

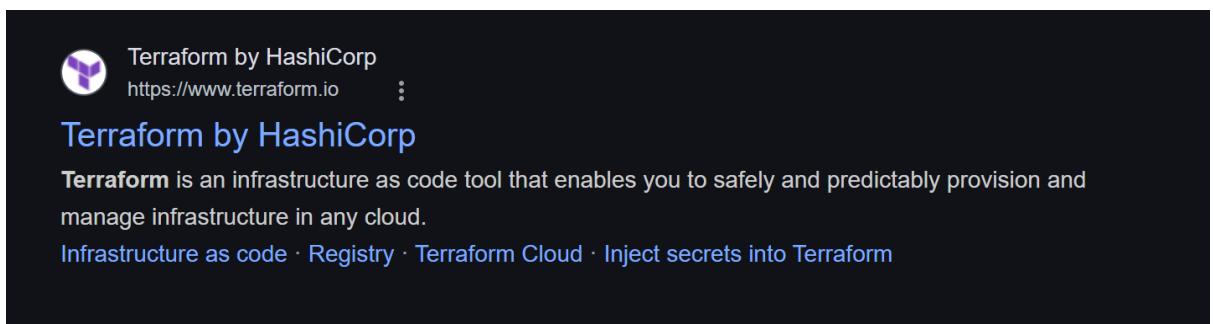
Host a Dynamic Ecommerce Website on AWS with Terraform

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.



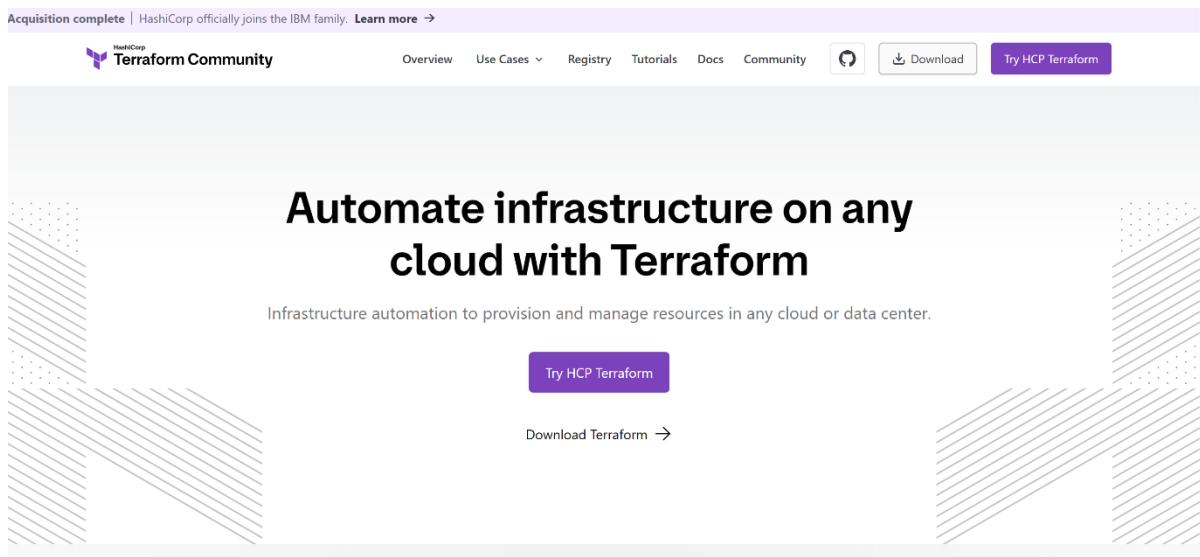
- **AWS Infrastructure:** Our site will run on AWS. Key resources include a Virtual Private Cloud (VPC), subnets, an internet gateway, NAT gateways, an RDS database, security groups, an Application Load Balancer (ALB), an Auto Scaling Group, and a Route 53 DNS record. Terraform will manage these.
- **GitHub & 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.

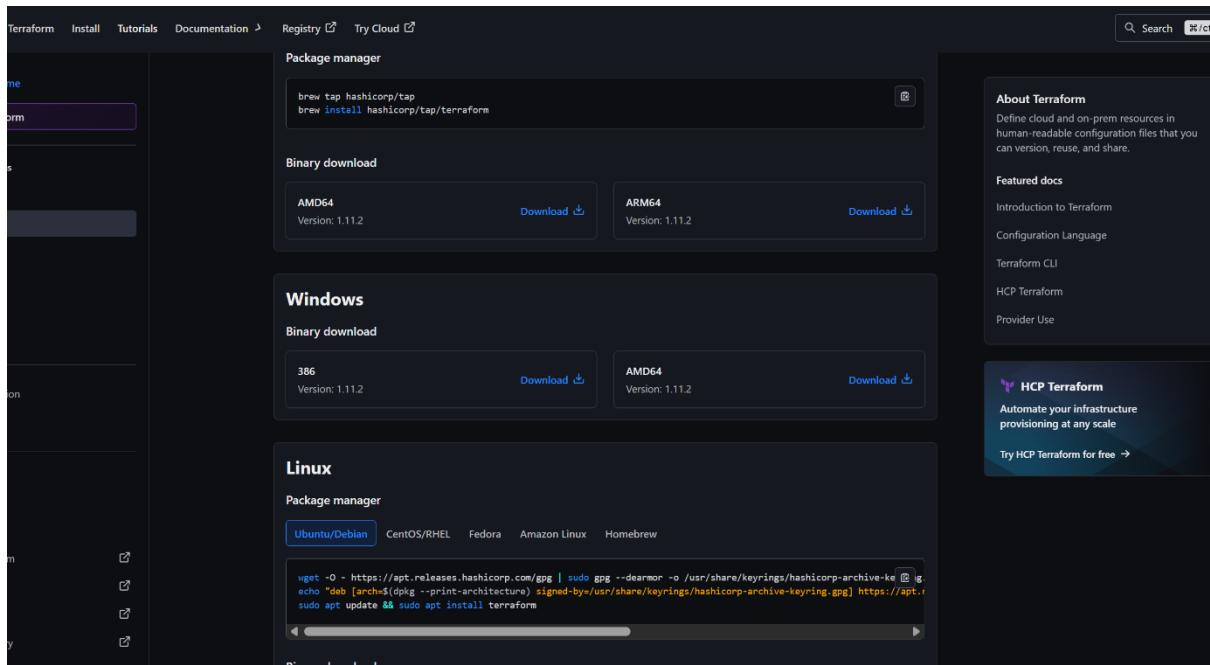
- **AWS CLI:** The AWS Command Line Interface will let us authenticate and interact with AWS from our local machine. We'll use it to configure credentials and verify connectivity.

2. Install Terraform (Windows)

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

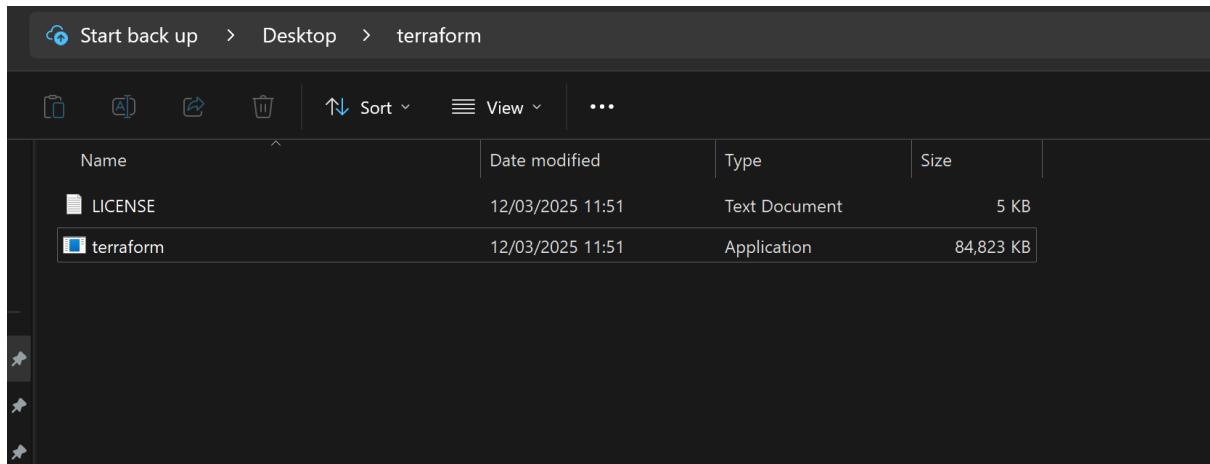
1. **Download Terraform:** Visit the official Terraform website and download the appropriate Windows ZIP archive (32-bit or 64-bit depending on your system).
Screenshot below shows Terraform's website with download options.





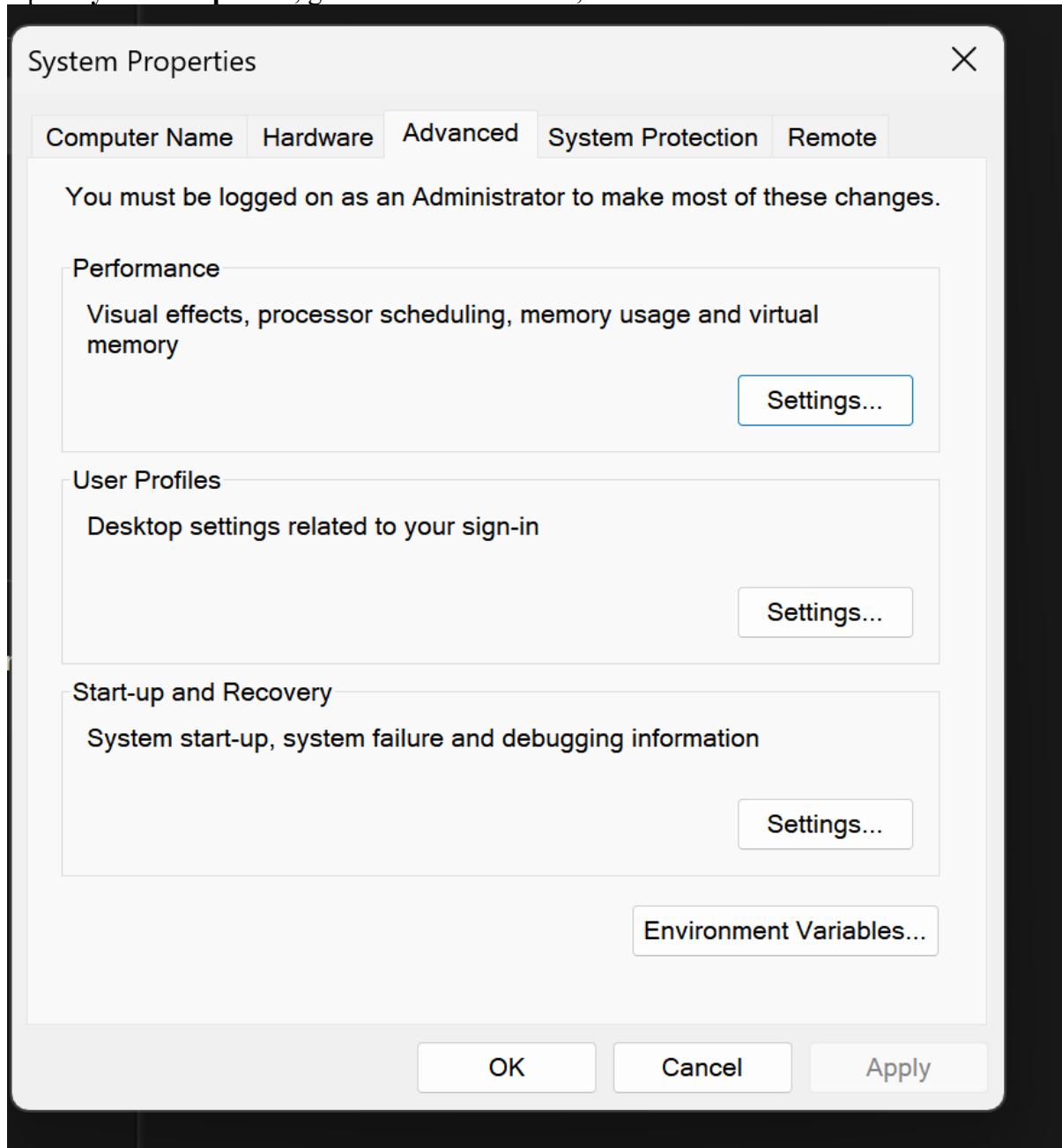
2. **Extract the ZIP:** Extract the downloaded `terraform.zip` to a folder (e.g., `C:\Users\<YourName>\Desktop\terraform`). You should see the `terraform.exe` binary in this folder.

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

Variable	Value
ChocolateyLastPathUpdate	133576771563142904
OneDrive	C:\Users\chido\OneDrive
OneDriveConsumer	C:\Users\chido\OneDrive
Path	C:\Python312\Scripts\;C:\Python312\;C:\Users\chido\AppData\Local\Temp\;C:\Users\chido\AppData\Local\Temp
TEMP	C:\Users\chido\AppData\Local\Temp
TMP	C:\Users\chido\AppData\Local\Temp

User variables for chido

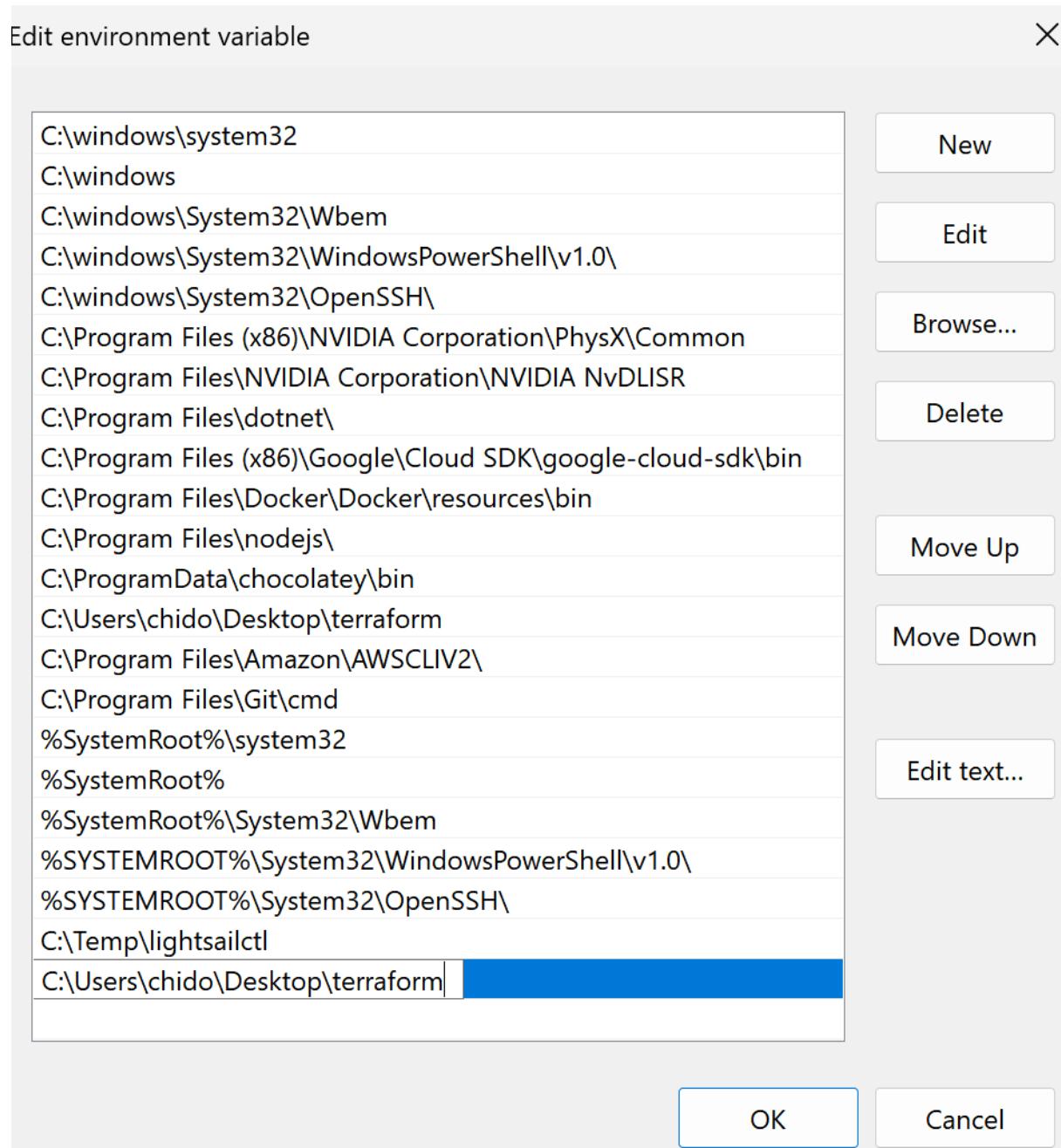
New... Edit... Delete

Variable	Value
ChocolateyInstall	C:\ProgramData\chocolatey
ComSpec	C:\WINDOWS\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
IGCCSVC_DB	AQAAANCMnd8BFdERjHoAwE/Cl+sBAAAI+AZzu+7fUG/vmZIX...
NUMBER_OF_PROCESSORS	20
OS	Windows_NT
Path	C:\windows\system32;C:\windows;C:\windows\System32\Wbem...;.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PATHEXT	

System variables

New... Edit... Delete

OK Cancel



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.

```
Microsoft Windows [Version 10.0.26100.3476]
(c) Microsoft Corporation. All rights reserved.

C:\Users\chido>terraform -v
Terraform v1.11.2
on windows_386

C:\Users\chido>
```

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.

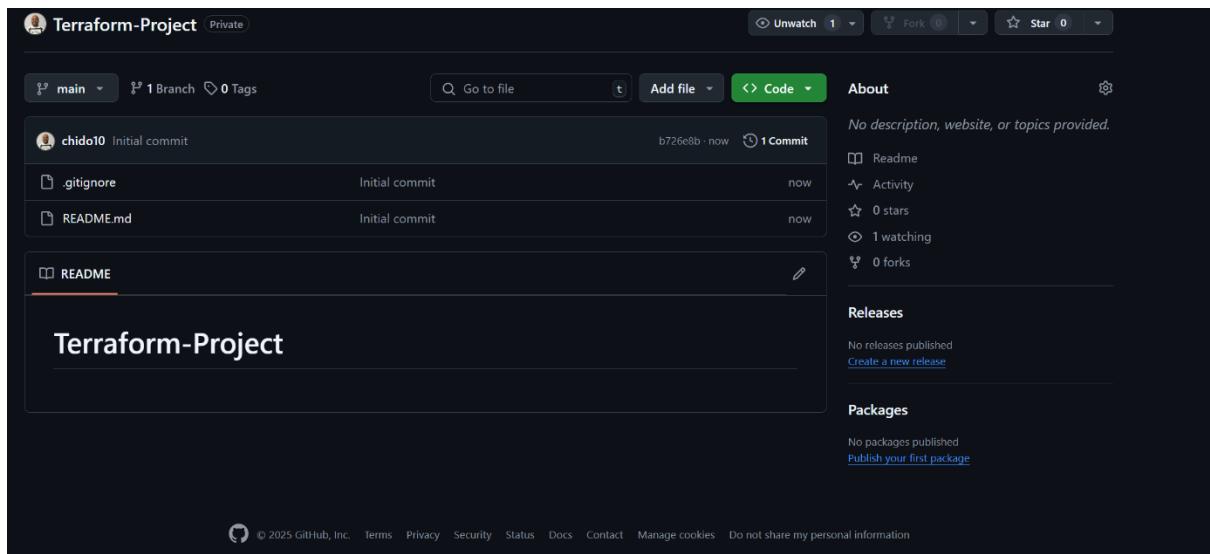
4. Create a GitHub Repository

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.*

The screenshot shows the GitHub 'Create a new repository' form. At the top, it says 'Create a new repository'. Below that, there's a note: 'A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)' A note below that says 'Required fields are marked with an asterisk (*).' The 'Repository template' section has a dropdown set to 'No template'. The 'Owner' field is set to 'chido10' and the 'Repository name' field is 'Terraform-Project', with a note 'Terraform-Project is available.' The 'Description (optional)' field is empty. The 'Visibility' section shows 'Public' (unchecked) and 'Private' (checked), with the note 'Anyone on the internet can see this repository. You choose who can commit.' The 'Initialize this repository with:' section has a checked checkbox for 'Add a README file', with the note 'This is where you can write a long description for your project. [Learn more about READMEs.](#)' The 'Add .gitignore' section has a dropdown for '.gitignore template' set to 'None'. A search bar shows 'terr' and a suggestion 'Terraform'. The note 'This will set `main` as the default branch. Change the default name in your [settings](#).' is at the bottom.



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 ...

