10.0.5.0/24"
description  = "private data subnet az2 cidr block"
type         = string
}
```



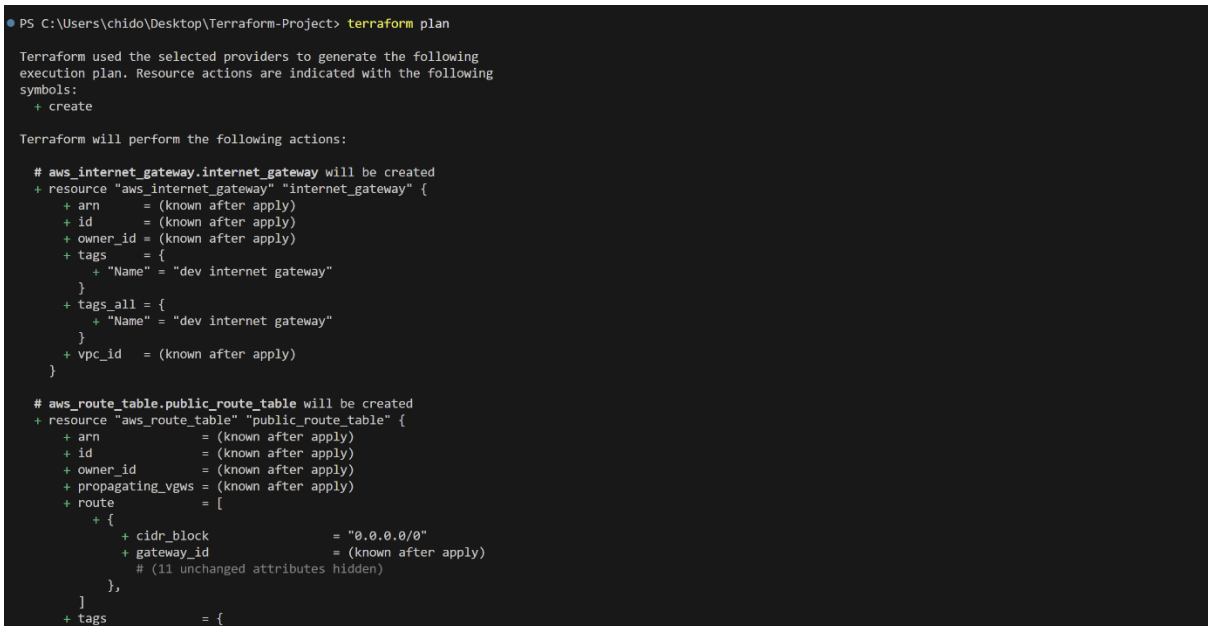
```
}

# create private data subnet az1
# terraform aws create subnet
resource "aws_subnet" "private_data_subnet_az1" {
  vpc_id                  = aws_vpc.vpc.id
  cidr_block              = var.private_data_subnet_az1_cidr
  availability_zone       = "us-east-1a"
  map_public_ip_on_launch = false

  tags      = {
    Name    = "private data subnet az1"
  }
}

# create private data subnet az2
# terraform aws create subnet
resource "aws_subnet" "private_data_subnet_az2" {
  vpc_id                  = aws_vpc.vpc.id
  cidr_block              = var.private_data_subnet_az2_cidr
  availability_zone       = "us-east-1b"
  map_public_ip_on_launch = false

  tags      = {
    Name    = "private data subnet az2"
  }
}
```

```
}

variable "private_app_subnet_az1_cidr" {
default      = "10.0.2.0/24"
description  = "private app subnet az1 cidr block"
type         = string
}

variable "private_app_subnet_az2_cidr" {
default      = "10.0.3.0/24"
description  = "private app subnet az2 cidr block"
type         = string
}

variable "private_data_subnet_az1_cidr" {
default      = "10.0.4.0/24"
description  = "private data subnet az1 cidr block"
type         = string
}

variable "private_data_subnet_az2_cidr" {
default      = "10.0.5.0/24"
description  = "private data subnet az2 cidr block"
type         = string
}
```

- **VPC & CIDR:** Define an aws_vpc resource with a CIDR block (e.g., 10.0.0.0/16). Enable DNS support if needed (usually on by default). Tag it with a Name for easy identification in AWS console.
- **Subnets:** Create multiple aws_subnet resources:

    - Public subnets (e.g., one in each availability zone, with CIDR like 10.0.1.0/24, 10.0.2.0/24). These will be used for public resources like Load Balancers or NAT Gateways.
    - Private subnets (for application servers and databases, e.g., 10.0.3.0/24, 10.0.4.0/24 in different AZs).
- **Internet Gateway:** Add an aws_internet_gateway attached to the VPC to allow internet access for public subnets.
- **Route Table (Public):** Create an aws_route_table for public subnets with a default route (0.0.0.0/0) pointing to the Internet Gateway. Associate this route table with the public subnets via aws_route_table_association.

The **vpc.tf** file in the project contains these resources. After writing it, you'd run terraform plan/apply to create them.



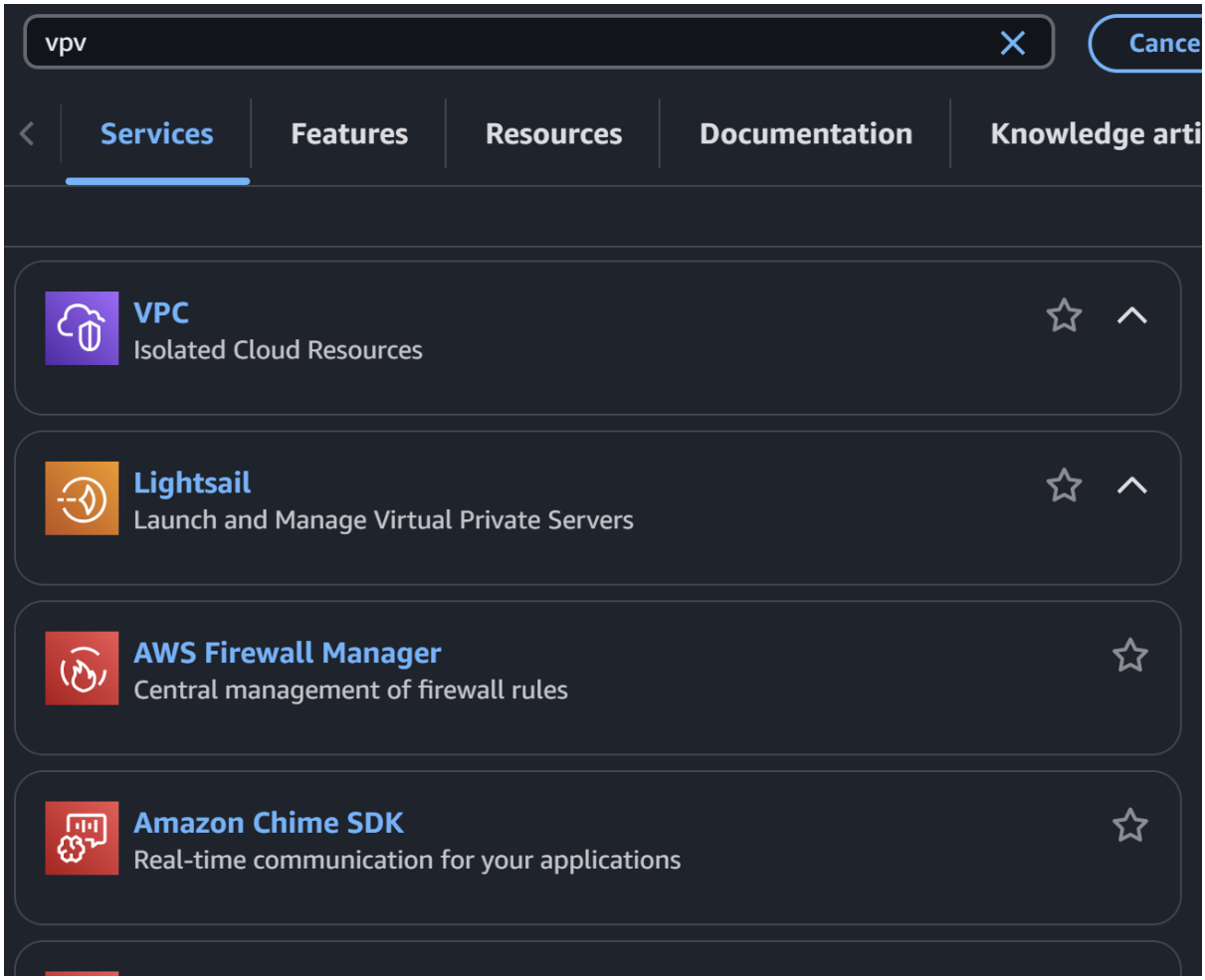
```
PS C:\Users\chido\Desktop\Terraform-Project> terraform apply

Terraform used the selected providers to generate the following
execution plan. Resource actions are indicated with the following
symbols:
  + create

Terraform will perform the following actions:

  # aws_internet_gateway.internet_gateway will be created
  + resource "aws_internet_gateway" "internet_gateway" {
      + arn      = (known after apply)
      + id       = (known after apply)
      + owner_id = (known after apply)
      + tags     = {
          + "Name" = "dev internet gateway"
        }
      + tags_all = {
          + "Name" = "dev internet gateway"
        }
      + vpc_id   = (known after apply)
    }

  # aws_route_table.public_route_table will be created
  + resource "aws_route_table" "public_route_table" {
      + arn              = (known after apply)
      + id               = (known after apply)
```

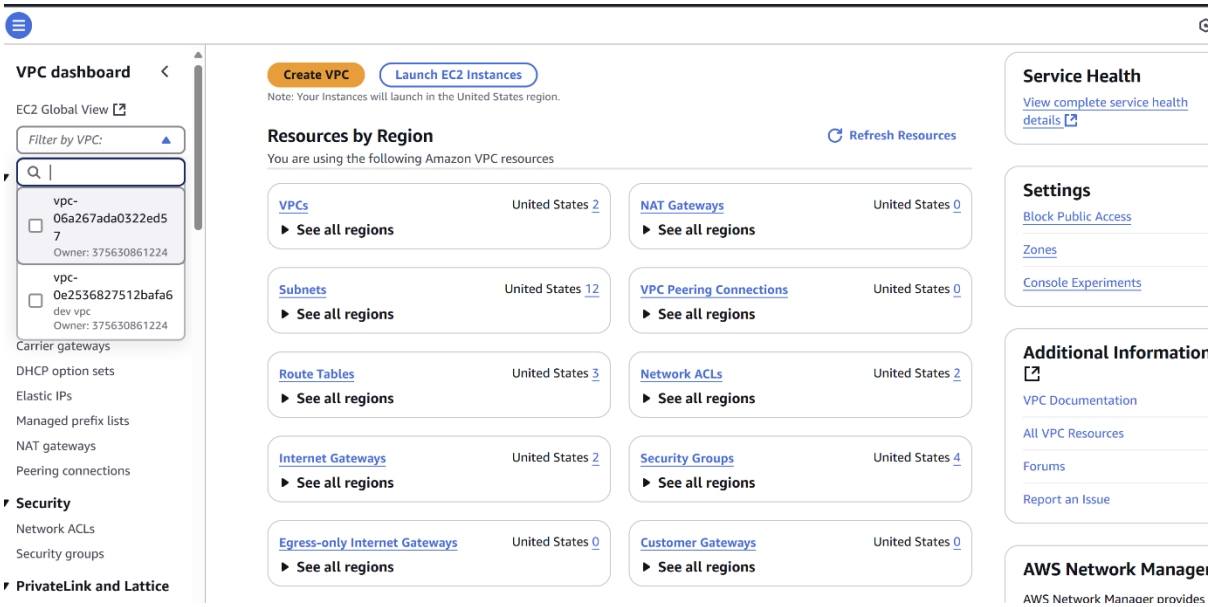
Terraform will output the VPC ID, subnet IDs, etc., which we'll use in later resources.

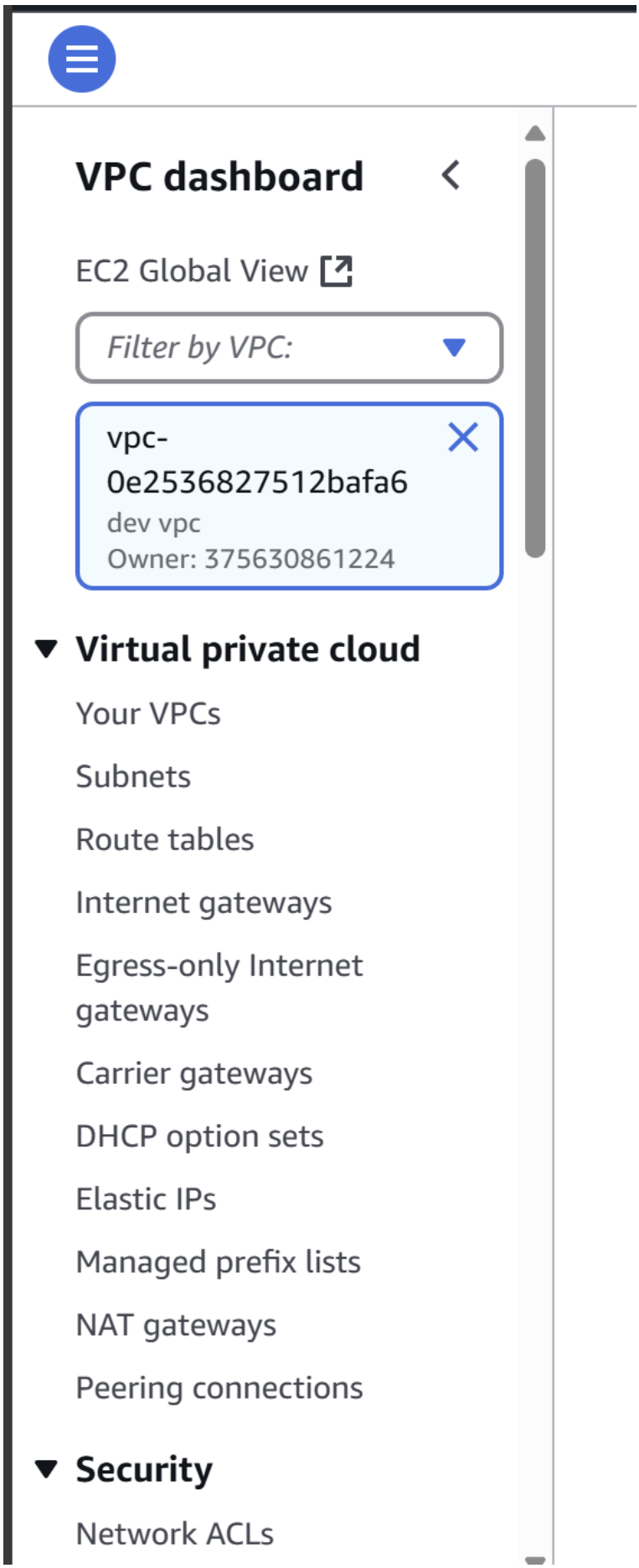
We will then confirm that we want to perform these changes with 'Yes'

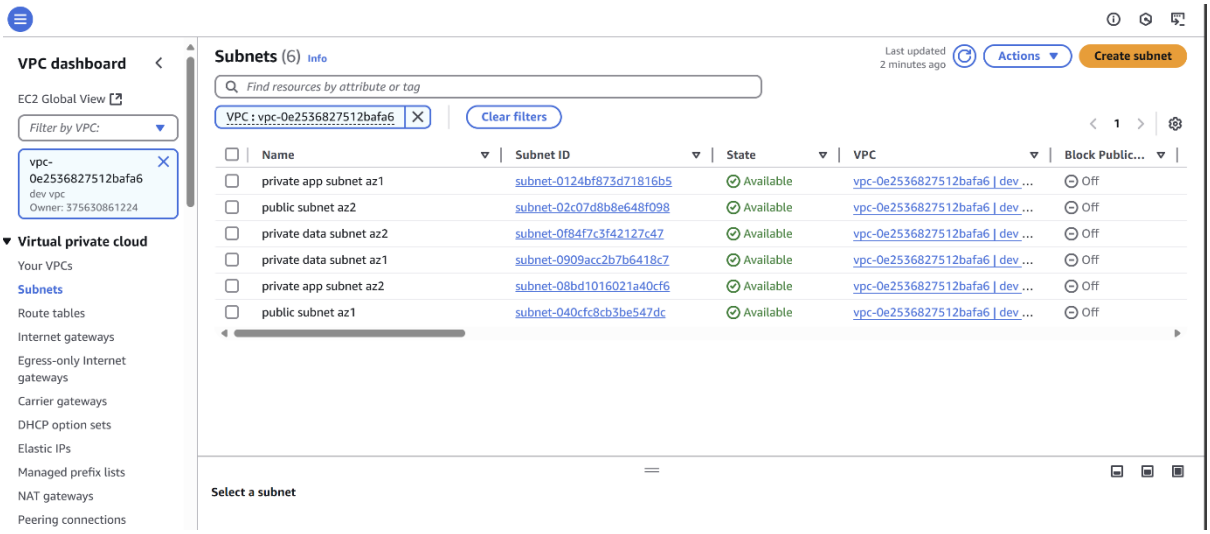
```
aws_subnet.private_app_subnet_az1: Creating...
aws_internet_gateway.internet_gateway: Creating...
aws_subnet.public_subnet_az1: Creating...
aws_subnet.public_subnet_az2: Creating...
aws_subnet.private_data_subnet_az1: Creating...
aws_subnet.private_app_subnet_az2: Creating...
aws_subnet.private_data_subnet_az2: Creating...
aws_internet_gateway.internet_gateway: Creation complete after 1s [id=igw-0550600374ce71ab9]
aws_route_table.public_route_table: Creating...
aws_subnet.private_app_subnet_az2: Creation complete after 1s [id=subnet-08bd1016021a40cf6]
aws_subnet.private_app_subnet_az1: Creation complete after 1s [id=subnet-0124bf873d71816b5]
aws_subnet.private_data_subnet_az1: Creation complete after 1s [id=subnet-0909acc2b7b6418c7]
aws_route_table.public_route_table: Creation complete after 2s [id=rtb-0aad8ffe33ba8d07d]
aws_subnet.private_data_subnet_az2: Creation complete after 4s [id=subnet-0f84f7c3f42127c47]
aws_subnet.public_subnet_az1: Still creating... [10s elapsed]
aws_subnet.public_subnet_az2: Still creating... [10s elapsed]
aws_subnet.public_subnet_az1: Creation complete after 12s [id=subnet-040cfc8cb3be547dc]
aws_route_table_association.public_subnet_az1_route_table_association: Creating...
aws_subnet.public_subnet_az2: Creation complete after 12s [id=subnet-02c07d8b8e648f098]
aws_route_table_association.public_subnet_2_route_table_association: Creating...
aws_route_table_association.public_subnet_2_route_table_association: Creation complete after 1s [id=rtbassoc-06b480f92e14ecdc7]
aws_route_table_association.public_subnet_az1_route_table_association: Creation complete after 1s [id=rtbassoc-087ca3c385245022c]

Apply complete! Resources: 11 added, 0 changed, 0 destroyed.
PS C:\Users\chido\Desktop\Terraform-Project>
```

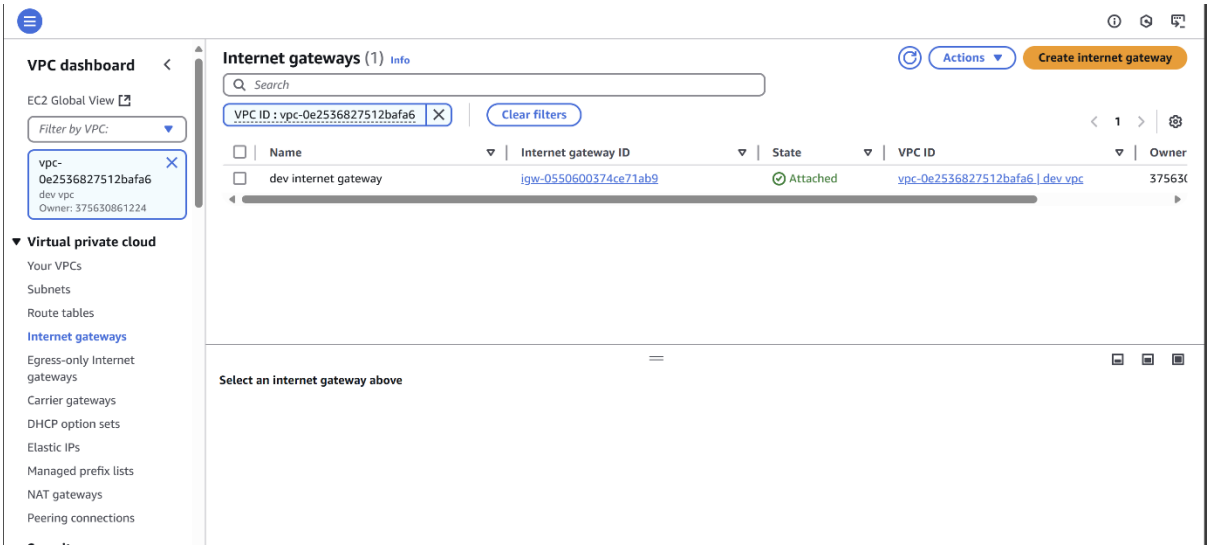
After applying, you can verify in the AWS console under VPC > Your VPCs that the new VPC is created with the correct CIDR and has the subnets, gateway, and route table set up.

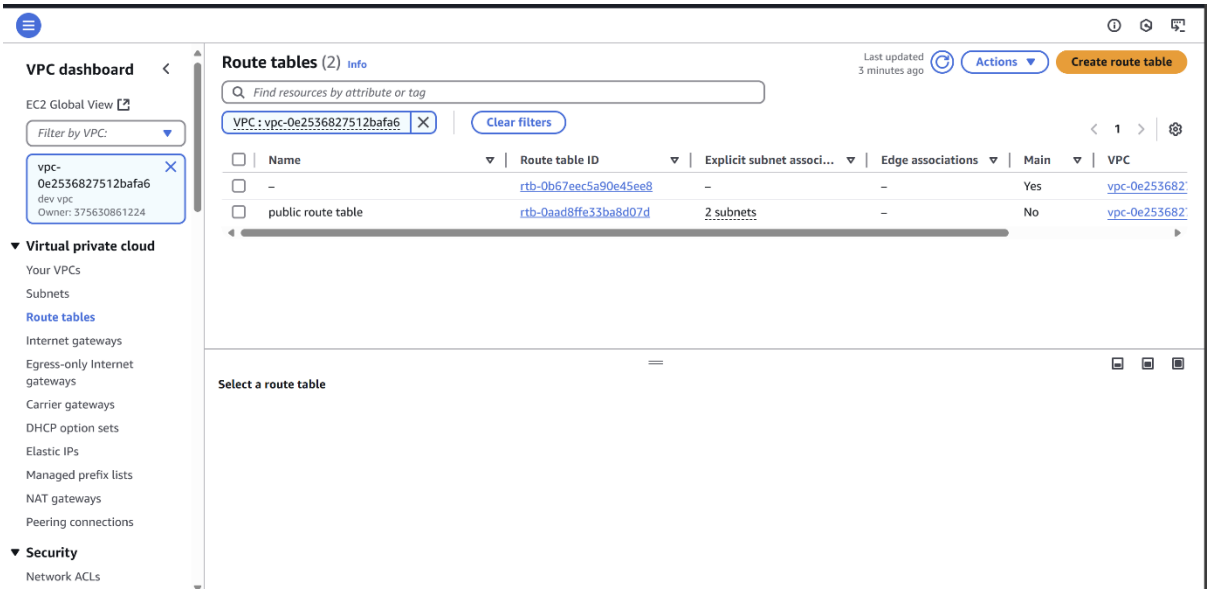

vpv
Cance
Services
Features
Resources
Documentation
Knowledge arti
VPC
Isolated Cloud Resources
Lightsail
Launch and Manage Virtual Private Servers
AWS Firewall Manager
Central management of firewall rules
Amazon Chime SDK
Real-time communication for your applications

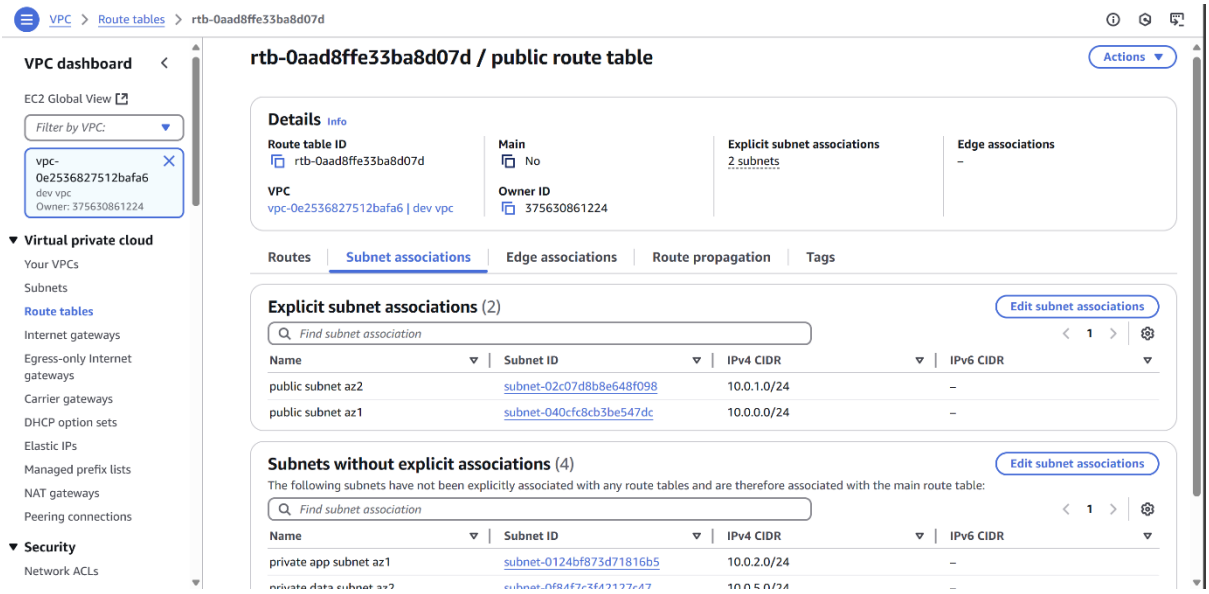

VPC dashboard
EC2 Global View
Filter by VPC:
vpc-06a267ada0322ed57
Owner: 375630861224
vpc-0e2536827512bafa6
dev vpc
Owner: 375630861224
Carrier gateways
DHCP option sets
Elastic IPs
Managed prefix lists
NAT gateways
Peering connections
Security
Network ACLs
Security groups
PrivateLink and Lattice
Create VPC
Launch EC2 Instances
Note: Your Instances will launch in the United States region.
Resources by Region
Refresh Resources
You are using the following Amazon VPC resources
VPCs United States 2
See all regions
NAT Gateways United States 0
See all regions
Subnets United States 12
See all regions
VPC Peering Connections United States 0
See all regions
Route Tables United States 3
See all regions
Network ACLs United States 2
See all regions
Internet Gateways United States 2
See all regions
Security Groups United States 4
See all regions
Egress-only Internet Gateways United States 0
See all regions
Customer Gateways United States 0
See all regions
Service Health
View complete service health details
Settings
Block Public Access
Zones
Console Experiments
Additional Information
VPC Documentation
All VPC Resources
Forums
Report an Issue
AWS Network Manager
AWS Network Manager provides


VPC dashboard
EC2 Global View
Filter by VPC:
vpc-
0e2536827512bafa6
dev vpc
Owner: 375630861224
Virtual private cloud
Your VPCs
Subnets
Route tables
Internet gateways
Egress-only Internet gateways
Carrier gateways
DHCP option sets
Elastic IPs
Managed prefix lists
NAT gateways
Peering connections
Security
Network ACLs

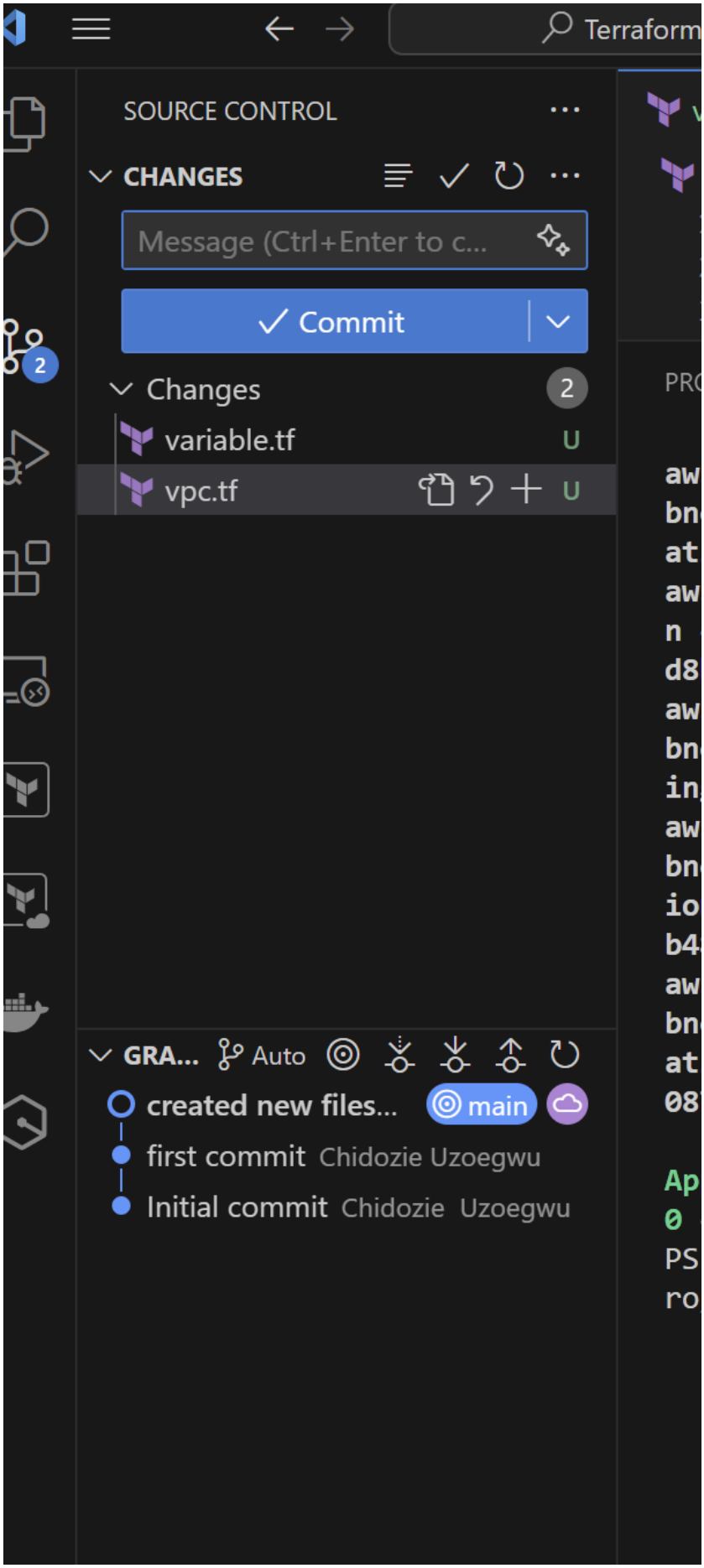

VPC dashboard
EC2 Global View
Filter by VPC:
vpc-0e2536827512bafa6
dev vpc
Owner: 375630861224
Virtual private cloud
Your VPCs
Subnets
Route tables
Internet gateways
Egress-only Internet gateways
Carrier gateways
DHCP option sets
Elastic IPs
Managed prefix lists
NAT gateways
Peering connections
Subnets (6) Info
Last updated 2 minutes ago
Actions
Create subnet
Find resources by attribute or tag
VPC : vpc-0e2536827512bafa6
Clear filters
Name
Subnet ID
State
VPC
Block Public...
private app subnet az1
subnet-0124bf873d71816b5
Available
vpc-0e2536827512bafa6 | dev ...
Off
public subnet az2
subnet-02c07d8b8e648f098
private data subnet az2
subnet-0f84f7c3f42127c47
private data subnet az1
subnet-0909acc2b7b6418c7
private app subnet az2
subnet-08bd1016021a40cf6
public subnet az1
subnet-040cfc8cb3be547dc
Select a subnet

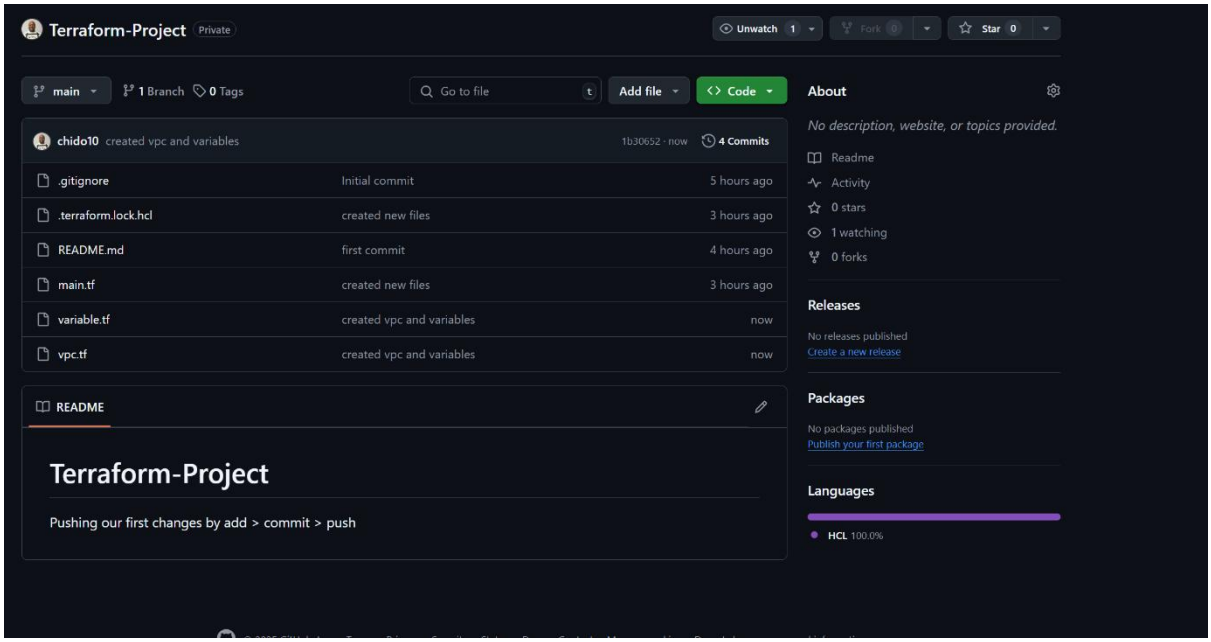

VPC dashboard
EC2 Global View
Filter by VPC:
vpc-0e2536827512bafa6
dev vpc
Owner: 375630861224
Virtual private cloud
Your VPCs
Subnets
Route tables
Internet gateways
Egress-only Internet gateways
Carrier gateways
DHCP option sets
Elastic IPs
Managed prefix lists
NAT gateways
Peering connections
Internet gateways (1) Info
Actions
Create internet gateway
Search
VPC ID : vpc-0e2536827512bafa6
Clear filters
Name
Internet gateway ID
State
VPC ID
Owner
dev internet gateway
igw-0550600374ce71ab9
Attached
vpc-0e2536827512bafa6 | dev vpc
Select an internet gateway above


VPC dashboard
EC2 Global View
Filter by VPC:
vpc-0e2536827512bafa6
dev vpc
Owner: 375630861224
Virtual private cloud
Your VPCs
Subnets
Route tables
Internet gateways
Egress-only Internet gateways
Carrier gateways
DHCP option sets
Elastic IPs
Managed prefix lists
NAT gateways
Peering connections
Security
Network ACLs
Route tables (2) Info
Last updated 3 minutes ago
Actions
Create route table
Find resources by attribute or tag
VPC : vpc-0e2536827512bafa6
Clear filters
Name
Route table ID
Explicit subnet associ...
Edge associations
Main
VPC
–
rtb-0b67eec5a90e45ee8
Yes
public route table
rtb-0aad8ffe33ba8d07d
2 subnets
No
Select a route table



After confirming this we can as well commit and push to our GitHub and confirm it henceforth. See below:


Terraform
SOURCE CONTROL
CHANGES
Message (Ctrl+Enter to c...
Commit
Changes
2
variable.tf
U
vpc.tf
U
GRA...
Auto
created new files...
main
first commit Chidozie Uzoegwu
Initial commit Chidozie Uzoegwu



21. Create NAT Gateway

Our private subnets need a way to reach the internet (for updates, etc.) without being directly exposed. NAT Gateways provide this:

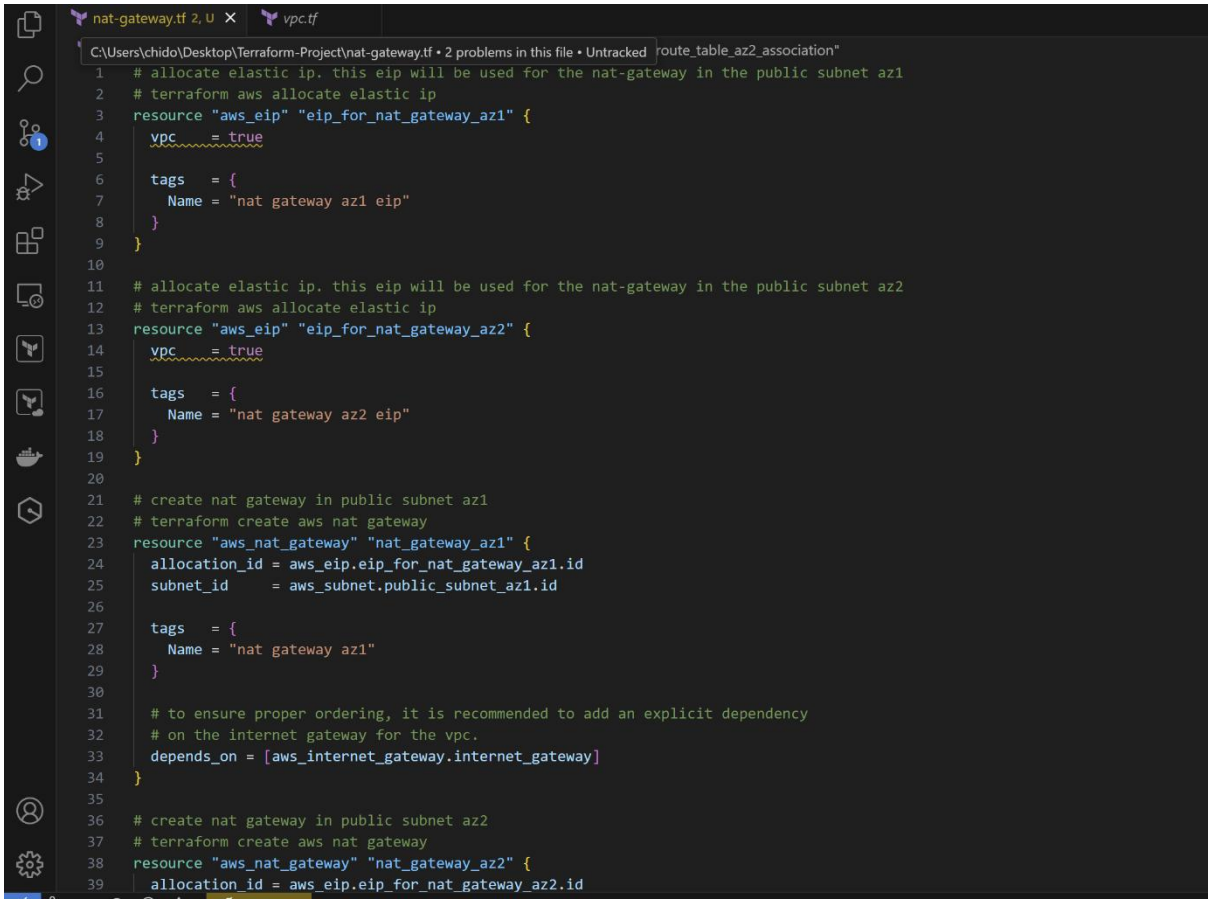
- **Elastic IPs:** For each NAT Gateway, allocate an Elastic IP (aws_eip). In our setup, we create two NAT Gateways (for high availability across AZs), thus two EIPs:
- **NAT Gateways:** Create aws_nat_gateway resources, one in each public subnet, using the above EIPs:

We add a depends_on = [aws_internet_gateway.internet_gateway] in the Terraform code to ensure the internet gateway is created before NAT (Terraform sometimes needs explicit ordering when IP addresses are involved).

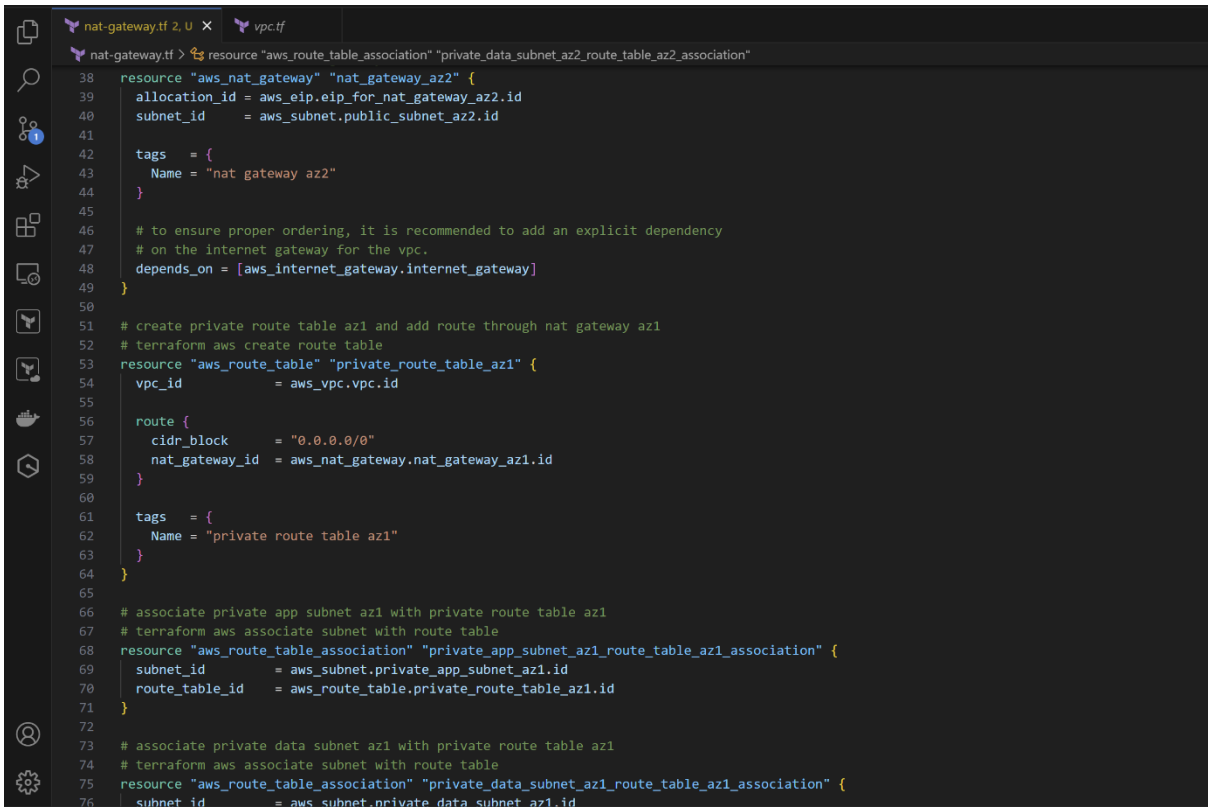
- **Private Route Tables:** For each AZ's private subnet(s), create an aws_route_table that routes 0.0.0.0/0 to the NAT Gateway in that AZ:

Similarly, for az2. Then associate each private subnet with its respective route table using aws_route_table_association.

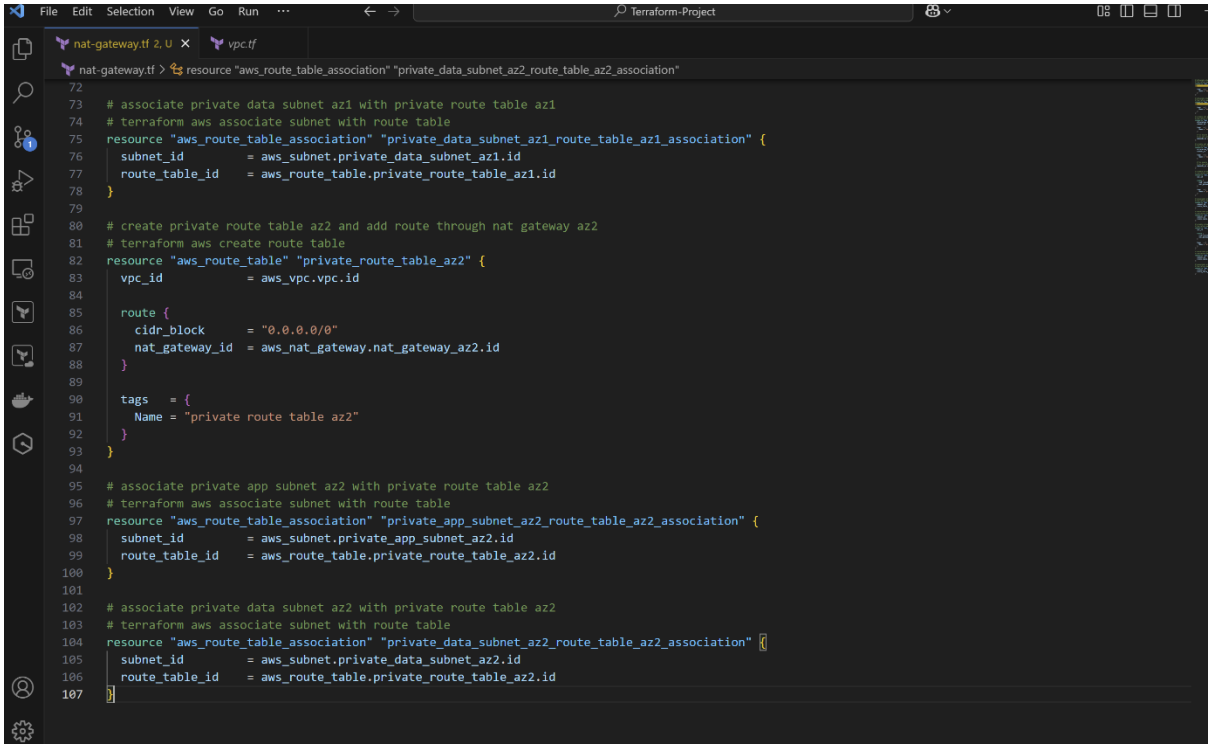
*Screenshot: Terraform code in **nat-gateway.tf** showing EIP and NAT gateway resources, and route table associations.*



```
nat-gateway.tf 2, U  vpc.tf
C:\Users\chido\Desktop\Terraform-Project\nat-gateway.tf • 2 problems in this file • Untracked
route_table_az2_association"
# allocate elastic ip. this eip will be used for the nat-gateway in the public subnet az1
# terraform aws allocate elastic ip
resource "aws_eip" "eip_for_nat_gateway_az1" {
  vpc      = true

  tags   = {
    Name = "nat gateway az1 eip"
  }
}

# allocate elastic ip. this eip will be used for the nat-gateway in the public subnet az2
# terraform aws allocate elastic ip
resource "aws_eip" "eip_for_nat_gateway_az2" {
  vpc      = true

  tags   = {
    Name = "nat gateway az2 eip"
  }
}

# create nat gateway in public subnet az1
# terraform create aws nat gateway
resource "aws_nat_gateway" "nat_gateway_az1" {
  allocation_id = aws_eip.eip_for_nat_gateway_az1.id
  subnet_id     = aws_subnet.public_subnet_az1.id

  tags   = {
    Name = "nat gateway az1"
  }

  # to ensure proper ordering, it is recommended to add an explicit dependency
  # on the internet gateway for the vpc.
  depends_on = [aws_internet_gateway.internet_gateway]
}

# create nat gateway in public subnet az2
# terraform create aws nat gateway
resource "aws_nat_gateway" "nat_gateway_az2" {
  allocation_id = aws_eip.eip_for_nat_gateway_az2.id
```

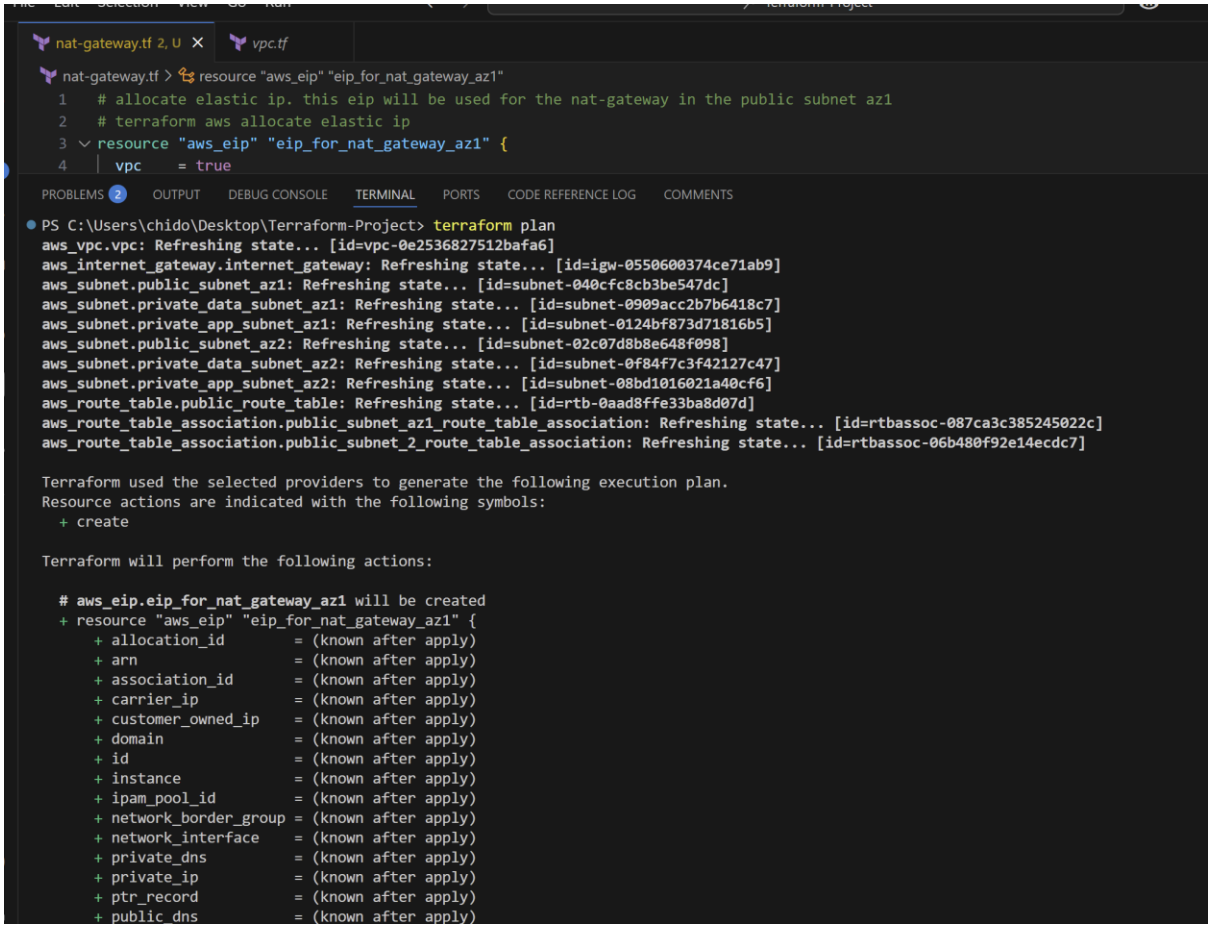


```
nat-gateway.tf 2, U  vpc.tf
nat-gateway.tf > resource "aws_route_table_association" "private_data_subnet_az2_route_table_az2_association"
resource "aws_nat_gateway" "nat_gateway_az2" {
  allocation_id = aws_eip.eip_for_nat_gateway_az2.id
  subnet_id     = aws_subnet.public_subnet_az2.id

  tags   = {
    Name = "nat gateway az2"
  }

  # to ensure proper ordering, it is recommended to add an explicit dependency
  # on the internet gateway for the vpc.
  depends_on = [aws_internet_gateway.internet_gateway]
}

# create private route table az1 and add route through nat gateway az1
# terraform aws create route table
resource "aws_route_table" "private_route_table_az1" {
  vpc_id            = aws_vpc.vpc.id

  route {
    cidr_block      = "0.0.0.0/0"
    nat_gateway_id  = aws_nat_gateway.nat_gateway_az1.id
  }

  tags   = {
    Name = "private route table az1"
  }
}

# associate private app subnet az1 with private route table az1
# terraform aws associate subnet with route table
resource "aws_route_table_association" "private_app_subnet_az1_route_table_az1_association" {
  subnet_id         = aws_subnet.private_app_subnet_az1.id
  route_table_id    = aws_route_table.private_route_table_az1.id
}

# associate private data subnet az1 with private route table az1
# terraform aws associate subnet with route table
resource "aws_route_table_association" "private_data_subnet_az1_route_table_az1_association" {
  subnet_id         = aws_subnet.private_data_subnet_az1.id
```



From the screenshot:

- NAT Gateways for AZ1 and AZ2 (aws_nat_gateway.nat_gateway_az1/az2).
- Route tables for private subnets (with nat_gateway_id pointing to the NAT gateways).
- Route table associations linking private subnets to those route tables.

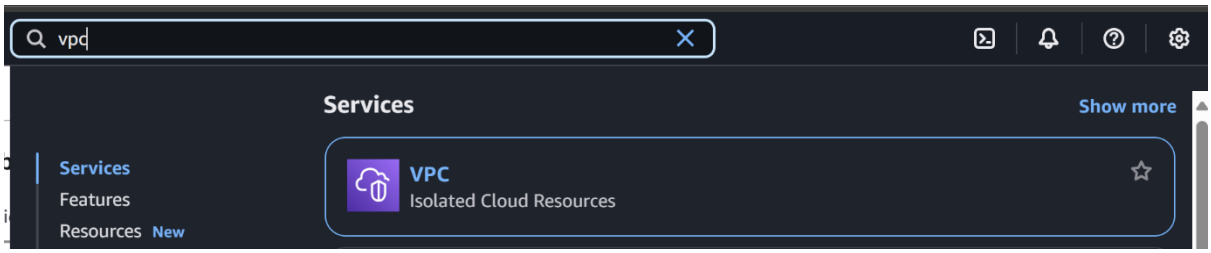


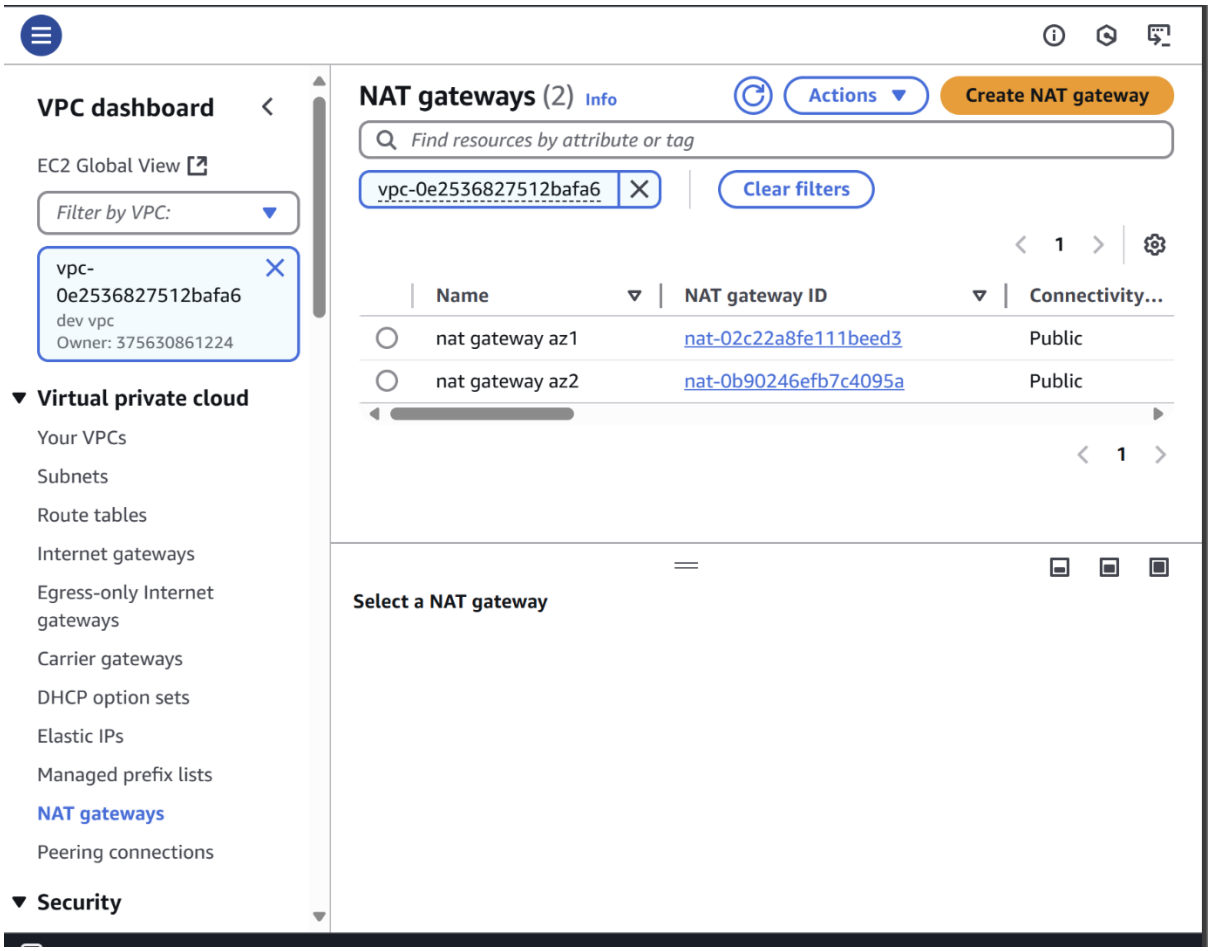
```
nat-gateway.tf 2, U   vpc.tf
nat-gateway.tf > resource "aws_eip" "eip_for_nat_gateway_az1"
1  # allocate elastic ip. this eip will be used for the nat-gateway in the public subnet az1
2  # terraform aws allocate elastic ip
3  resource "aws_eip" "eip_for_nat_gateway_az1" {
4    vpc      = true

PROBLEMS 2   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   CODE REFERENCE LOG   COMMENTS

PS C:\Users\chido\Desktop\Terraform-Project> terraform plan
aws_vpc.vpc: Refreshing state... [id=vpc-0e2536827512bafa6]
aws_internet_gateway.internet_gateway: Refreshing state... [id=igw-0550600374ce71ab9]
aws_subnet.public_subnet_az1: Refreshing state... [id=subnet-040cfc8cb3be547dc]
aws_subnet.private_data_subnet_az1: Refreshing state... [id=subnet-0909acc2b7b6418c7]
aws_subnet.private_app_subnet_az1: Refreshing state... [id=subnet-0124bf873d71816b5]
aws_subnet.public_subnet_az2: Refreshing state... [id=subnet-02c07d8b8e648f098]
aws_subnet.private_data_subnet_az2: Refreshing state... [id=subnet-0f84f7c3f42127c47]
aws_subnet.private_app_subnet_az2: Refreshing state... [id=subnet-08bd1016021a40cf6]
aws_route_table.public_route_table: Refreshing state... [id=rtb-0aad8ffe33ba8d07d]
aws_route_table_association.public_subnet_az1_route_table_association: Refreshing state... [id=rtbassoc-087ca3c385245022c]
aws_route_table_association.public_subnet_2_route_table_association: Refreshing state... [id=rtbassoc-06b480f92e14ecdc7]