More results from git-scm.com v

Git
System software :

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

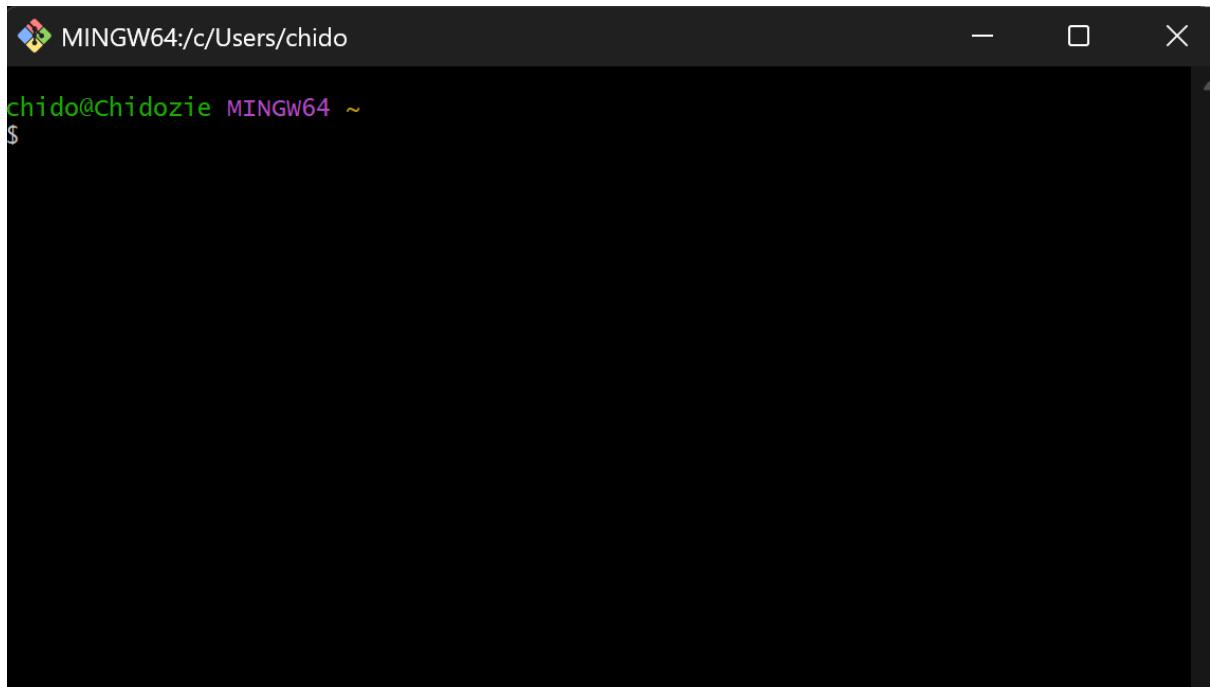
Install **winget tool** if you don't already have it, then type this command in command prompt or Powershell.

```
winget install --id Git.Git --source winget
```

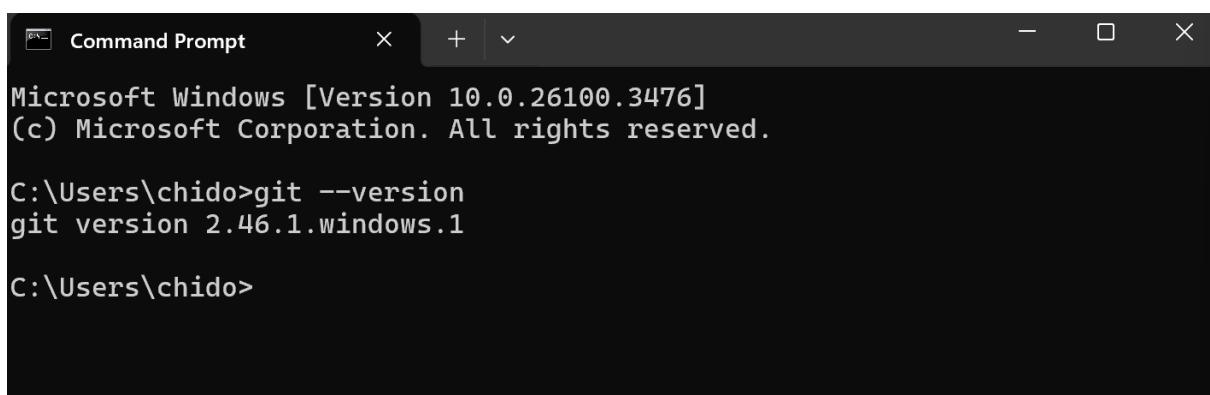
The current source code release is version **2.49.0**. If you want the newer version, you can build it from [the source code](#).

Now What?

Now that you have downloaded Git, it's time to start using it.



A screenshot of a terminal window titled "MINGW64:/c/Users/chido". The window has a dark background and light-colored text. It shows the command prompt "chido@chidozie MINGW64 ~" followed by a blank line where the user can type commands.



A screenshot of a Windows Command Prompt window titled "Command Prompt". The window has a dark background and light-colored text. It shows the command prompt "C:\Users\chido>" followed by the output of the "git --version" command: "git version 2.46.1.windows.1".

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.

```
Microsoft Windows [Version 10.0.26100.3476]
(c) Microsoft Corporation. All rights reserved.

C:\Users\chido>git config --global user.name "Chidozie Uzoegwu"
C:\Users\chido>
C:\Users\chido>git config --global user.email "chidozieuzoegwu@gmail.com"

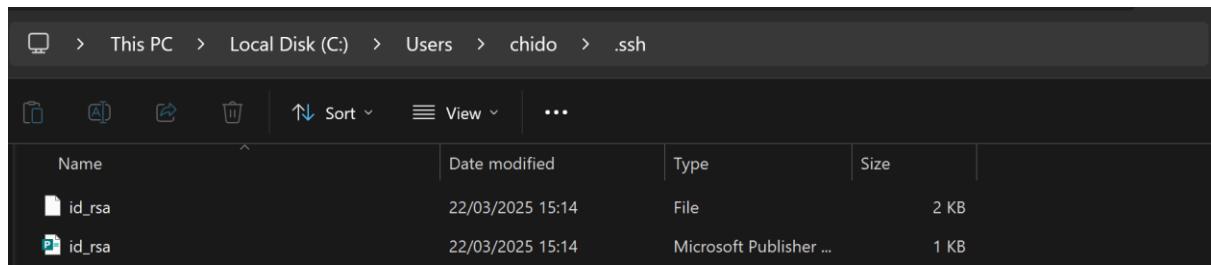
C:\Users\chido>
```

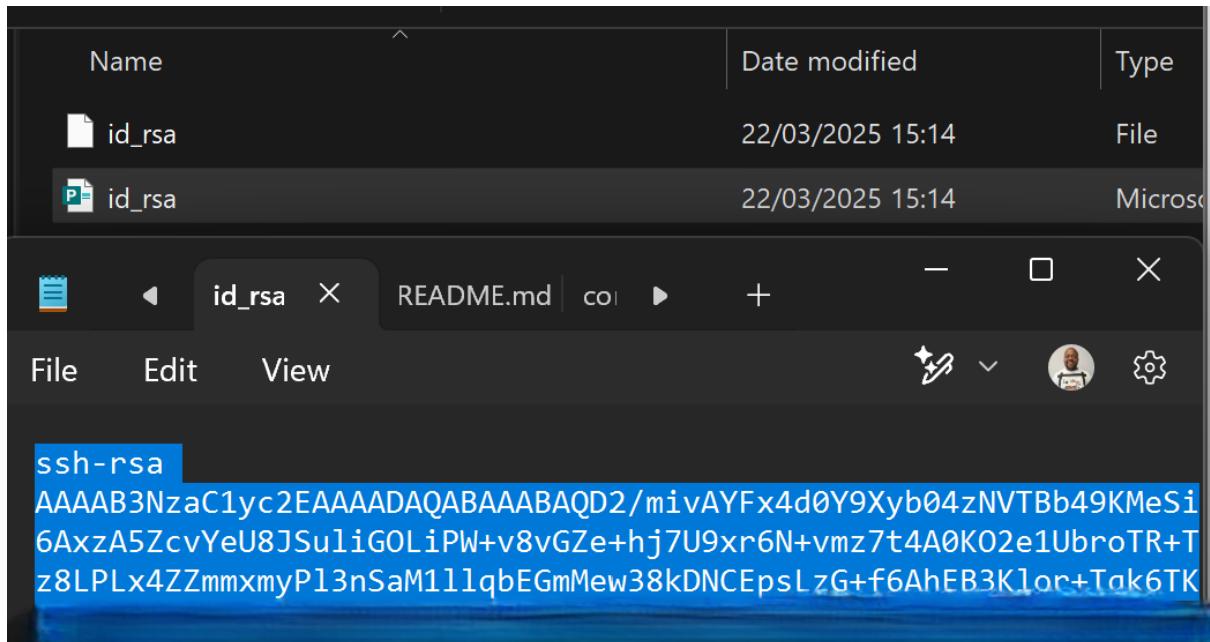
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\chido> ssh-keygen -t rsa -b 2048
Generating public/private rsa key pair.
Enter file in which to save the key (C:\Users\chido/.ssh/id_rsa):
C:\Users\chido/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in C:\Users\chido/.ssh/id_rsa
Your public key has been saved in C:\Users\chido/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:ossBG+J0/ha+FDZ6AcQBexbiXM8fgIMaLW/iRPqUAHI chido@Chidozie
The key's randomart image is:
+---[RSA 2048]---+
|++E=o.
|B***+. .
|+0 =.o .
|+.B . . .
|+=o =..S
|.o.+o.+
| o...o
| + oo+
| oo.=o.
+---[SHA256]---+
PS C:\Users\chido>
```

- **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).

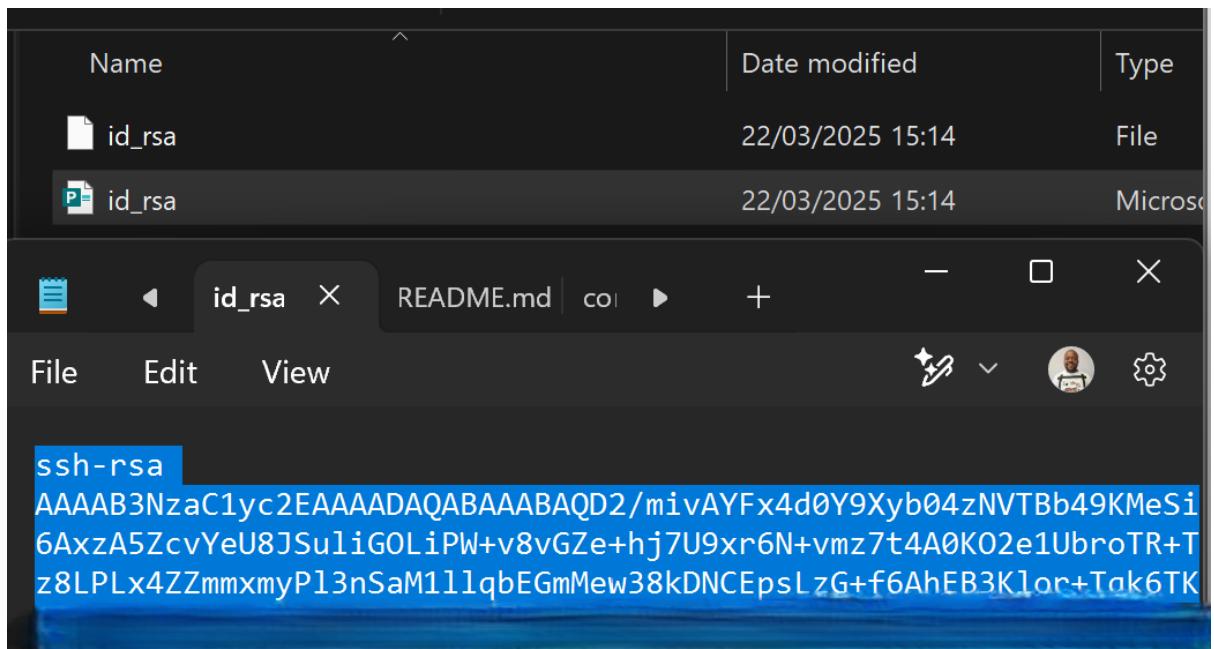
The SSH key pair will allow GitHub to recognize your computer. Next, we'll add the public key to your GitHub account.

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

The screenshot shows the GitHub profile settings page for a user named Chidozie Uzoegwu (chido10). The left sidebar contains navigation links for Public profile, Account, Appearance, Accessibility, Notifications, Access (Billing and plans, Emails, Password and authentication, Sessions), SSH and GPG keys (selected), Organizations, Enterprises, and Moderation. The main content area is titled "SSH keys" and displays a message: "There are no SSH keys associated with your account." It includes a link to "connecting to GitHub using SSH keys" and "common SSH problems". A green "New SSH key" button is located at the top right of this section. Below it is a "GPG keys" section with a similar message and a "New GPG key" button. At the bottom of the page is a "Vigilant mode" section with a checkbox for "Flag unsigned commits as unverified".

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 the "Add new SSH Key" form. It has fields for "Title" (containing "my-public-Key"), "Key type" (set to "Authentication Key"), and a large "Key" text area containing a long public SSH key starting with "ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQD2/mivAYFx4d0Y9Xyb04zNVTBb49KMeSi6AxzA5ZcvYeU8JSuliGOLiPW+v8vGZe+hj7U9xr6N+vmz7t4A0K O2e1ubroTR+Tz8LPLx4ZZmmxmyPl3nSaM1llqbEGmMeww38kDNCFncl zG+f6AhEB3Klor+Tak6TKnGarc5ViIloOo3A3lvs6ml lP-Rw7al ncelIR38xFXC". A green "Add SSH key" button is at the bottom left.

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.

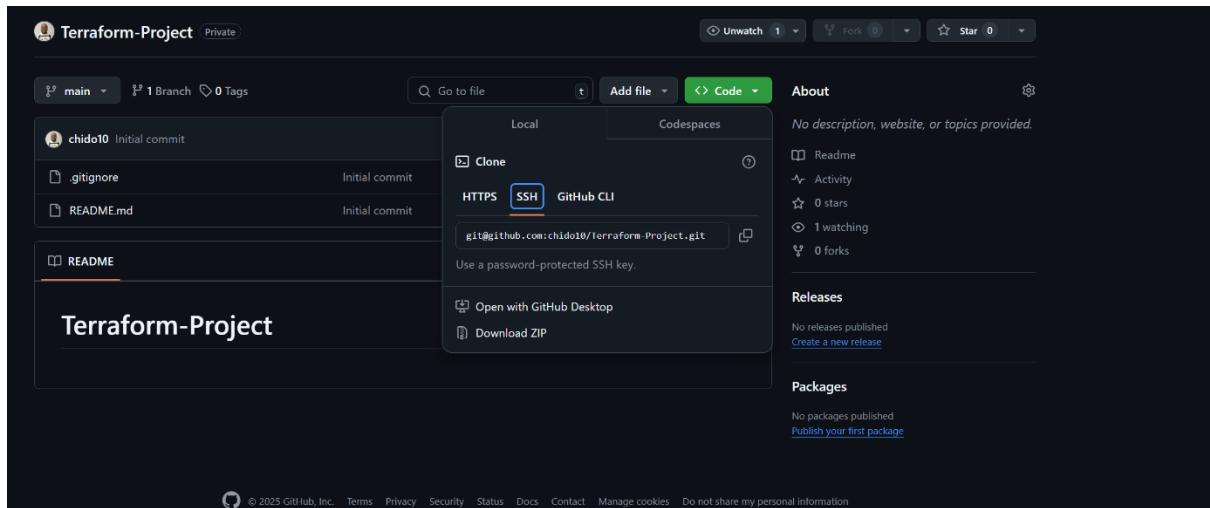
The screenshot shows the GitHub user profile page for Chidozie Uzoegwu (chido10). The left sidebar has sections like Public profile, Account, Appearance, Accessibility, Notifications, Billing and plans, Emails, Password and authentication, Sessions, SSH and GPG keys (which is selected), Organizations, Enterprises, and Moderation. The main content area is titled "SSH keys" and shows one key: "my-public-Key". It includes the SSH key value, a timestamp (Added on Mar 22, 2025), and a note (Never used — Read/write). A "Delete" button is also present. Below this is a section for "GPG keys" which states "There are no GPG keys associated with your account." A "Vigilant mode" section is also shown.

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.

A screenshot of a Microsoft Windows Command Prompt window. The title bar says 'Command Prompt'. The window displays the following text:

```
Microsoft Windows [Version 10.0.26100.3476]
(c) Microsoft Corporation. All rights reserved.

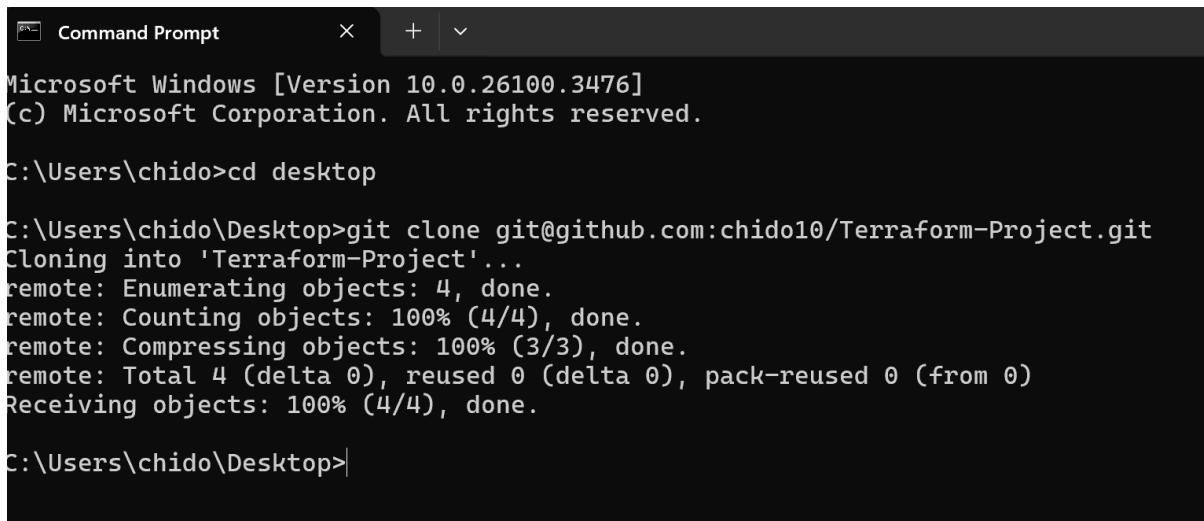
C:\Users\chido>cd desktop

C:\Users\chido\Desktop>git clone git@github.com:chido10/Terraform-Project.git
Cloning into 'Terraform-Project'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (4/4), done.

C:\Users\chido\Desktop>
```

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.).

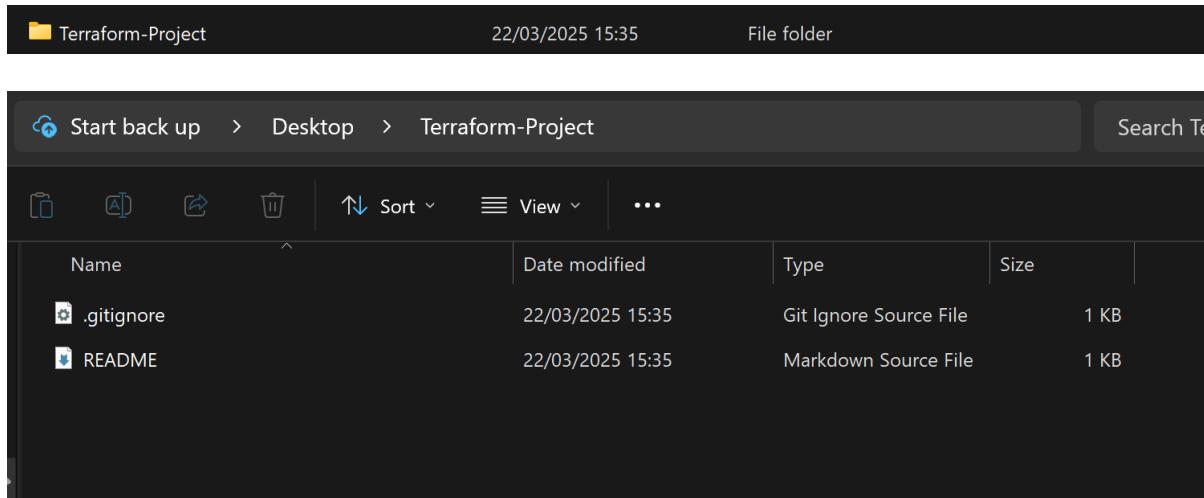


```
Command Prompt
Microsoft Windows [Version 10.0.26100.3476]
(c) Microsoft Corporation. All rights reserved.

C:\Users\chido>cd desktop

C:\Users\chido\Desktop>git clone git@github.com:chido10/Terraform-Project.git
Cloning into 'Terraform-Project'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (4/4), done.

C:\Users\chido\Desktop>
```



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.”

A screenshot of a GitHub repository page for 'Terraform-Project'. The repository is private. It shows a single commit by 'chido10' titled 'Initial commit' made 40 minutes ago. The commit includes changes to '.gitignore' and 'README.md'. The 'About' section indicates no description, website, or topics provided. The 'Releases' section shows no releases published, with a link to 'Create a new release'. The 'Packages' section shows no packages published, with a link to 'Publish your first package'.

A screenshot of a file manager interface showing the contents of the 'Terraform-Project' directory. The directory contains two files: '.gitignore' and 'README'. Both files were modified on 22/03/2025 at 15:35. The 'README' file is a Markdown source file. A preview window is open, displaying the content of the 'README' file:

```
# 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.

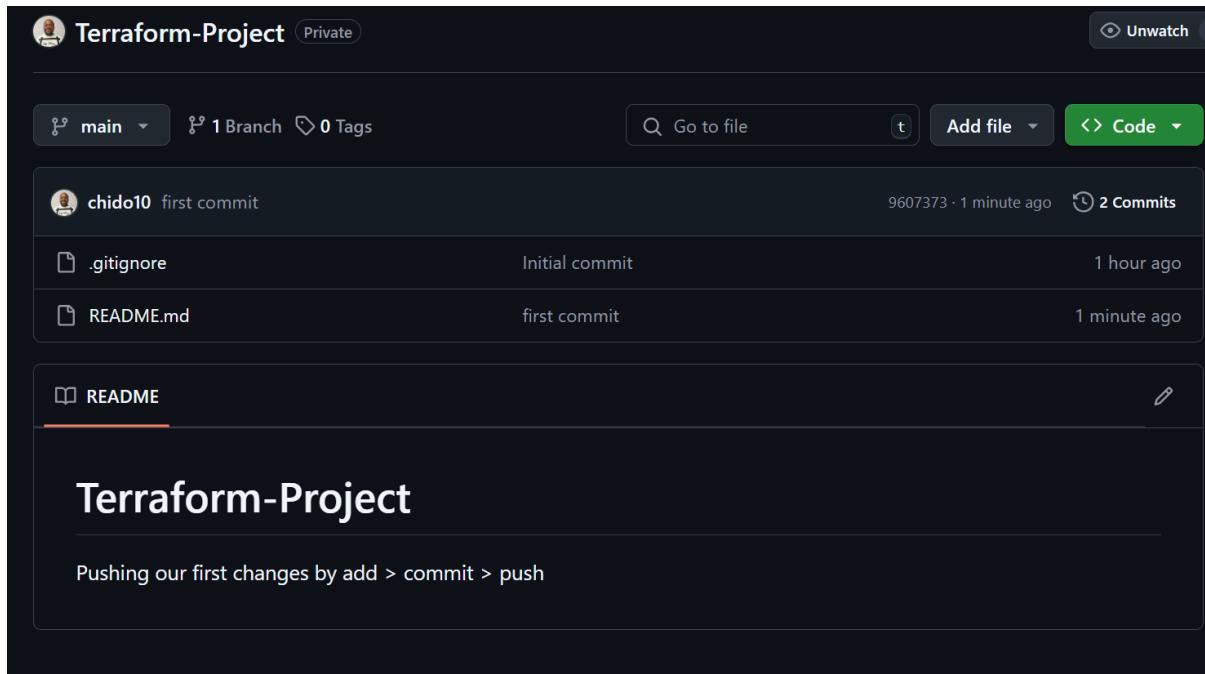
```
C:\Users\chido\Desktop>cd C:\Users\chido\Desktop\Terraform-Project
C:\Users\chido\Desktop\Terraform-Project>git add -A
C:\Users\chido\Desktop\Terraform-Project>git commit -m "first commit"
[main 9607373] first commit
 1 file changed, 3 insertions(+), 1 deletion(-)

C:\Users\chido\Desktop\Terraform-Project>git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 20 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 350 bytes | 87.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To github.com:chido10/Terraform-Project.git
 b726e8b..9607373  main -> main

C:\Users\chido\Desktop\Terraform-Project>
```

- **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:

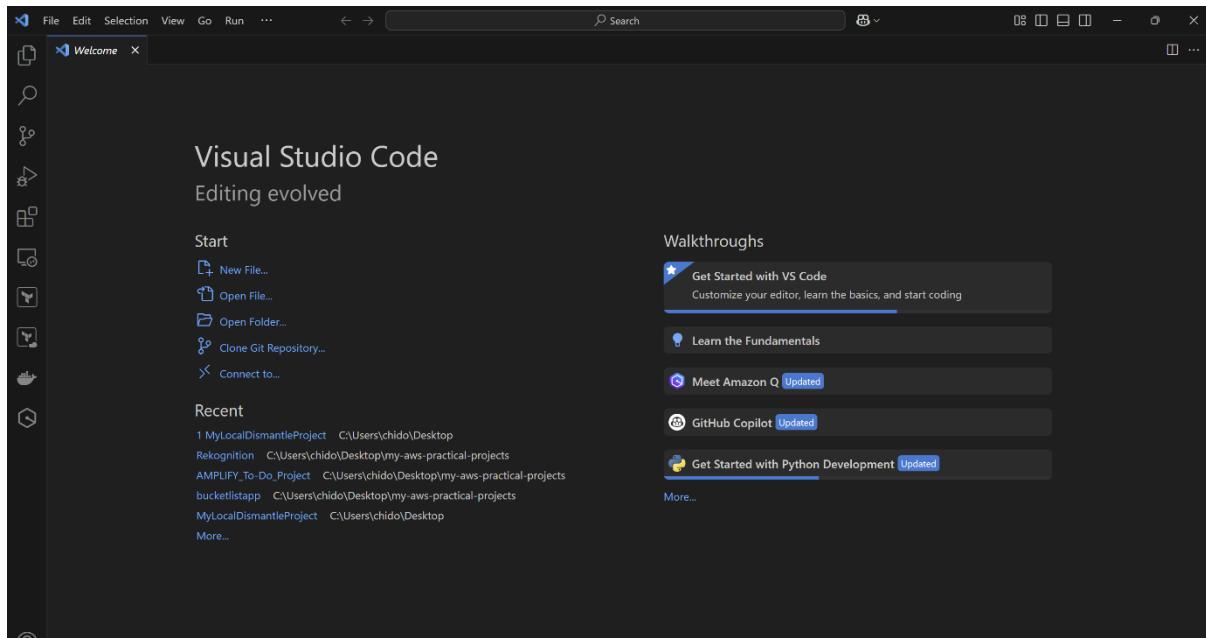
- **Download VS Code:** Go to the [Visual Studio Code website](#) and download the installer for Windows. Choose the correct installer (User Installer or System Installer, 64-bit for most modern systems).