Terraform used the selected providers to generate the following execution plan.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # aws_eip.eip_for_nat_gateway_az1 will be created
  + resource "aws_eip" "eip_for_nat_gateway_az1" {
      + allocation_id        = (known after apply)
      + arn                  = (known after apply)
      + association_id       = (known after apply)
      + carrier_ip           = (known after apply)
      + customer_owned_ip    = (known after apply)
      + domain               = (known after apply)
      + id                   = (known after apply)
      + instance             = (known after apply)
      + ipam_pool_id         = (known after apply)
      + network_border_group = (known after apply)
      + network_interface    = (known after apply)
      + private_dns          = (known after apply)
      + private_ip           = (known after apply)
      + ptr_record           = (known after apply)
      + public_dns           = (known after apply)
```

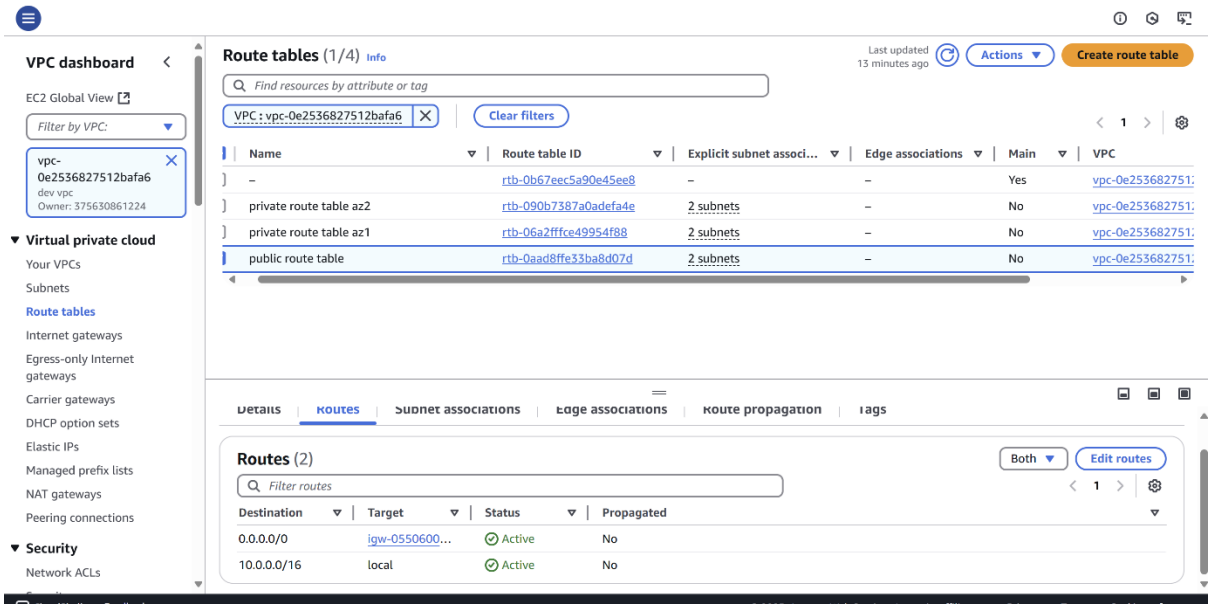
```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_eip.eip_for_nat_gateway_az2: Creating...
aws_eip.eip_for_nat_gateway_az1: Creating...
aws_eip.eip_for_nat_gateway_az2: Creation complete after 2s [id=eipalloc-0a3c649035e2ef090]
aws_eip.eip_for_nat_gateway_az1: Creation complete after 2s [id=eipalloc-0353a7e76907b2048]
aws_nat_gateway.nat_gateway_az2: Creating...
aws_nat_gateway.nat_gateway_az1: Creating...
aws_nat_gateway.nat_gateway_az2: Still creating... [10s elapsed]
aws_nat_gateway.nat_gateway_az1: Still creating... [10s elapsed]
aws_nat_gateway.nat_gateway_az2: Still creating... [20s elapsed]
aws_nat_gateway.nat_gateway_az1: Still creating... [20s elapsed]
aws_nat_gateway.nat_gateway_az2: Still creating... [30s elapsed]
aws_nat_gateway.nat_gateway_az1: Still creating... [30s elapsed]
aws_nat_gateway.nat_gateway_az2: Still creating... [40s elapsed]
aws_nat_gateway.nat_gateway_az1: Still creating... [40s elapsed]
aws_nat_gateway.nat_gateway_az2: Still creating... [50s elapsed]
aws_nat_gateway.nat_gateway_az1: Still creating... [50s elapsed]
aws_nat_gateway.nat_gateway_az2: Still creating... [1m0s elapsed]
aws_nat_gateway.nat_gateway_az1: Still creating... [1m0s elapsed]
```

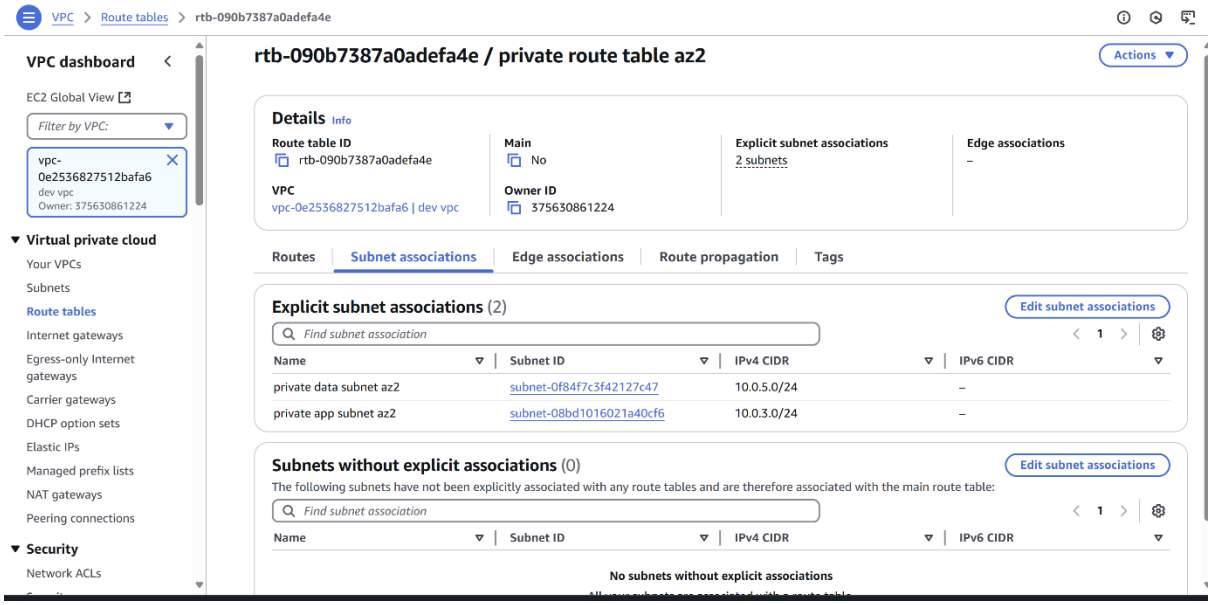
After applying these, the private subnets will have internet access through NAT. You can confirm in AWS VPC console under Route Tables: each private route table has a route to a

NAT gateway. The NAT gateways themselves will show up in the VPC console's NAT Gateway section, each with an Elastic IP.

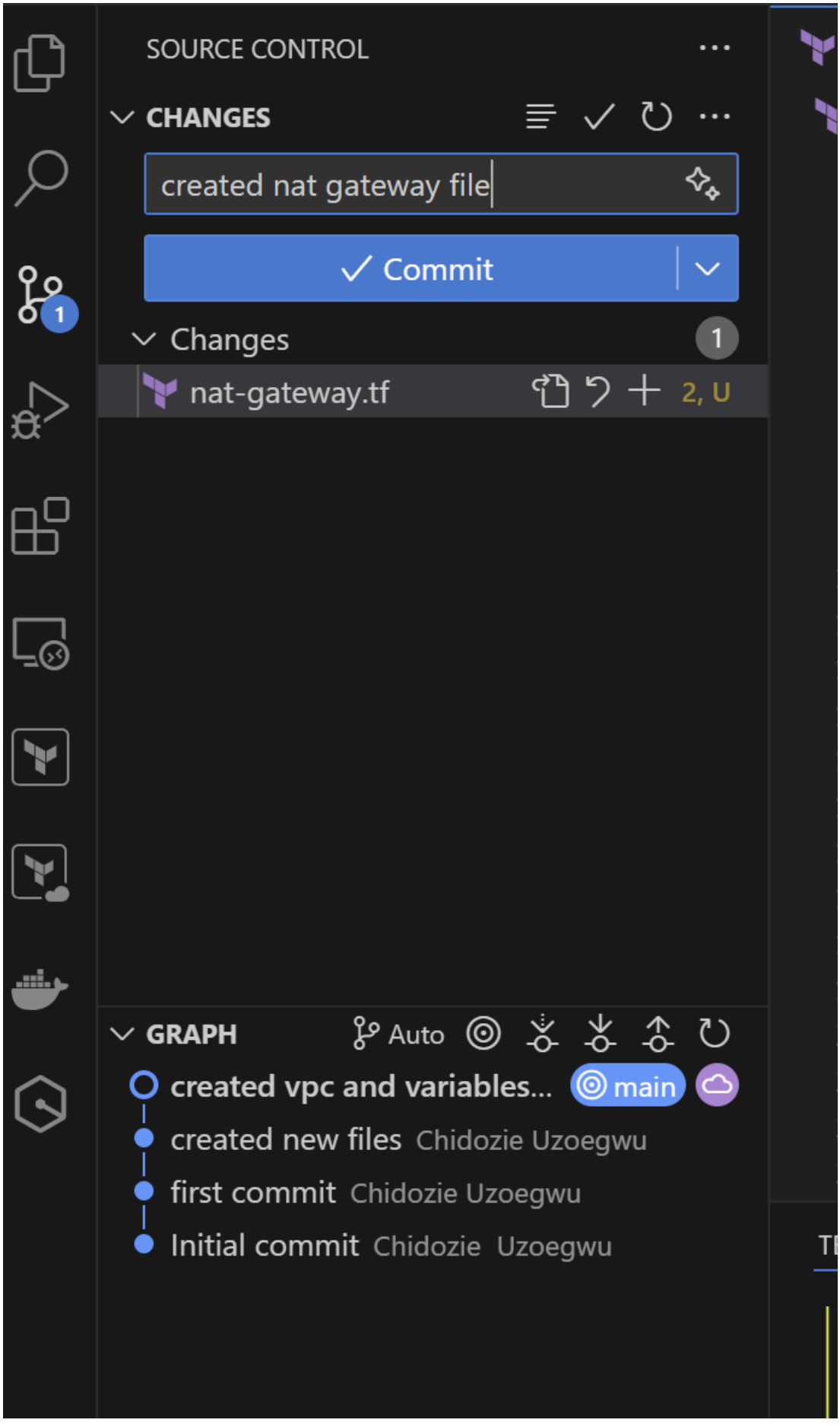


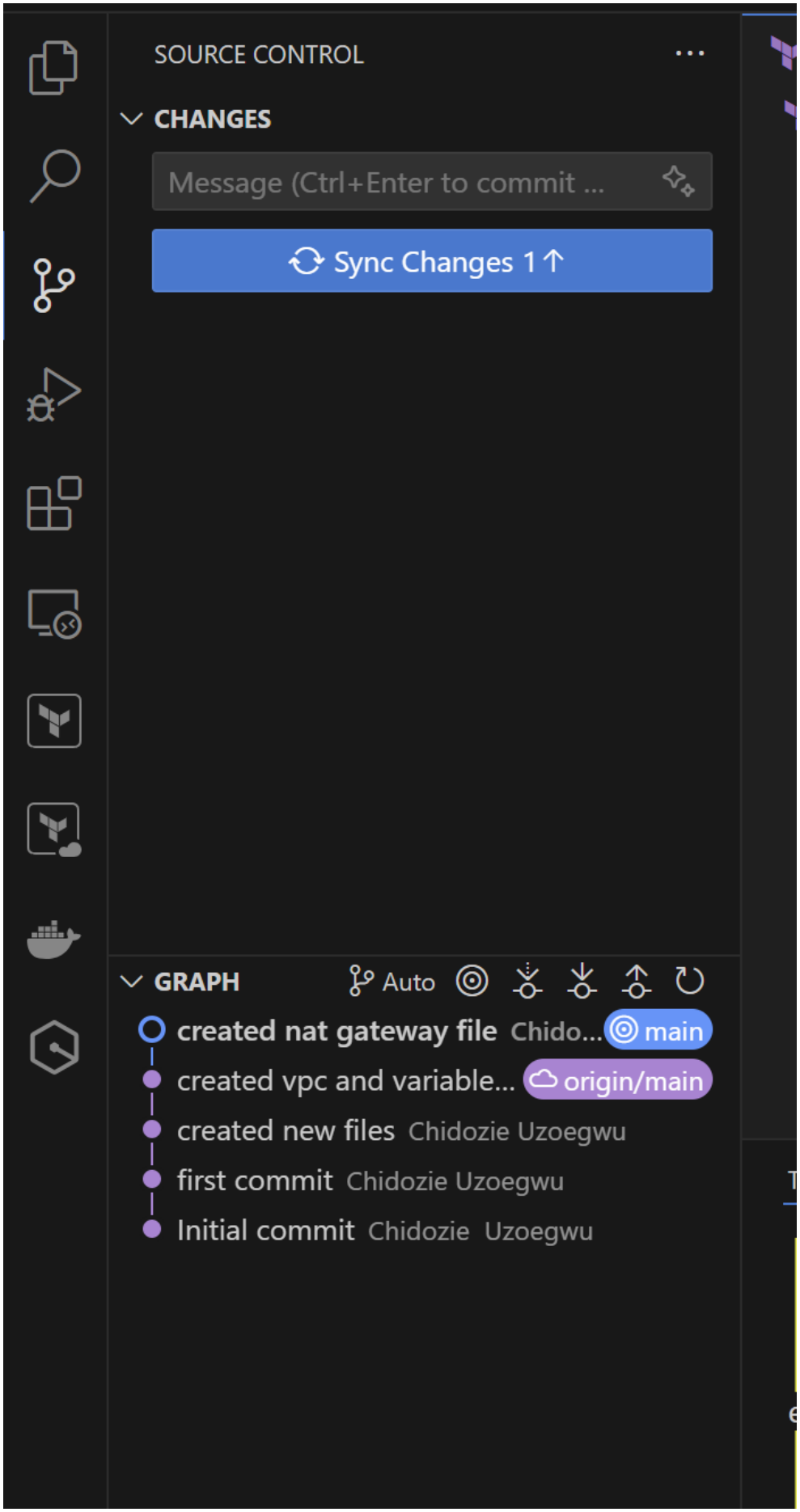


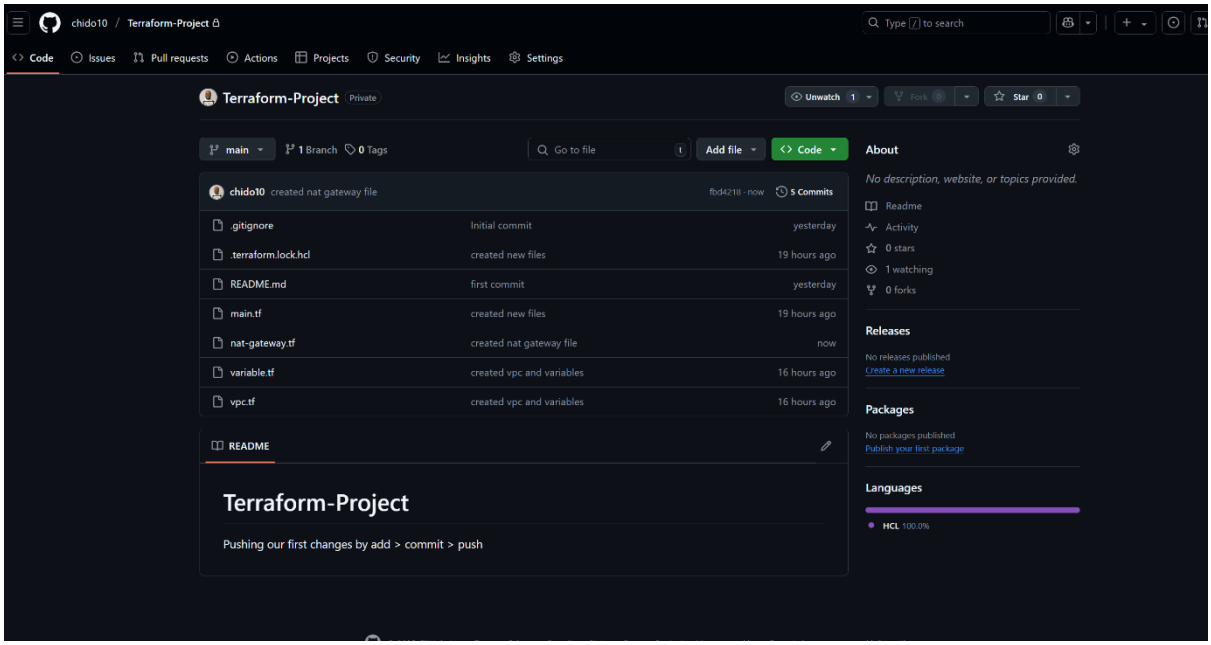


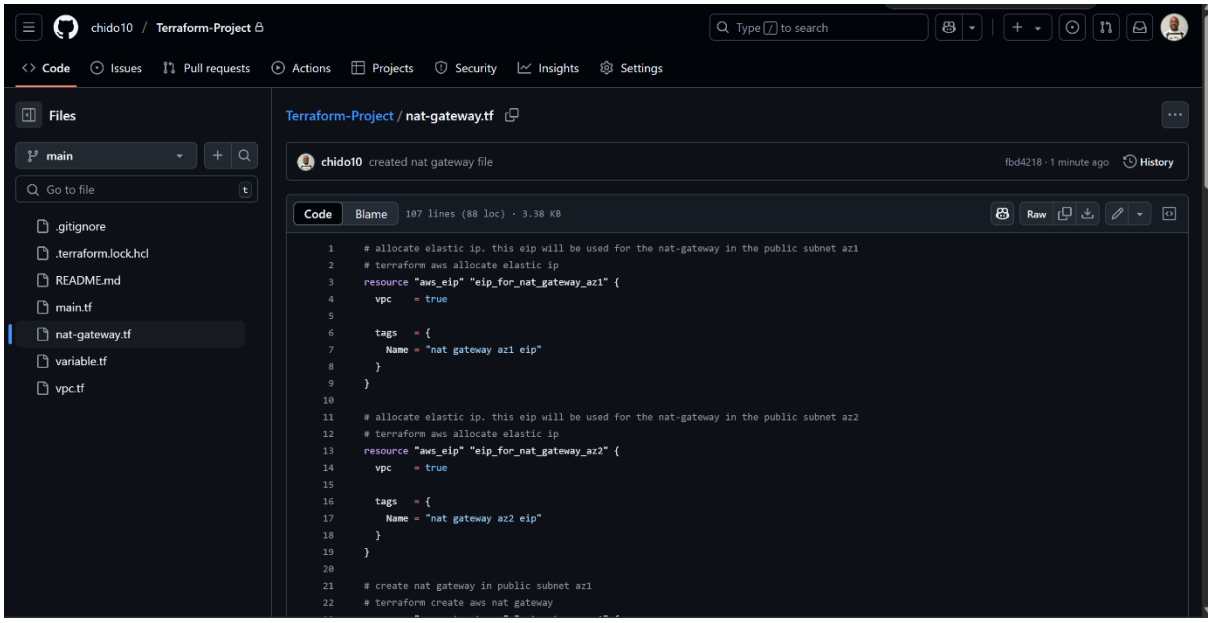


After confirming this we can as well commit and push to our GitHub and confirm it henceforth.

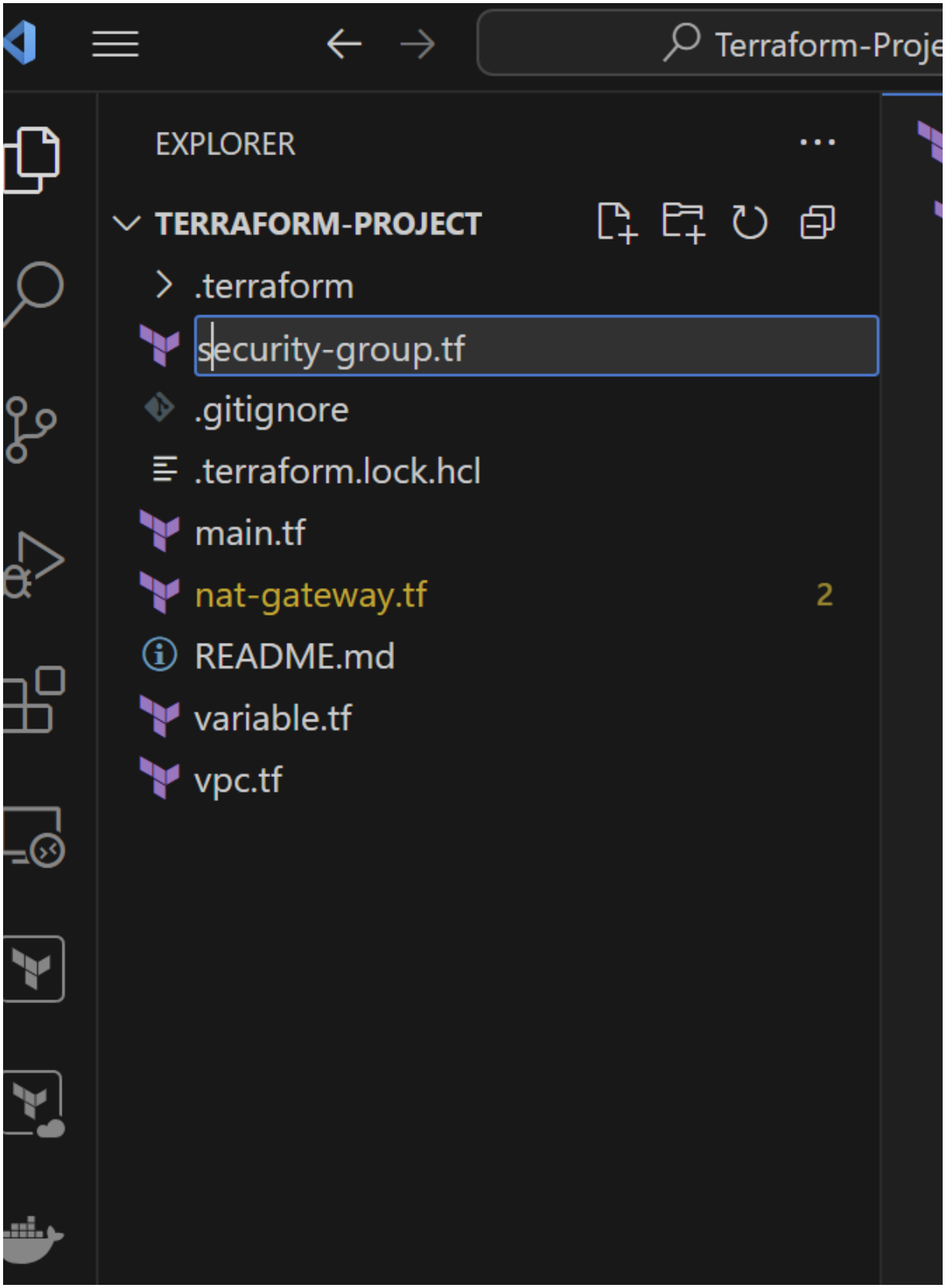

SOURCE CONTROL
CHANGES
created nat gateway file
Commit
Changes
1
nat-gateway.tf
2, U
GRAPH
Auto
created vpc and variables...
main
created new files Chidozie Uzoegwu
first commit Chidozie Uzoegwu
Initial commit Chidozie Uzoegwu


SOURCE CONTROL
CHANGES
Message (Ctrl+Enter to commit ...
Sync Changes 1↑
GRAPH
Auto
created nat gateway file Chido... main
created vpc and variable... origin/main
created new files Chidozie Uzoegwu
first commit Chidozie Uzoegwu
Initial commit Chidozie Uzoegwu





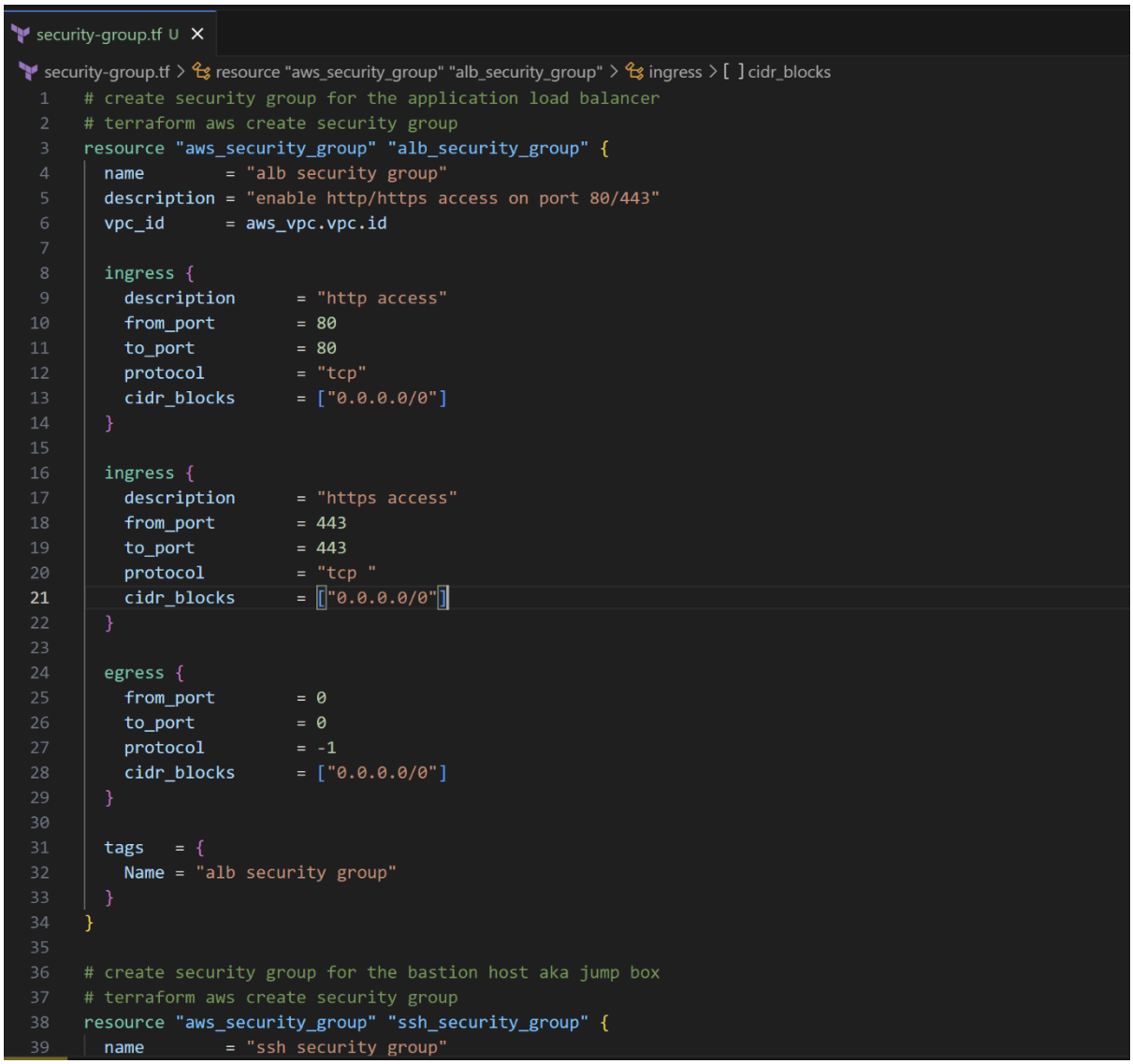
22. Create Security Group



Security Groups act as virtual firewalls for your EC2 instances, ALB, RDS, and SSH access. In this project, we created four security groups:

- Application Load Balancer (ALB) Security Group
- SSH (Bastion Host) Security Group
- Web Server Security Group
- Database Security Group

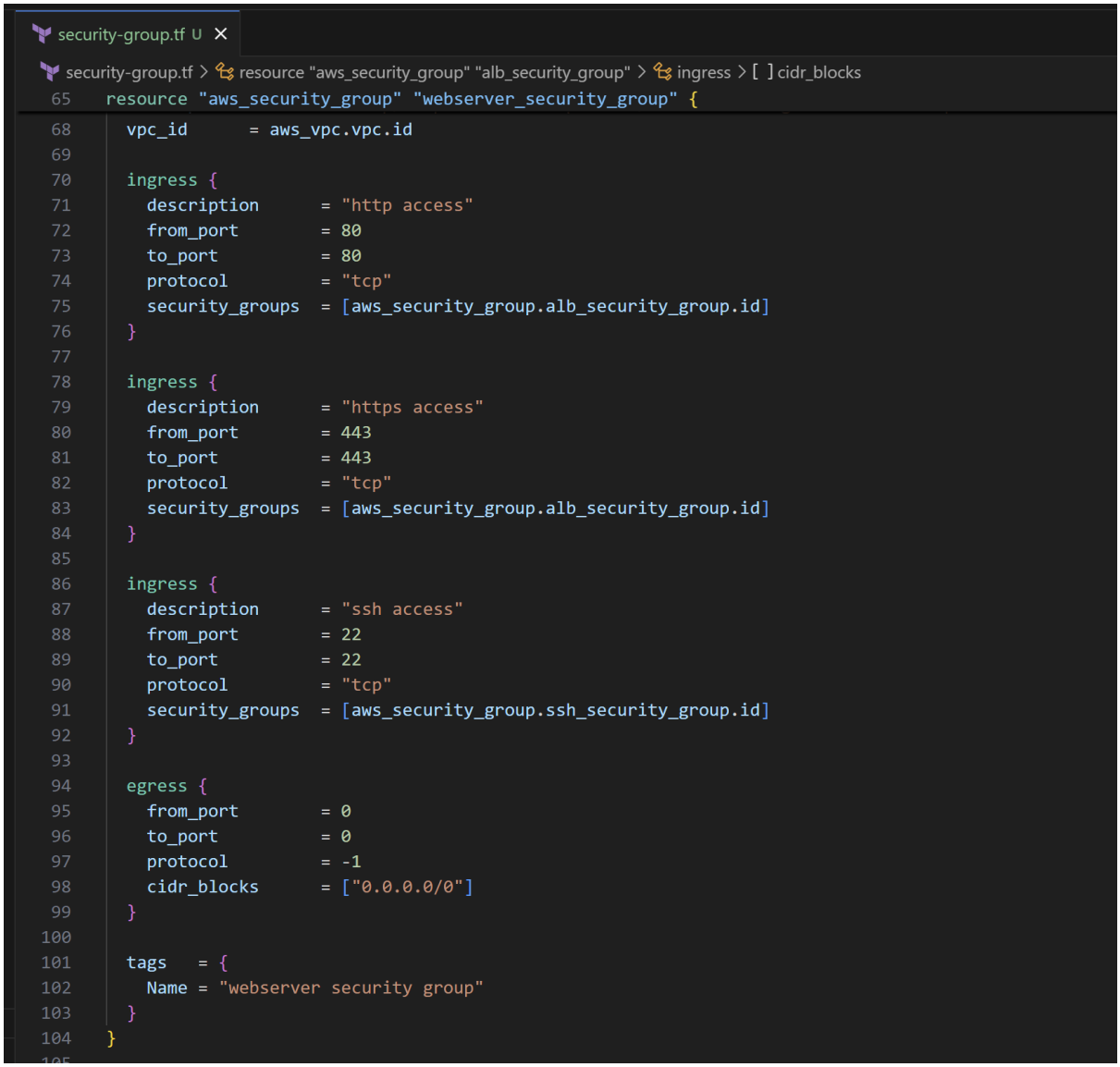
Each group controls which traffic is allowed in and out based on port, protocol, and source.

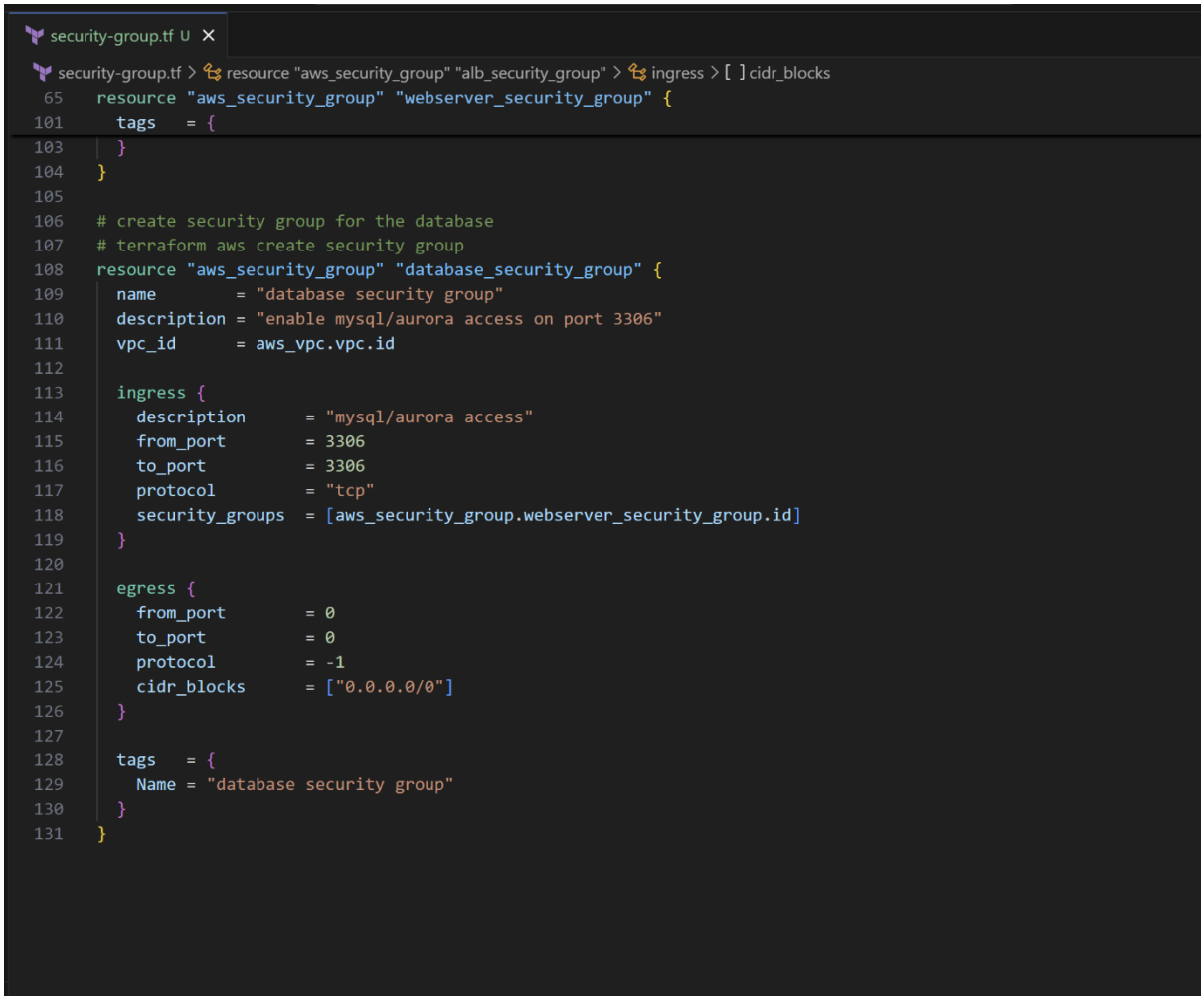


security-group.tf U

security-group.tf > resource "aws_security_group" "alb_security_group" > ingress > [ ] cidr_blocks