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.



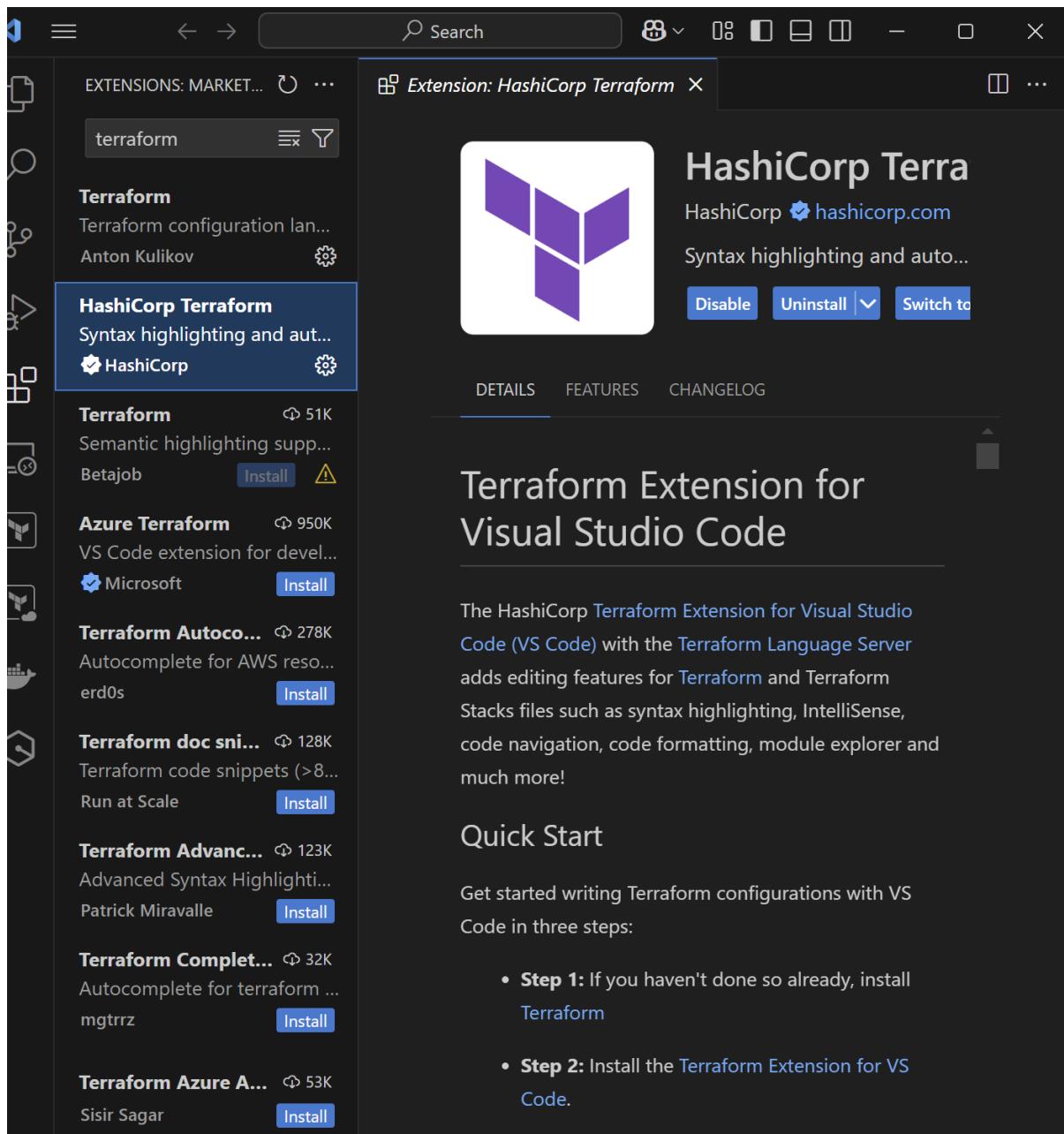
Having VS Code will make it easier to edit multiple Terraform files, and next we'll enhance it with Terraform-specific extensions.

12. Install the Extensions for Terraform

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

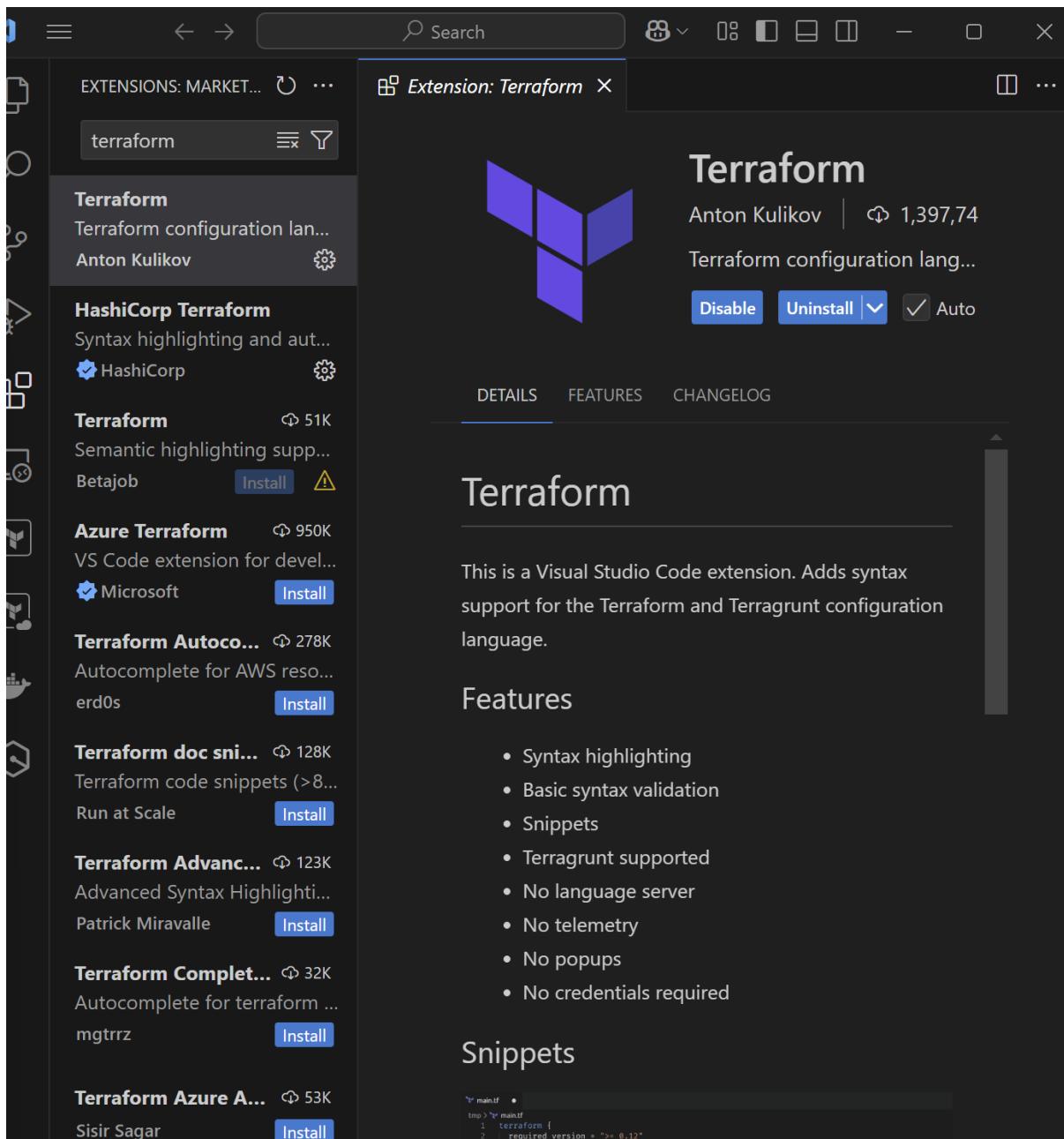
- **HashiCorp Terraform Extension:** In VS Code, click the Extensions icon (on the left toolbar, looks like building blocks) and search for “HashiCorp Terraform”. Install the official **HashiCorp Terraform** extension. This adds syntax highlighting, autocompletion, and integration with Terraform Language 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

aws command line interface

All Videos Images Short videos Shopping Web News More Tools

 Amazon AWS Documentation
<https://docs.aws.amazon.com/cli/getting-started-install> :

Installing or updating to the latest version of the AWS CLI

This topic describes how to install or update the latest release of the **AWS Command Line Interface** (AWS CLI) on supported operating systems.

Windows
Install version 1 of the AWS Command Line Interface (AWS ...)

Linux
Download using the curl command. ... Download using the direct ...

macOS
Install the AWS Command Line Interface (AWS CLI) version 1 ...

Past releases
... AWS Command Line Interface version 2 (AWS CLI) on ...

Setting up the AWS CLI
AWS CLI installation guide: Download installer, unzip, run ...

[More results from amazon.com »](#)

Install or update the AWS CLI

To update your current installation of AWS CLI on Windows, download a new installer each time you update to overwrite previous versions. AWS CLI is updated regularly. To see when the latest version was released, see the [AWS CLI version 2 Changelog](#) on GitHub.

1. Download and run the AWS CLI MSI installer for Windows (64-bit):
<https://awscli.amazonaws.com/AWSCLIV2.msi>
Alternatively, you can run the `msiexec` command to run the MSI installer.

- **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.

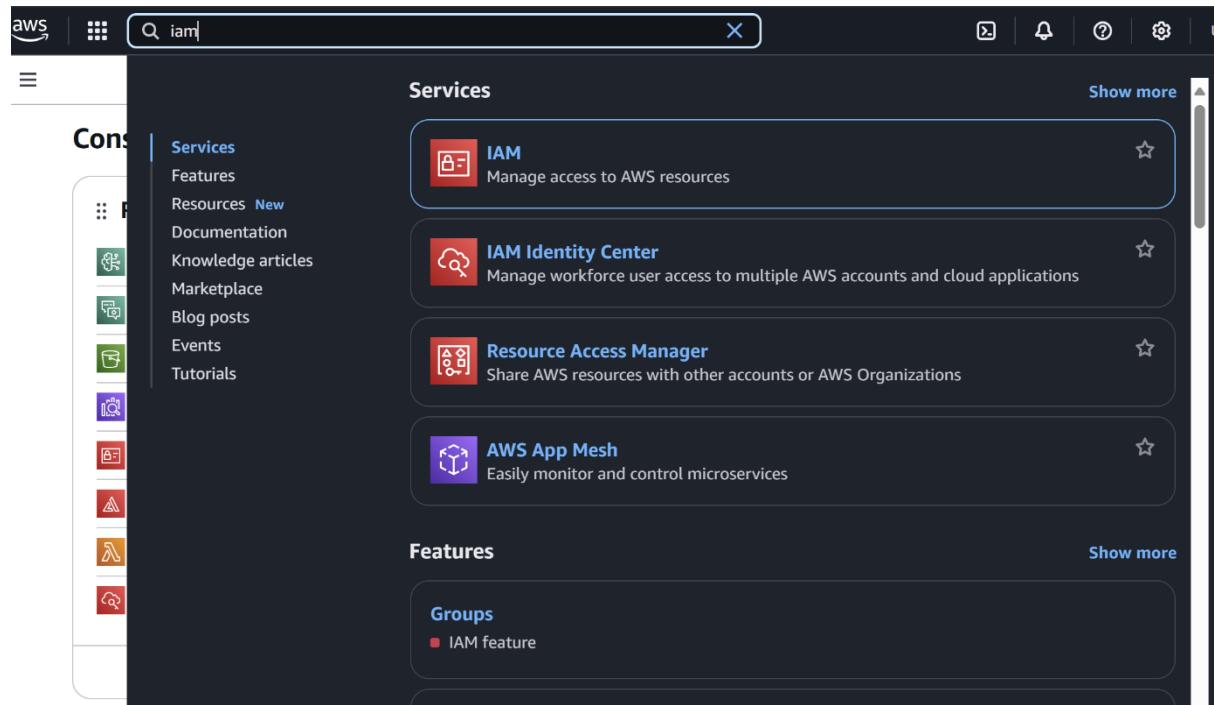
```
Command Prompt X + ▾  
Microsoft Windows [Version 10.0.26100.3476]  
(c) Microsoft Corporation. All rights reserved.  
  
C:\Users\chido>aws --version  
aws-cli/2.15.59 Python/3.11.8 Windows/10 exe/AMD64  
  
C:\Users\chido>
```

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.

The screenshot shows the 'Create user' wizard in the AWS IAM console. The navigation bar at the top shows 'IAM > Users > Create user'. The title 'Step 1 of 3' and section 'Specify user details' are displayed. A 'User details' card contains a 'User name' field with 'terraform-user' entered. Below it is a note about character restrictions and a checkbox for 'Provide user access to the AWS Management Console - optional'. A callout box provides information about generating programmatic access after user creation. At the bottom are 'Cancel' and 'Next' buttons.

User details

User name

terraform-user

The user name can have up to 64 characters. Valid characters: A-Z, a-z, 0-9, and + = , . @ _ - (hyphen)

Provide user access to the AWS Management Console - optional
If you're providing console access to a person, it's a [best practice](#) to manage their access in IAM Identity Center.

i If you are creating programmatic access through access keys or service-specific credentials for AWS CodeCommit or Amazon Keypairs, you can generate them after you create this IAM user. [Learn more](#)

Cancel **Next**

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.



Permissions options

- Add user to group
Add user to an existing group, or create a new group. We recommend using groups to manage user permissions by job function.
- Copy permissions
Copy all group memberships, attached managed policies, and inline policies from an existing user.
- Attach policies directly
Attach a managed policy directly to a user. As a best practice, we recommend attaching policies to a group instead. Then, add the user to the appropriate group.

Permissions policies (1368)

Choose one or more policies to attach to your new user.

Filter by Type

Search

All types ▾

< 1 2 3 4 5 6 7 ... 69 > ⚙

Create policy

The screenshot shows the AWS IAM 'Create user' interface. At the top, the navigation path is 'IAM > Users > Create user'. Below the path, a message says 'Choose one or more policies to attach to your new user.' A search bar and a 'Filter by Type' dropdown are present. The main area displays a table of policies:

	Policy name	Type	Attached e...
<input type="checkbox"/>	AccessAnalyzerSer...	AWS managed	0
<input checked="" type="checkbox"/>	AdministratorAccess	AWS managed - job f...	5
<input type="checkbox"/>	AdministratorAcce...	AWS managed	0
<input type="checkbox"/>	AdministratorAcce...	AWS managed	0
<input type="checkbox"/>	AIOpsAssistantPolicy	AWS managed	0
<input type="checkbox"/>	AIOpsConsoleAdmi...	AWS managed	0
<input type="checkbox"/>	AIOpsOperatorAcc...	AWS managed	0
<input type="checkbox"/>	AIOpsReadOnlyAcc...	AWS managed	0
<input type="checkbox"/>	AlexaForBusinessD...	AWS managed	0
<input type="checkbox"/>	AlexaForBusinessF...	AWS managed	0
<input type="checkbox"/>	AlexaForBusinessG...	AWS managed	0

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.

The screenshot shows the AWS IAM Users page. On the left, there's a sidebar with navigation links for Identity and Access Management (IAM), Dashboard, Access management (User groups, Roles, Policies, Identity providers, Account settings, Root access management), and Access reports (Access Analyzer, External access, Unused access, Analyzer settings, Credential report, Organization activity, Service control policies). The main area has a green success message: "User created successfully. You can view and download the user's password and email instructions for signing in to the AWS Management Console." Below this, it says "Users (5) Info" and provides a brief description: "An IAM user is an identity with long-term credentials that is used to interact with AWS in an account." A search bar is present. The main content is a table with columns: User name, Path, Groups, Last activity, MFA, Password age, and Console last sign-in. The table lists five users, with "terraform-user1" being the most recent addition.

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.

The screenshot shows the AWS IAM User Details page for a user named 'terraform-user1'. The left sidebar contains navigation links for IAM, Users, Access management, and Access reports. The main content area displays the user's ARN, creation date, and access status. It also includes tabs for Permissions, Groups, Tags, Security credentials (which is selected), and Last Accessed. Under the Security credentials tab, there is a 'Console sign-in' section with a 'Console sign-in link' and a 'Console password' field. A 'Multi-factor authentication (MFA)' section indicates 0 MFA devices assigned. A 'Create access key' button is located in the top right corner of the main content area.

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).

The screenshot shows the AWS IAM Access Keys page, which is currently empty. It features a 'Create access key' button in the top right corner. Below it, there is a note about using access keys for programmatic calls and a warning against using long-term credentials like access keys. A 'Create access key' button is also located at the bottom center of the page.

- Step 1
 - Access key best practices & alternatives
 - Step 2 - optional
 - Set description tag
 - Step 3
 - Retrieve access keys

Access key best practices & alternatives Info

Avoid using long-term credentials like access keys to improve your security. Consider the following use cases and alternatives.

Use case

- Command Line Interface (CLI)
You plan to use this access key to enable the AWS CLI to access your AWS account.
- Local code
You plan to use this access key to enable application code in a local development environment to access your AWS account.
- Application running on an AWS compute service
You plan to use this access key to enable application code running on an AWS compute service like Amazon EC2, Amazon ECS, or AWS Lambda to access your AWS account.
- Third-party service
You plan to use this access key to enable access for a third-party application or service that monitors or manages your AWS resources.
- Application running outside AWS
You plan to use this access key to authenticate workloads running in your data center or other infrastructure outside of AWS that needs to access your AWS resources.
- Other
Your use case is not listed here.

- **(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.*

Access key created

This is the only time that the secret access key can be viewed or downloaded. You cannot recover it later. However, you can create a new access key any time.

- Step 1
 - Access key best practices & alternatives
 - Step 2 - optional
 - Set description tag
 - Retrieve access keys

Retrieve access keys Info

Access key

If you lose or forget your secret access key, you cannot retrieve it. Instead, create a new access key and make the old key inactive.

Access key | Secret access key

 AKIAVWVZU2QD4T2F1LFA  ***** Show

Access key best practices

- Never store your access key in plain text, in a code repository, or in code.
- Disable or delete access key when no longer needed.
- Enable least-privilege permissions.
- Rotate access keys regularly.

For more details about managing access keys, see the [best practices for managing AWS access keys](#).

[Download .csv file](#)

[Done](#)

The screenshot shows the AWS IAM User Details page for a user named 'terraform-user1'. The user was created on March 22, 2025, at 16:13 UTC. They have an ARN of arn:aws:iam::375630861224:user/terraform-user1. Console access is disabled, and the last console sign-in was on March 22, 2025, at 16:13 UTC. There are two access keys: Access key 1 (AKIAVORIUVJF...), which is never used, and Access key 2 (Create access key). The Security credentials tab is selected. Other tabs include Permissions, Groups, Tags (1), and Last Accessed.

Console sign-in

Console sign-in link: <https://signin.aws.amazon.com/console>

Console password: Not enabled

Multi-factor authentication (MFA) (0)

Use MFA to increase the security of your AWS environment. Signing in with MFA requires an authentication code from an MFA device. Each user can have a maximum of 8 MFA devices assigned. [Learn more](#)

Buttons: Remove, Resync, Assign MFA device

Type	Identifier	Certifications	Created on
------	------------	----------------	------------

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:

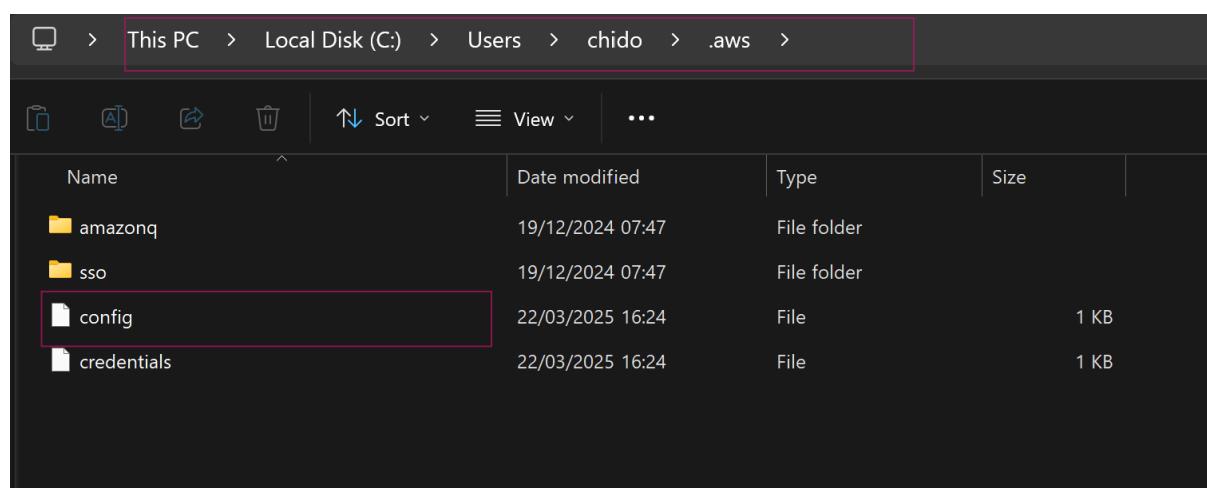
- **AWS Configure:** Open Command Prompt and run:
aws configure --profile terraform-user
When prompted:
 - Enter the **Access Key ID** (from previous step).
 - Enter the **Secret Access Key**.
 - Default region name: choose your desired region (e.g., **us-east-1** or the region you want to create resources 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.

```
Command Prompt  
Microsoft Windows [Version 10.0.26100.3476]  
(c) Microsoft Corporation. All rights reserved.  
  
C:\Users\chido>aws --version  
aws-cli/2.15.59 Python/3.11.8 Windows/10 exe/AMD64  
  
C:\Users\chido>aws configure --profile terraform-user  
AWS Access Key ID [*****JYXE]: AKIAVQF*****  
AWS Secret Access Key [*****cCa0]: Q*****  
Default region name [us-east-1]: us-east-1  
Default output format [None]:  
  
C:\Users\chido>
```



The screenshot shows the Windows File Explorer interface. The path is: This PC > Local Disk (C:) > Users > chido > .aws >. The contents of the .aws folder are:

Name	Date modified	Type
amazonq	19/12/2024 07:47	File folder
sso	19/12/2024 07:47	File folder
config	22/03/2025 16:24	File
credentials	22/03/2025 16:24	File

Below is a screenshot of a code editor showing the AWS config file. The file content is:

```
[default]
region=eu-west-2
output=json

[profile terraform-user]
region=us-east-1

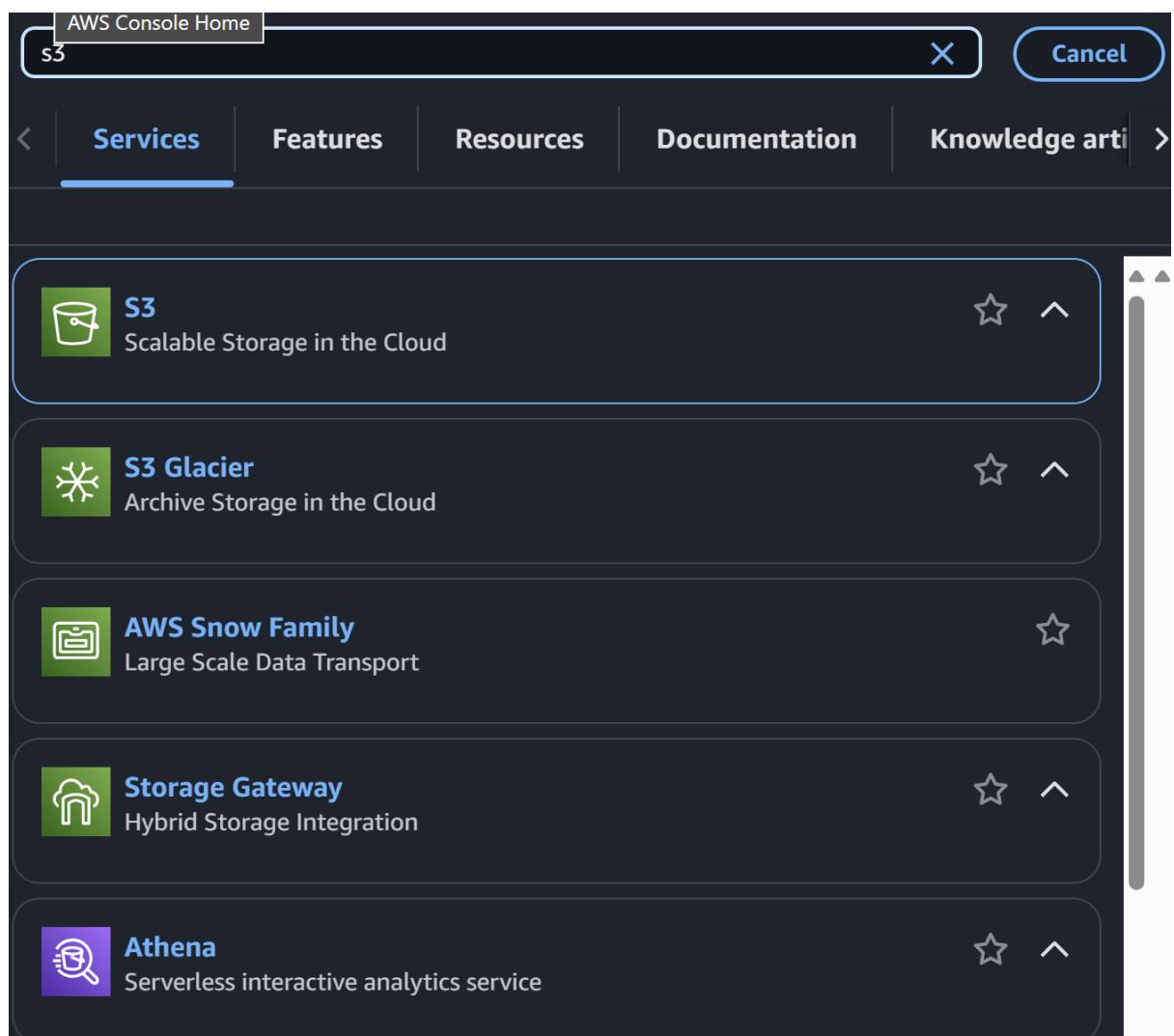
[profile amplify-dev]
region=eu-west-2
[profile myOtherProfile]
region = eu-west-2
output = json
```

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).



Create bucket Info

Buckets are containers for data stored in S3.

General configuration

AWS Region

US East (N. Virginia) us-east-1

Bucket type Info

General purpose

Recommended for most use cases and access patterns. General purpose buckets are the original S3 bucket type. They allow a mix of storage classes that redundantly store objects across multiple Availability Zones.

Directory

Recommended for low-latency use cases. These buckets use only the S3 Express One Zone storage class, which provides faster processing of data within a single Availability Zone.

Bucket name Info

chidiozie-terraform-remote-state1

Bucket names must be 3 to 63 characters and unique within the global namespace. Bucket names must also begin and end with a letter or number. Valid characters are a-z, 0-9, periods (.), and hyphens (-). [Learn More](#)

Copy settings from existing bucket - optional

Only the bucket settings in the following configuration are copied.

[Choose bucket](#)

Format: s3://bucket/prefix

Object Ownership Info

Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects.

ACLs disabled (recommended)

All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using

ACLs enabled

Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be

▶ Account snapshot - updated every 24 hours All AWS Regions

Storage lens provides visibility into storage usage and activity trends. Metrics don't include directory buckets. [Learn more](#)

[View Storage Lens dashboard](#)

[General purpose buckets](#)

[Directory buckets](#)

General purpose buckets (15) Info All AWS Regions

Buckets are containers for data stored in S3.

Find buckets by name



[Copy ARN](#)

[Empty](#)

[Delete](#)

[Create bucket](#)

< 1 > |

Bucket Versioning

Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. [Learn more](#)

Bucket Versioning

Disable

Enable

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**.

Bucket Versioning

Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. [Learn more](#)

Bucket Versioning

Disable

Enable

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

The screenshot displays a search result from Google and a code editor side-by-side.

Search Results (Left):

- Terraform Registry:** https://registry.terraform.io/docs/resources/vpc
 - aws_vpc | Resources | hashicorp/aws**
 - Aws_vpc_peering_connection · Aws_vpc_endpoint · Aws_vpc_endpoint_service
- Spacelift:** https://spacelift.io/blog/terraform-aws-vpc
 - How to Build AWS VPC using Terraform – Step by Step**
 - Learn how to create and develop a basic AWS VPC using Terraform and how to set up VPC endpoints and peering. Terraform VPC module explained.
- GitHub:** https://github.com/terraform-aws-modules/terraform-aws-vpc
 - Terraform module to create AWS VPC resources**
 - This module supports three scenarios for creating NAT gateways. Each will be explained in further detail in the corresponding sections.
 - Main.tf · Vpc-flow-logs.tf · Outputs.tf · Variables.tf
- Terraform Registry:** https://registry.terraform.io/resources/cloudfront_distribution
 - Terraform AWS Provider**
 - Use HCP Terraform for free. Browse. Providers Modules Policy Libraries Beta ... aws · Intro Learn Docs

Code Editor (Right):

```
example.tf
C: > Users > chido > Desktop > example.tf
1 Press [Ctrl]+[I] to ask Code to do something. Start typing to dismiss

resource "aws_vpc" "main" {
  cidr_block      = "10.0.0.0/16"
  instance_tenancy = "default"

  tags = {
    Name = "main"
  }
}
```

The above code creates a VPC with a CIDR block of 10.0.0.0/16.

Bottom Panel:

Basic usage with tags:

```
resource "aws_vpc" "main" {
  cidr_block      = "10.0.0.0/16"
  instance_tenancy = "default"

  tags = {
    Name = "main"
  }
}
```

Copy

VPC with CIDR from AWS IPAM:

AWS VPC Dashboard (Bottom):

Create VPC Launch EC2 Instances

Note: Your instances will launch in the United States region.

Resources by Region:

- VPCs (United States 1) See all regions
- NAT Gateways (United States 0) See all regions
- Subnets (United States 6) See all regions
- VPC Peering Connections (United States 0) See all regions
- Route Tables (United States 1) See all regions
- Network ACLs (United States 1) See all regions
- Internet Gateways (United States 1) See all regions
- Security Groups (United States 3) See all regions
- Egress-only Internet Gateways (United States 0) See all regions
- Customer Gateways (United States 0) See all regions
- DHCP option sets (United States 1)
- Virtual Private Gateways (United States 0)

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 tools and features to help you manage and monitor your network on AWS. Network Manager makes it easier to perform connectivity management, network monitoring and

AWS | Search | AWS Console Home > Create VPC

Create VPC Info

A VPC is an isolated portion of the AWS Cloud populated by AWS objects, such as .

VPC settings

Resources to create Info
Create only the VPC resource or the VPC and other networking resources.

VPC only VPC and more

Name tag auto-generation Info

Enter a value for the Name tag. This value will be used to auto-generate Name tags for all resources in the VPC.

Auto-generate
project

IPv4 CIDR block Info

Determine the starting IP and the size of your VPC using CIDR notation.

10.0.0.0/16 65,536 IPs

CIDR block size must be between /16 and /28.

IPv6 CIDR block Info

No IPv6 CIDR block Amazon-provided IPv6 CIDR block

Tenancy Info

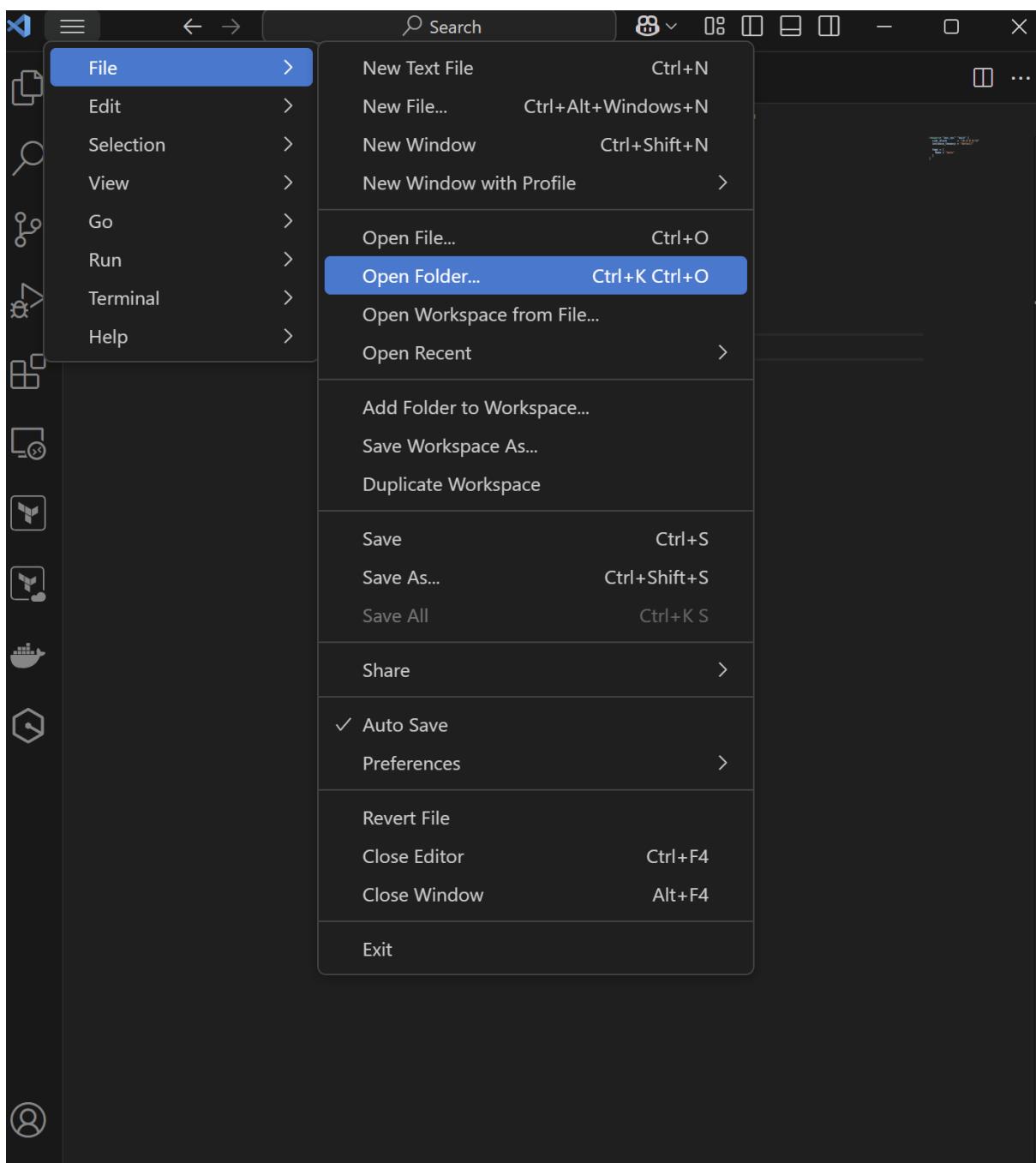
Default ▾

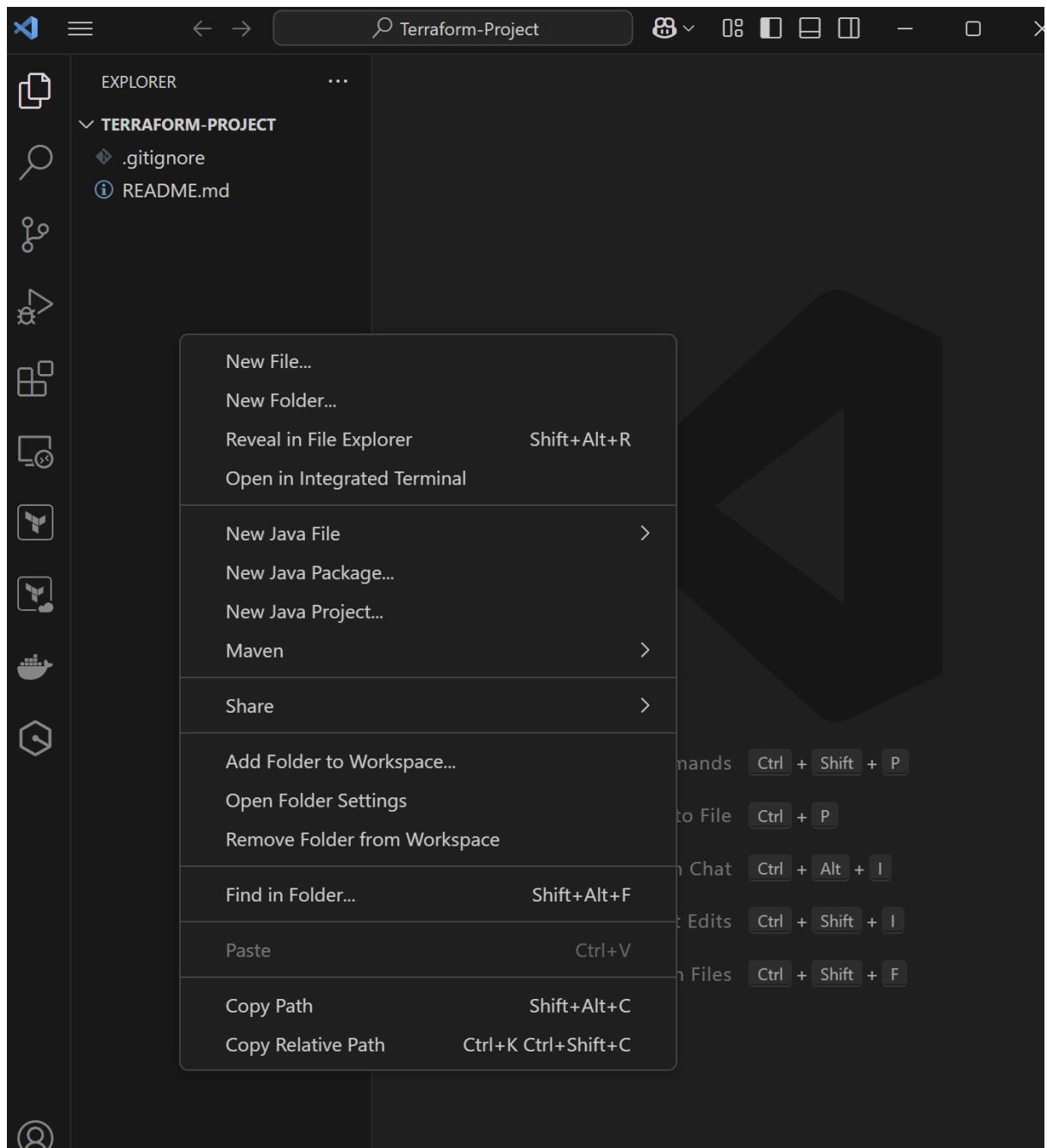
- Reference resources by name to connect them (e.g., subnet IDs in other resources).

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

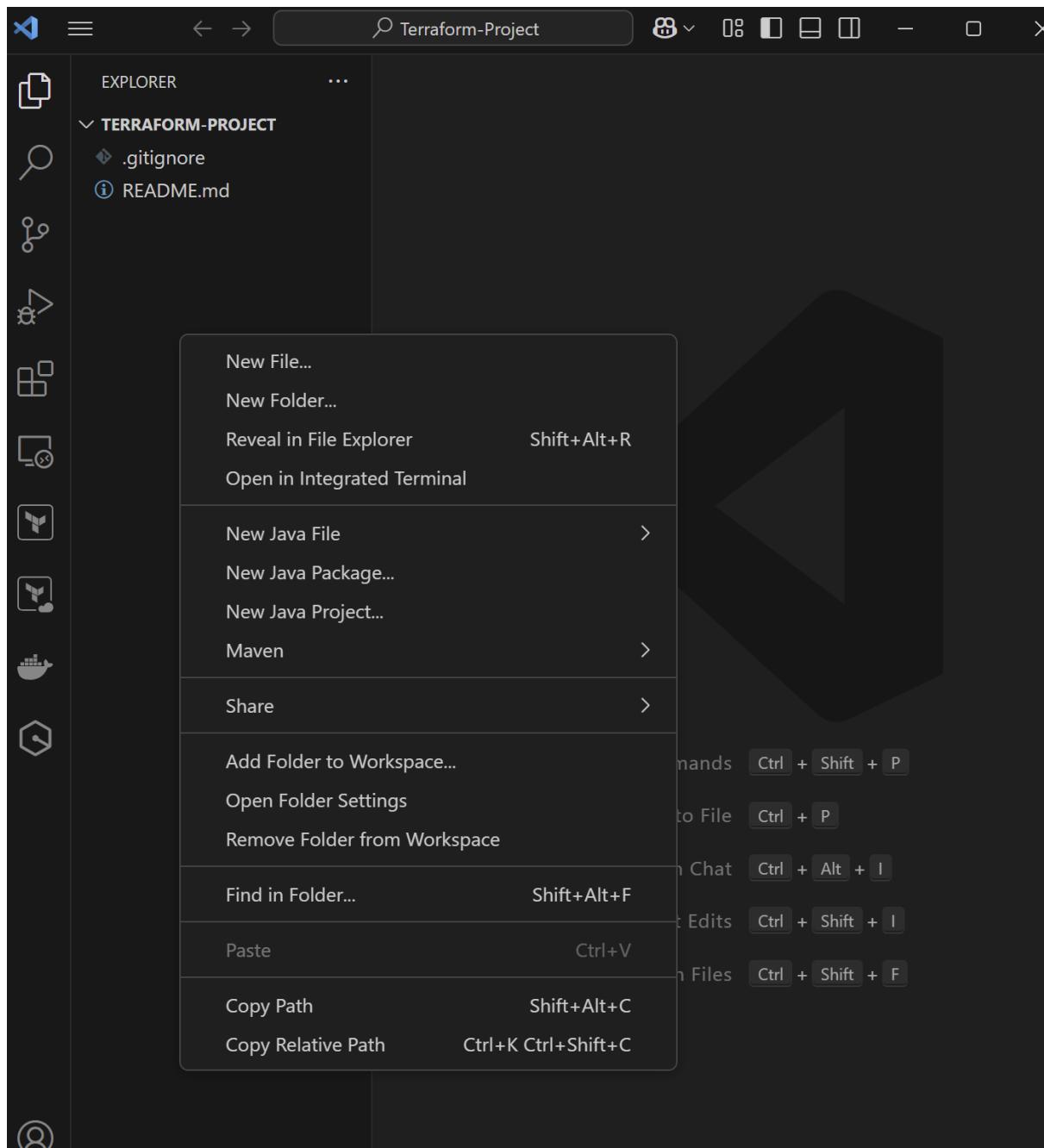
19. Authenticate Terraform with AWS

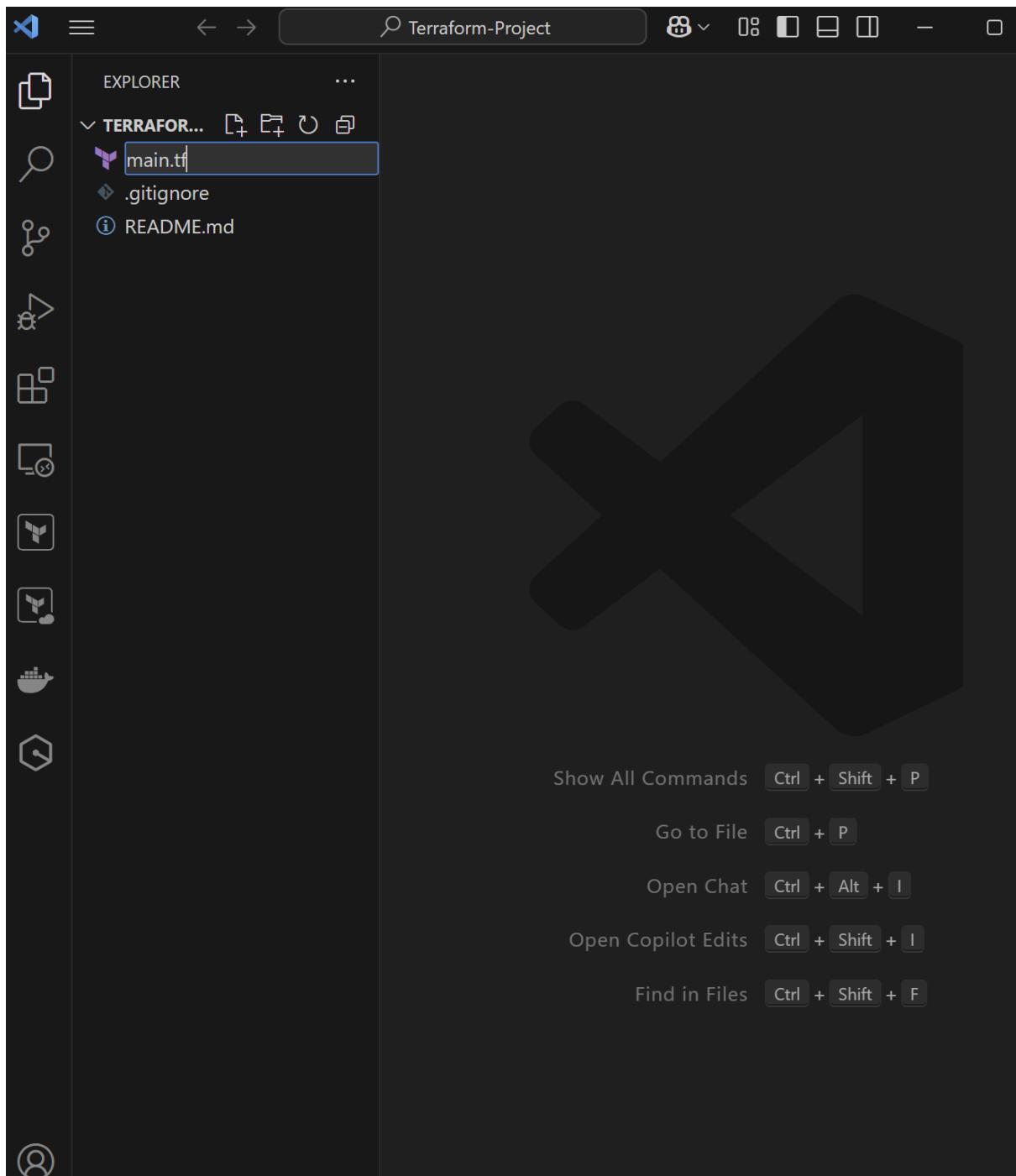
Before provisioning, we will first open our terraform folder in VS code.