```
35
36  # create security group for the bastion host aka jump box
37  # terraform aws create security group
38  resource "aws_security_group" "ssh_security_group" {
39    name        = "ssh security group"
40    description = "enable ssh access on port 22"
41    vpc_id      = aws_vpc.vpc.id
42
43    ingress {
44      description      = "ssh access"
45      from_port        = 22
46      to_port          = 22
47      protocol         = "tcp"
48      cidr_blocks      = [var.ssh_location]
49    }
50
51    egress {
52      from_port        = 0
53      to_port          = 0
54      protocol         = -1
55      cidr_blocks      = ["0.0.0.0/0"]
56    }
57
58    tags   = {
59      Name = "ssh security group"
60    }
61  }
62
63  # create security group for the web server
64  # terraform aws create security group
65  resource "aws_security_group" "webserver_security_group" {
66    name        = "webserver security group"
67    description = "enable http/https access on port 80/443 via alb sg and access on port 22 via ssh sg"
68    vpc_id      = aws_vpc.vpc.id
69
70    ingress {
71      description      = "http access"
72      from_port        = 80
73      to_port          = 80
```


security-group.tf U
security-group.tf > resource "aws_security_group" "alb_security_group" > ingress > [ ] cidr_blocks
65 resource "aws_security_group" "webserver_security_group" {
68 vpc_id = aws_vpc.vpc.id
69
70 ingress {
71 description = "http access"
72 from_port = 80
73 to_port = 80
74 protocol = "tcp"
75 security_groups = [aws_security_group.alb_security_group.id]
76 }
77
78 ingress {
79 description = "https access"
80 from_port = 443
81 to_port = 443
82 protocol = "tcp"
83 security_groups = [aws_security_group.alb_security_group.id]
84 }
85
86 ingress {
87 description = "ssh access"
88 from_port = 22
89 to_port = 22
90 protocol = "tcp"
91 security_groups = [aws_security_group.ssh_security_group.id]
92 }
93
94 egress {
95 from_port = 0
96 to_port = 0
97 protocol = -1
98 cidr_blocks = ["0.0.0.0/0"]
99 }
100
101 tags = {
102 Name = "webserver security group"
103 }
104 }



ALB Security Group: This allows external traffic to the Application Load Balancer via HTTP (port 80) and HTTPS (port 443).

SSH Security Group (Bastion Host): Allows SSH access (port 22) only from a specific IP or range defined in the ssh_location variable.

Web Server Security Group: Allows HTTP, HTTPS from the ALB and SSH from the bastion host security group.

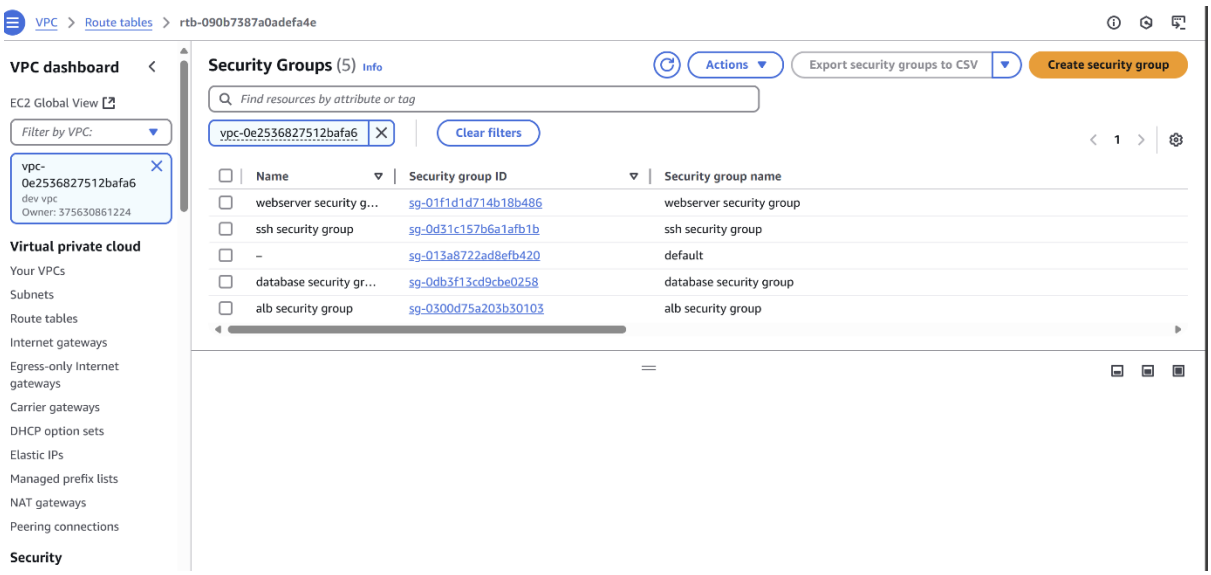
Database Security Group: Allows MySQL/Aurora access on port 3306 only from the Web Server Security Group.

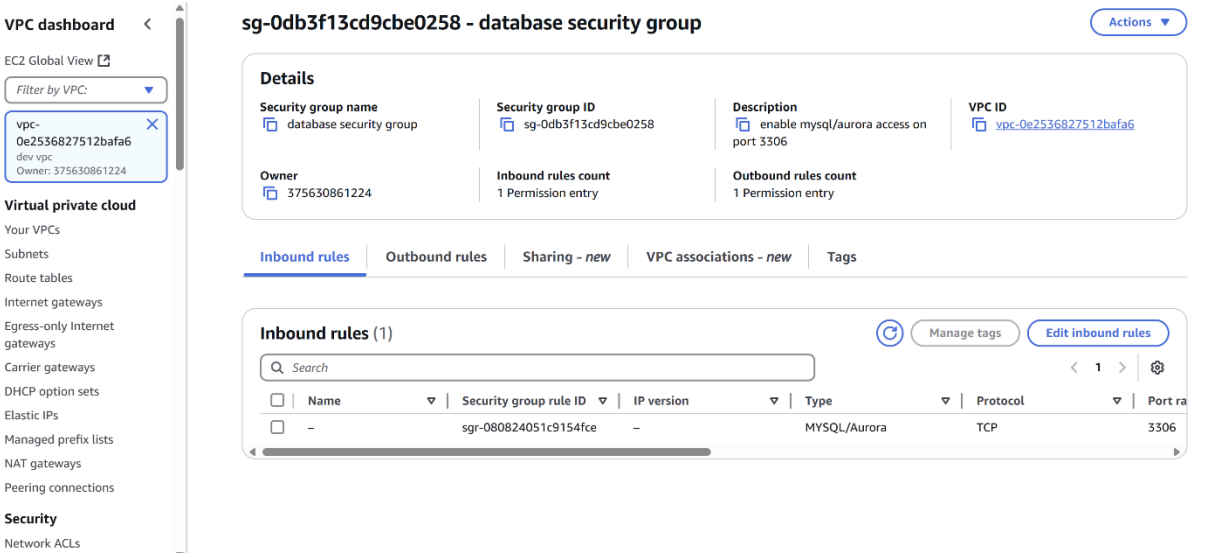
Terraform Apply Output and Verification

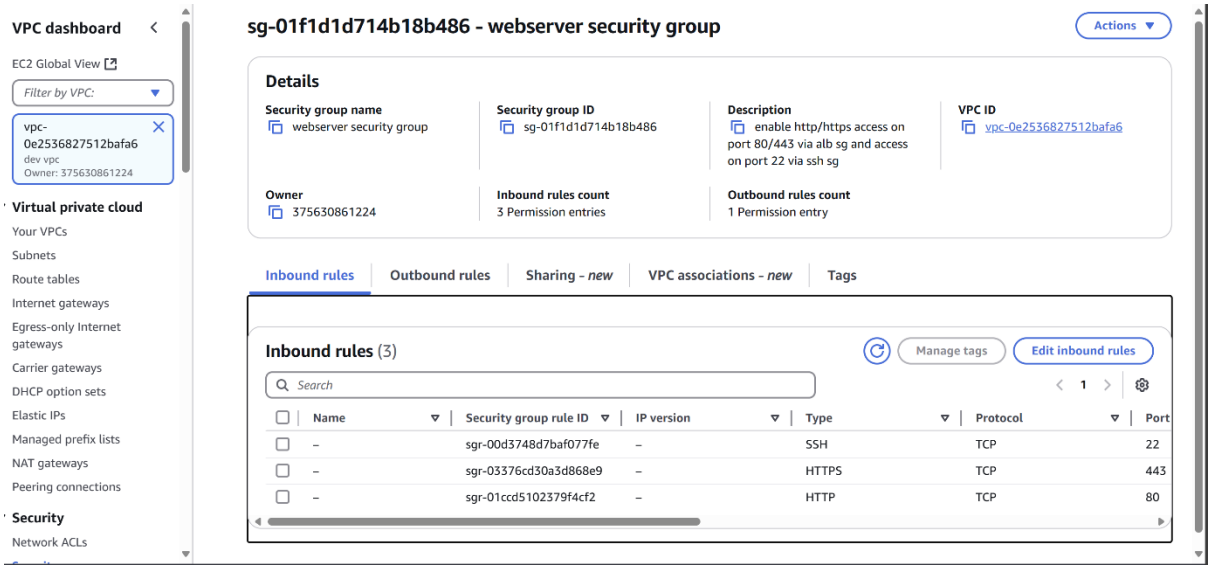
After applying the configuration using terraform apply, Terraform provisions all defined resources and prints out the results.

```
PROBLEMS 2   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   CODE REFERENCE LOG   COMMENTS

PS C:\Users\chido\Desktop\Terraform-Project> terraform apply
aws_vpc.vpc: Refreshing state... [id=vpc-0e2536827512bafa6]
aws_eip.eip_for_nat_gateway_az1: Refreshing state... [id=eipalloc-0353a7e76907b2048]
aws_eip.eip_for_nat_gateway_az2: Refreshing state... [id=eipalloc-0a3c649035e2ef090]
aws_subnet.private_app_subnet_az2: Refreshing state... [id=subnet-08bd1016021a40cf6]
aws_subnet.public_subnet_az1: Refreshing state... [id=subnet-040cfc8cb3be547dc]
aws_internet_gateway.internet_gateway: Refreshing state... [id=igw-0550600374ce71ab9]
aws_subnet.private_data_subnet_az1: Refreshing state... [id=subnet-0909acc2b7b6418c7]
aws_subnet.private_app_subnet_az1: Refreshing state... [id=subnet-0124bf873d71816b5]
aws_subnet.public_subnet_az2: Refreshing state... [id=subnet-02c07d8b8e648f098]
aws_subnet.private_data_subnet_az2: Refreshing state... [id=subnet-0f84f7c3f42127c47]
aws_route_table.public_route_table: Refreshing state... [id=rtb-0aad8ffe33ba8d07d]
aws_nat_gateway.nat_gateway_az1: Refreshing state... [id=nat-02c22a8fe111beed3]
aws_nat_gateway.nat_gateway_az2: Refreshing state... [id=nat-0b90246efb7c4095a]
aws_route_table.private_route_table_az1: Refreshing state... [id=rtb-06a2fffce49954f88]
aws_route_table_association.public_subnet_2_route_table_association: Refreshing state... [id=rtbassoc-06b480f92e14ecdc7]
aws_route_table_association.public_subnet_az1_route_table_association: Refreshing state... [id=rtbassoc-087ca3c385245022c]
aws_route_table.private_route_table_az2: Refreshing state... [id=rtb-090b7387a0adefa4e]
aws_route_table_association.private_data_subnet_az1_route_table_az1_association: Refreshing state... [id=rtbassoc-0315320328a7ef0df]
aws_route_table_association.private_app_subnet_az1_route_table_az1_association: Refreshing state... [id=rtbassoc-0ec84a573ffedf081]
aws_route_table_association.private_app_subnet_az2_route_table_az2_association: Refreshing state... [id=rtbassoc-01dd1209bc8bbb9c1]
aws_route_table_association.private_data_subnet_az2_route_table_az2_association: Refreshing state... [id=rtbassoc-01d7257cfed278331]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following sym
  + create
```

In the AWS Console, we can verify that all security groups have been created and are correctly attached to the right resources:

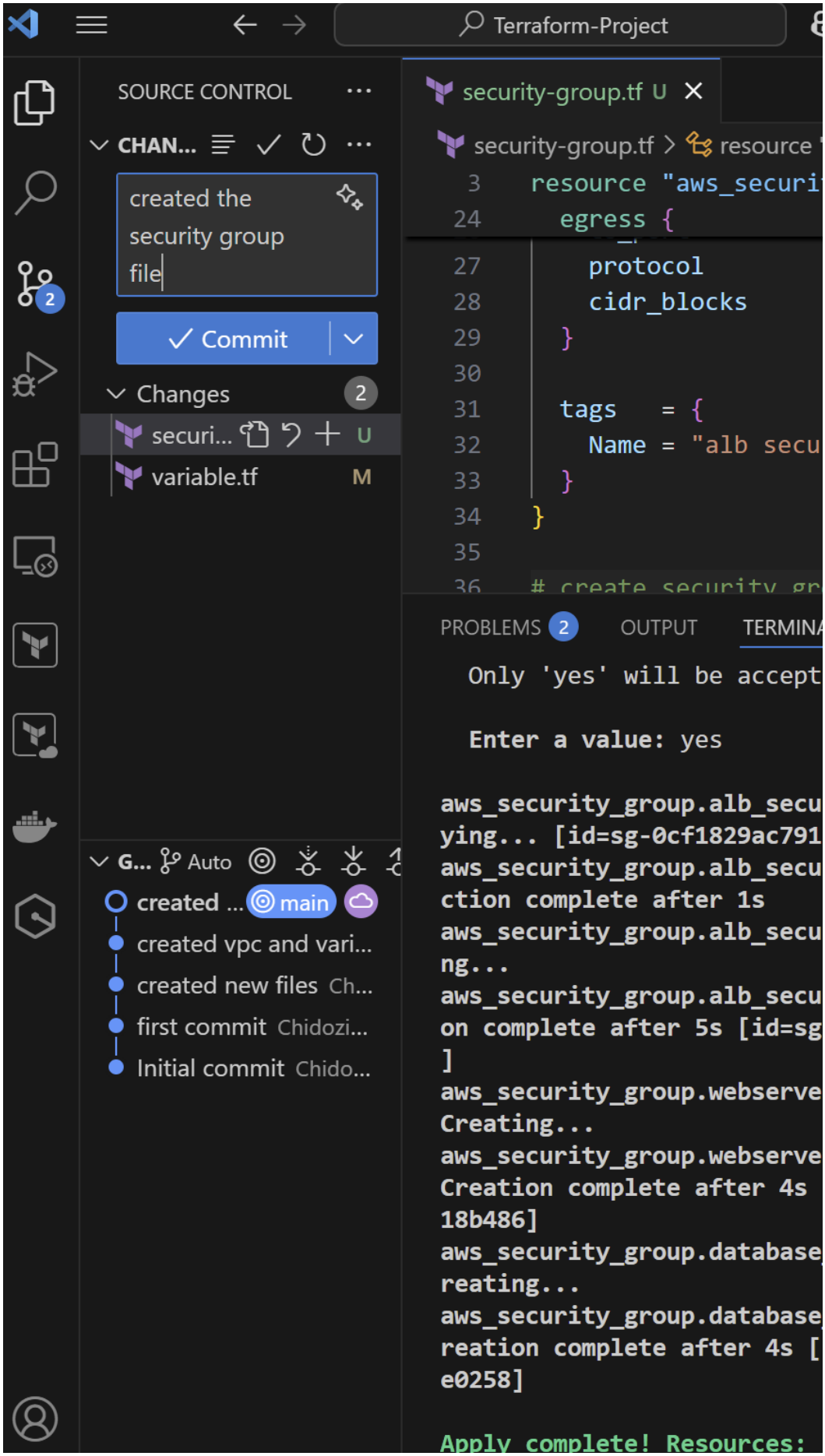


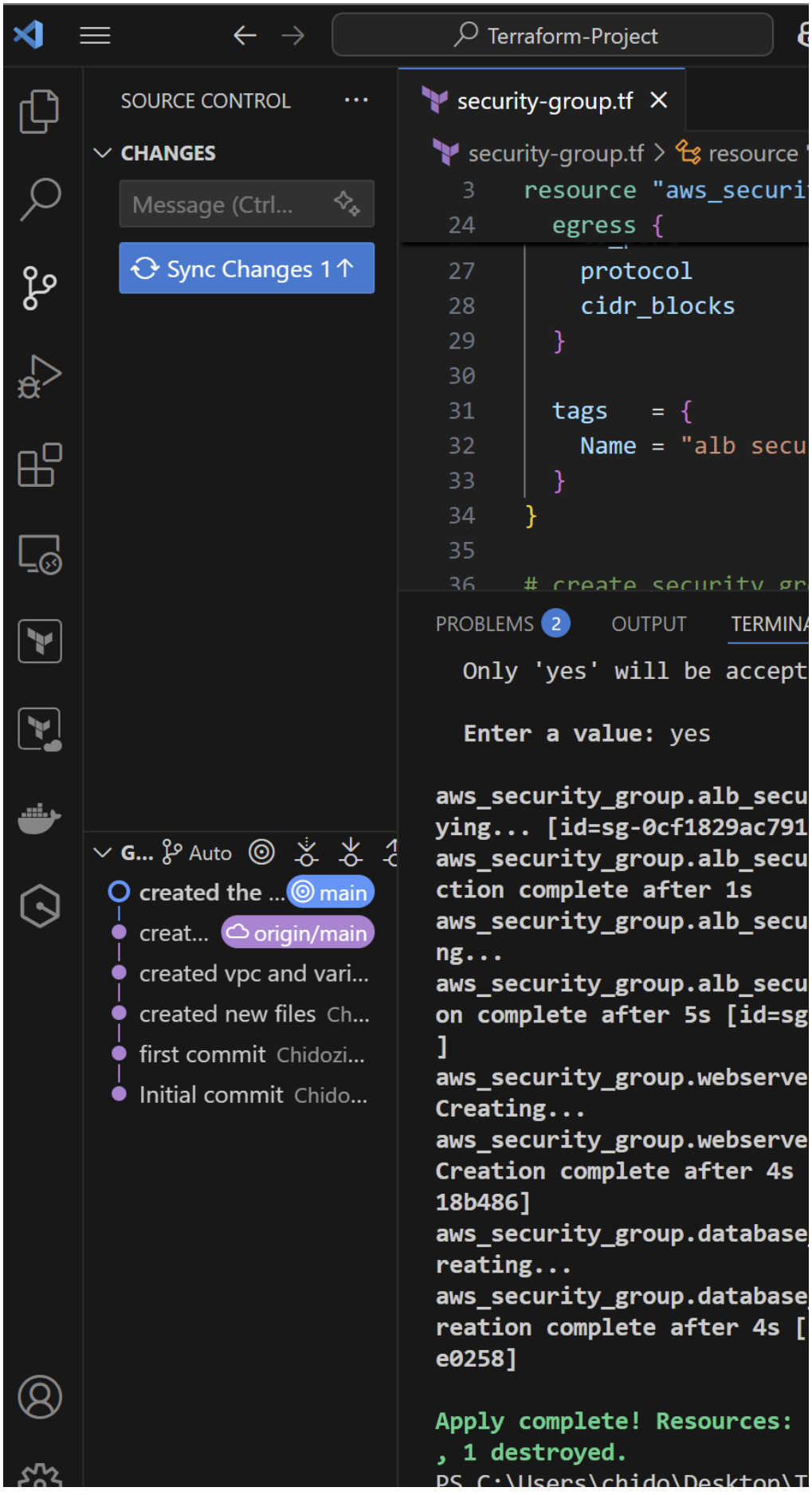


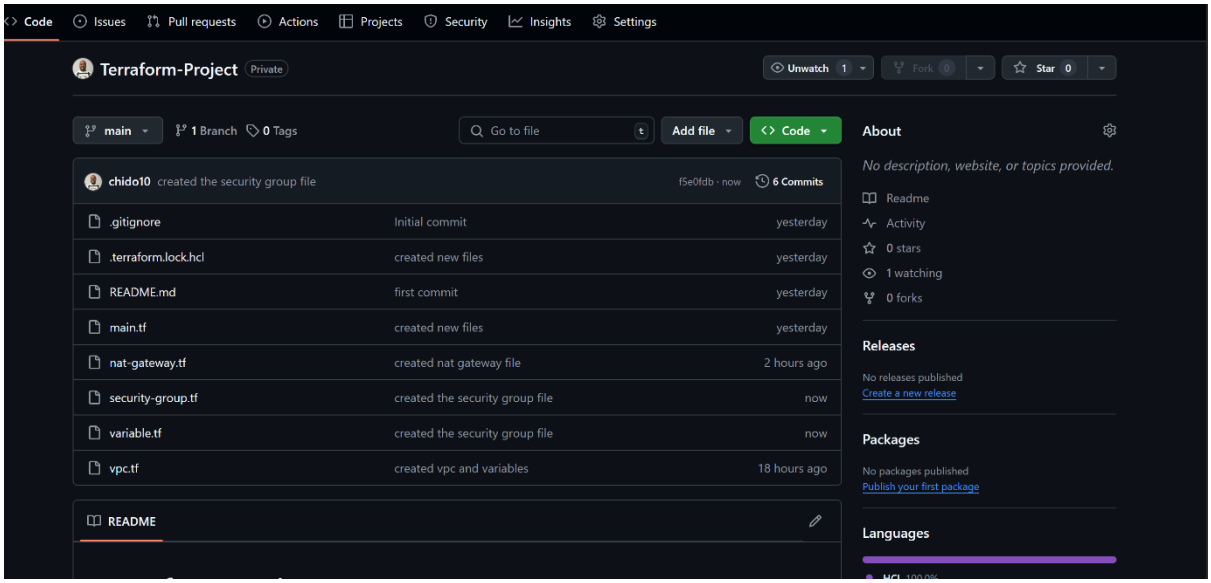


Commit to GitHub

After creating the security-group.tf file, changes were committed to the Git repository with an appropriate message.


Terraform-Project
SOURCE CONTROL
CHAN...
created the security group file
Commit
Changes 2
securi... U
variable.tf M
security-group.tf U
security-group.tf > resource
3 resource "aws_securi
24 egress {
27 protocol
28 cidr_blocks
29 }
30
31 tags = {
32 Name = "alb secu
33 }
34 }
35
36 # create security gr
PROBLEMS 2
OUTPUT
TERMINA
Only 'yes' will be accept
Enter a value: yes
aws_security_group.alb_secu
ying... [id=sg-0cf1829ac791
aws_security_group.alb_secu
ction complete after 1s
aws_security_group.alb_secu
ng...
aws_security_group.alb_secu
on complete after 5s [id=sg
]
aws_security_group.webserve
Creating...
aws_security_group.webserve
Creation complete after 4s
18b486]
aws_security_group.database
reating...
aws_security_group.database
reation complete after 4s [
e0258]
Apply complete! Resources:
G... Auto
created ... main
created vpc and vari...
created new files Ch...
first commit Chidozi...
Initial commit Chido...


Terraform-Project
SOURCE CONTROL
CHANGES
Message (Ctrl...
Sync Changes 1
security-group.tf
security-group.tf > resource
3 resource "aws_securi
24 egress {
27 protocol
28 cidr_blocks
29 }
30
31 tags = {
32 Name = "alb secu
33 }
34 }
35
36 # create security gr
PROBLEMS 2
OUTPUT
TERMINA
Only 'yes' will be accept
Enter a value: yes
aws_security_group.alb_secu
ying... [id=sg-0cf1829ac791
aws_security_group.alb_secu
ction complete after 1s
aws_security_group.alb_secu
ng...
aws_security_group.alb_secu
on complete after 5s [id=sg
]
aws_security_group.webserve
Creating...
aws_security_group.webserve
Creation complete after 4s
18b486]
aws_security_group.database
reating...
aws_security_group.database
reation complete after 4s [
e0258]
Apply complete! Resources:
, 1 destroyed.
PS C:\Users\chido\Desktop\T
G... Auto
created the ... main
creat... origin/main
created vpc and vari...
created new files Ch...
first commit Chidozi...
Initial commit Chido...



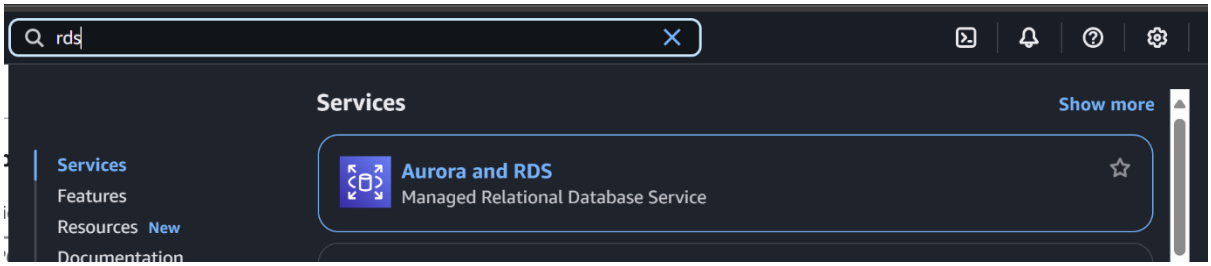
23. Create RDS (Database)

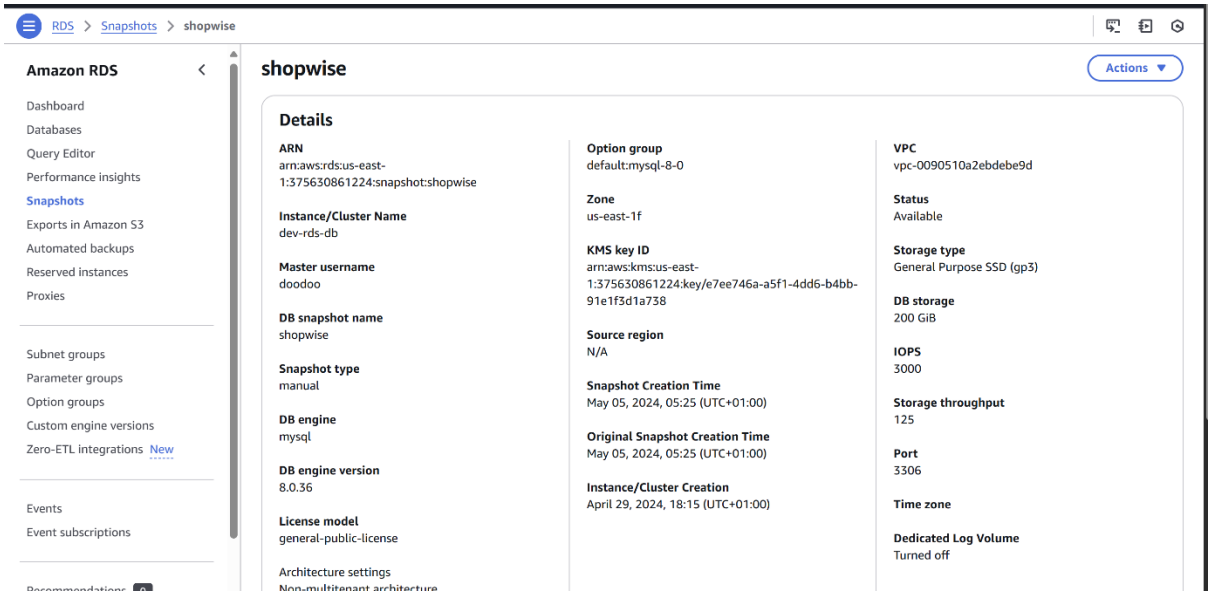
In this step, we will provision an Amazon RDS database instance using Terraform, restored from an existing manual snapshot named “shopwise.” This approach ensures that your eCommerce website’s database is consistently deployed with the same data and settings, facilitating disaster recovery and simplified configuration management.

Step 1: Search for RDS in the AWS Console

In the AWS Management Console, go to the search bar and type RDS.

Select “Aurora and RDS” from the services list to open the RDS dashboard.





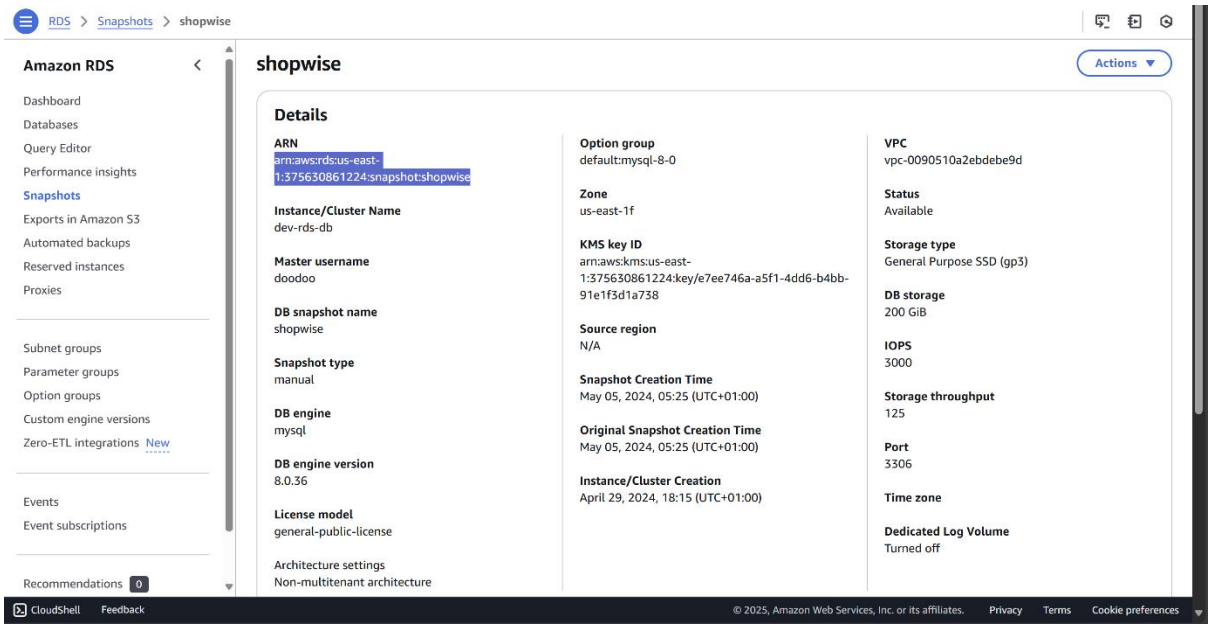
(Here, the main RDS console page is displayed, showing an overview of databases, snapshots, and quick links.)

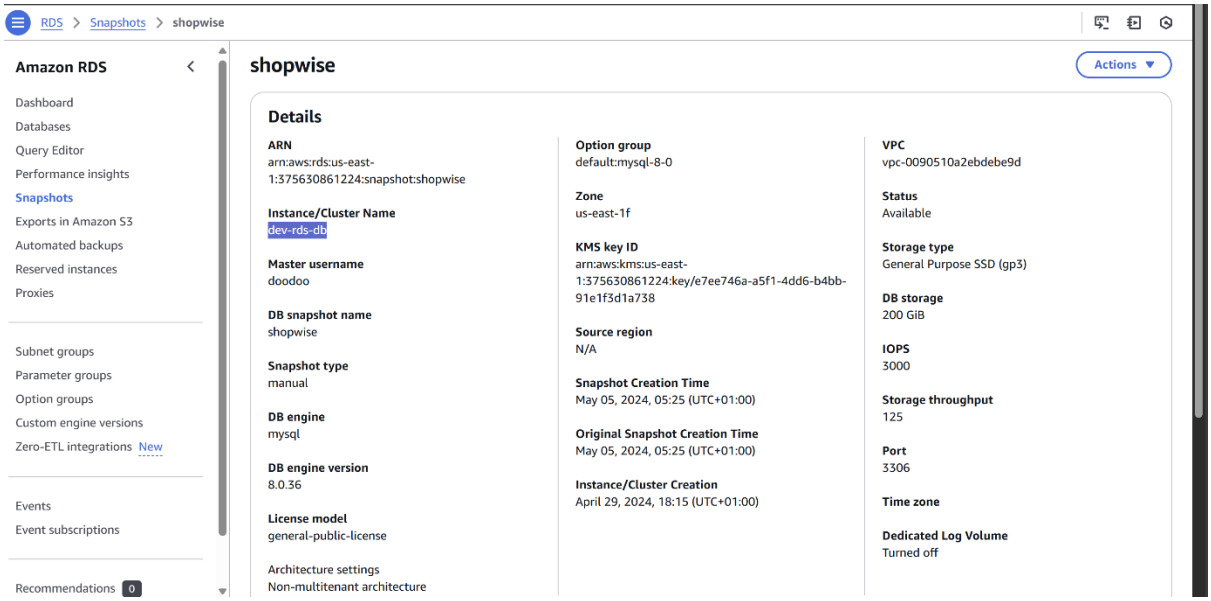
Step 2: Identify the Manual Snapshot

From the left navigation pane, click on “Snapshots.”

Find the manual snapshot named “shopwise.”

Click on it to see its ARN (Amazon Resource Name), size, encryption status, and other details.





(This screenshot shows the snapshot's status and region. You can confirm encryption and creation date as well.)

Step 3: Declare RDS Variables

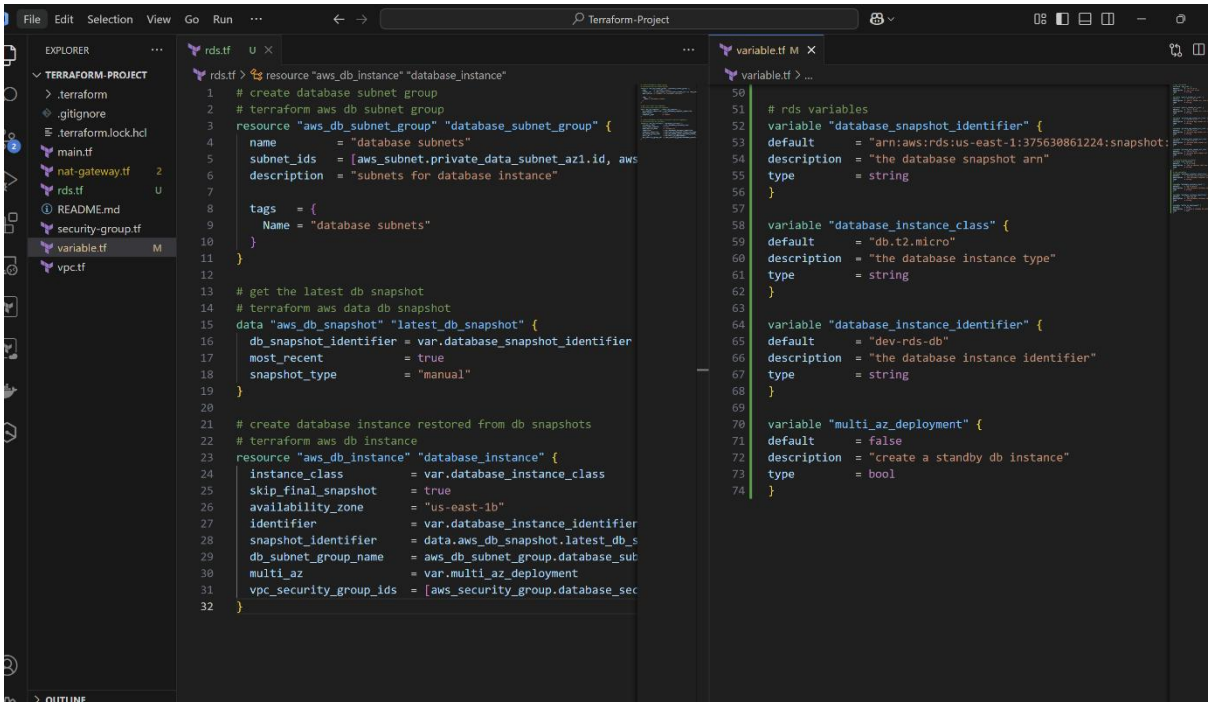
Before proceeding, ensure that your Terraform project includes the relevant variables in variable.tf. These variables hold references such as:

Snapshot ARN

Database instance class

Multi-AZ deployment preference

Database instance identifier



(This screenshot shows the variables you defined, such as database_snapshot_identifier, database_instance_class, and more.)

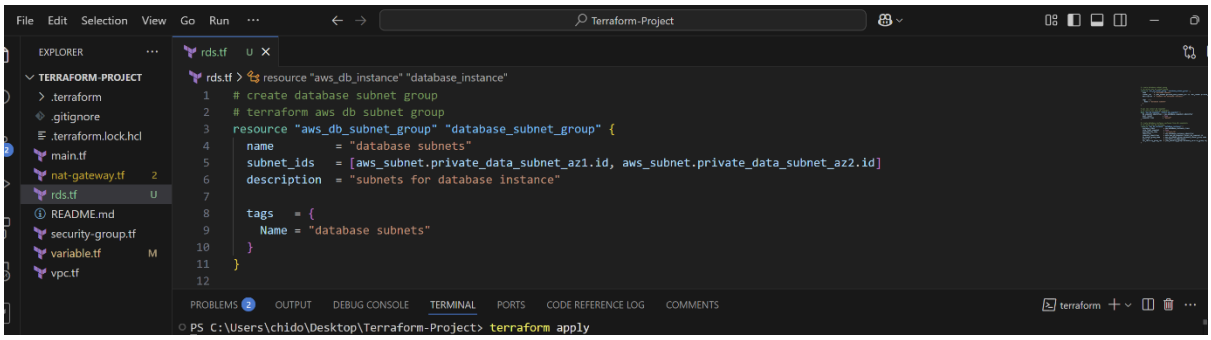
Step 4: Configure RDS (Restore) in Terraform

You will have a file (commonly rds.tf) where you:

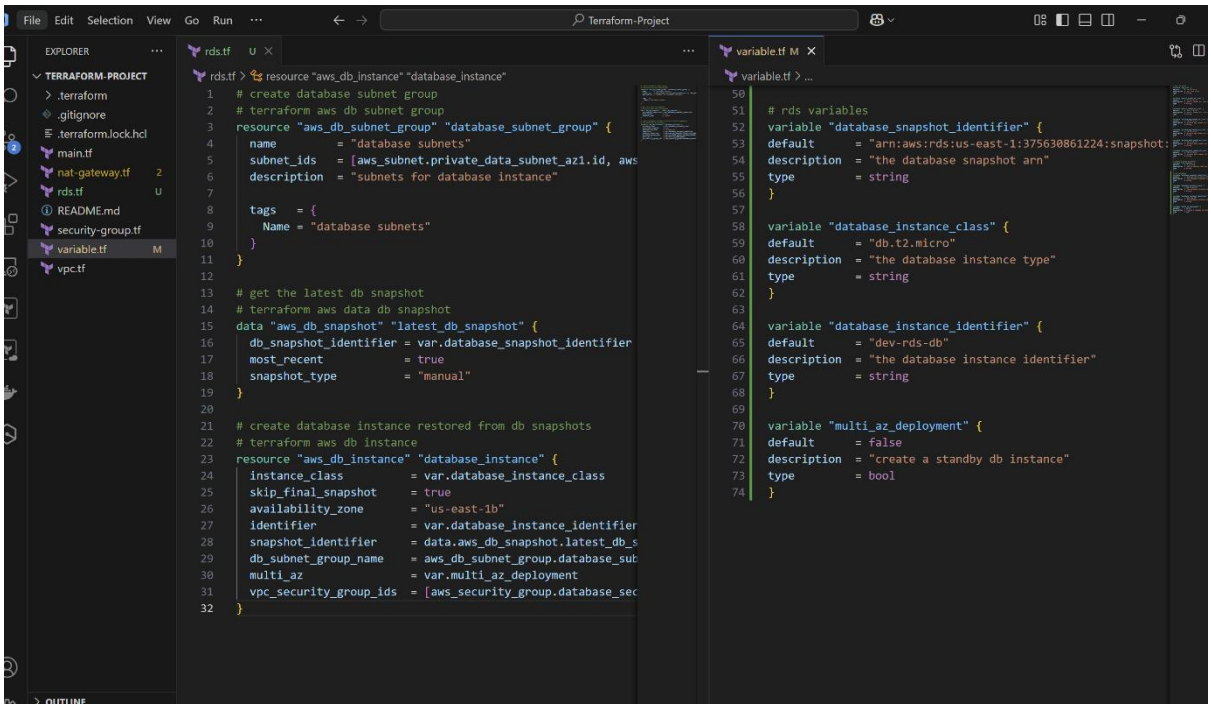
Define a Database Subnet Group to specify which private subnets the RDS instance will reside in.

Refer to the Snapshot using a Terraform data source to fetch the snapshot's details.

Create the RDS Instance that restores from the snapshot. Ensure the instance class supports encryption at rest (for example, db.t3. micro instead of db.t2.micro).



(In this screenshot, your RDS configuration references the subnet group and uses the snapshot data source.)



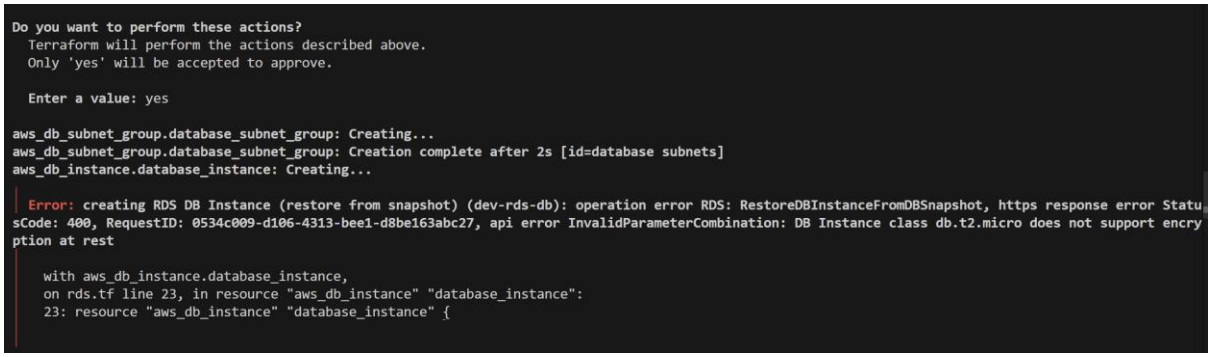
Step 5: Run terraform apply

Open your terminal in the directory containing your Terraform files.

Run the command terraform apply and review the plan.

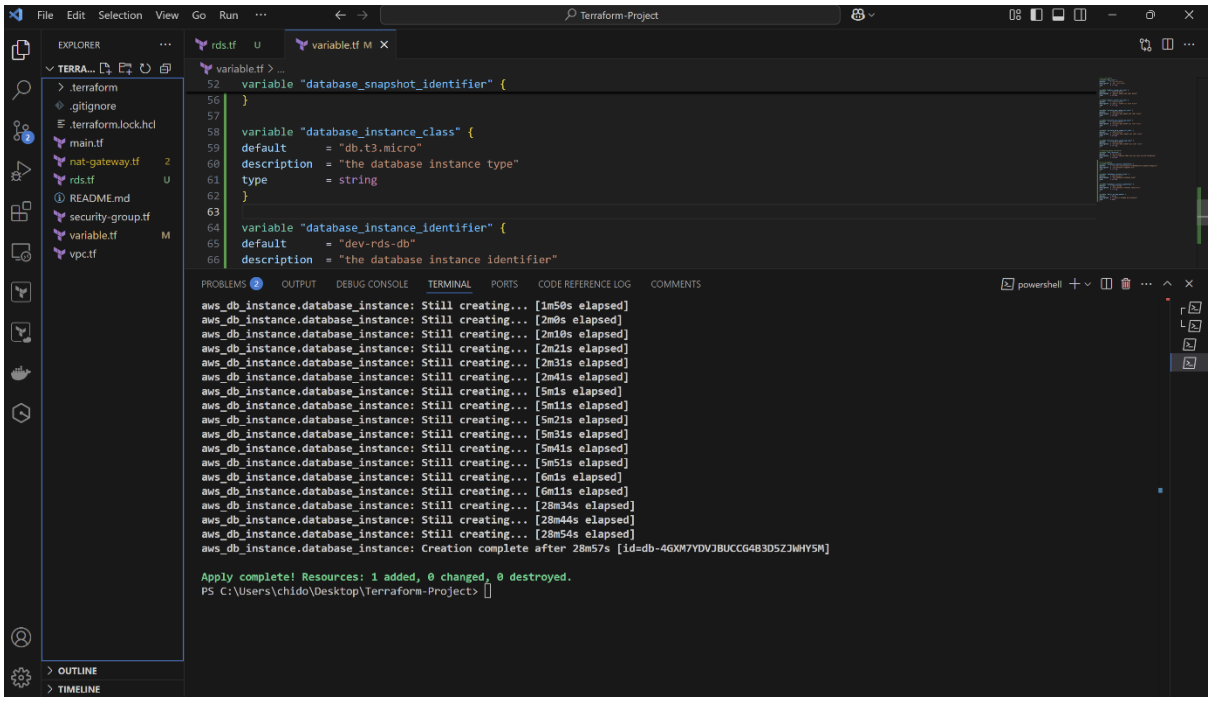
Confirm to proceed—Terraform will start creating the RDS instance, referencing the snapshot you specified.

Important: If you encounter an error about db.t2.micro not supporting encryption, change the instance class to something like db.t3.micro or another class that supports encryption at rest.