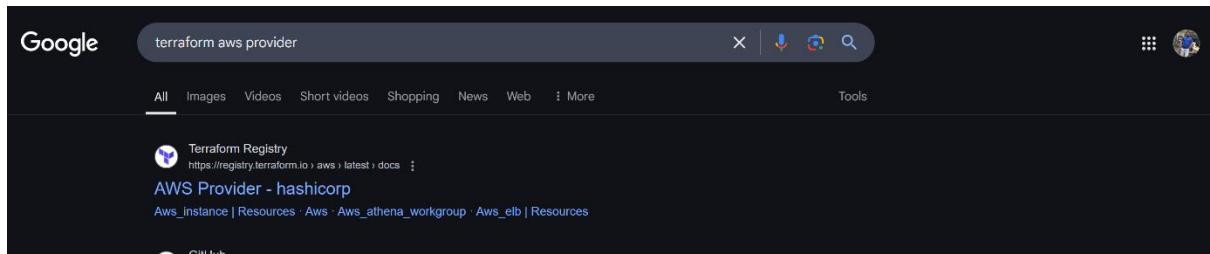


Create a main.tf file inside our folder.





- **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:



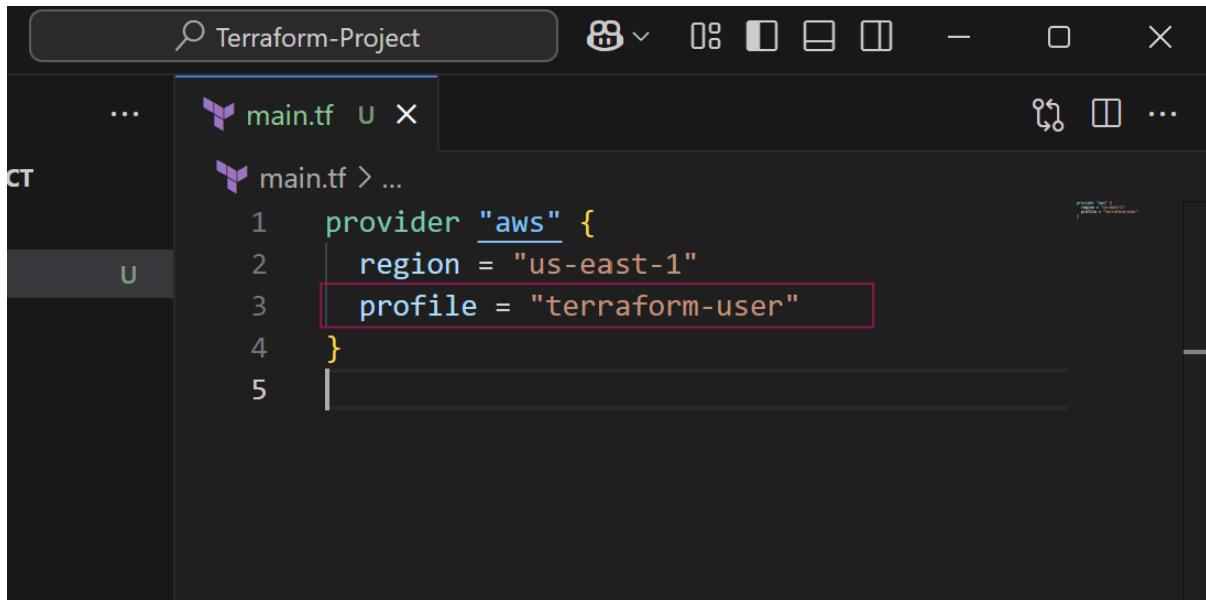
This screenshot shows the "Example Usage" section of the Terraform Registry documentation for the AWS Provider. It includes examples for Terraform 0.13 and later, and Terraform 0.12 and earlier. The code snippets demonstrate how to configure the AWS provider with a specific region (us-east-1) and create a VPC resource.

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

resource "aws_vpc" "example" {
  cidr_block = "10.0.0.0/16"
}
```

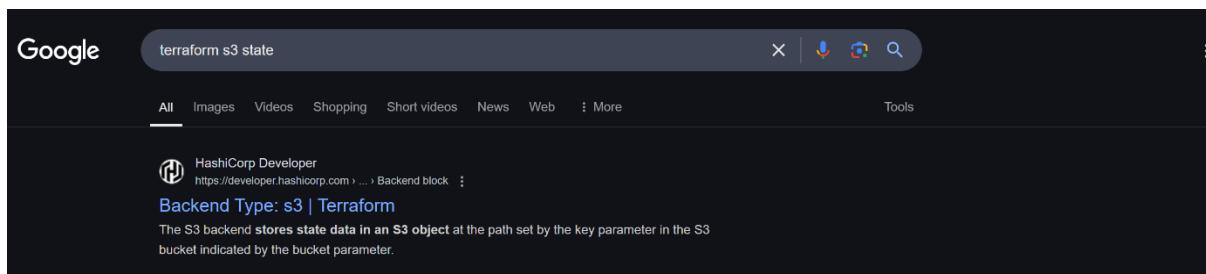
This screenshot shows a code editor window displaying a Terraform configuration file named "main.tf". The code defines a provider block for "aws" with the region set to "us-east-1".

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



A screenshot of a code editor window titled "Terraform-Project". The main pane displays a file named "main.tf" with the following content:

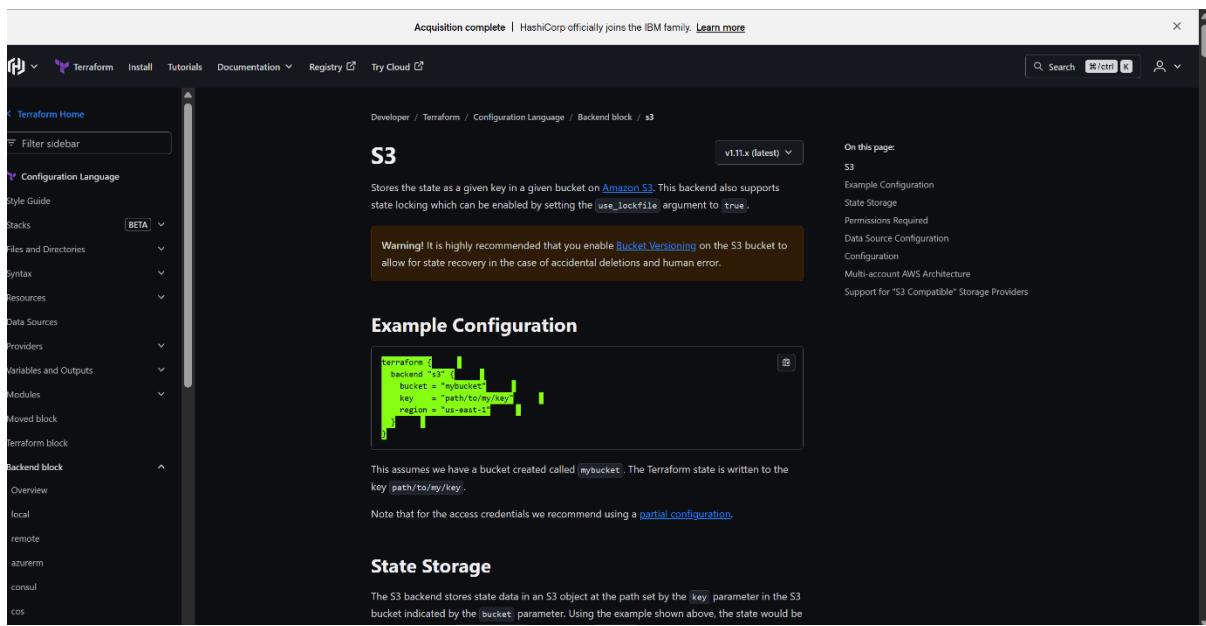
```
provider "aws" {
  region = "us-east-1"
  profile = "terraform-user"
}
```



A screenshot of a Google search results page. The search query is "terraform s3 state". The top result is a link from HashiCorp Developer to a page about the S3 Backend block.

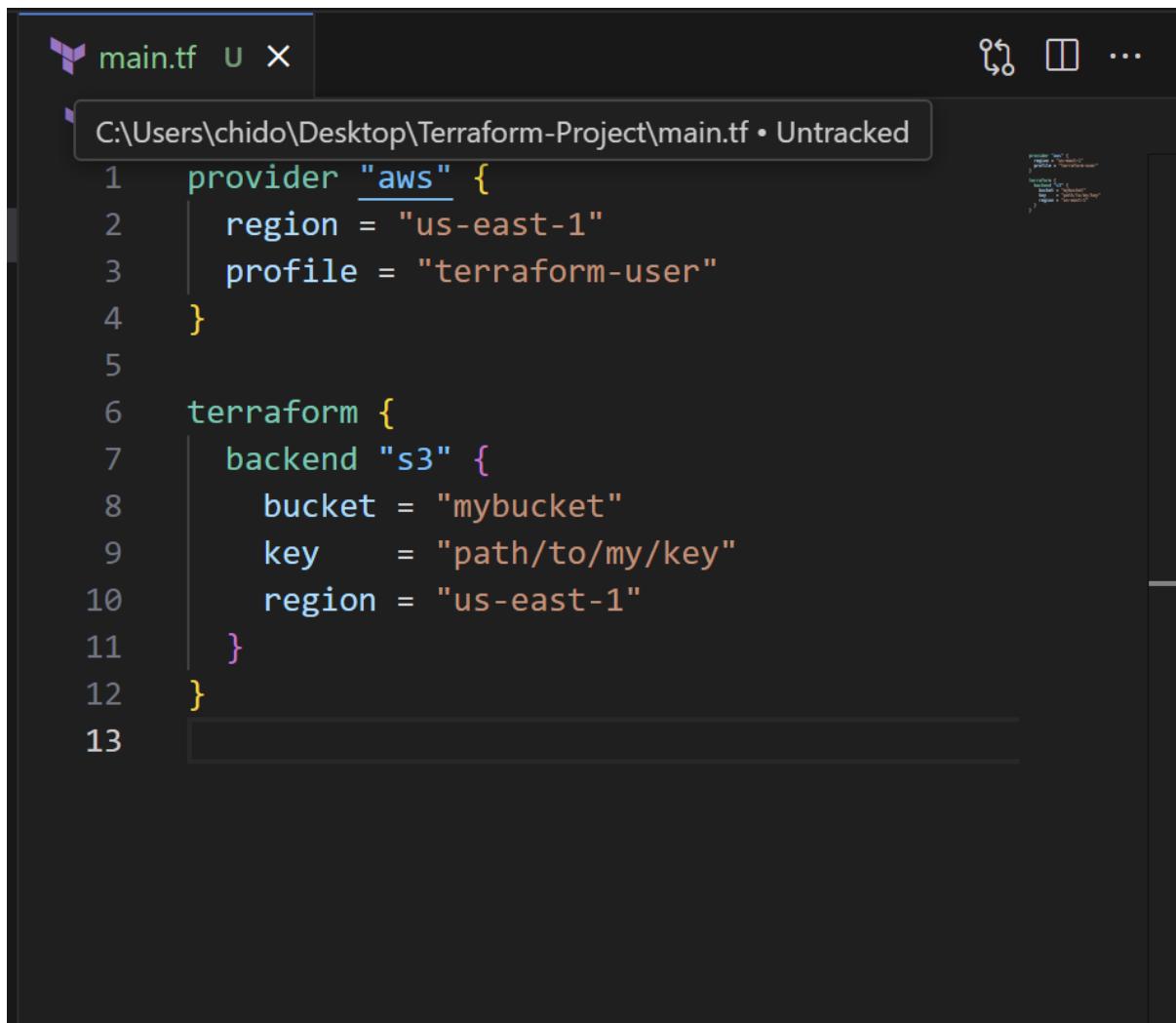
Backend Type: s3 | Terraform

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.



A screenshot of the Terraform documentation website. The URL is [https://developer.hashicorp.com/terraform/language/backends/s3](#). The page title is "S3". It provides information about storing state in an S3 bucket, including a warning about enabling Bucket Versioning. It includes an "Example Configuration" block with code snippets and a "State Storage" section.

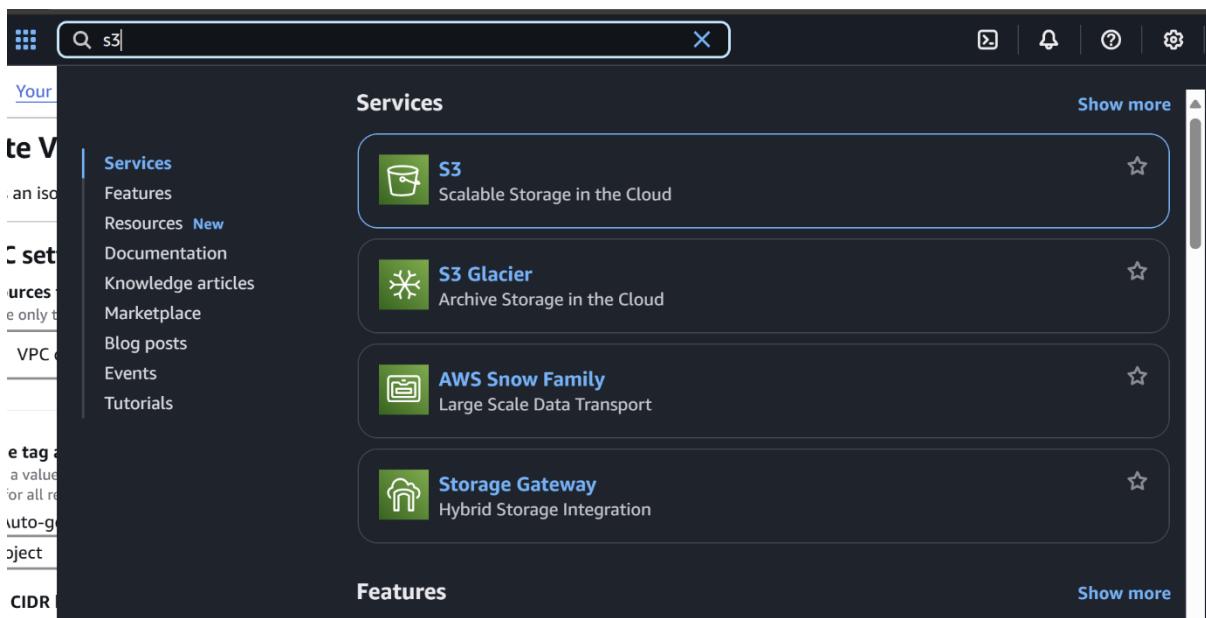
```
provider "s3" {
  bucket = "mybucket"
  key    = "path/to/my/key"
  region = "us-east-1"
}
```



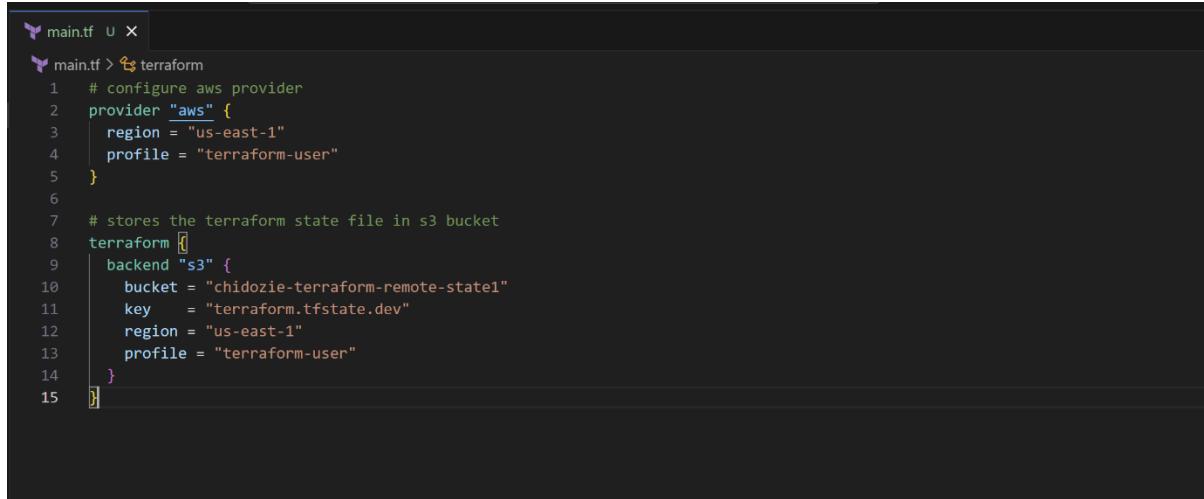
```
main.tf
provider "aws" {
  region = "us-east-1"
  profile = "terraform-user"
}

terraform {
  backend "s3" {
    bucket = "mybucket"
    key    = "path/to/my/key"
    region = "us-east-1"
  }
}
```

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



The screenshot shows the AWS Management Console search results for "s3". The search bar at the top contains "s3". Below the search bar, there is a sidebar with links to "Your VPCs", "Services", "Features", "Resources New", "Documentation", "Knowledge articles", "Marketplace", "Blog posts", "Events", and "Tutorials". The main content area is titled "Services" and lists four services: "S3 Scalable Storage in the Cloud", "S3 Glacier Archive Storage in the Cloud", "AWS Snow Family Large Scale Data Transport", and "Storage Gateway Hybrid Storage Integration". Each service item has a star icon to its right.



The screenshot shows a code editor window with a dark theme. The file is named 'main.tf'. The code defines an AWS provider and an S3 backend for storing Terraform state.

```
main.tf
main.tf > terraform
1 # configure aws provider
2 provider "aws" {
3   region = "us-east-1"
4   profile = "terraform-user"
5 }
6
7 # stores the terraform state file in s3 bucket
8 terraform [
9   backend "s3" {
10     bucket = "chidozie-terraform-remote-state1"
11     key    = "terraform.tfstate.dev"
12     region = "us-east-1"
13     profile = "terraform-user"
14   }
15 ]
```

- 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.

The screenshot shows a VS Code interface with a dark theme. On the left, a code editor displays the `main.tf` file containing Terraform configuration code. On the right, a terminal window shows the output of the `terraform init` command.

```

main.tf
1 # configure aws provider
2 provider "aws" {
3   region = "us-east-1"
4   profile = "terraform-user"
5 }
6
7 # stores the terraform state file in s3 bucket
8 terraform {
9   backend "s3" {
10    bucket = "chidozie-terraform-remote-state1"
11    key    = "terraform.tfstate.dev"
12    region = "us-east-1"
13    profile = "terraform-user"
14  }
}

```

TERMINAL

```

● PS C:\Users\chido\Desktop\Terraform-Project> terraform init
Initializing the backend...

Successfully configured the backend "s3"! Terraform will automatically
use this backend unless the backend configuration changes.

Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.92.0...
- Installed hashicorp/aws v5.92.0 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

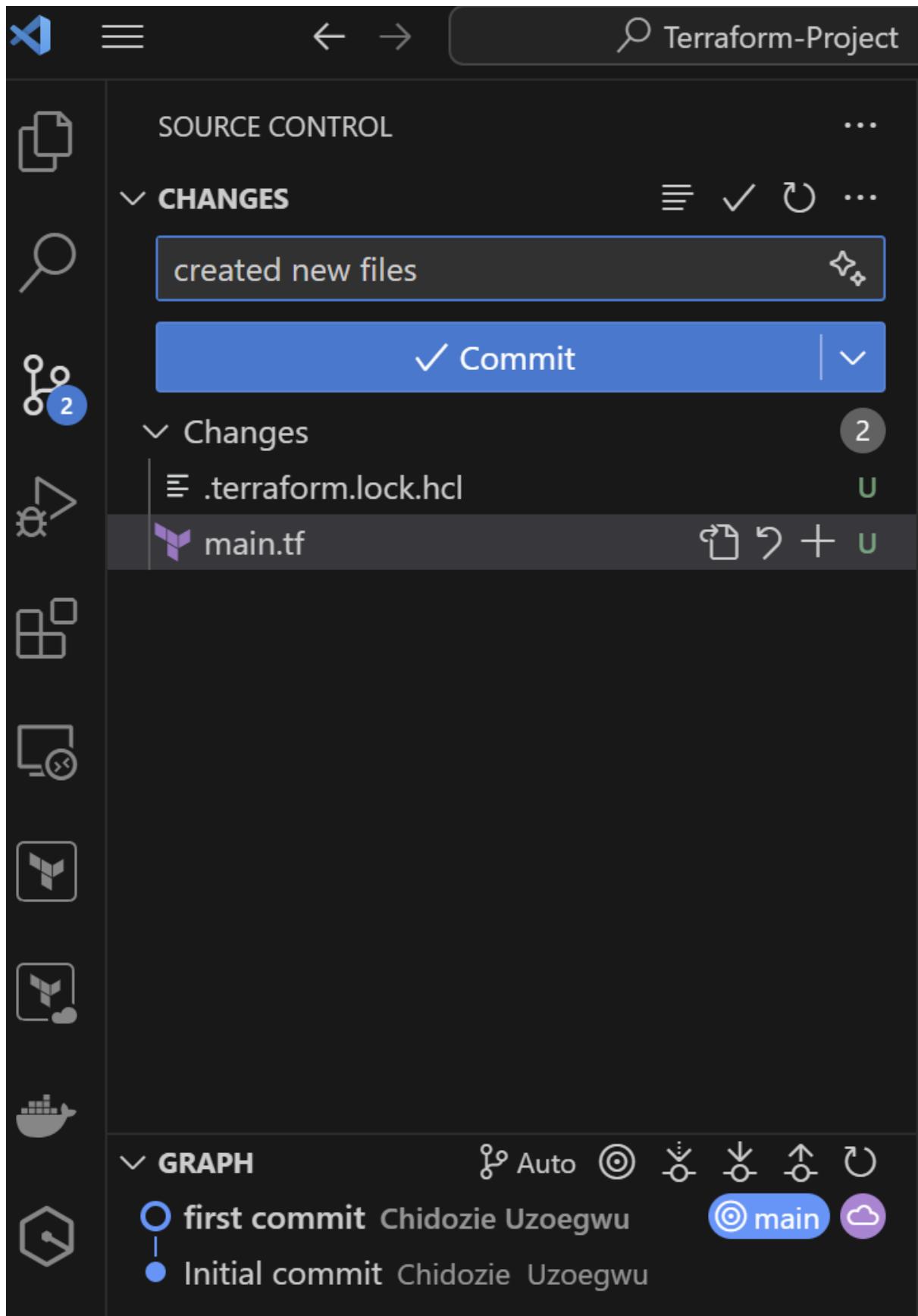
You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

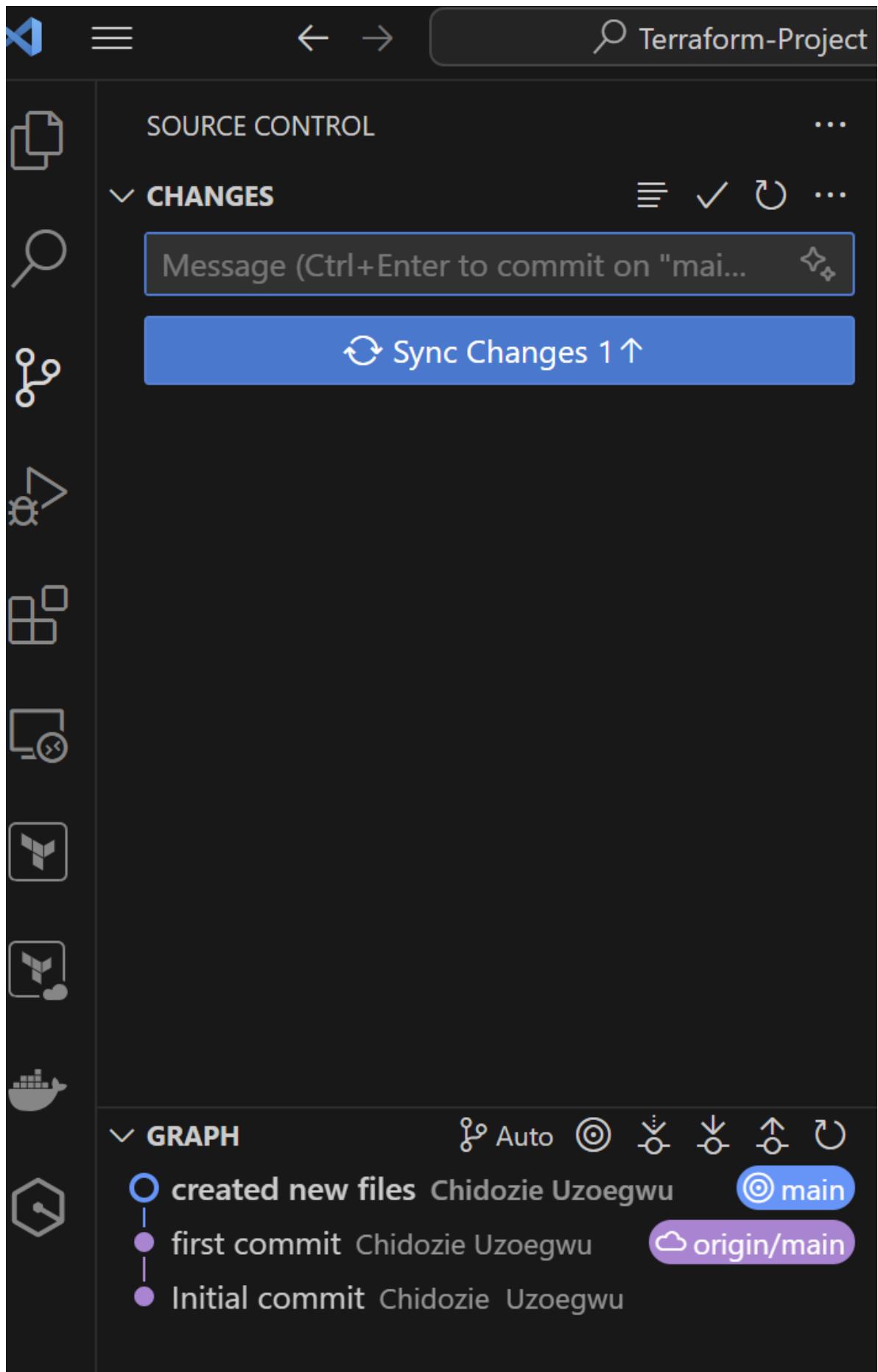
If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.

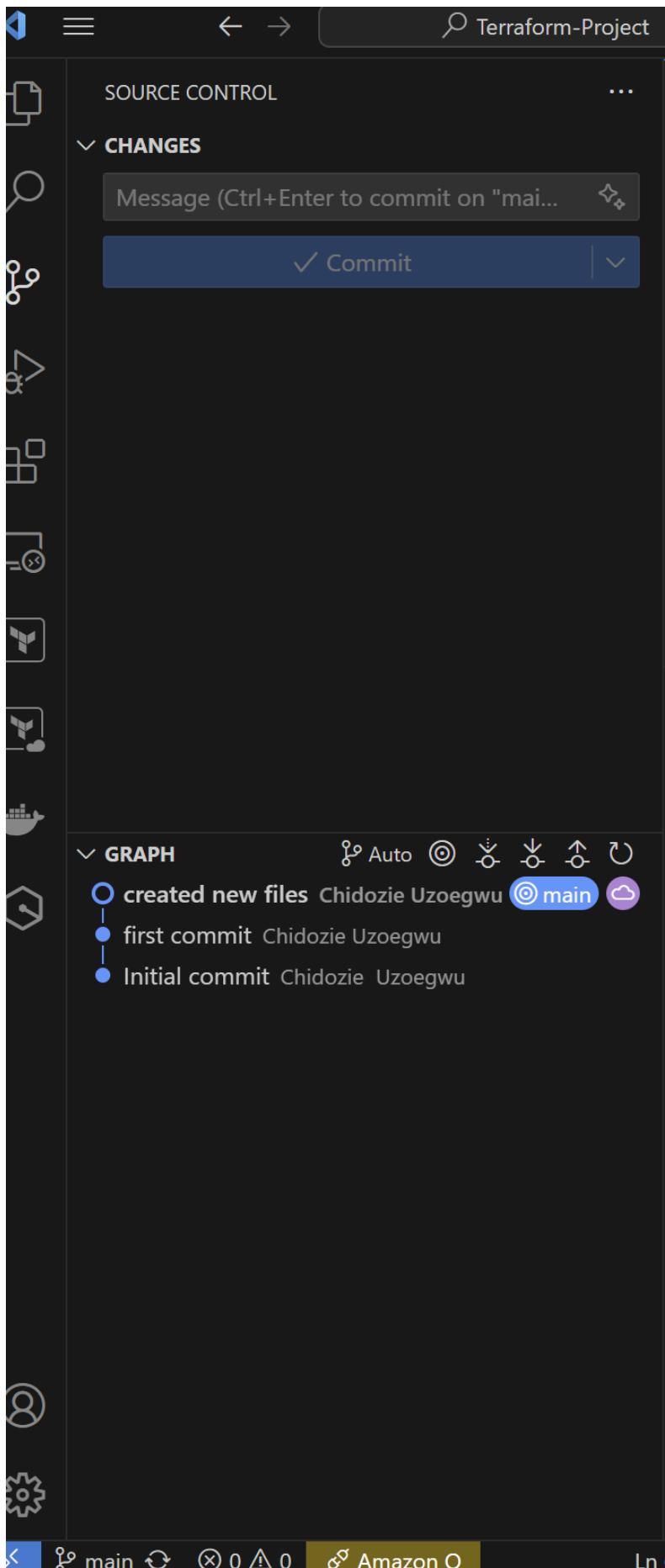
```

PS C:\Users\chido\Desktop\Terraform-Project>

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







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

The screenshot shows a GitHub repository named "Terraform-Project". The repository is private and has 1 branch and 0 tags. The commit history shows three commits by user "chido10": "created new files" (initial commit), ".gitignore" (2 hours ago), ".terraform.lock.hcl" (created new files, 2 minutes ago), "README.md" (first commit, 1 hour ago), and "main.tf" (created new files, 2 minutes ago). The README file contains the text "Terraform-Project" and "Pushing our first changes by add > commit > push". The repository page also includes sections for About, Releases, Packages, and Languages, all of which are currently empty or inactive.

The screenshot shows the content of the "main.tf" file in the "Terraform-Project" repository. The file contains the following Terraform configuration:

```
1  # configure aws provider
2  provider "aws" {
3    region = "us-east-1"
4    profile = "terraform-user"
5  }
6
7  # stores the terraform state file in s3 bucket
8  terraform {
9    backend "s3" {
10      bucket = "chidozie-terraform-remote-state"
11      key    = "terraform.tfstate.dev"
12      region = "us-east-1"
13      profile = "terraform-user"
14    }
15 }
```

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

```

1 # create vpc
2 # terraform aws create vpc
3 resource "aws_vpc" "vpc" {
4   cidr_block      = var.vpc_cidr
5   instance_tenancy = "default"
6   enable_dns_hostnames = true
7
8   tags    = {
9     Name    = "dev vpc"
10    }
11  }
12
13 # create internet gateway and attach it to vpc
14 # terraform aws create internet gateway
15 resource "aws_internet_gateway" "internet_gateway" {
16   vpc_id    = aws_vpc.vpc.id
17
18   tags    = {
19     Name    = "dev internet gateway"
20    }
21  }
22
23 # create public subnet az1
24 # terraform aws create subnet
25 resource "aws_subnet" "public_subnet_az1" {
26   vpc_id      = aws_vpc.vpc.id
27   cidr_block   = var.public_subnet_az1_cidr
28   availability_zone = "us-east-1a"
29   map_public_ip_on_launch = true
30
31   tags    = {
32     Name    = "public subnet az1"
33    }
34  }
35
36 # create public subnet az2
37 # terraform aws create subnet
38 resource "aws_subnet" "public_subnet_az2" {
39   vpc_id      = aws_vpc.vpc.id

```

variable.tf

```

variable.tf
1 # vpc variables
2 variable "vpc_cidr" {
3   default    = "10.0.0.0/16"
4   description = "VPC cidr block"
5   type       = string
6 }
7
8 variable "public_subnet_az1_cidr" {
9   default    = "10.0.0.0/24"
10  description = "public subnet az1 cidr block"
11  type       = string
12 }
13
14 variable "public_subnet_az2_cidr" {
15   default    = "10.0.1.0/24"
16  description = "public subnet az2 cidr block"
17  type       = string
18 }
19
20 variable "private_app_subnet_az1_cidr" {
21   default    = "10.0.2.0/24"
22  description = "private app subnet az1 cidr block"
23  type       = string
24 }
25
26 variable "private_app_subnet_az2_cidr" {
27   default    = "10.0.3.0/24"
28  description = "private app subnet az2 cidr block"
29  type       = string
30 }
31
32 variable "private_data_subnet_az1_cidr" {
33   default    = "10.0.4.0/24"
34  description = "private data subnet az1 cidr block"
35  type       = string
36 }
37
38 variable "private_data_subnet_az2_cidr" {
39   default    = "10.0.5.0/24"
40  description = "private data subnet az2 cidr block"
41  type       = string
42 }

```

vpc.tf

```

35
36 # create public subnet az2
37 # terraform aws create subnet
38 resource "aws_subnet" "public_subnet_az2" {
39   vpc_id      = aws_vpc.vpc.id
40   cidr_block   = var.public_subnet_az2_cidr
41   availability_zone = "us-east-1b"
42   map_public_ip_on_launch = true
43
44   tags    = {
45     Name    = "public subnet az2"
46    }
47  }
48
49 # create route table and add public route
50 # terraform aws create route table
51 resource "aws_route_table" "public_route_table" {
52   vpc_id      = aws_vpc.vpc.id
53
54   route {
55     cidr_block = "0.0.0.0/0"
56     gateway_id = aws_internet_gateway.internet_gateway.id
57   }
58
59   tags    = {
60     Name    = "public route table"
61    }
62  }
63
64 # associate public subnet az1 to "public route table"
65 # terraform aws associate subnet with route table
66 resource "aws_route_table_association" "public_subnet_az1_route_table"
67   subnet_id      = aws_subnet.public_subnet_az1.id
68   route_table_id = aws_route_table.public_route_table.id
69 }
70
71 # associate public subnet az2 to "public route table"
72 # terraform aws associate subnet with route table
73 resource "aws_route_table_association" "public_subnet_2_route_table_a"

```

variable.tf

```

variable.tf
18 }
19
20 variable "private_app_subnet_az1_cidr" {
21   default    = "10.0.2.0/24"
22  description = "private app subnet az1 cidr block"
23  type       = string
24 }
25
26 variable "private_app_subnet_az2_cidr" {
27   default    = "10.0.3.0/24"
28  description = "private app subnet az2 cidr block"
29  type       = string
30 }
31
32 variable "private_data_subnet_az1_cidr" {
33   default    = "10.0.4.0/24"
34  description = "private data subnet az1 cidr block"
35  type       = string
36 }
37
38 variable "private_data_subnet_az2_cidr" {
39   default    = "10.0.5.0/24"
40  description = "private data subnet az2 cidr block"
41  type       = string
42 }

```

```

vpc.tf > resource "aws_subnet" "private_data_subnet_az2" > tags > Name
...
71 # associate public subnet az2 to "public route table"
72 # terraform aws associate subnet with route table
73 resource "aws_route_table_association" "public_subnet_2_route_table_az2" {
74   subnet_id      = aws_subnet.public_subnet_az2.id
75   route_table_id = aws_route_table.public_route_table.id
76 }
77
78 # create private app subnet az1
79 # terraform aws create subnet
80 resource "aws_subnet" "private_app_subnet_az1" {
81   vpc_id          = aws_vpc.vpc.id
82   cidr_block      = var.private_app_subnet_az1_cidr
83   availability_zone = "us-east-1a"
84   map_public_ip_on_launch = false
85
86   tags = {
87     Name = "private app subnet az1"
88   }
89 }
90
91 # create private app subnet az2
92 # terraform aws create subnet
93 resource "aws_subnet" "private_app_subnet_az2" {
94   vpc_id          = aws_vpc.vpc.id
95   cidr_block      = var.private_app_subnet_az2_cidr
96   availability_zone = "us-east-1b"
97   map_public_ip_on_launch = false
98
99   tags = {
100    Name = "private app subnet az2"
101  }
102 }
103
104 # create private data subnet az1
105 # terraform aws create subnet
106 resource "aws_subnet" "private_data_subnet_az1" {
107   vpc_id          = aws_vpc.vpc.id
108   cidr_block      = var.private_data_subnet_az1_cidr

```

```

variable.tf > variable "public_subnet_az1_cidr" > description
...
18 }
19
20 variable "private_app_subnet_az1_cidr" {
21   default = "10.0.2.0/24"
22   description = "private app subnet az1 cidr block"
23   type = string
24 }
25
26 variable "private_app_subnet_az2_cidr" {
27   default = "10.0.3.0/24"
28   description = "private app subnet az2 cidr block"
29   type = string
30 }
31
32 variable "private_data_subnet_az1_cidr" {
33   default = "10.0.4.0/24"
34   description = "private data subnet az1 cidr block"
35   type = string
36 }
37
38 variable "private_data_subnet_az2_cidr" {
39   default = "10.0.5.0/24"
40   description = "private data subnet az2 cidr block"
41   type = string
42 }

```

```

vpc.tf > resource "aws_subnet" "private_data_subnet_az2" > tags > Name
...
102 }
103
104 # create private data subnet az1
105 # terraform aws create subnet
106 resource "aws_subnet" "private_data_subnet_az1" {
107   vpc_id          = aws_vpc.vpc.id
108   cidr_block      = var.private_data_subnet_az1_cidr
109   availability_zone = "us-east-1a"
110   map_public_ip_on_launch = false
111
112   tags = {
113     Name = "private data subnet az1"
114   }
115 }
116
117 # create private data subnet az2
118 # terraform aws create subnet
119 resource "aws_subnet" "private_data_subnet_az2" {
120   vpc_id          = aws_vpc.vpc.id
121   cidr_block      = var.private_data_subnet_az2_cidr
122   availability_zone = "us-east-1b"
123   map_public_ip_on_launch = false
124
125   tags = {
126     Name = "private data subnet az2"
127   }
128 }

```