```
variable "database_instance_class" {
default       = "db.t3.micro"
description   = "the database instance type"
type          = string
}
```

```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_db_instance.database_instance: Creating...
aws_db_instance.database_instance: Still creating... [10s elapsed]
aws_db_instance.database_instance: Still creating... [20s elapsed]
aws_db_instance.database_instance: Still creating... [30s elapsed]
aws_db_instance.database_instance: Still creating... [40s elapsed]
aws_db_instance.database_instance: Still creating... [50s elapsed]
aws_db_instance.database_instance: Still creating... [1m0s elapsed]
aws_db_instance.database_instance: Still creating... [1m10s elapsed]
aws_db_instance.database_instance: Still creating... [1m20s elapsed]
aws_db_instance.database_instance: Still creating... [1m30s elapsed]
aws_db_instance.database_instance: Still creating... [1m40s elapsed]
aws_db_instance.database_instance: Still creating... [1m50s elapsed]
aws_db_instance.database_instance: Still creating... [2m0s elapsed]
aws_db_instance.database_instance: Still creating... [2m10s elapsed]
aws_db_instance.database_instance: Still creating... [2m21s elapsed]
aws_db_instance.database_instance: Still creating... [2m31s elapsed]
aws_db_instance.database_instance: Still creating... [2m41s elapsed]
```



Step 6: Verify the Restored DB in the AWS Console

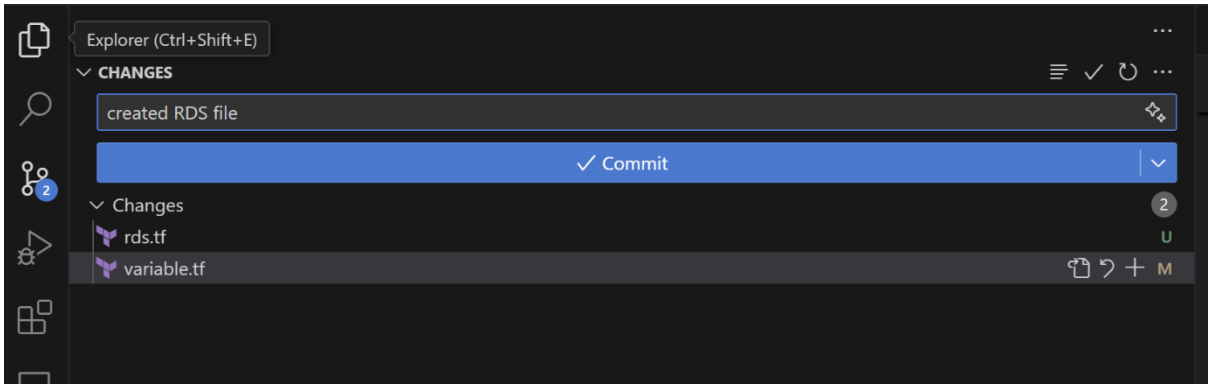
Once the apply is finished without errors, go back to the AWS RDS console:

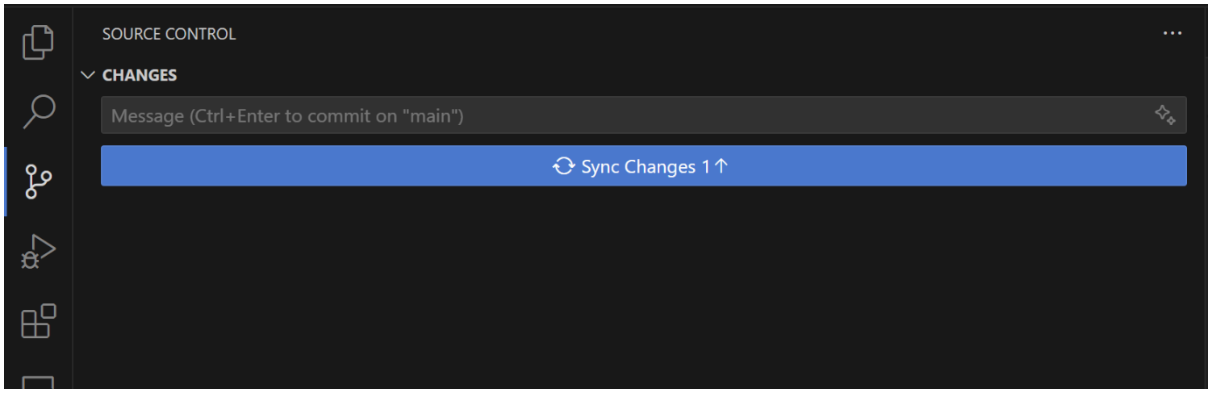
Under Databases, the new instance should be listed.

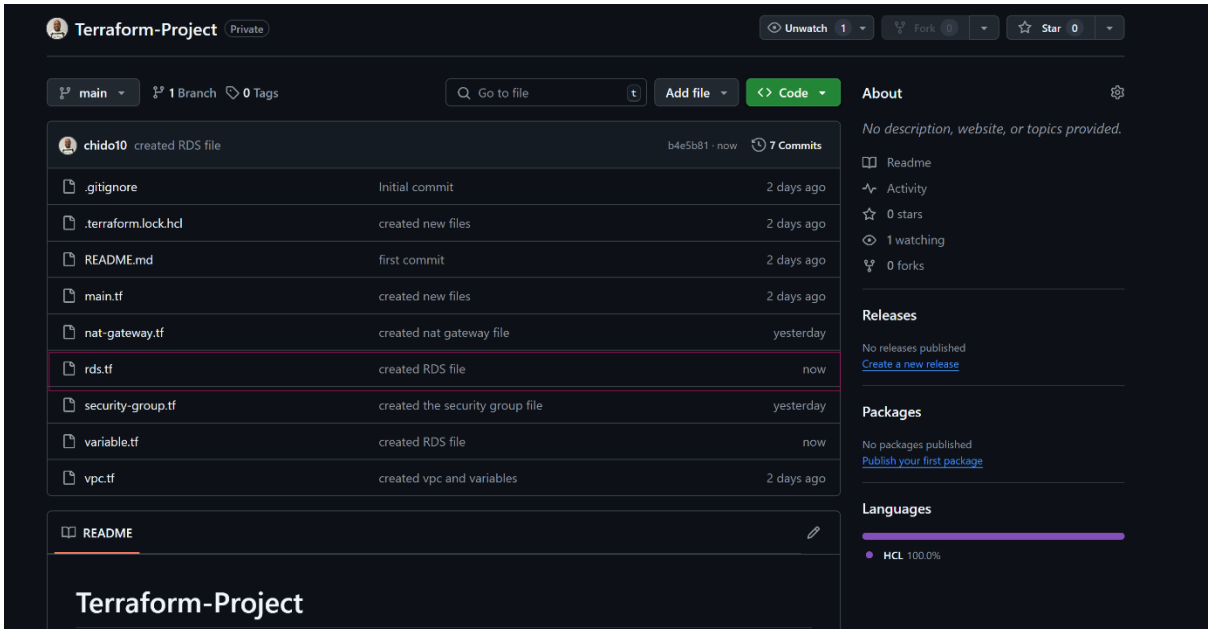
Confirm its status is “Available.”

Check the Connectivity & security tab to verify it’s using your intended subnet group, security group, and that it matches the region and availability zone you specified.

Once satisfied, we committed and pushed the changes to our GitHub.







Conclusion

Restoring Amazon RDS from a snapshot streamlines database provisioning by leveraging existing data and configurations. This method reduces manual setup steps and helps maintain consistency across multiple environments (for example, Dev, Staging, and Production). Always ensure the instance class you select supports the encryption requirements of your snapshot.

24. Create Application Load Balancer (ALB)

In this section, we will provision an Application Load Balancer (ALB) to distribute incoming HTTP/HTTPS traffic across multiple EC2 instances in different Availability Zones, ensuring high availability and efficient traffic management for our eCommerce application.

An Application Load Balancer operates at Layer 7 (the application layer) and provides advanced request routing. In our setup, we will:

- Create an ALB resource.
- Define a target group for our EC2 instances.
- Create listeners to handle HTTP (and optionally HTTPS) traffic.
- Output the ALB DNS name.

The ALB will be placed in public subnets and forward traffic to the private subnets where our application servers (EC2 instances) reside.

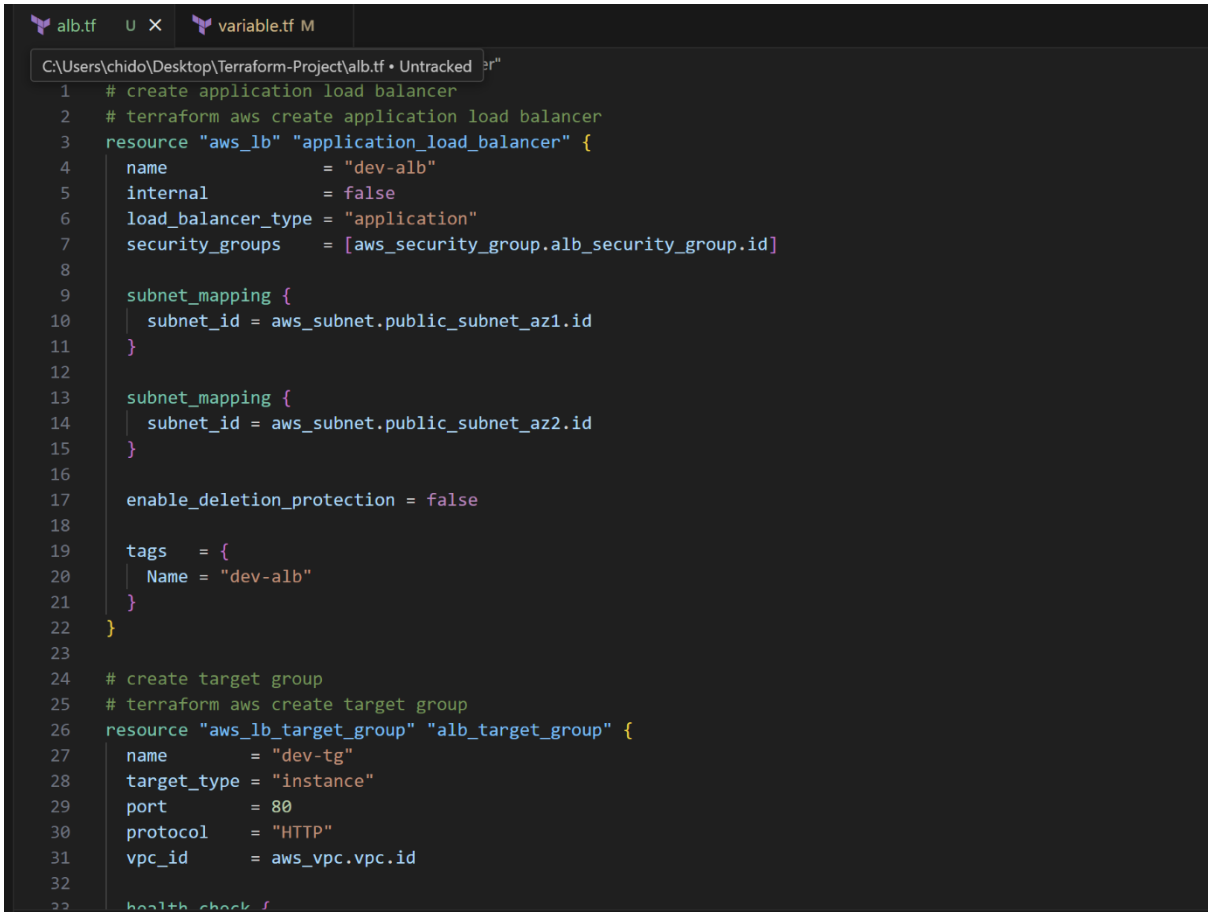
ALB Resource: The ALB is declared with attributes such as name, type, subnets, security groups, and tags. We ensure it is internet-facing so external clients can reach our application. For example, we set:

internal = false for public exposure.

load_balancer_type = "application" to specify an Application Load Balancer.

security_groups to allow inbound traffic on ports 80 (HTTP) or 443 (HTTPS).

subnets referencing the VPC's public subnets in at least two Availability Zones.



Target Group

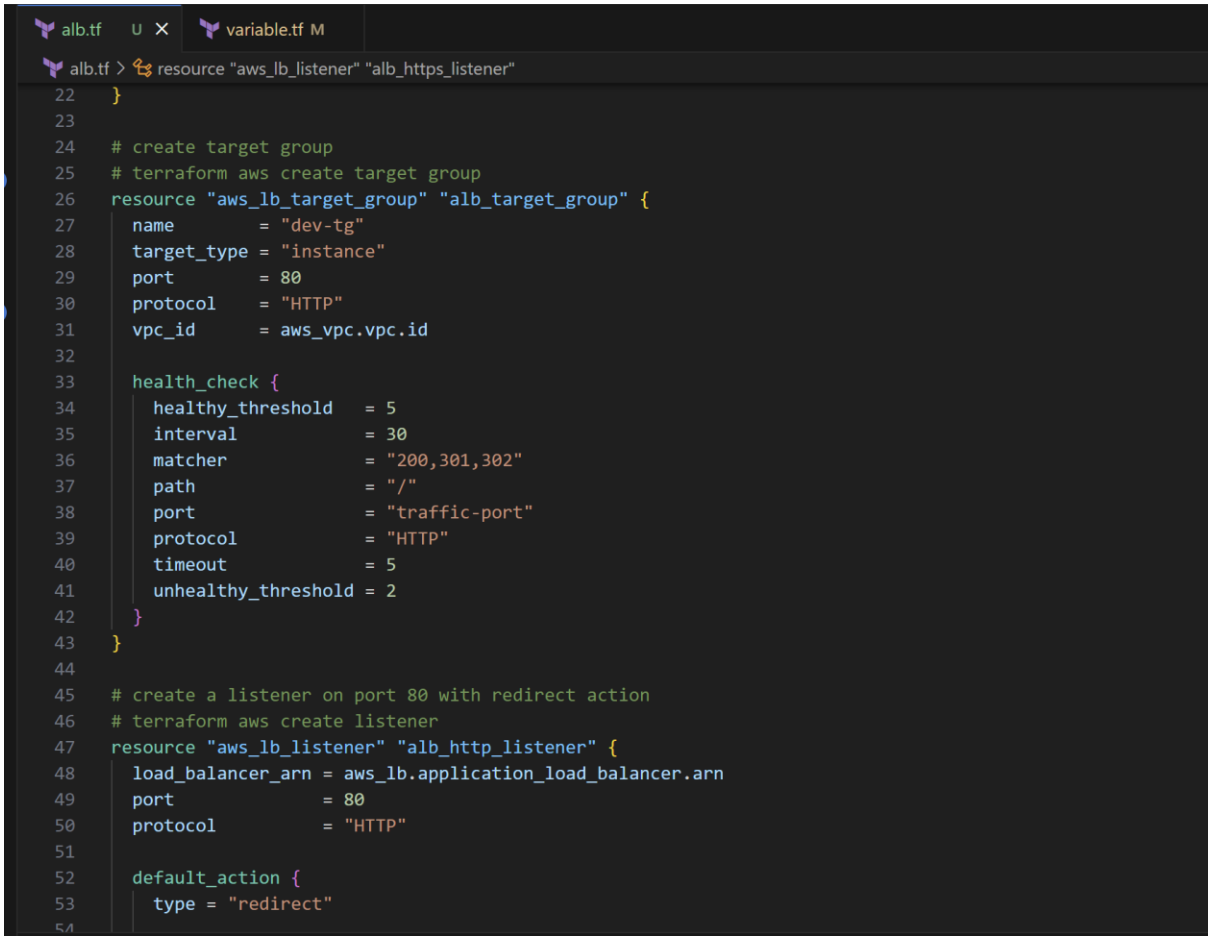
Next, we create a target group to register the backend EC2 instances. Key attributes:

Name and Protocol/Port (e.g., HTTP on port 80).

VPC ID that matches our existing VPC.

Target Type set to either instance or ip. Here, we use instance to register EC2 instances.

Health Check configuration (optional) to specify how ALB checks instance health (health check path, interval, etc.).



Listener

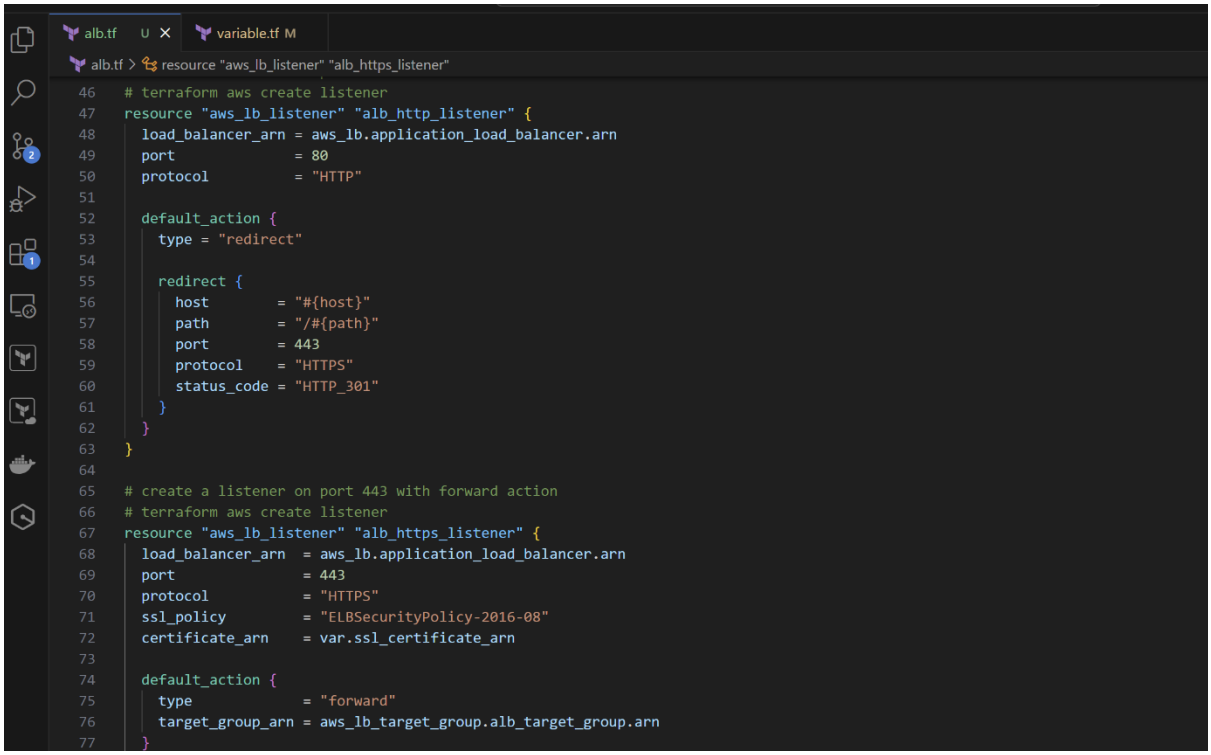
A listener is what actually receives client requests on a specified port (for example, port 80 for HTTP). Within the listener configuration, we define an action that forwards requests to the target group:

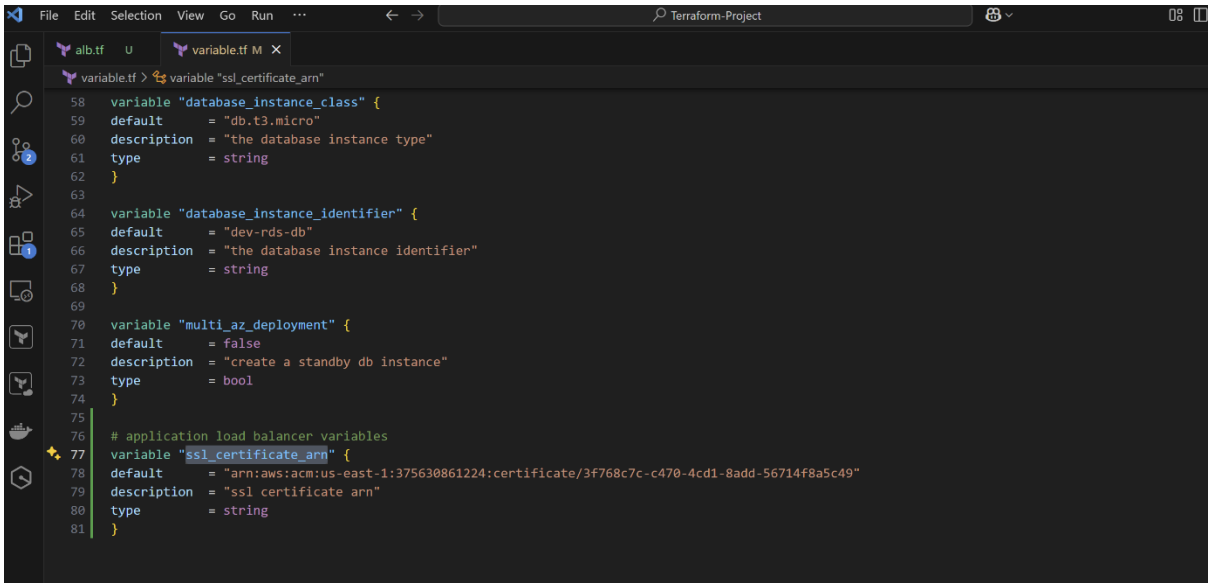
port = 80

protocol = "HTTP"

default_action = forward to the target group created above

If you need HTTPS, you can add another listener on port 443 and associate an SSL certificate (from ACM) with it.





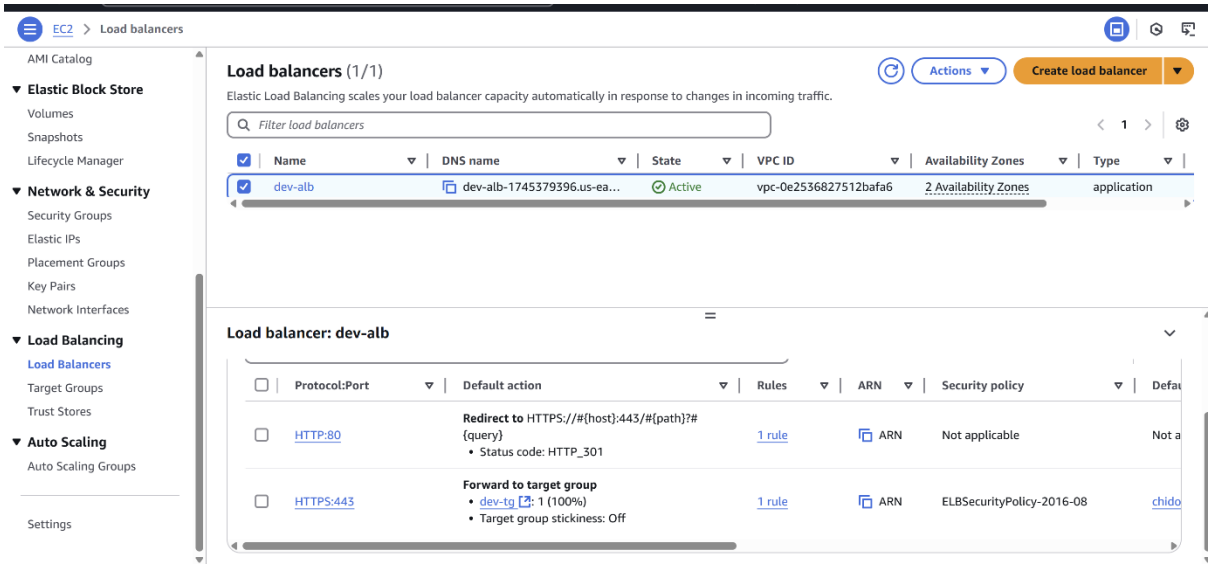
Output

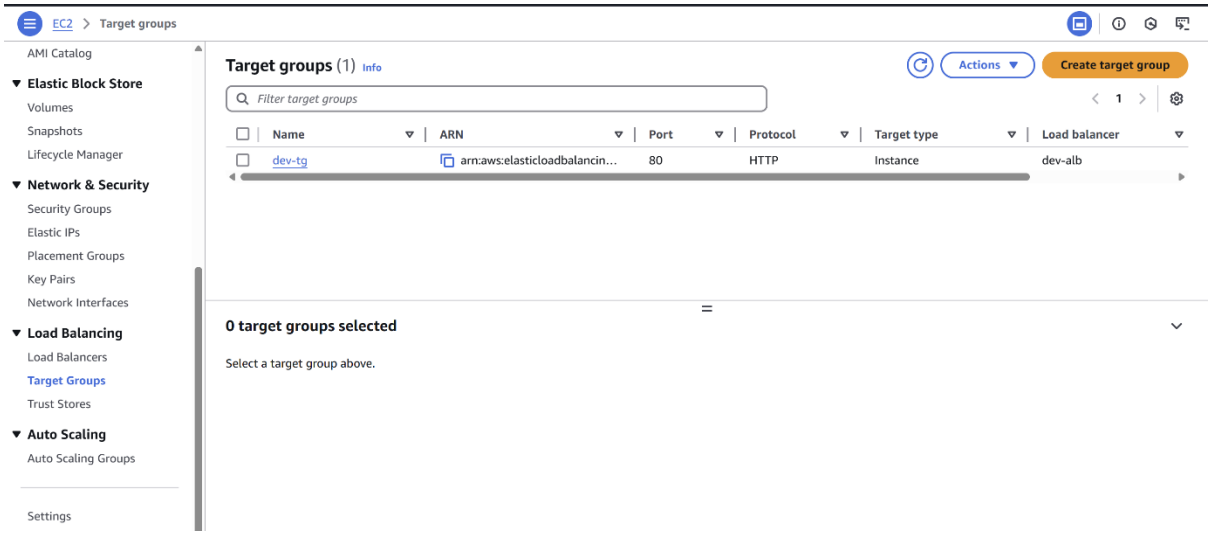