```

variable.tf > variable "public_subnet_az1_cidr" > description
...
18 }
19
20 variable "private_app_subnet_az1_cidr" {
21   default = "10.0.2.0/24"
22   description = "private app subnet az1 cidr block"
23   type = string
24 }
25
26 variable "private_app_subnet_az2_cidr" {
27   default = "10.0.3.0/24"
28   description = "private app subnet az2 cidr block"
29   type = string
30 }
31
32 variable "private_data_subnet_az1_cidr" {
33   default = "10.0.4.0/24"
34   description = "private data subnet az1 cidr block"
35   type = string
36 }
37
38 variable "private_data_subnet_az2_cidr" {
39   default = "10.0.5.0/24"
40   description = "private data subnet az2 cidr block"
41   type = string
42 }

```

- 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 plan

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)
  + owner_id = (known after apply)
  + propagating_vgws = (known after apply)
  + route    = [
    + {
      + {
        + cidr_block          = "0.0.0.0/0"
        + gateway_id         = (known after apply)
        # (11 unchanged attributes hidden)
      },
    ],
  + tags     = {
}
```

```
% 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.private_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_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-0aad8ffe33babd07d]
aws_subnet.private_data_subnet_az2: Creation complete after 4s [id=subnet-0f84f7c3f42127c47]
aws_subnet.private_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-040fc8cb3be547dc]
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

C Cancel

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 <

Create VPC Launch EC2 Instances

Note: Your Instances will launch in the United States region.

Refresh Resources

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

Resources by Region You are using the following Amazon VPC resources

VPCs	United States 2	NAT Gateways	United States 0
▶ See all regions		▶ See all regions	

Subnets	United States 12	VPC Peering Connections	United States 0
▶ See all regions		▶ See all regions	

Route Tables	United States 3	Network ACLs	United States 2
▶ See all regions		▶ See all regions	

Internet Gateways	United States 2	Security Groups	United States 4
▶ See all regions		▶ See all regions	

Egress-only Internet Gateways	United States 0	Customer Gateways	United States 0
▶ See all regions		▶ See all regions	

Carrier gateways

DHCP option sets

Elastic IPs

Managed prefix lists

NAT gateways

Peering connections

Security

Network ACLs

Security groups

PrivateLink and Lattice



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 

Subnets (6)    

Last updated 2 minutes ago

Subnets (6)

<input type="checkbox"/>	Name	Subnet ID	State	VPC	Block Public...
<input type="checkbox"/>	private app subnet az1	subnet-0124bf873d71816b5	Available	vpc-0e2536827512bafa6 dev ...	
<input type="checkbox"/>	public subnet az2	subnet-02c07d8b8e648f098	Available	vpc-0e2536827512bafa6 dev ...	
<input type="checkbox"/>	private data subnet az2	subnet-0f84f7c3f42127c47	Available	vpc-0e2536827512bafa6 dev ...	
<input type="checkbox"/>	private data subnet az1	subnet-0909acc2b7b6418c7	Available	vpc-0e2536827512bafa6 dev ...	
<input type="checkbox"/>	private app subnet az2	subnet-08bd1016021a40cf6	Available	vpc-0e2536827512bafa6 dev ...	
<input type="checkbox"/>	public subnet az1	subnet-040fc8cb5be547dc	Available	vpc-0e2536827512bafa6 dev ...	

Select a subnet

VPC dashboard

EC2 Global View 

Internet gateways (1)    

Last updated 3 minutes ago

Internet gateways (1)

<input type="checkbox"/>	Name	Internet gateway ID	State	VPC ID	Owner
<input type="checkbox"/>	dev internet gateway	igw-0550600374ce71ab9	Attached	vpc-0e2536827512bafa6 dev vpc	37563C

Select an internet gateway above

VPC dashboard

EC2 Global View 

Route tables (2)    

Last updated 3 minutes ago

Route tables (2)

<input type="checkbox"/>	Name	Route table ID	Explicit subnet assoc...	Edge associations	Main	VPC
<input type="checkbox"/>	-	rtb-0b67eec5a90e45ee8	-	-	Yes	vpc-0e2536827512bafa6
<input type="checkbox"/>	public route table	rtb-0aad8ffe33ba8d07d	2 subnets	-	No	vpc-0e2536827512bafa6

Select a route table

The screenshot shows the AWS VPC Route Tables page. The top navigation bar includes 'VPC' > 'Route tables' > 'rtb-0aad8ffe33ba8d07d'. On the left sidebar, under 'Route tables', there is a 'Filter by VPC' dropdown set to 'vpc-0e2536827512bafab6 dev vpc Owner: 375630861224'. The main content area is titled 'rtb-0aad8ffe33ba8d07d / public route table'. It displays the following details:

Route table ID	Main	Explicit subnet associations	Edge associations
rtb-0aad8ffe33ba8d07d	No	2 subnets	-
VPC	Owner ID		
vpc-0e2536827512bafab6 dev vpc	375630861224		

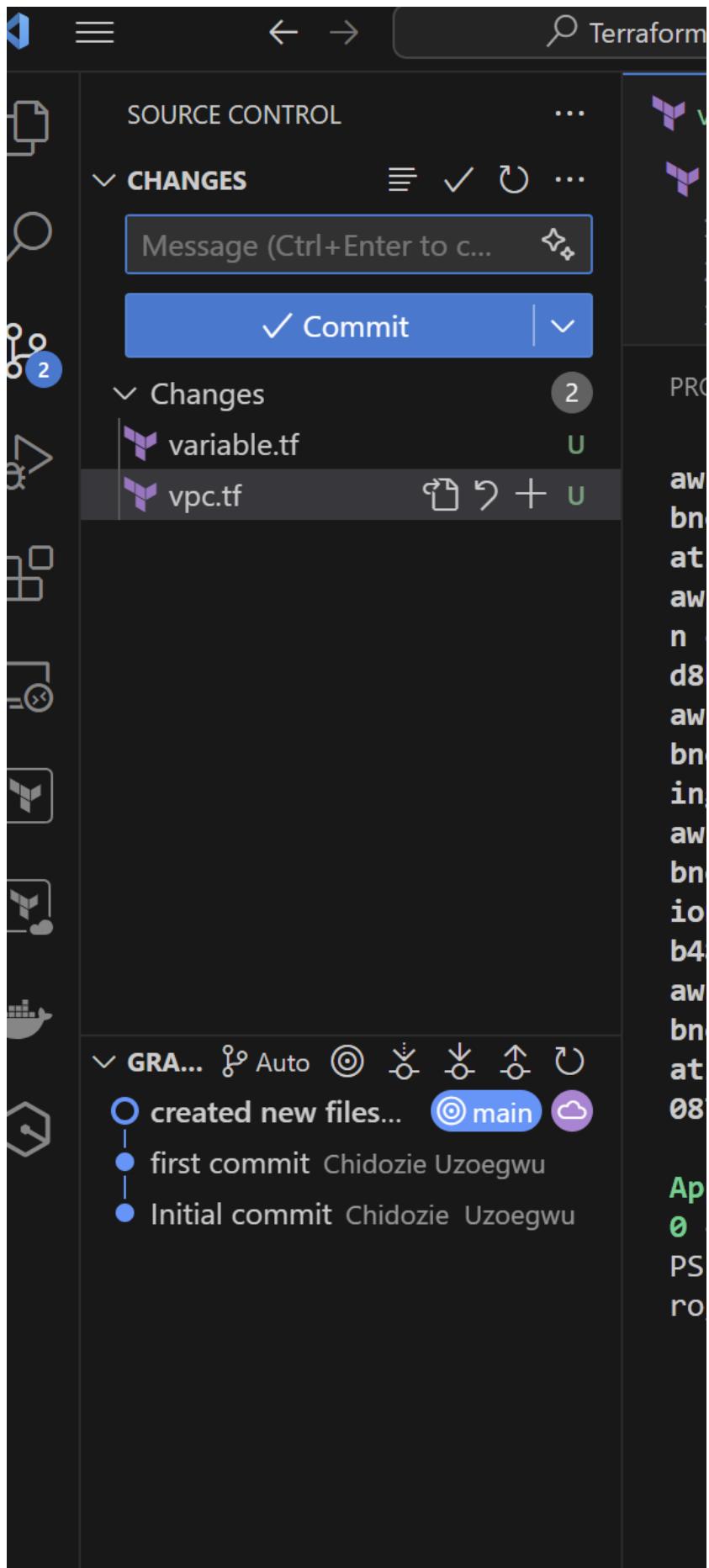
Below the details, there are tabs for 'Routes', 'Subnet associations' (which is selected), 'Edge associations', 'Route propagation', and 'Tags'. The 'Subnet associations' section shows 'Explicit subnet associations (2)' with the following data:

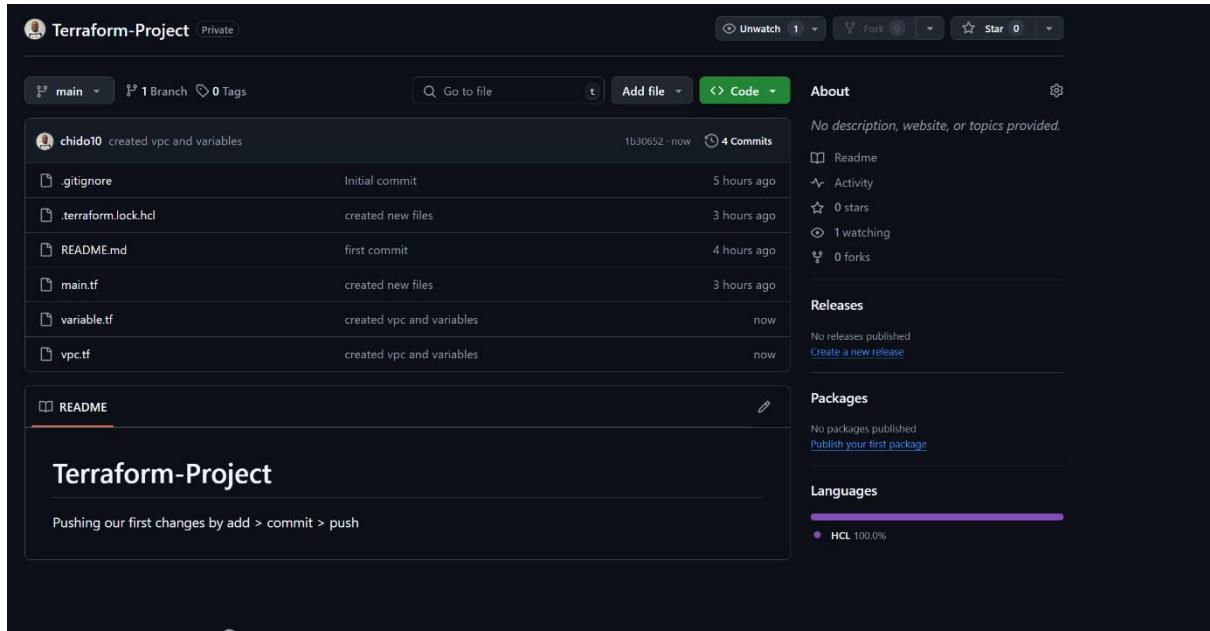
Name	Subnet ID	IPv4 CIDR	IPv6 CIDR
public subnet az2	subnet-02c07dbb8e648f098	10.0.1.0/24	-
public subnet az1	subnet-040fcfc8cb3be547dc	10.0.0.0/24	-

The 'Subnets without explicit associations (4)' section lists two subnets:

Name	Subnet ID	IPv4 CIDR	IPv6 CIDR
private app subnet az1	subnet-0124bf873d71816b5	10.0.2.0/24	-
private data subnet az2	subnet-0fbaf77726a7177a7	10.0.5.0/24	-

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





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.

```
C:\Users\chido\Desktop\Terraform-Project\nat-gateway.tf • 2 problems in this file • Untracked | route_table_az2_association"
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
5
6   tags    = {
7     Name = "nat gateway az1 eip"
8   }
9 }
10
11 # allocate elastic ip. this eip will be used for the nat-gateway in the public subnet az2
12 # terraform aws allocate elastic ip
13 resource "aws_eip" "eip_for_nat_gateway_az2" {
14   vpc      = true
15
16   tags    = {
17     Name = "nat gateway az2 eip"
18   }
19 }
20
21 # create nat gateway in public subnet az1
22 # terraform create aws nat gateway
23 resource "aws_nat_gateway" "nat_gateway_az1" {
24   allocation_id = aws_eip.eip_for_nat_gateway_az1.id
25   subnet_id     = aws_subnet.public_subnet_az1.id
26
27   tags    = {
28     Name = "nat gateway az1"
29   }
30
31   # to ensure proper ordering, it is recommended to add an explicit dependency
32   # on the internet gateway for the vpc.
33   depends_on = [aws_internet_gateway.internet_gateway]
34 }
35
36 # create nat gateway in public subnet az2
37 # terraform create aws nat gateway
38 resource "aws_nat_gateway" "nat_gateway_az2" {
39   allocation_id = aws_eip.eip_for_nat_gateway_az2.id
```

```
nat-gateway.tf > resource "aws_route_table_association" "private_data_subnet_az2_route_table_az2_association"
38 resource "aws_nat_gateway" "nat_gateway_az2" {
39   allocation_id = aws_eip.eip_for_nat_gateway_az2.id
40   subnet_id     = aws_subnet.public_subnet_az2.id
41
42   tags    = {
43     Name = "nat gateway az2"
44   }
45
46   # to ensure proper ordering, it is recommended to add an explicit dependency
47   # on the internet gateway for the vpc.
48   depends_on = [aws_internet_gateway.internet_gateway]
49 }
50
51 # create private route table az1 and add route through nat gateway az1
52 # terraform aws create route table
53 resource "aws_route_table" "private_route_table_az1" {
54   vpc_id          = aws_vpc.vpc.id
55
56   route {
57     cidr_block    = "0.0.0.0/0"
58     nat_gateway_id = aws_nat_gateway.nat_gateway_az1.id
59   }
60
61   tags    = {
62     Name = "private route table az1"
63   }
64 }
65
66 # associate private app subnet az1 with private route table az1
67 # terraform aws associate subnet with route table
68 resource "aws_route_table_association" "private_app_subnet_az1_route_table_az1_association" {
69   subnet_id      = aws_subnet.private_app_subnet_az1.id
70   route_table_id = aws_route_table.private_route_table_az1.id
71 }
72
73 # associate private data subnet az1 with private route table az1
74 # terraform aws associate subnet with route table
75 resource "aws_route_table_association" "private_data_subnet_az1_route_table_az1_association" {
76   subnet_id      = aws_subnet.private_data_subnet_az1.id
```

The screenshot shows a code editor interface with two tabs open: 'nat-gateway.tf' and 'vpc.tf'. The 'nat-gateway.tf' tab contains the following Terraform code:

```
72 # associate private data subnet az1 with private route table az1
73 # terraform aws associate subnet with route table
74 resource "aws_route_table_association" "private_data_subnet_az1_route_table_az1_association" {
75   subnet_id      = aws_subnet.private_data_subnet_az1.id
76   route_table_id = aws_route_table.private_route_table_az1.id
77 }
78 }

79 # create private route table az2 and add route through nat gateway az2
80 # terraform aws create route table
81 resource "aws_route_table" "private_route_table_az2" {
82   vpc_id          = aws_vpc.vpc.id
83
84   route {
85     cidr_block    = "0.0.0.0/0"
86     nat_gateway_id = aws_nat_gateway.nat_gateway_az2.id
87   }
88 }

89 tags  = {
90   Name = "private route table az2"
91 }
92 }

93 }

94 # associate private app subnet az2 with private route table az2
95 # terraform aws associate subnet with route table
96 resource "aws_route_table_association" "private_app_subnet_az2_route_table_az2_association" {
97   subnet_id      = aws_subnet.private_app_subnet_az2.id
98   route_table_id = aws_route_table.private_route_table_az2.id
99 }

100 }

101 # associate private data subnet az2 with private route table az2
102 # terraform aws associate subnet with route table
103 resource "aws_route_table_association" "private_data_subnet_az2_route_table_az2_association" [
104   subnet_id      = aws_subnet.private_data_subnet_az2.id
105   route_table_id = aws_route_table.private_route_table_az2.id
106 ]
107 ]
```

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-040fc8cb3be547dc]
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-02c07d8bb8e648f098]
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.

The screenshot shows the AWS VPC dashboard. At the top, there is a search bar with the text "vpc". Below it, a navigation bar has "Services" selected. A "VPC" icon with the text "Isolated Cloud Resources" is highlighted. On the left, a sidebar lists various VPC-related services: EC2 Global View, Filter by VPC (with a dropdown showing "vpc-0e2536827512bafa6 dev vpc Owner: 375630861224"), Virtual private cloud (with sub-options like Your VPCs, Subnets, Route tables, Internet gateways, Egress-only Internet gateways, Carrier gateways, DHCP option sets, Elastic IPs, Managed prefix lists), and NAT gateways (which is currently selected). The main content area is titled "NAT gateways (2)" and shows a table with two entries:

Name	NAT gateway ID	Connectivity...
nat gateway az1	nat-02c22a8fe111beed3	Public
nat gateway az2	nat-0b90246efb7c4095a	Public

Below the table, a section titled "Select a NAT gateway" is visible.

The screenshot shows the AWS VPC dashboard with the 'Route tables' section selected. A search bar at the top right contains the filter 'VPC:vpc-0e2536827512bafab6'. The main table lists four route tables:

Name	Route table ID	Explicit subnet associations	Main	VPC
-	rtb-0b67eecc5a90e45ee8	-	Yes	vpc-0e2536827512bafab6
private route table az2	rtb-090b7387a0adefaa4e	2 subnets	No	vpc-0e2536827512bafab6
private route table az1	rtb-06a2fffcce49954f88	2 subnets	No	vpc-0e2536827512bafab6
public route table	rtb-0aad0ffe33ba8d07d	2 subnets	No	vpc-0e2536827512bafab6

Below the table, there are tabs for 'Details', 'Routes', 'Subnet associations', 'Edge associations', 'Route propagation', and 'Tags'. The 'Routes' tab is active, showing two routes:

Destination	Target	Status	Propagated
0.0.0.0/0	igw-0550600...	Active	No
10.0.0.0/16	local	Active	No

The screenshot shows the AWS VPC dashboard with the 'Route tables' section selected. A search bar at the top right contains the filter 'VPC:vpc-0e2536827512bafab6'. The main table lists four route tables:

Name	Route table ID	Main	Explicit subnet associations	Edge associations
-	rtb-0b67eecc5a90e45ee8	No	2 subnets	-
private route table az2	rtb-090b7387a0adefaa4e	No	2 subnets	-
private route table az1	rtb-06a2fffcce49954f88	No	2 subnets	-
public route table	rtb-0aad0ffe33ba8d07d	No	2 subnets	-

Below the table, there are tabs for 'Details', 'Subnet associations', 'Edge associations', 'Route propagation', and 'Tags'. The 'Subnet associations' tab is active, showing explicit subnet associations:

Name	Subnet ID	IPv4 CIDR	IPv6 CIDR
private data subnet az2	subnet-0fb4f7c3f42127c47	10.0.5.0/24	-
private app subnet az2	subnet-08bd1016021a40cf6	10.0.3.0/24	-

Below this, there is a section for 'Subnets without explicit associations' which states: 'The following subnets have not been explicitly associated with any route tables and are therefore associated with the main route table:'. A table below shows these subnets:

Name	Subnet ID	IPv4 CIDR	IPv6 CIDR
------	-----------	-----------	-----------

The table is currently empty, displaying 'No subnets without explicit associations'.

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

nat-gateway.tf

1

Graph

Auto

main

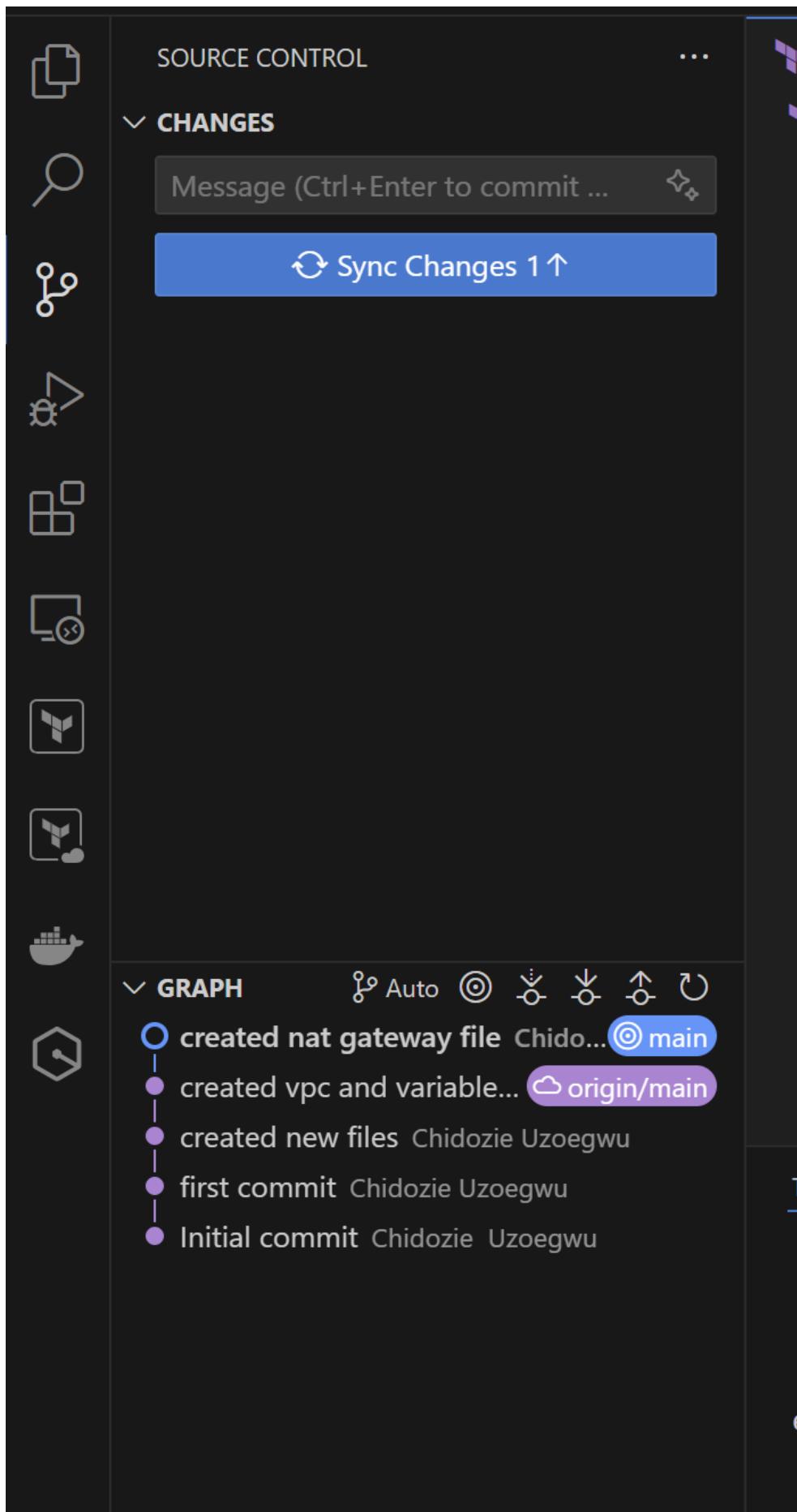
Cloud

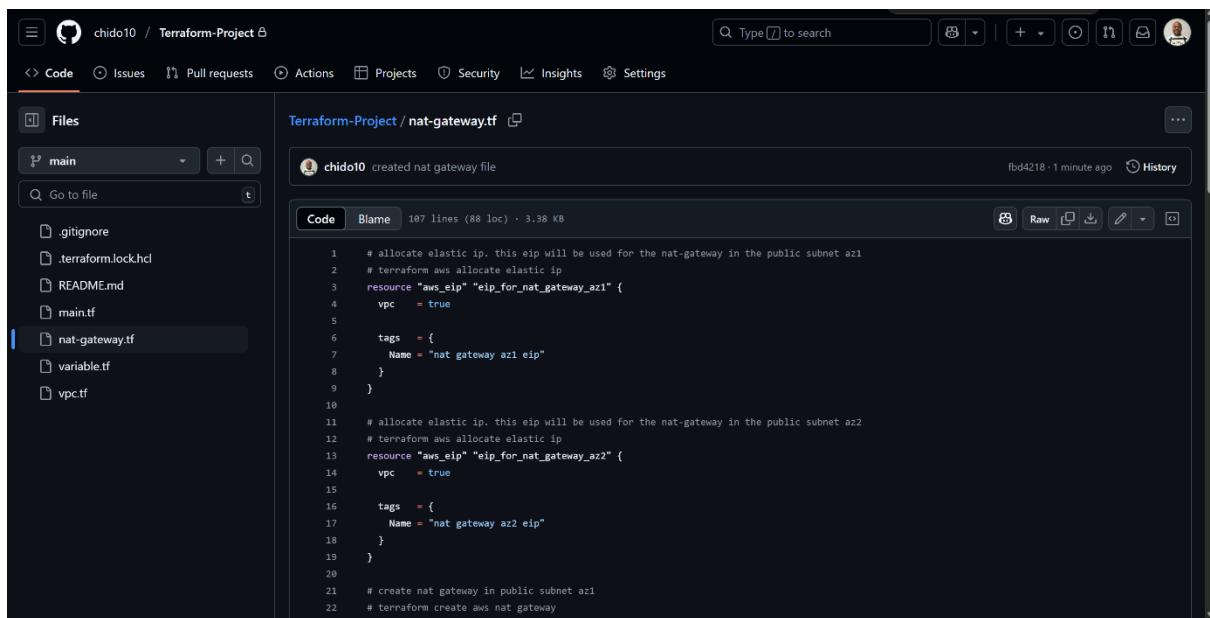
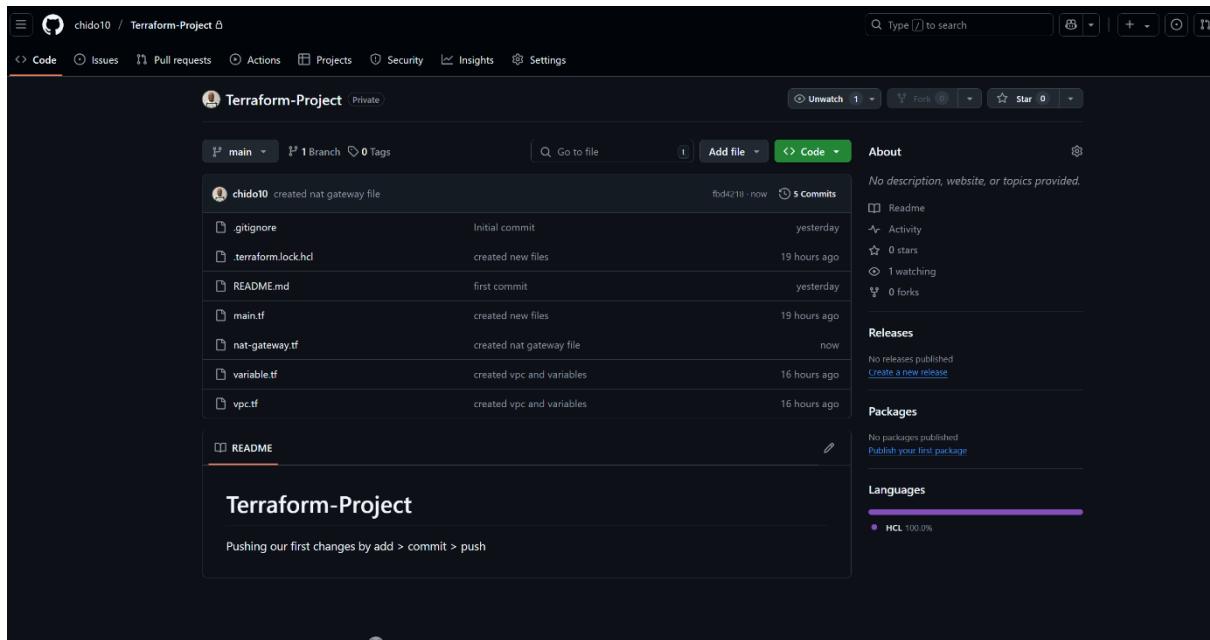
created vpc and variables...

created new files Chidozie Uzoegwu

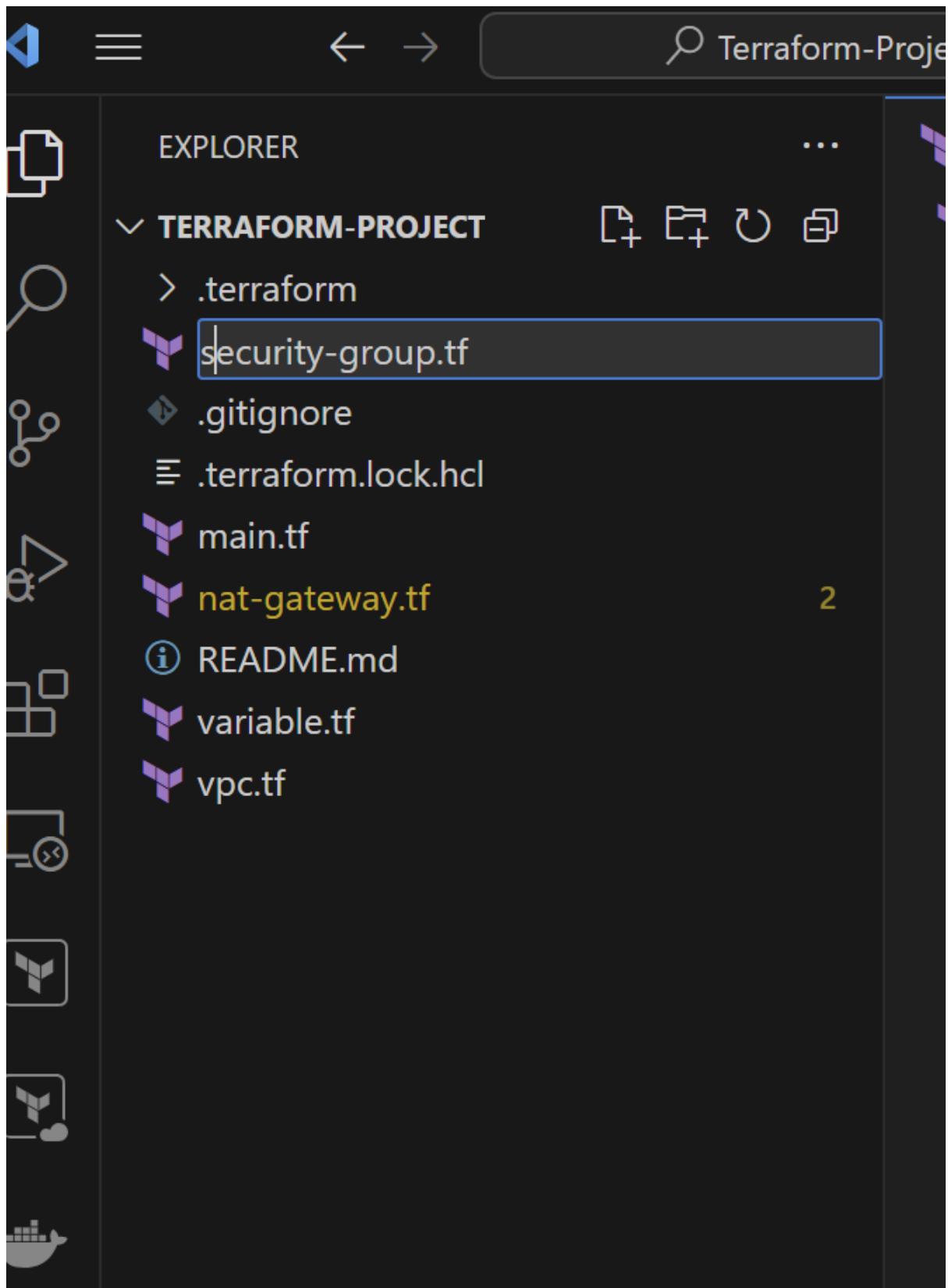
first commit Chidozie Uzoegwu

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 X
security-group.tf > resource "aws_security_group" "alb_security_group" > ingress > [ ] cidr_blocks
1 # create security group for the application load balancer
2 # terraform aws create security group
3 resource "aws_security_group" "alb_security_group" {
4   name        = "alb security group"
5   description = "enable http/https access on port 80/443"
6   vpc_id      = aws_vpc.vpc.id
7
8   ingress {
9     description = "http access"
10    from_port  = 80
11    to_port    = 80
12    protocol   = "tcp"
13    cidr_blocks = ["0.0.0.0/0"]
14  }
15
16  ingress {
17    description = "https access"
18    from_port   = 443
19    to_port     = 443
20    protocol   = "tcp"
21    cidr_blocks = ["0.0.0.0/0"]
22  }
23
24  egress {
25    from_port  = 0
26    to_port    = 0
27    protocol   = -1
28    cidr_blocks = ["0.0.0.0/0"]
29  }
30
31  tags  = {
32    Name = "alb security group"
33  }
34}
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"