For convenience, we often define an output variable in Terraform to retrieve the ALB's DNS name, making it easy to test or reference:

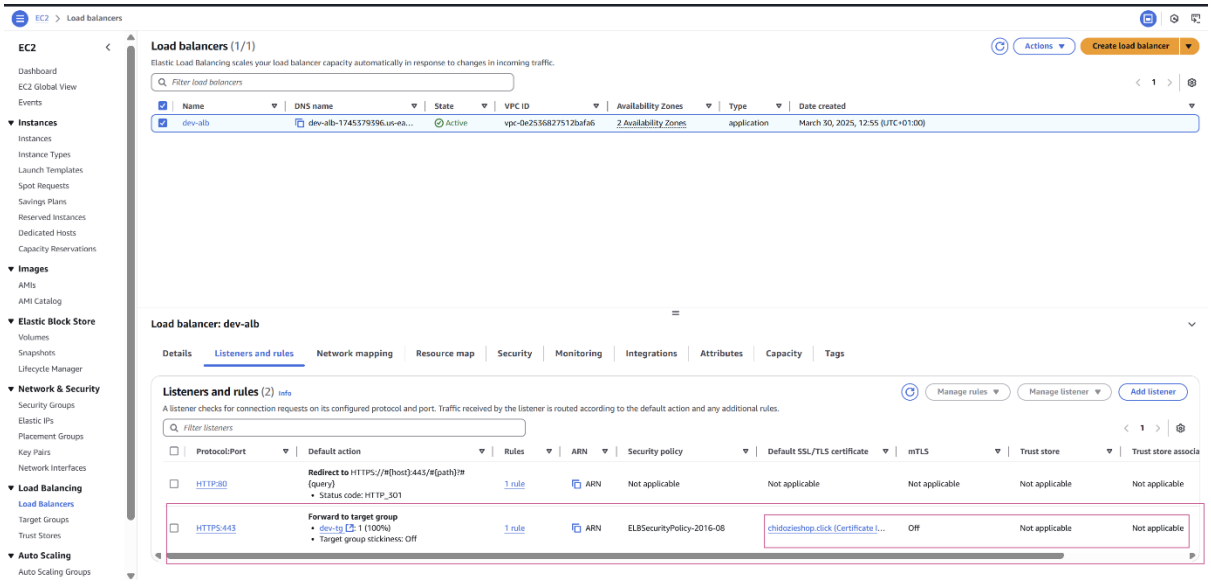
Once Terraform applies the configuration (or once you finish in the AWS console wizard), you can retrieve this DNS name from either the Terraform output or from the Load Balancer

description in the AWS console. Entering that DNS name in a browser should route you to your application (if the target instances are healthy).

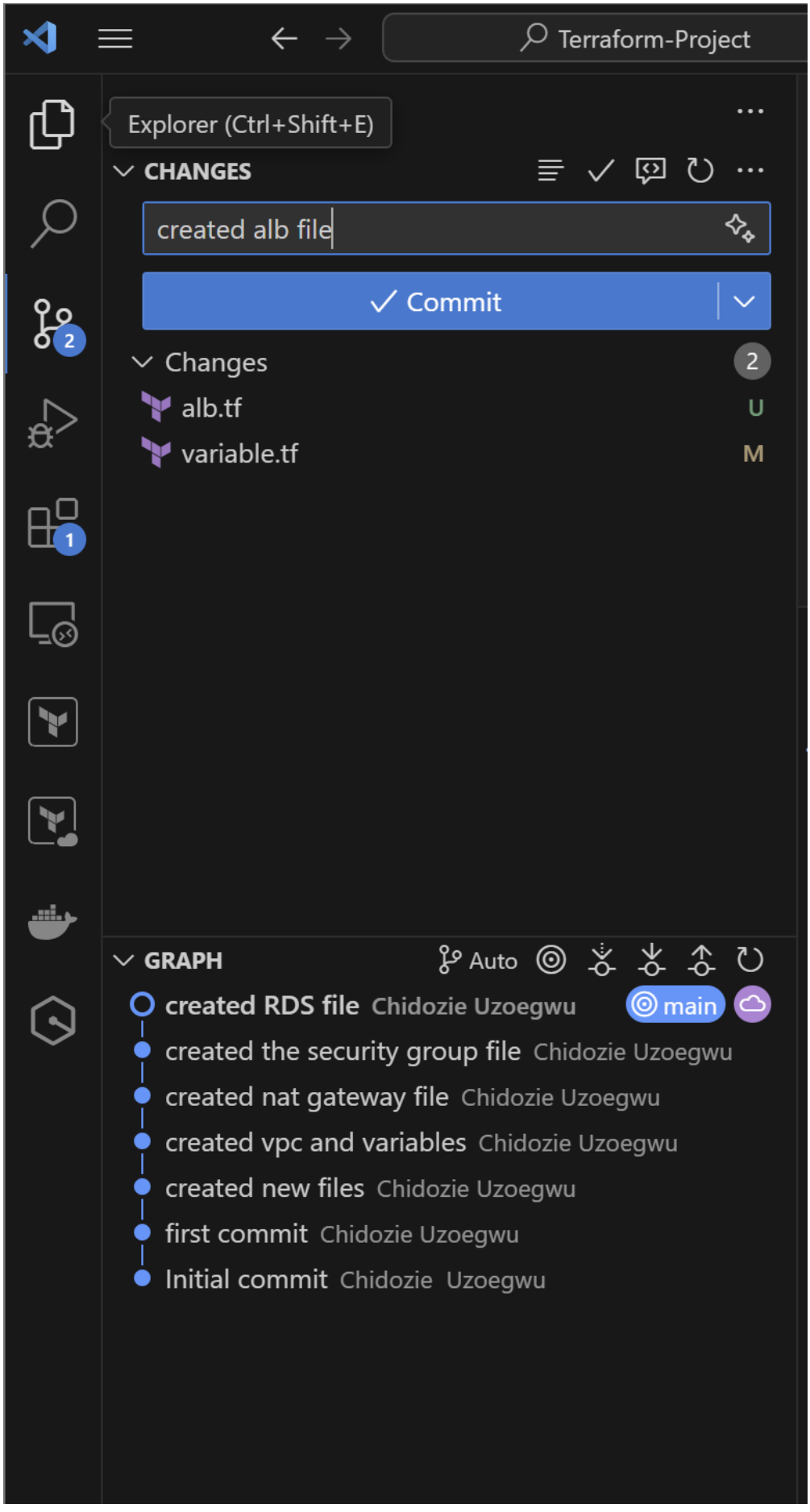
```
PS C:\Users\chido\Desktop\Terraform-Project> terraform apply
data.aws_db_snapshot.latest_db_snapshot: Reading...
aws_eip.eip_for_nat_gateway_az1: Refreshing state... [id=eipalloc-0353a7e76907b2048]
aws_vpc.vpc: Refreshing state... [id=vpc-0e2536827512bafa6]
aws_eip.eip_for_nat_gateway_az2: Refreshing state... [id=eipalloc-0a3c649035e2ef090]
data.aws_db_snapshot.latest_db_snapshot: Read complete after 1s [id=shopwise]
aws_internet_gateway.internet_gateway: Refreshing state... [id=igw-0550600374ce71ab9]
aws_subnet.public_subnet_az2: Refreshing state... [id=subnet-02c07d8b8e648f098]
aws_subnet.private_data_subnet_az1: Refreshing state... [id=subnet-0909acc2b7b6418c7]
aws_subnet.private_app_subnet_az1: Refreshing state... [id=subnet-0124bf873d71816b5]
aws_subnet.private_app_subnet_az2: Refreshing state... [id=subnet-08bd1016021a40cf6]
aws_subnet.private_data_subnet_az2: Refreshing state... [id=subnet-0f84f7c3f42127c47]
aws_subnet.public_subnet_az1: Refreshing state... [id=subnet-040cfc8cb3be547dc]
aws_security_group.ssh_security_group: Refreshing state... [id=sg-0d31c157b6a1afb1b]
aws_security_group.alb_security_group: Refreshing state... [id=sg-0300d75a203b30103]
aws_route_table.public_route_table: Refreshing state... [id=rtb-0aad8ffe33ba8d07d]
aws_nat_gateway.nat_gateway_az2: Refreshing state... [id=nat-0b90246efb7c4095a]
aws_db_subnet_group.database_subnet_group: Refreshing state... [id=database subnets]
aws_route_table.private_route_table_az2: Refreshing state... [id=rtb-090b7387a0adefa4e]
aws_route_table_association.public_subnet_2_route_table_association: Refreshing state... [id=rtbassoc-06b480f92e14ecdc7]
aws_route_table_association.public_subnet_az1_route_table_association: Refreshing state... [id=rtbassoc-087ca3c385245022c]
aws_nat_gateway.nat_gateway_az1: Refreshing state... [id=nat-02c22a8fe111beed3]
aws_security_group.webserver_security_group: Refreshing state... [id=sg-01f1d1d714b18b486]
aws_route_table.private_route_table_az1: Refreshing state... [id=rtb-06a2fffce49954f88]
aws_route_table_association.private_data_subnet_az2_route_table_az2_association: Refreshing state... [id=rtbassoc-01d7257cfed278
aws_route_table_association.private_app_subnet_az2_route_table_az2_association: Refreshing state... [id=rtbassoc-01dd1209bc8bbb9
aws_security_group.database_security_group: Refreshing state... [id=sg-0db3f13cd9cbe0258]
aws_route_table_association.private_app_subnet_az1_route_table_az1_association: Refreshing state... [id=rtbassoc-0ec84a573ffedf0
```

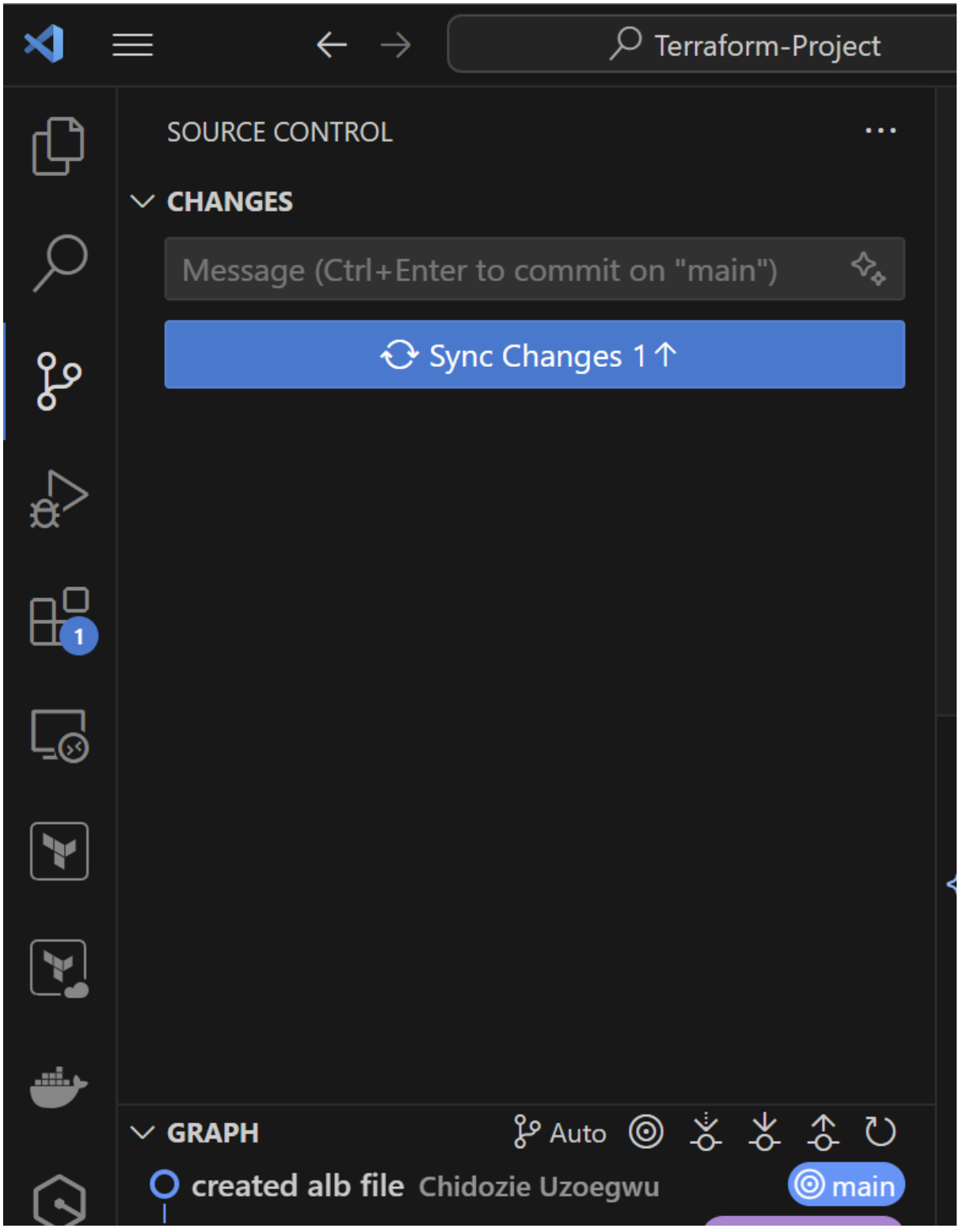


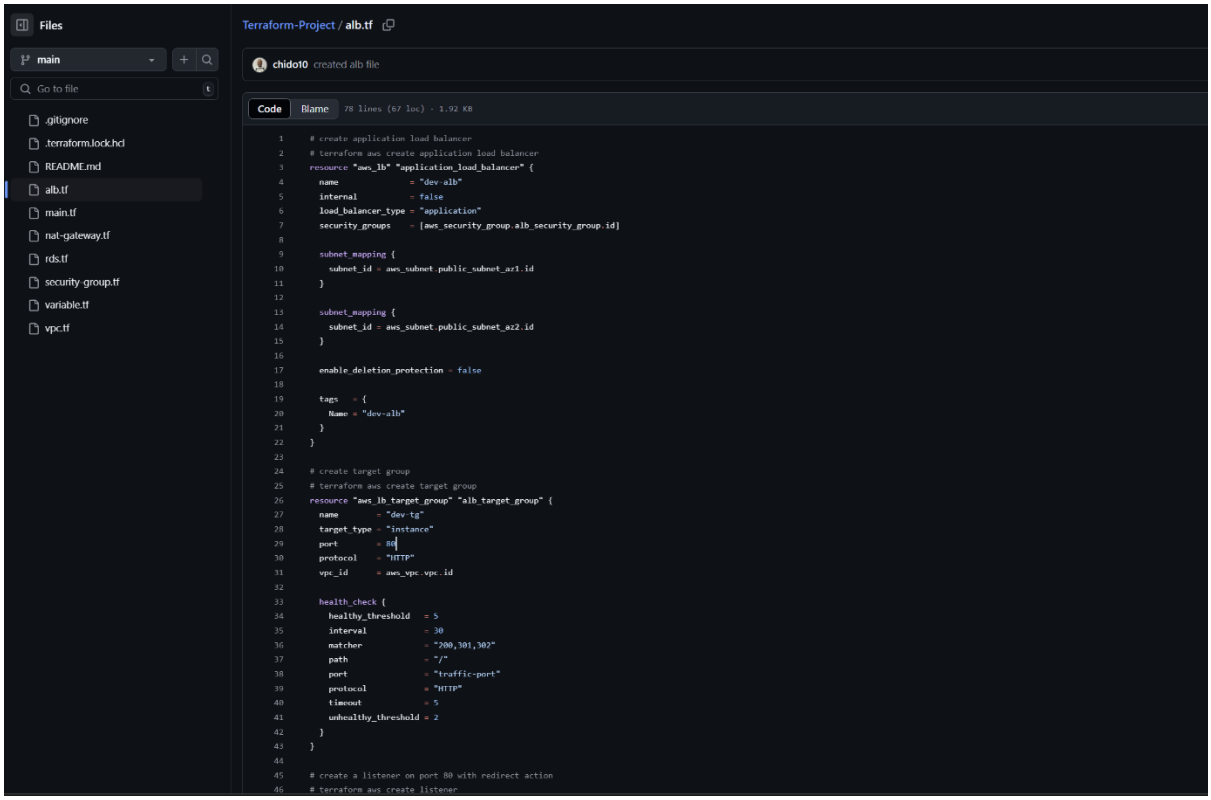




Afterwards, we can push our code to GitHub.


Terraform-Project
Explorer (Ctrl+Shift+E)
CHANGES
created alb file
Commit
Changes
2
alb.tf
U
variable.tf
M
GRAPH
Auto
created RDS file Chidozie Uzoegwu
main
created the security group file Chidozie Uzoegwu
created nat gateway file Chidozie Uzoegwu
created vpc and variables Chidozie Uzoegwu
created new files Chidozie Uzoegwu
first commit Chidozie Uzoegwu
Initial commit Chidozie Uzoegwu


Terraform-Project
SOURCE CONTROL
CHANGES
Message (Ctrl+Enter to commit on "main")
Sync Changes 1
GRAPH
Auto
created alb file Chidozie Uzoegwu
main



25. Create SNS Topic

Amazon SNS (Simple Notification Service) enables applications, systems, and services to instantly send notifications. It's commonly used in infrastructure for alerting or integrating with other AWS services like Auto Scaling or Lambda.

In this step, we will:

1. *Create an SNS topic named dev-sns-topic*

Subscribe an email address to the topic for receiving notifications

Create SNS Topic

A Terraform file named sns.tf was created and contains the aws_sns_topic resource block. This block defines a topic named dev-sns-topic.

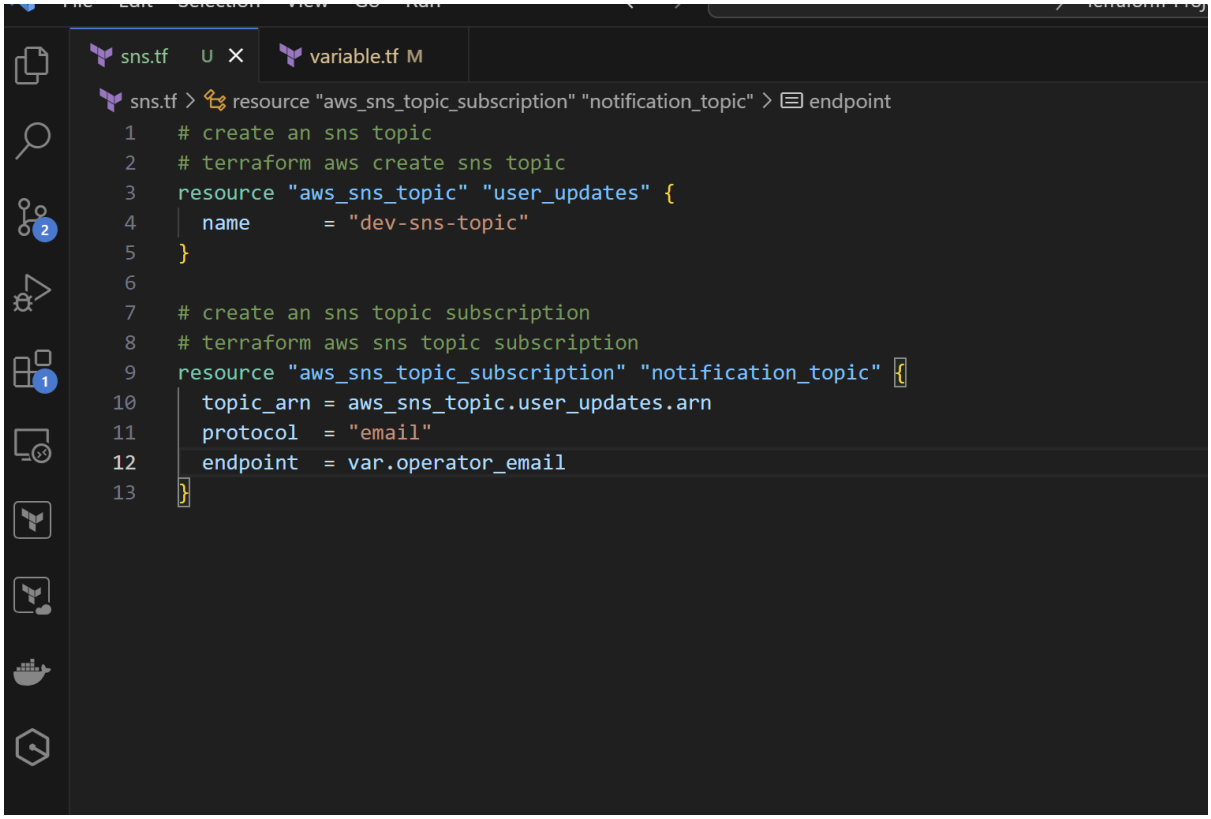
2. Create SNS Email Subscription

Still in sns.tf, a second resource is used to subscribe an email to the SNS topic:

Uses aws_sns_topic_subscription

Protocol is email

Email is injected via variable var.operator_email



3. Define the Email Variable

In variable.tf, the variable operator_email is declared:

Default value: loudoodoo@gmail.com

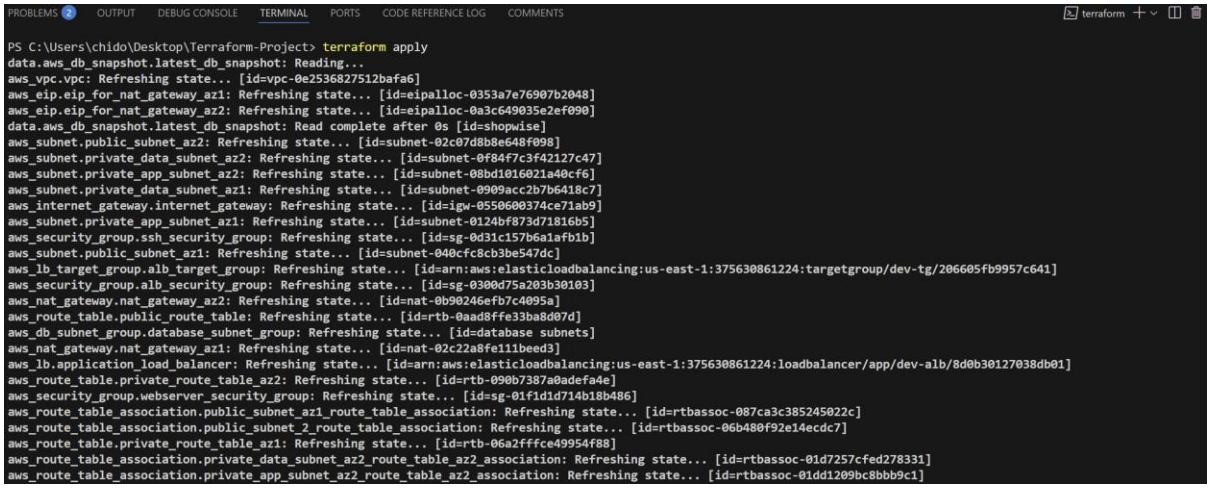
Description provided

Type: string

```
#sns topic variables
variable "operator_email" {
default      = "loudoodoo@gmail.com"
description  = "a valid email address to receive sns notifications"
type         = string
}
```

4. Terraform Apply

You ran terraform apply to provision resources. The SNS topic and subscription were successfully created along with other infrastructure.



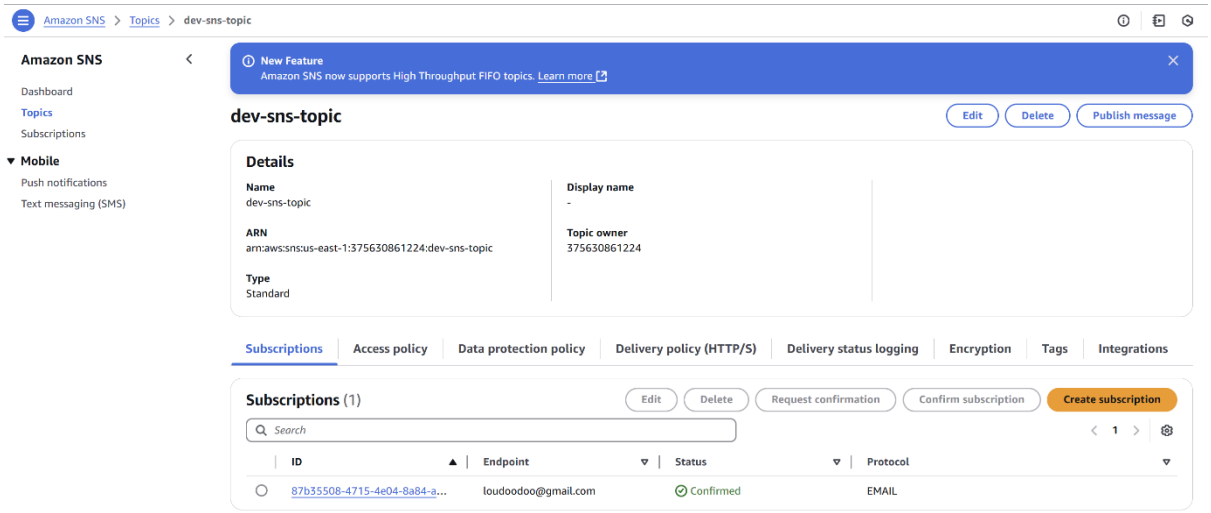
5. Validate SNS Topic on AWS Console

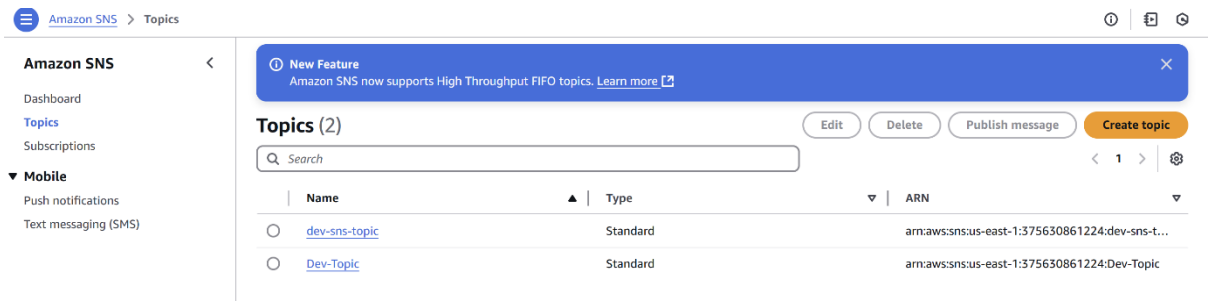
In the AWS SNS Dashboard:

The topic dev-sns-topic is listed

Protocol shows as EMAIL

The subscription status is Confirmed for the provided email

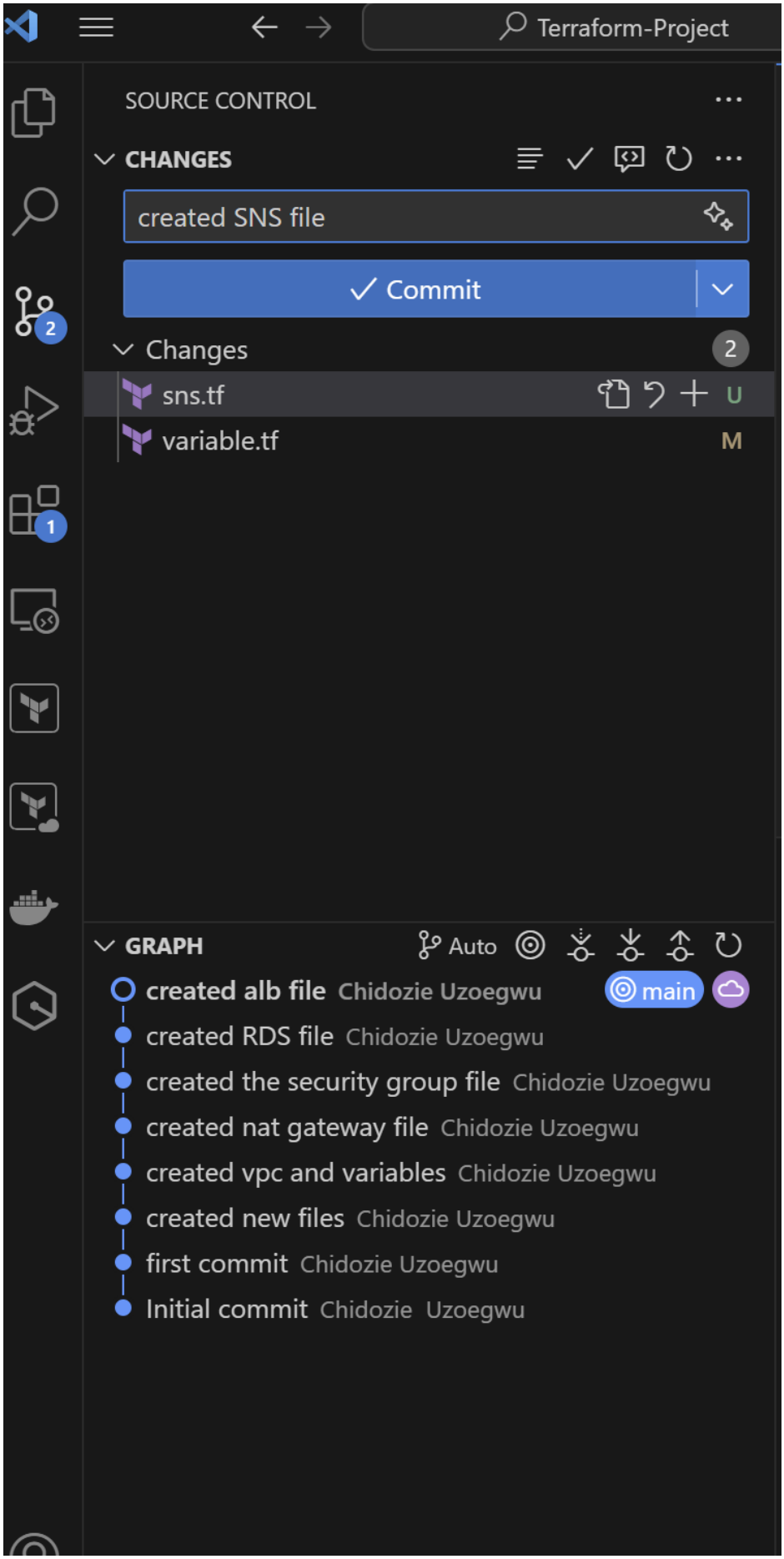




6. Version Control Confirmation

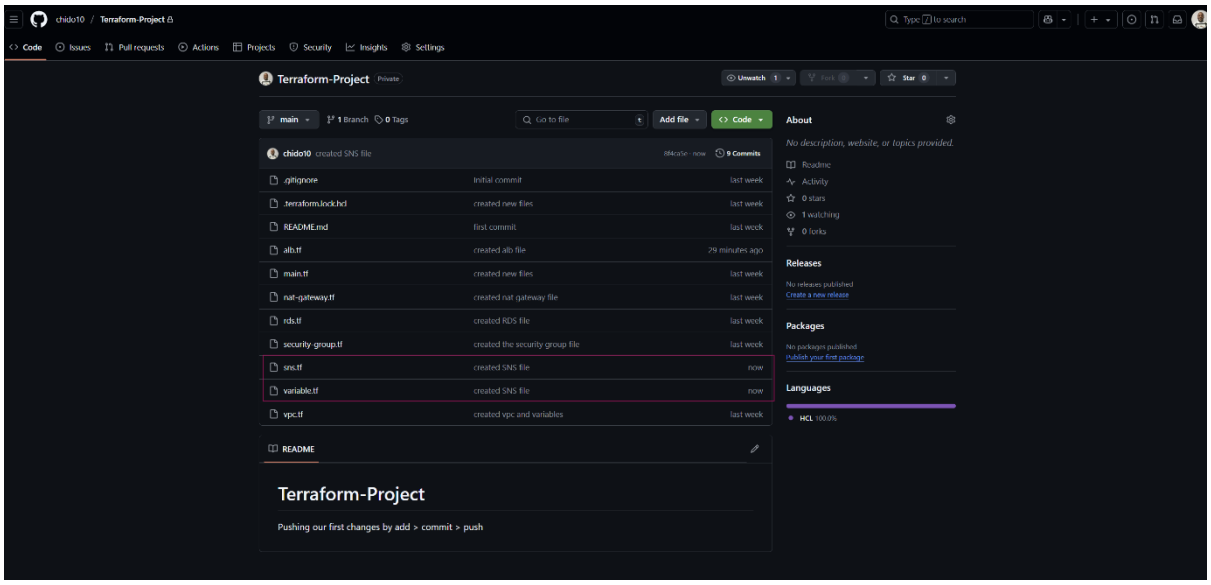
In Visual Studio Code:

The sns.tf and variable.tf files were committed with the message "created SNS file"


Terraform-Project
SOURCE CONTROL
CHANGES
created SNS file
Commit
Changes
2
sns.tf
U
variable.tf
M
GRAPH
Auto
created alb file Chidozie Uzoegwu
main
created RDS file Chidozie Uzoegwu
created the security group file Chidozie Uzoegwu
created nat gateway file Chidozie Uzoegwu
created vpc and variables Chidozie Uzoegwu
created new files Chidozie Uzoegwu
first commit Chidozie Uzoegwu
Initial commit Chidozie Uzoegwu

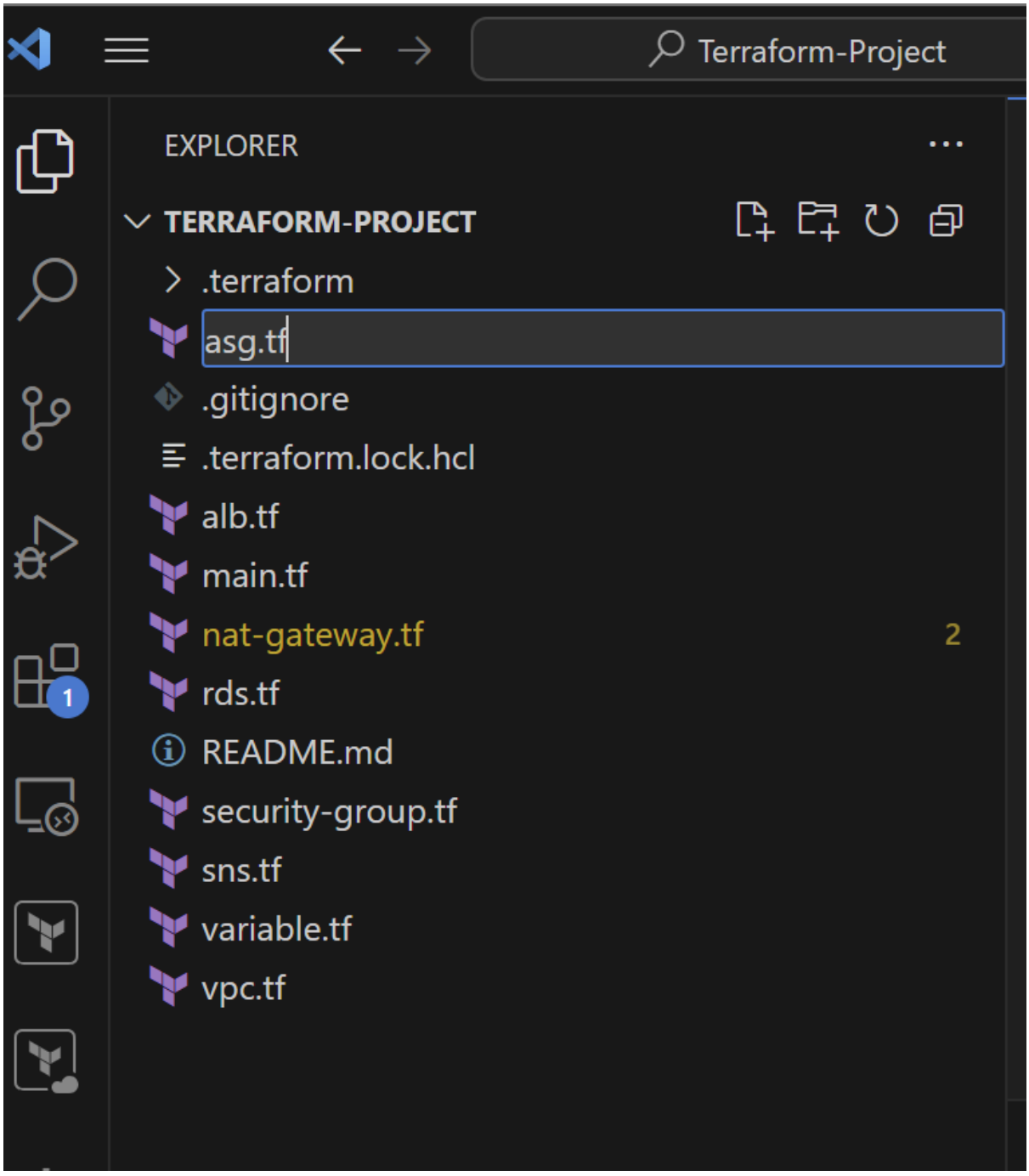
7. Pushed to GitHub

Your project (Terraform-Project) on GitHub includes the committed sns.tf and variable.tf files reflecting the changes.



26. Create Auto Scaling Group (ASG)

Create asg.tf file in VS Code



A new file named asg.tf is added to the Terraform project to define the Auto Scaling Group resources.

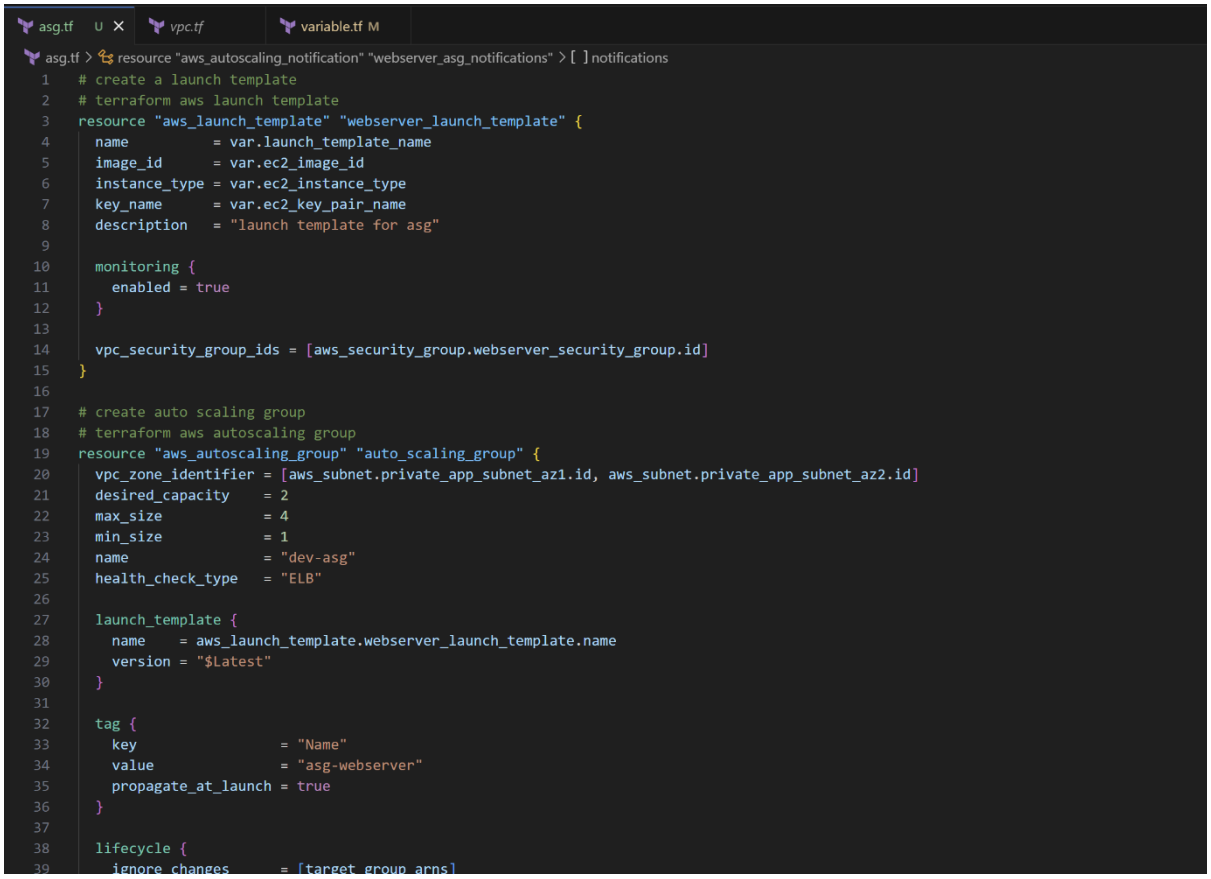
Write Terraform code for Launch Template and ASG

Defines the aws_launch_template with parameters such as image_id, instance_type, and key_name. Also defines the aws_autoscaling_group with properties like min_size, max_size, and desired_capacity.

Adds a name tag to the ASG and configures the lifecycle block to ignore changes to target_group_arns to avoid unnecessary diffs in Terraform plans.

Attach ASG to ALB target group: Uses the aws_autoscaling_attachment resource to link the ASG to the ALB target group.

Add ASG notifications: Defines aws_autoscaling_notification for the ASG to send lifecycle event notifications such as instance launch and termination to an SNS topic.



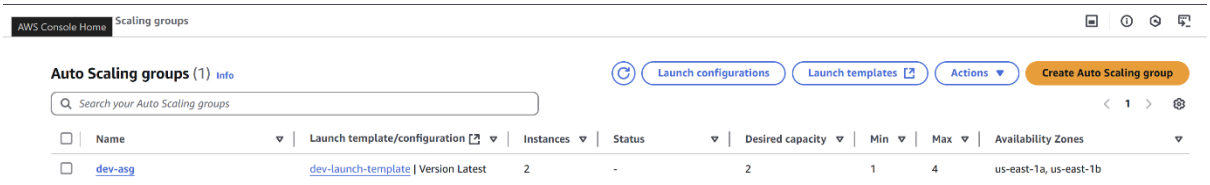
```
asg.tf U   vpc.tf   variable.tf M
asg.tf > resource "aws_autoscaling_notification" "webserver_asg_notifications" > [ ] notifications
19  resource "aws_autoscaling_group" "auto_scaling_group" {
32    tag {
34      value               = "asg-webserver"
35      propagate_at_launch = true
36    }
37
38    lifecycle {
39      ignore_changes      = [target_group_arns]
40    }
41  }
42
43  # attach auto scaling group to alb target group
44  # terraform aws autoscaling attachment
45  resource "aws_autoscaling_attachment" "asg_alb_target_group_attachment" {
46    autoscaling_group_name = aws_autoscaling_group.auto_scaling_group.id
47    lb_target_group_arn    = aws_lb_target_group.alb_target_group.arn
48  }
49
50  # create an auto scaling group notification
51  # terraform aws autoscaling notification
52  resource "aws_autoscaling_notification" "webserver_asg_notifications" {
53    group_names = [aws_autoscaling_group.auto_scaling_group.name]
54
55    notifications = [
56      "autoscaling:EC2_INSTANCE_LAUNCH",
57      "autoscaling:EC2_INSTANCE_TERMINATE",
58      "autoscaling:EC2_INSTANCE_LAUNCH_ERROR",
59      "autoscaling:EC2_INSTANCE_TERMINATE_ERROR",
60    ]
61
62    topic_arn = aws_sns_topic.user_updates.arn
63  }
```

Run terraform apply to deploy resources

```
PS C:\Users\chido\Desktop\Terraform-Project> terraform apply
data.aws_db_snapshot.latest_db_snapshot: Reading...
aws_vpc.vpc: Refreshing state... [id=vpc-0e2536827512bafa6]
aws_eip.eip_for_nat_gateway_az2: Refreshing state... [id=eipalloc-0a3c649035e2ef090]
aws_sns_topic.user_updates: Refreshing state... [id=arn:aws:sns:us-east-1:375630861224:dev-sns-topic]
aws_eip.eip_for_nat_gateway_az1: Refreshing state... [id=eipalloc-0353a7e76907b2048]
data.aws_db_snapshot.latest_db_snapshot: Read complete after 1s [id=shopwise]
aws_sns_topic_subscription.notification_topic: Refreshing state... [id=arn:aws:sns:us-east-1:375630861224:dev-sns-topic:87b35508-4715-4e04-8a84-a3a787ae1cde]
aws_internet_gateway.internet_gateway: Refreshing state... [id=igw-0550600374ce71ab9]
aws_subnet.private_app_subnet_az1: Refreshing state... [id=subnet-0124bf873d71816b5]
aws_subnet.private_data_subnet_az1: Refreshing state... [id=subnet-0909acc2b7b6418c7]
aws_subnet.public_subnet_az1: Refreshing state... [id=subnet-040cfc8cb3be547dc]
aws_subnet.private_app_subnet_az2: Refreshing state... [id=subnet-08bd1016021a40cf6]
aws_subnet.public_subnet_az2: Refreshing state... [id=subnet-02c07d8b8e648f098]
aws_subnet.private_data_subnet_az2: Refreshing state... [id=subnet-0f84f7c3f42127c47]
aws_security_group.ssh_security_group: Refreshing state... [id=sg-0d31c157b6a1afb1b]
aws_security_group.alb_security_group: Refreshing state... [id=sg-0300d75a203b30103]
aws_lb_target_group.alb_target_group: Refreshing state... [id=arn:aws:elasticloadbalancing:us-east-1:375630861224:targetgroup/dev-tg/206605fb9957c641]
aws_route_table.public_route_table: Refreshing state... [id=rtb-0aad8ffe33ba8d07d]
aws_nat_gateway.nat_gateway_az1: Refreshing state... [id=nat-02c22a8fe111beed3]
```

Terraform execution successfully applies the configuration, creating and refreshing the necessary resources including the ASG, subnets, security groups, and SNS topic.

Verify ASG in AWS Console

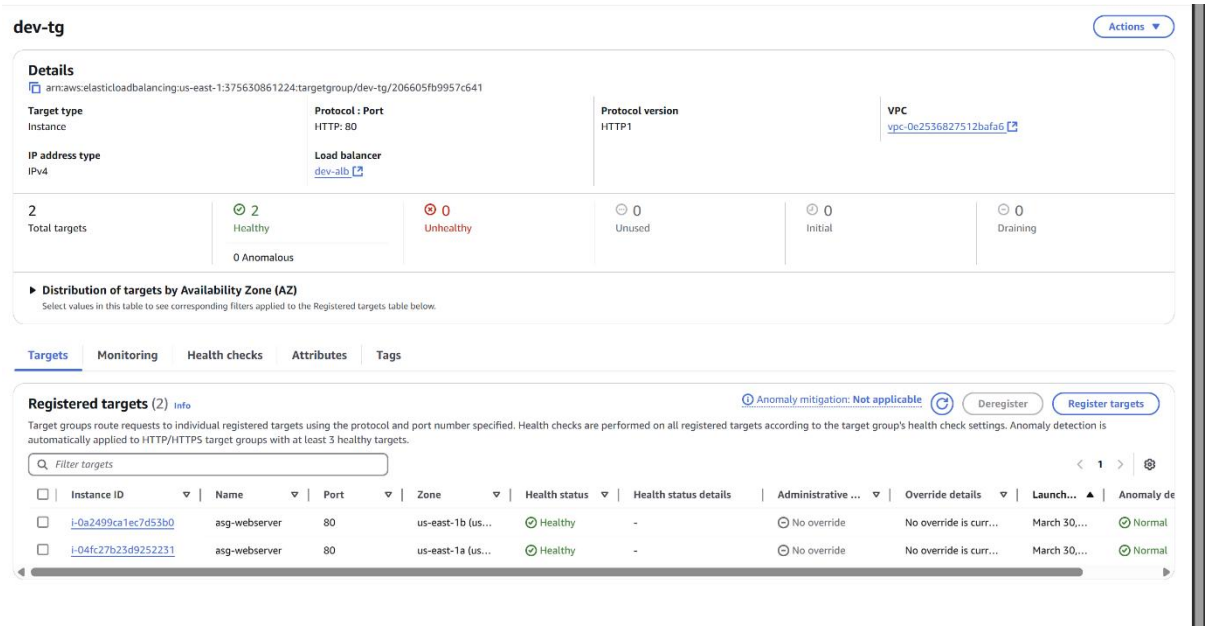


The Auto Scaling Group named dev-asg appears in the AWS Console with two running instances and the correct launch template attached.

Check Launch Templates in AWS Console

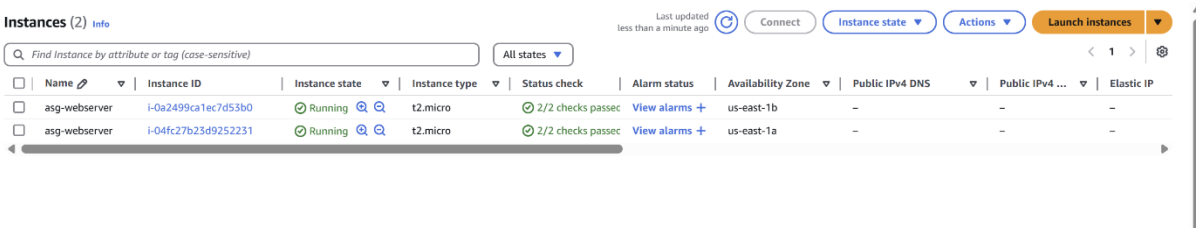
Confirms the creation of the launch template dev-launch-template with version 1, as expected.

Check Target Group health in ALB



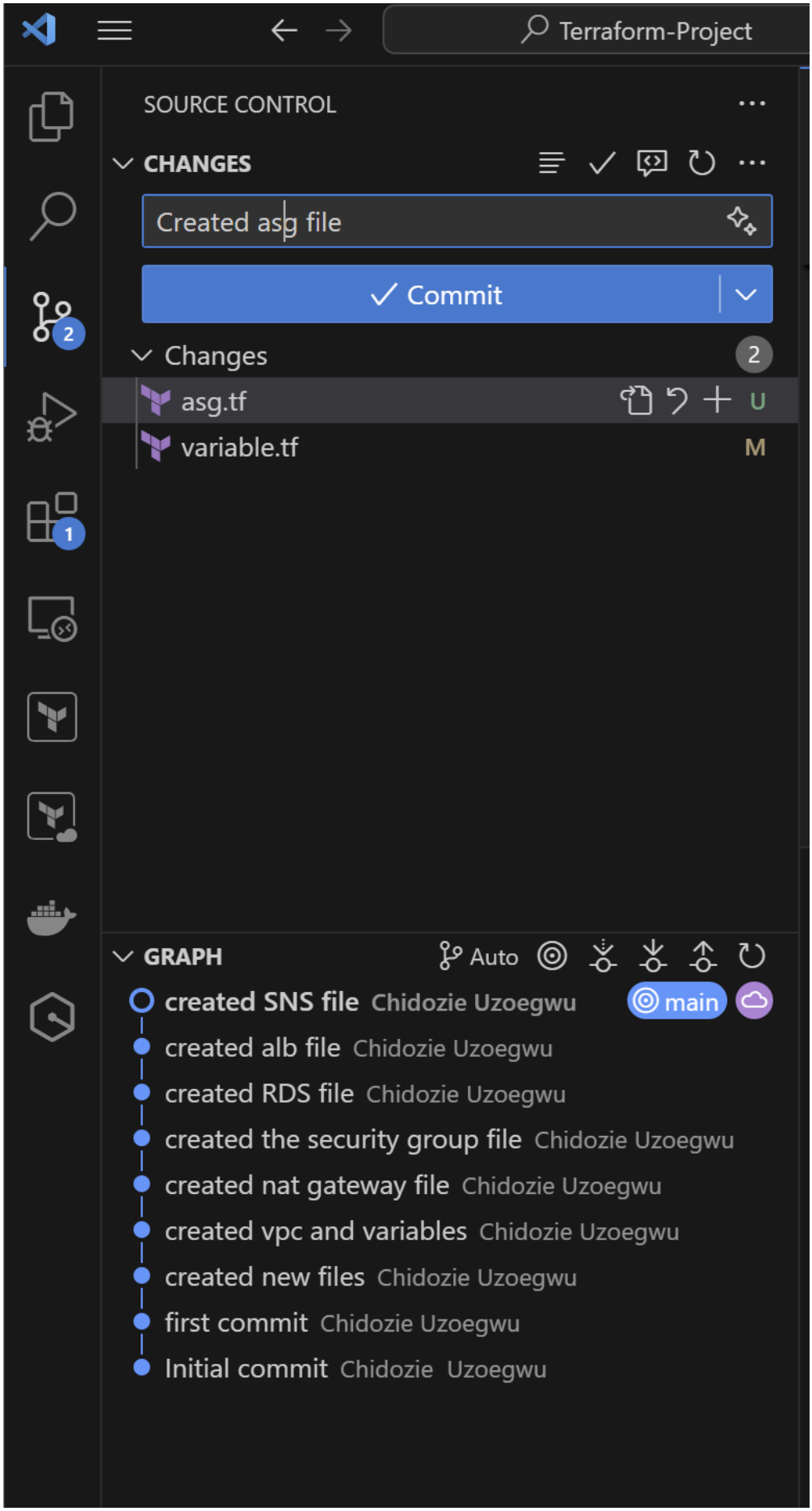
The target group dev-tg displays two registered instances that are healthy and serving traffic.

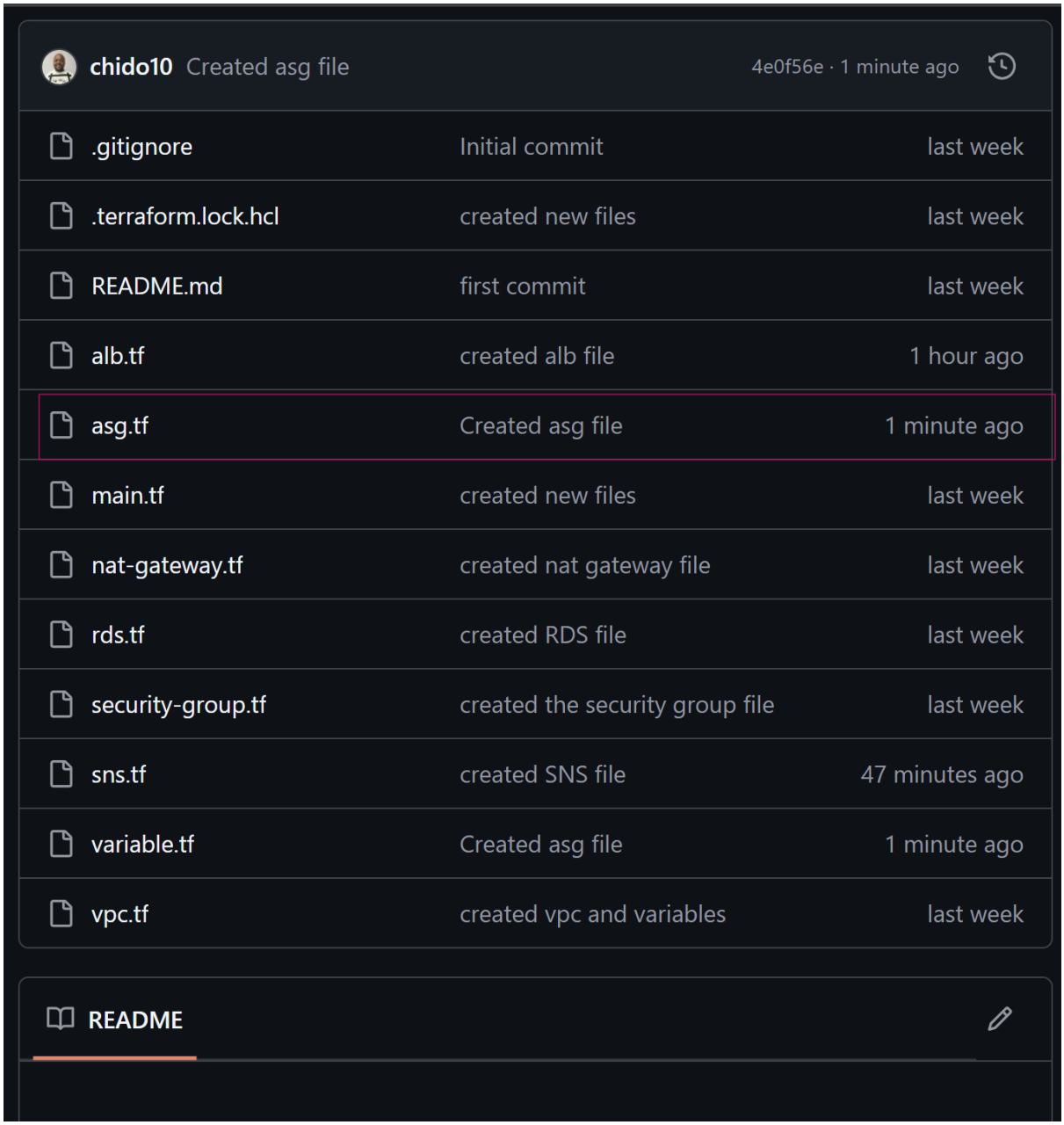
Verify running EC2 instances

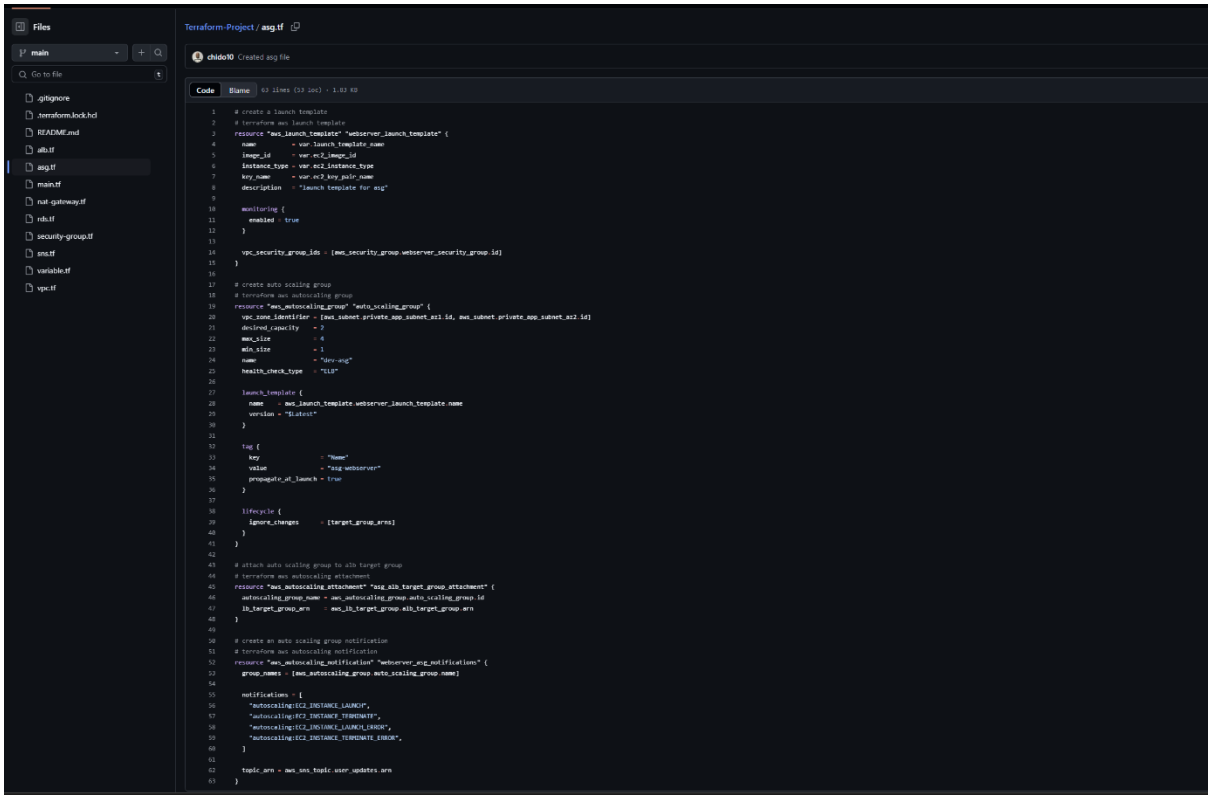


AWS EC2 console shows two instances running under the ASG, both tagged as asg-webserver and passing health checks.

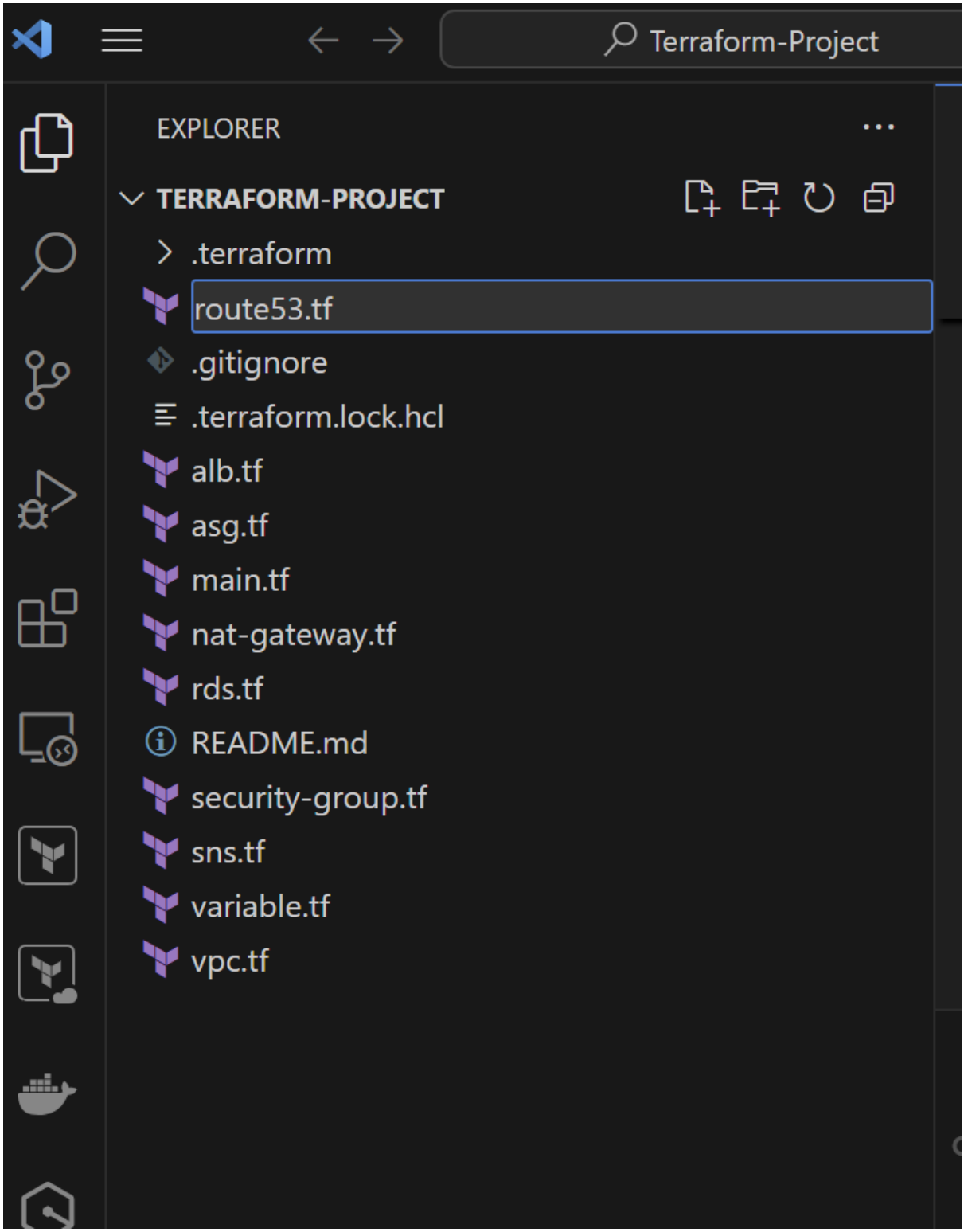
we can the commit and push to our GitHub


Terraform-Project
SOURCE CONTROL
CHANGES
Created asg file
Commit
Changes
2
asg.tf
U
variable.tf
M
GRAPH
Auto
created SNS file Chidozie Uzoegwu
main
created alb file Chidozie Uzoegwu
created RDS file Chidozie Uzoegwu
created the security group file Chidozie Uzoegwu
created nat gateway file Chidozie Uzoegwu
created vpc and variables Chidozie Uzoegwu
created new files Chidozie Uzoegwu
first commit Chidozie Uzoegwu
Initial commit Chidozie Uzoegwu


chido10 Created asg file
4e0f56e · 1 minute ago
.gitignore Initial commit last week
.terraform.lock.hcl created new files last week
README.md first commit last week
alb.tf created alb file 1 hour ago
asg.tf Created asg file 1 minute ago
main.tf created new files last week
nat-gateway.tf created nat gateway file last week
rds.tf created RDS file last week
security-group.tf created the security group file last week
sns.tf created SNS file 47 minutes ago
variable.tf Created asg file 1 minute ago
vpc.tf created vpc and variables last week
README

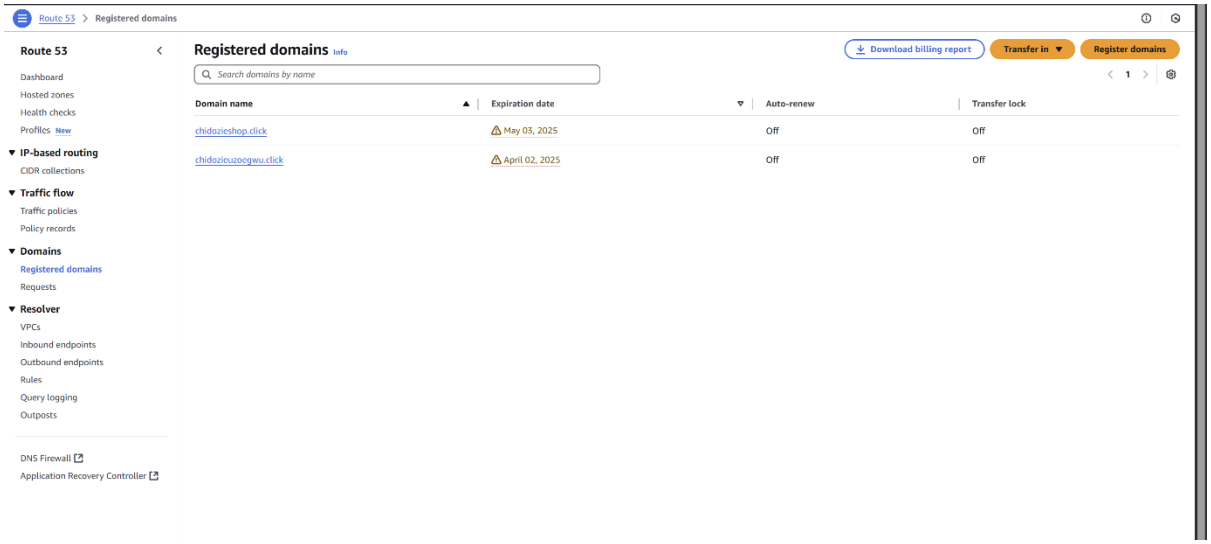


27. Create Record Set in Route 53



To access the site with a friendly domain name, create a DNS record pointing to the ALB:

- **Hosted Zone:** This assumes you have a Route 53 Hosted Zone for your domain (e.g., example.com). If not, you can use the ALB DNS as-is or create a dummy Hosted Zone for learning. We already have our own domain and we used it instead.



We created our code below in our route53 file and set our variable to link with our DNS as it is below

route53.tf

route53.tf > resource "aws_route53_record" "site_domain" > alias > evaluate_target_health

```
# get hosted zone details
# terraform aws data hosted zone
data "aws_route53_zone" "hosted_zone" {
  name = var.domain_name
}

# create a record set in route 53
# terraform aws route 53 record
resource "aws_route53_record" "site_domain" {
  zone_id = data.aws_route53_zone.hosted_zone.zone_id
  name    = var.record_name
  type    = "A"

  alias {
    name                   = aws_lb.application_load_balancer.dns_name
    zone_id                = aws_lb.application_load_balancer.zone_id
    evaluate_target_health = true
  }
}
```

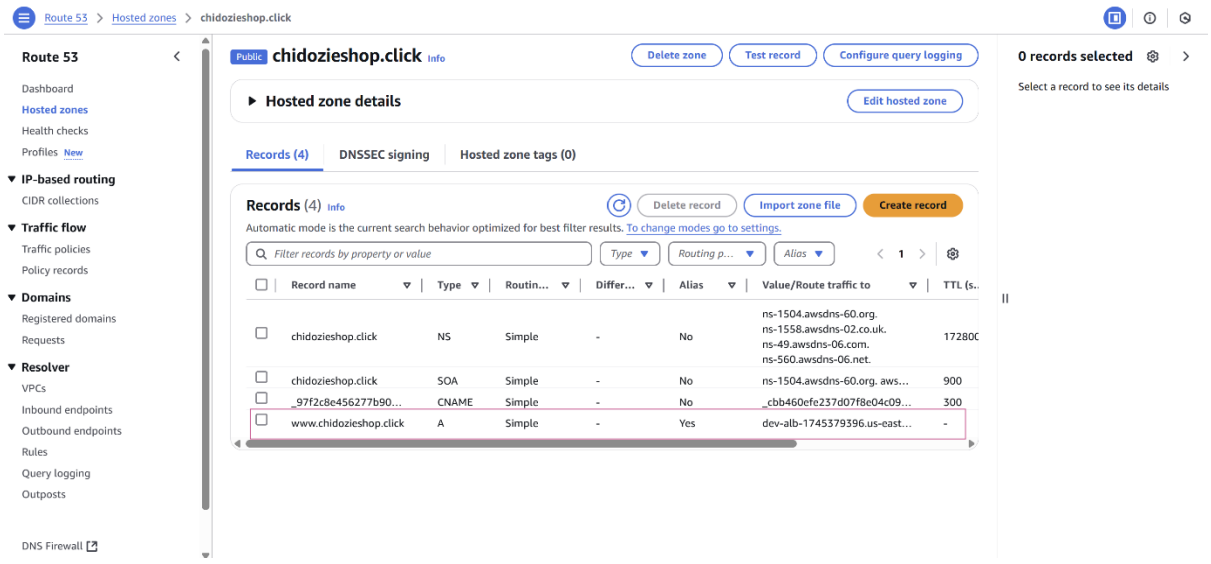
```
# route 53 variables
variable "domain_name" {
default      = "chidozieshop.click"
description  = "domain name"
type         = string
}

variable "record_name" {
default      = "www"
description  = "sub domain name"
type         = string
}
```

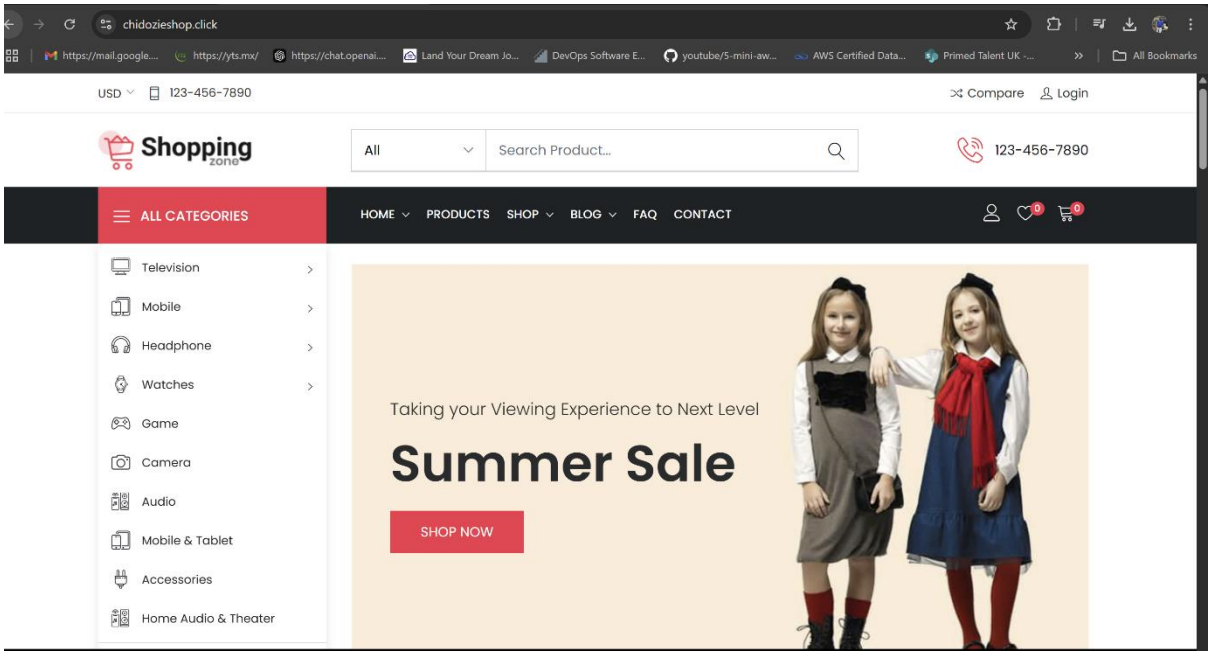
Next, we terraform apply and confirm it successful

```
Apply complete! Resources: 2 added, 0 changed,
Manage Users\chido\Desktop\Terraform-Project>
```

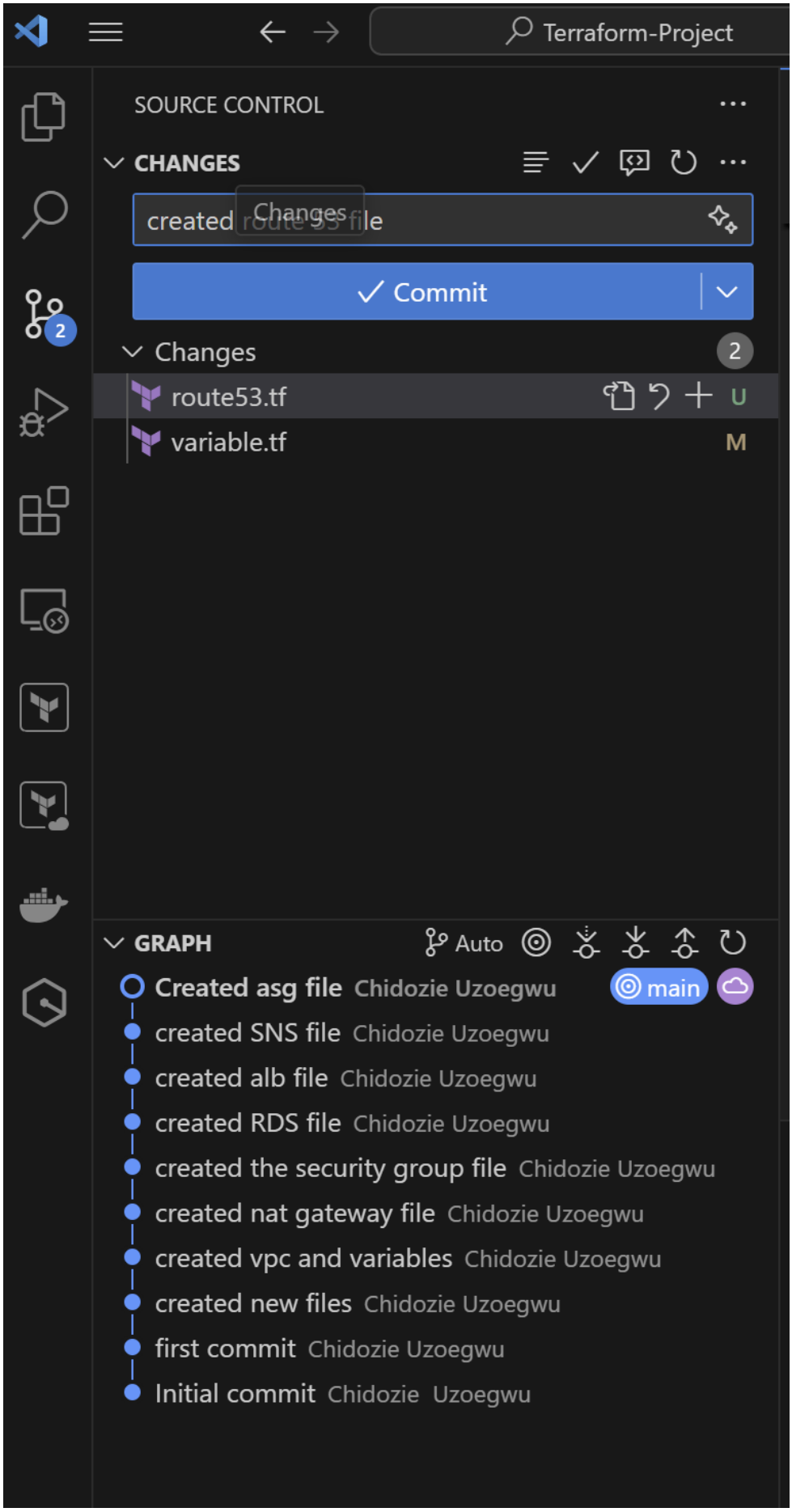
After applying, Route 53 will have the record. It may take a few minutes to propagate. If you then go to http://www.example.com (if that's what you set, and your domain is pointing to Route 53's name servers), it should resolve to the ALB and show the site.

We then can go back to our console and confirm it is been created and test the link as well

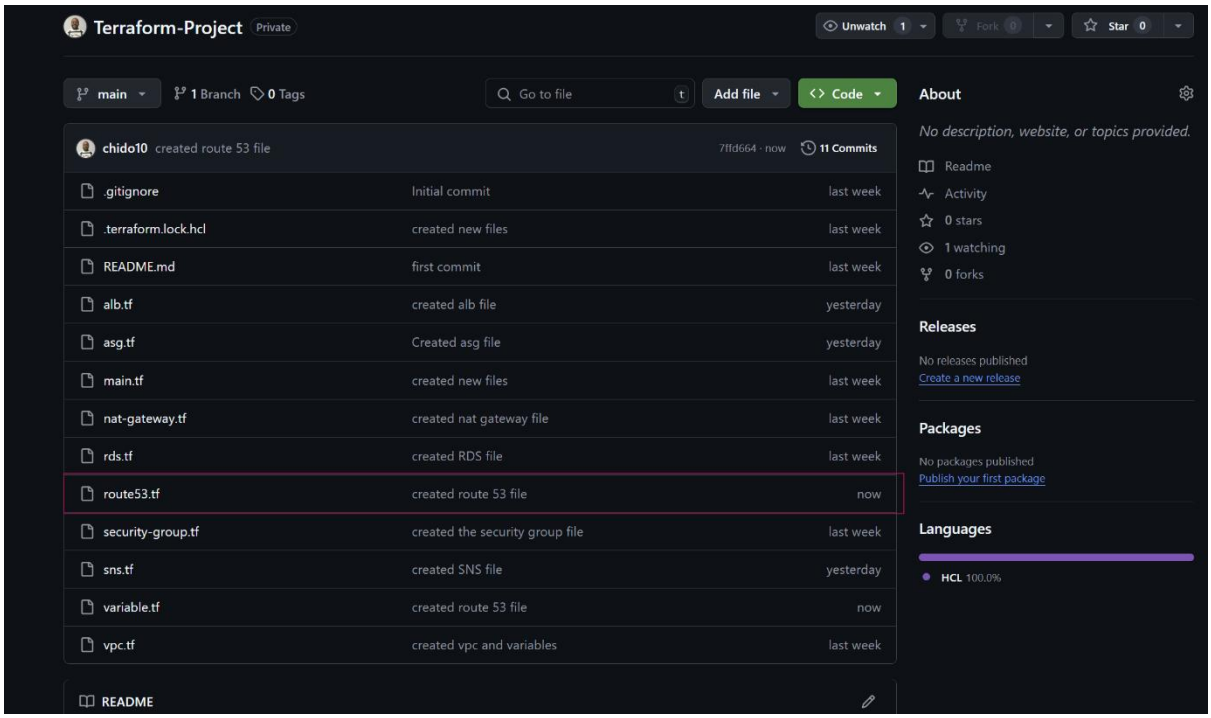


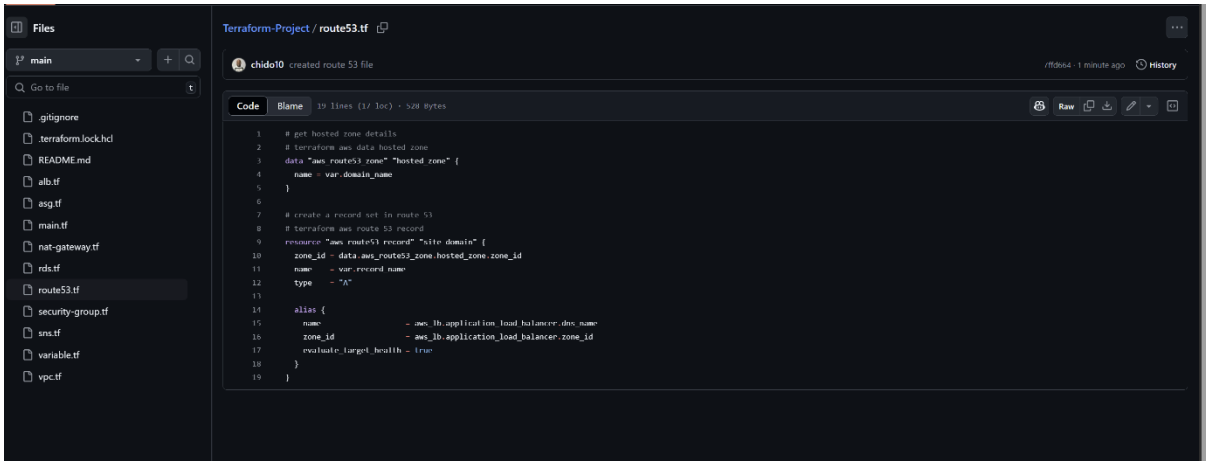


After confirming we can then as well commit and push to our GitHub

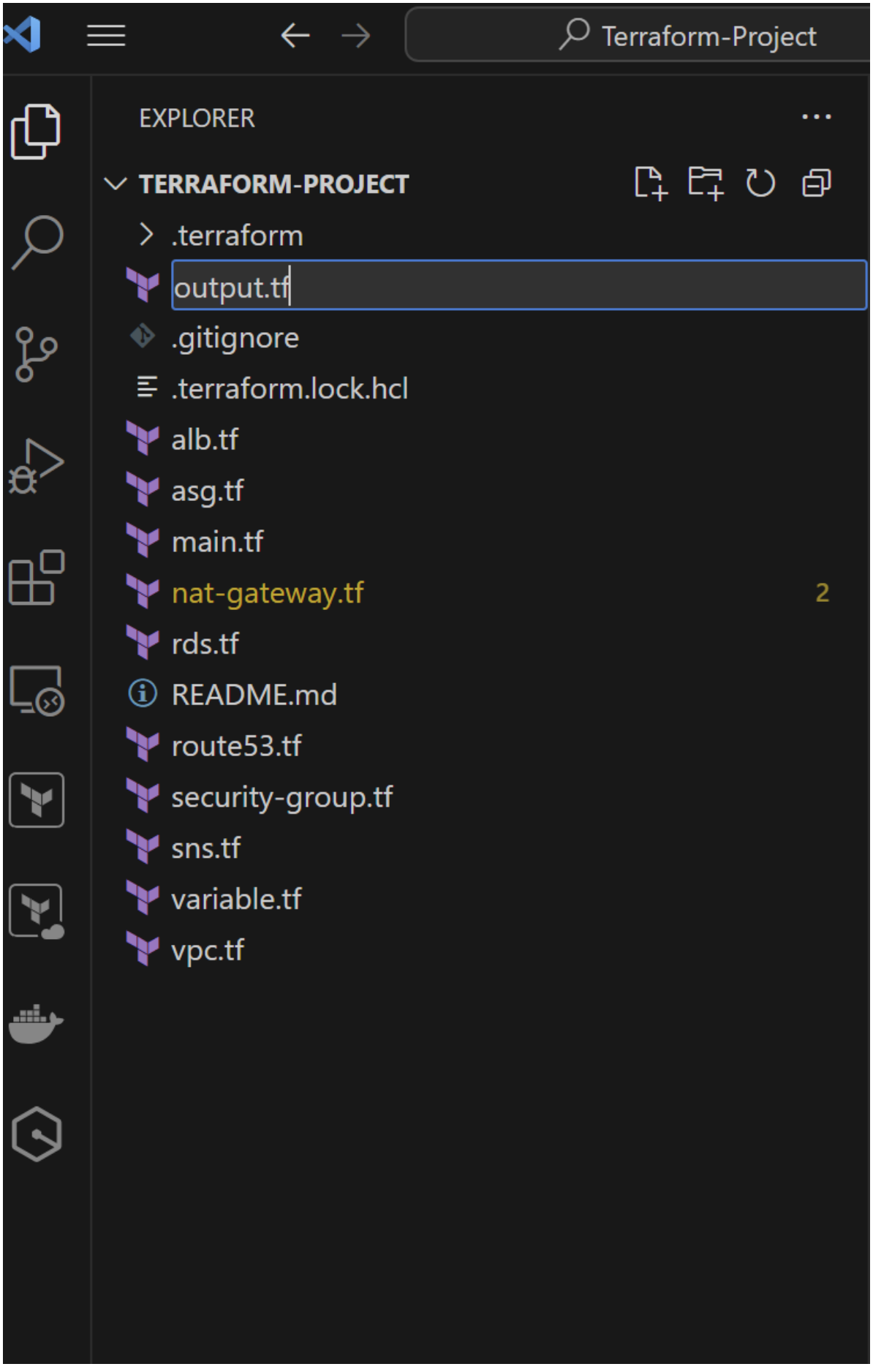

Terraform-Project
SOURCE CONTROL
CHANGES
created route 53 file
Changes
Commit
Changes
2
route53.tf
U
variable.tf
M
GRAPH
Auto
Created asg file Chidozie Uzoegwu
main
created SNS file Chidozie Uzoegwu
created alb file Chidozie Uzoegwu
created RDS file Chidozie Uzoegwu
created the security group file Chidozie Uzoegwu
created nat gateway file Chidozie Uzoegwu
created vpc and variables Chidozie Uzoegwu
created new files Chidozie Uzoegwu
first commit Chidozie Uzoegwu
Initial commit Chidozie Uzoegwu

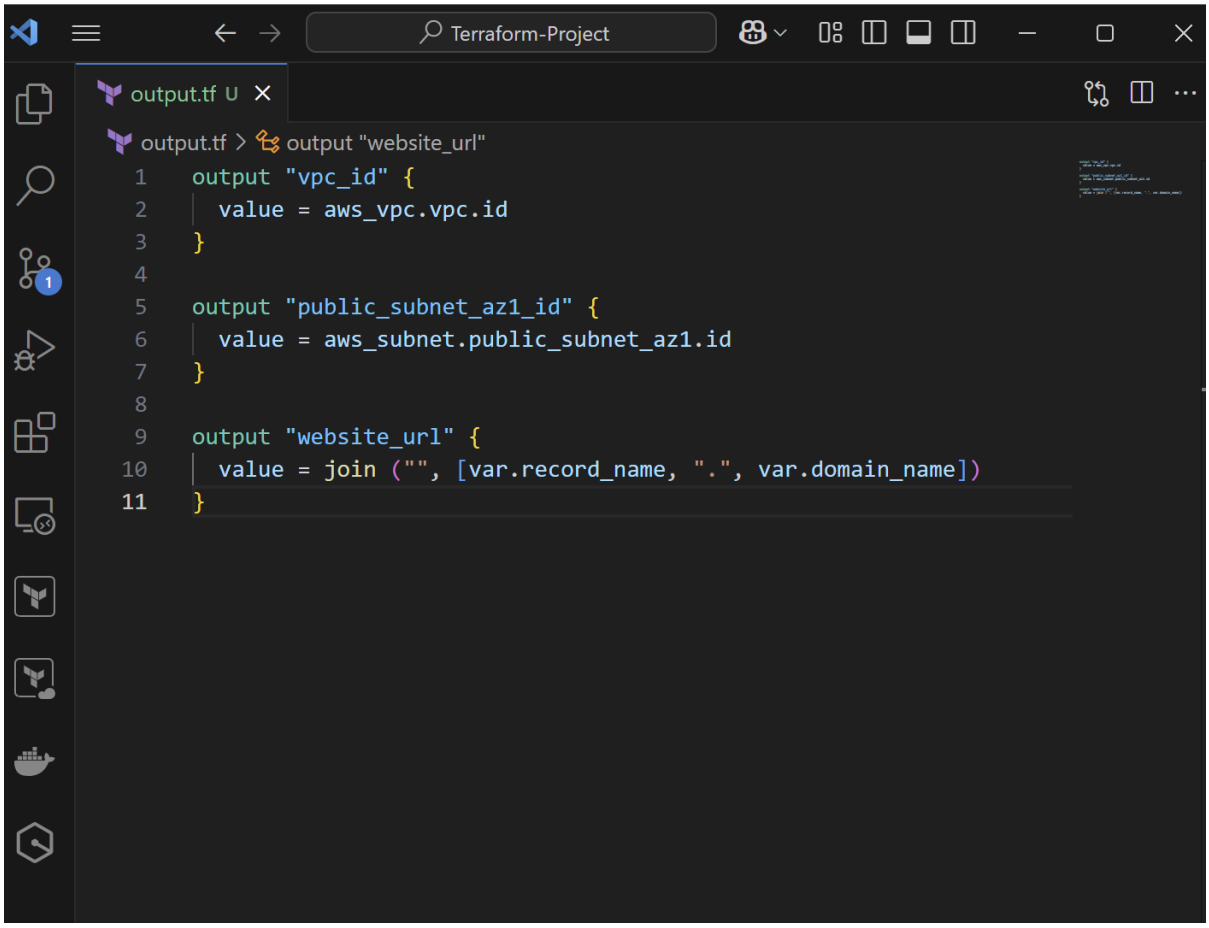
Afterwards, confirm in our repository





28. Terraform Outputs


Terraform-Project
EXPLORER
TERRAFORM-PROJECT
.terraform
output.tf
.gitignore
.terraform.lock.hcl
alb.tf
asg.tf
main.tf
nat-gateway.tf
2
rds.tf
README.md
route53.tf
security-group.tf
sns.tf
variable.tf
vpc.tf

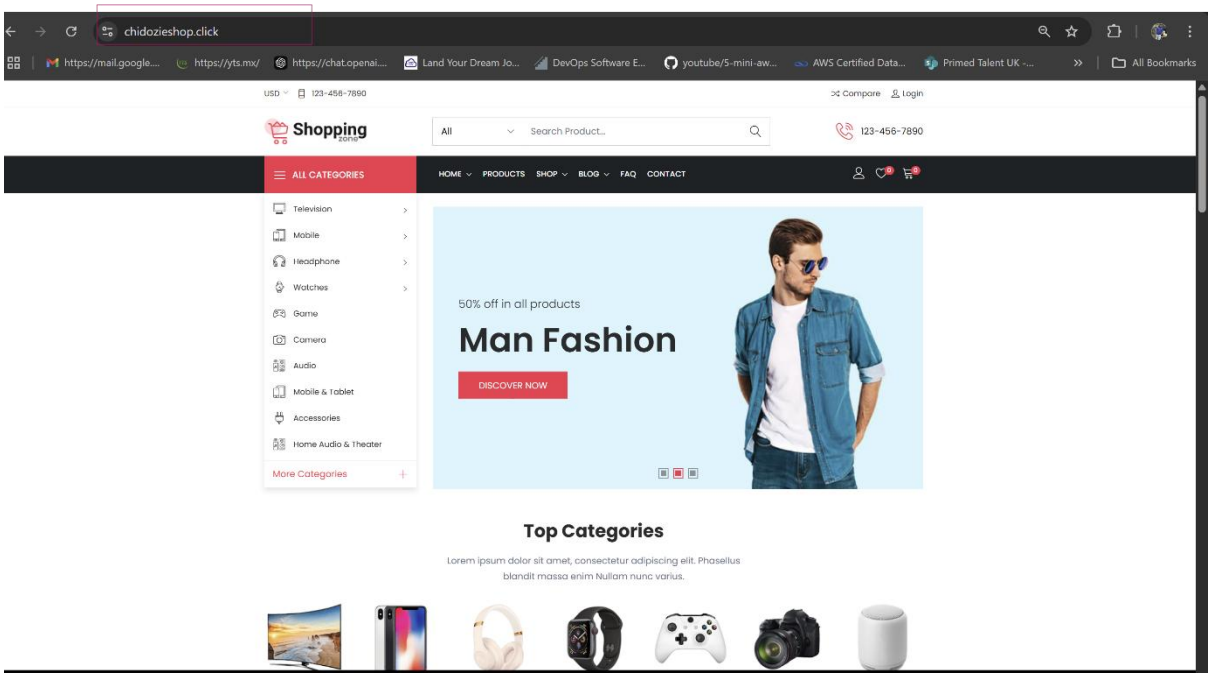


We configured outputs in Terraform to easily retrieve important information. After a successful terraform apply, Terraform prints these outputs.

```
PS C:\Users\chido\Desktop\Terraform-Project> terraform apply
data.aws_db_snapshot.latest_db_snapshot: Reading...
data.aws_route53_zone.hosted_zone: Reading...
aws_vpc.vpc: Refreshing state... [id=vpc-0e2536827512bafa6]
aws_eip.eip_for_nat_gateway_az2: Refreshing state... [id=eipalloc-0a3c649035e2ef090]
aws_eip.eip_for_nat_gateway_az1: Refreshing state... [id=eipalloc-0353a7e76907b2048]
aws_sns_topic.user_updates: Refreshing state... [id=arn:aws:sns:us-east-1:375630861224:dev-
data.aws_db_snapshot.latest_db_snapshot: Read complete after 1s [id=shopwise]
aws_sns_topic_subscription.notification_topic: Refreshing state... [id=arn:aws:sns:us-east-
data.aws_route53_zone.hosted_zone: Read complete after 1s [id=Z0386233250UG6M7UVWTD]
aws_internet_gateway.internet_gateway: Refreshing state... [id=igw-0550600374ce71ab9]
aws_subnet.public_subnet_az1: Refreshing state... [id=subnet-040cfc8cb3be547dc]
aws_subnet.private_app_subnet_az1: Refreshing state... [id=subnet-0124bf873d71816b5]
```

```
Changes to Outputs:
  + public_subnet_az1_id = "subnet-040cfc8cb3be547dc"
  + vpc_id               = "vpc-0e2536827512bafa6"
  + website_url          = "www.chidozieshop.click"
```

```
Apply complete! Resources: 1 added, 0 changed, 1 destroyed.

Outputs:

public_subnet_az1_id = "subnet-040cfc8cb3be547dc"
vpc_id = "vpc-0e2536827512bafa6"
website_url = "www.chidozieshop.click"
PS C:\Users\chido\Desktop\Terraform-Project>
```



29. Clean Up

After finishing the project or for routine maintenance, you should clean up resources to avoid ongoing costs (especially in personal demos or sandbox accounts):

- **Destroy with Terraform:** The safest way to remove all resources is terraform destroy. This will read the state file and attempt to delete every resource it created, in the proper order.

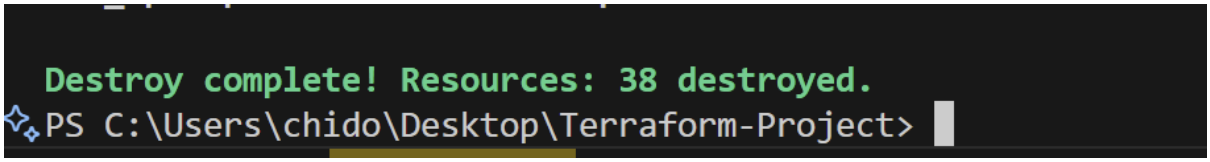
```
PS C:\Users\chido\Desktop\Terraform-Project> terraform destroy
```

Run: terraform destroy -auto-approve (auto-approve skips the yes prompt, use with

caution) in your project directory. Terraform will show all resources marked for destruction (it's essentially the reverse of apply).

```
Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes
```

It will then proceed to delete, showing progress. DB instances can take a bit to delete (unless skip_final_snapshot = true was set, which we did to speed up). If everything is set up correctly, Terraform will delete the resources and conclude with Destroy complete! and the number of resources destroyed.



- **Verify in AWS:** Check the AWS console for each resource type:
  - VPCs: the custom VPC should be gone (or if not, its dependencies might not have fully cleaned – ensure subnets, gateways are gone).
  - EC2: no running instances or load balancers.
  - RDS: the database is gone.
  - S3: The state file might still be in the bucket. You can delete the bucket manually (Terraform doesn't automatically remove the state file or the bucket, since it's the backend).
  - Route 53: the DNS record should be removed.
  - IAM: The IAM user still exists (since Terraform didn't create it, we did manually). You may keep it for future projects or remove its access key.
- **Remove Local Files:** Delete any local credentials if they were solely for this project (or at least the specific profile from your AWS credentials file if desired). And you can delete the repository or local folder if you're done with the code.

Remember to **NEVER leave stray resources** in your AWS if you're not actively using them—especially RDS or EC2 instances—to avoid unexpected charges.

Conclusion: You have now hosted a scalable, dynamic e-commerce website's infrastructure on AWS using Terraform. From networking (VPC, subnets, NAT) to compute (EC2 Auto Scaling), load balancing (ALB), database (RDS), and domain setup (Route 53), all aspects

were automated with Terraform configurations. This approach makes it easy to version, share, and reuse infrastructure code.