```

```
security-group.tf ×
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 X
security-group.tf > resource "aws_security_group" "alb_security_group" > ingress > [ ] cidr_blocks
65   resource "aws_security_group" "webserver_security_group" {
66     vpc_id      = aws_vpc.vpc.id
67
68     ingress {
69       description      = "http access"
70       from_port        = 80
71       to_port          = 80
72       protocol         = "tcp"
73       security_groups = [aws_security_group.alb_security_group.id]
74     }
75
76     ingress {
77       description      = "https access"
78       from_port        = 443
79       to_port          = 443
80       protocol         = "tcp"
81       security_groups = [aws_security_group.alb_security_group.id]
82     }
83
84     ingress {
85       description      = "ssh access"
86       from_port        = 22
87       to_port          = 22
88       protocol         = "tcp"
89       security_groups = [aws_security_group.ssh_security_group.id]
90     }
91
92     egress {
93       from_port        = 0
94       to_port          = 0
95       protocol         = -1
96       cidr_blocks     = ["0.0.0.0/0"]
97     }
98
99   }
100
101   tags  = {
102     Name = "Webserver security group"
103   }
104 }
105 }
```

```
security-group.tf U X
security-group.tf > resource "aws_security_group" "alb_security_group" > ingress > [ ] cidr_blocks
  65  resource "aws_security_group" "webserver_security_group" {
  66    tags      = {
  67    }
  68  }
  69
  70  # create security group for the database
  71  # terraform aws create security group
  72  resource "aws_security_group" "database_security_group" {
  73    name        = "database security group"
  74    description = "enable mysql/aurora access on port 3306"
  75    vpc_id      = aws_vpc.vpc.id
  76
  77    ingress {
  78      description      = "mysql/aurora access"
  79      from_port        = 3306
  80      to_port          = 3306
  81      protocol         = "tcp"
  82      security_groups = [aws_security_group.webserver_security_group.id]
  83    }
  84
  85    egress {
  86      from_port        = 0
  87      to_port          = 0
  88      protocol         = -1
  89      cidr_blocks     = ["0.0.0.0/0"]
  90    }
  91
  92    tags      = {
  93      Name = "database security group"
  94    }
  95  }
  96
  97  
```

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 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-040cf8cb3be547dc]
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-02c07db8ee648f098]
aws_subnet.private_data_subnet_az2: Refreshing state... [id=subnet-0f84f7c3f42127c47]
aws_route_table.public_route_table: Refreshing state... [id=rtb-0aaad8ffe33ba807d]
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-06a2ffffce49954f88]
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_association.private_route_table_az2_association: Refreshing state... [id=rtbassoc-02c22a8fe111beed3]
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-01dd1209bc8bb9c1]
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 symbols:
+ create

```

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

VPC > Route tables > rtb-090b7387a0adef4e

Security Groups (5) Info

Find resources by attribute or tag

Filter by VPC: vpc-0e2536827512bafa6

Name	Security group ID	Security group name
webserver security g...	sg-01f1d1d714b18b486	webserver security group
ssh security group	sg-0d31c157b6a1afb1b	ssh security group
-	sg-013a8722ad8efb420	default
database security gr...	sg-0db3f13cd9cbe0258	database security group
alb security group	sg-0300d75a203b30103	alb security group

VPC dashboard <

EC2 Global View Filter by VPC: vpc-0e2536827512bafa6

sg-0db3f13cd9cbe0258 - database security group

Details

Security group name database security group	Security group ID sg-0db3f13cd9cbe0258	Description enable mysql/aurora access on port 3306	VPC ID vpc-0e2536827512bafa6
Owner 375630861224	Inbound rules count 1 Permission entry	Outbound rules count 1 Permission entry	

Inbound rules | Outbound rules | Sharing - new | VPC associations - new | Tags

Inbound rules (1)

Name	Security group rule ID	IP version	Type	Protocol	Port range
-	sgr-080824051c9154fce	-	MySQL/Aurora	TCP	3306

The screenshot shows the AWS VPC dashboard with the following details:

Details

Security group name webserver security group	Security group ID sg-01f1d1d714b18b486	Description enable http/https access on port 80/443 via alb sg and access on port 22 via ssh sg	VPC ID vpc-0e2536827512baf6
Owner 375630861224	Inbound rules count 3 Permission entries	Outbound rules count 1 Permission entry	

Inbound rules (3)

Name	Security group rule ID	IP version	Type	Protocol	Port
-	sgr-00d3748d7baf077fe	-	SSH	TCP	22
-	sgr-03376cd30a3d868e9	-	HTTPS	TCP	443
-	sgr-01ccd5102379f4cf2	-	HTTP	TCP	80

Commit to GitHub

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

SOURCE CONTROL

CHAN... ✓ ⏪ ...

created the security group file

✓ Commit

Changes 2

securi... ↗ + M

variable.tf

G... Auto ⏪ ...

created ... @ main

- created vpc and vari...
- created new files Ch...
- first commit Chidozi...
- Initial commit Chido...

security-group.tf

```
resource "aws_security_group" "alb_secu" {
  egress {
    protocol
    cidr_blocks
  }
  tags = {
    Name = "alb secu"
  }
}
# create security group
```

PROBLEMS 2 OUTPUT TERMINAL

Only 'yes' will be accepted

Enter a value: yes

aws_security_group.alb_secu

yng... [id=sg-0cf1829ac791]

aws_security_group.alb_secu

cction 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:

SOURCE CONTROL

CHANGES

Message (Ctrl...)

Sync Changes 1 ↑

PROBLEMS 2 OUTPUT TERMINAL

Only 'yes' will be accepted

Enter a value: yes

```
aws_security_group.alb_secu
y... [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

The screenshot shows the Terraform Project interface in VS Code. On the left, the Source Control sidebar displays a commit history for a 'G...' repository. The most recent commit, 'created the ...', is highlighted with a blue oval and has a status of 'main'. Below it are five other commits: 'creat...', 'created vpc and vari...', 'created new files Ch...', 'first commit Chido...', and 'Initial commit Chido...'. The main editor area shows a Terraform configuration file for a security group, specifically defining an egress rule. The terminal panel at the bottom shows the output of a 'terraform apply' command, detailing the creation of various AWS resources like security groups and databases. A message in the terminal asks for confirmation of a value, which is being typed as 'yes'. The status bar at the bottom indicates the path 'C:\Users\chido\Desktop\T'.

Terraform-Project · Private

Code Issues Pull requests Actions Projects Security Insights Settings

main · 1 Branch · 0 Tags

Go to file Add file Code

chido10 created the security group file · f5e0fdb · now · 6 Commits

.gitignore Initial commit · yesterday

.terraform.lock.hcl created new files · yesterday

README.md first commit · yesterday

main.tf created new files · yesterday

nat-gateway.tf created nat gateway file · 2 hours ago

security-group.tf created the security group file · now

variable.tf created the security group file · now

vpc.tf created vpc and variables · 18 hours ago

README

About

No description, website, or topics provided.

Readme

Activity

0 stars

1 watching

0 forks

Releases

No releases published

Create a new release

Packages

No packages published

Publish your first package

Languages

HTML 100.0%

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.

rds

Services

Aurora and RDS

Managed Relational Database Service

Show more

The screenshot shows the Amazon RDS console with the navigation path: RDS > Snapshots > shopwise. The left sidebar includes links for Dashboard, Databases, Query Editor, Performance insights, Snapshots (selected), Exports in Amazon S3, Automated backups, Reserved instances, Proxies, Subnet groups, Parameter groups, Option groups, Custom engine versions, Zero-ETL integrations, Events, and Event subscriptions. The main content area displays the details of the 'shopwise' snapshot, including its ARN, Instance/Cluster Name, DB snapshot name, Snapshot type, DB engine, DB engine version, License model, Architecture settings, Option group, Zone, KMS key ID, Source region, Snapshot Creation Time, Original Snapshot Creation Time, Instance/Cluster Creation, VPC, Status, Storage type, DB storage, IOPS, Port, Time zone, and Dedicated Log Volume. The ARN is highlighted in blue.

Details	
ARN	arn:aws:rds:us-east-1:375630861224:snapshot:shopwise
Instance/Cluster Name	dev-rds-db
Master username	doodoo
DB snapshot name	shopwise
Snapshot type	manual
DB engine	mysql
DB engine version	8.0.36
License model	general-public-license
Architecture settings	Non-multitenant architecture
Option group	default:mysql-8-0
Zone	us-east-1f
KMS key ID	arn:aws:kms:us-east-1:375630861224:key/e7ee746a-a5f1-4dd6-b4bb-91e1f3d1a738
Source region	N/A
Snapshot Creation Time	May 05, 2024, 05:25 (UTC+01:00)
Original Snapshot Creation Time	May 05, 2024, 05:25 (UTC+01:00)
Instance/Cluster Creation	April 29, 2024, 18:15 (UTC+01:00)
VPC	vpc-0090510a2ebdebe9d
Status	Available
Storage type	General Purpose SSD (gp3)
DB storage	200 GiB
IOPS	3000
Storage throughput	125
Port	3306
Time zone	
Dedicated Log Volume	Turned off

(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.

The screenshot shows the Amazon RDS console with the navigation path: RDS > Snapshots > shopwise. The left sidebar includes links for Dashboard, Databases, Query Editor, Performance insights, Snapshots (selected), Exports in Amazon S3, Automated backups, Reserved instances, Proxies, Subnet groups, Parameter groups, Option groups, Custom engine versions, Zero-ETL integrations, Events, and Event subscriptions. The main content area displays the details of the 'shopwise' snapshot, including its ARN, Instance/Cluster Name, DB snapshot name, Snapshot type, DB engine, DB engine version, License model, Architecture settings, Option group, Zone, KMS key ID, Source region, Snapshot Creation Time, Original Snapshot Creation Time, Instance/Cluster Creation, VPC, Status, Storage type, DB storage, IOPS, Port, Time zone, and Dedicated Log Volume. The ARN is highlighted in blue.

Details	
ARN	arn:aws:rds:us-east-1:375630861224:snapshot:shopwise
Instance/Cluster Name	dev-rds-db
Master username	doodoo
DB snapshot name	shopwise
Snapshot type	manual
DB engine	mysql
DB engine version	8.0.36
License model	general-public-license
Architecture settings	Non-multitenant architecture
Option group	default:mysql-8-0
Zone	us-east-1f
KMS key ID	arn:aws:kms:us-east-1:375630861224:key/e7ee746a-a5f1-4dd6-b4bb-91e1f3d1a738
Source region	N/A
Snapshot Creation Time	May 05, 2024, 05:25 (UTC+01:00)
Original Snapshot Creation Time	May 05, 2024, 05:25 (UTC+01:00)
Instance/Cluster Creation	April 29, 2024, 18:15 (UTC+01:00)
VPC	vpc-0090510a2ebdebe9d
Status	Available
Storage type	General Purpose SSD (gp3)
DB storage	200 GiB
IOPS	3000
Storage throughput	125
Port	3306
Time zone	
Dedicated Log Volume	Turned off

The screenshot shows the Amazon RDS Snapshots page. On the left, there's a sidebar with navigation links like Dashboard, Databases, Query Editor, Performance insights, and Snapshots. The main area is titled 'shopwise' and contains a table with details about a database snapshot. The table has two columns: 'Details' and 'Options'. The 'Details' column includes fields such as ARN, Instance/Cluster Name, Master username, DB snapshot name, Snapshot type, DB engine, DB engine version, License model, and Architecture settings. The 'Options' column includes fields such as Option group, Zone, KMS key ID, Source region, Snapshot Creation Time, Original Snapshot Creation Time, Instance/Cluster Creation, VPC, Status, Storage type, DB storage, IOPS, Storage throughput, Port, and Time zone. At the top right of the main area, there's a blue 'Actions' button.

Details	
ARN	arn:aws:rds:us-east-1:375630861224:snapshot:shopwise
Instance/Cluster Name	dev-rds-db
Master username	doodoo
DB snapshot name	shopwise
Snapshot type	manual
DB engine	mysql
DB engine version	8.0.36
License model	general-public-license
Architecture settings	Non-multitenant architecture

Options	
Option group	default:mysql-8-0
Zone	us-east-1f
KMS key ID	arn:aws:kms:us-east-1:375630861224:key/e7ee746a-a5f1-4dd6-b4bb-91e1f3d1a738
Source region	N/A
Snapshot Creation Time	May 05, 2024, 05:25 (UTC+01:00)
Original Snapshot Creation Time	May 05, 2024, 05:25 (UTC+01:00)
Instance/Cluster Creation	April 29, 2024, 18:15 (UTC+01:00)
VPC	vpc-0090510a2ebdebe9d
Status	Available
Storage type	General Purpose SSD (gp3)
DB storage	200 GiB
IOPS	3000
Storage throughput	125
Port	3306
Time zone	
Dedicated Log Volume	Turned off

(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

```

File Edit Selection View Go Run ...
Terraform-Project
EXPLORER
> .terraform
> .gitignore
E .terraform.lock.hcl
main.tf
nat-gateway.tf 2
rds.tf U
README.md
security-group.tf
variable.tf M
vpc.tf

variable.tf > ...
variable.tf < ...
variable.tf > ...
variable.tf < ...

50 # rds variables
51 variable "database_snapshot_identifier" {
52   default = "arn:aws:rds:us-east-1:375630861224:snapshot:dev-rds-db"
53   description = "the database snapshot arn"
54   type = string
55 }
56
57 variable "database_instance_class" {
58   default = "db.t2.micro"
59   description = "the database instance type"
60   type = string
61 }
62
63 variable "database_instance_identifier" {
64   default = "dev-rds-db"
65   description = "the database instance identifier"
66   type = string
67 }
68
69 variable "multi_az_deployment" {
70   default = false
71   description = "create a standby db instance"
72   type = bool
73 }
74

```

(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`).

```

File Edit Selection View Go Run ...
Terraform-Project
EXPLORER
> .terraform
> .gitignore
E .terraform.lock.hcl
main.tf
nat-gateway.tf 2
rds.tf U
README.md
security-group.tf
variable.tf M
vpc.tf

rds.tf > & resource "aws_db_instance" "database_instance"
rds.tf < ...
resource "aws_db_instance" "database_instance" {
  # create database subnet group
  # terraform aws db subnet group
  resource "aws_db_subnet_group" "database_subnet_group" {
    name      = "database_subnets"
    subnet_ids = [aws_subnet.private_data_subnet_az1.id, aws_subnet.private_data_subnet_az2.id]
    description = "subnets for database instance"
  }
  tags = {
    Name = "database subnets"
  }
}

```

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

```
rds.tf
resource "aws_db_subnet_group" "database_subnet_group" {
  name            = "database_subnets"
  subnet_ids     = [aws_subnet.private_data_subnet_az1.id, aws_subnet.private_data_subnet_az2.id]
  description    = "subnets for database instance"
  tags           = [
    { Name = "database_subnets" }
  ]
}

# get the latest db snapshot
# terraform aws db snapshot
data "aws_db_snapshot" "latest_db_snapshot" {
  db_snapshot_identifier = var.database_snapshot_identifier
  most_recent            = true
  snapshot_type          = "manual"
}

# create database instance restored from db snapshots
# terraform aws db instance
resource "aws_db_instance" "database_instance" {
  instance_class      = var.database_instance_class
  skip_final_snapshot = true
  availability_zone   = "us-east-1b"
  identifier          = var.database_instance_identifier
  snapshot_identifier = data.aws_db_snapshot.latest_db_snapshot.id
  db_subnet_group_name = aws_db_subnet_group.database_subnet_group.name
  multi_az            = var.multi_az_deployment
  vpc_security_group_ids = [aws_security_group.database_security_group.id]
}
```

```
variable.tf
variable "database_snapshot_identifier" {
  default = "arn:aws:rds:us-east-1:375630861224:snapshot:dev-rds-db"
  description = "the database snapshot arn"
  type = string
}

variable "database_instance_class" {
  default = "db.t2.micro"
  description = "the database instance type"
  type = string
}

variable "database_instance_identifier" {
  default = "dev-rds-db"
  description = "the database instance identifier"
  type = string
}

variable "multi_az_deployment" {
  default = false
  description = "create a standby db instance"
  type = bool
}
```

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.

```
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_subnet_group.database_subnet_group: Creating...
aws_db_subnet_group.database_subnet_group: Creation complete after 2s [id=database_subnets]
aws_db_instance.database_instance: Creating...

Error: creating RDS DB Instance (restore from snapshot) (dev-rds-db): operation error RDS: RestoreDBInstanceFromDBSnapshot, https response error Status: StatusCode: 400, RequestID: 0534c009-d106-4313-bee1-d8be163abc27, api error InvalidParameterCombination: DB Instance class db.t2.micro does not support encryption at rest

with aws_db_instance.database_instance,
on rds.tf line 23, in resource "aws_db_instance" "database_instance":
23: resource "aws_db_instance" "database_instance" {
```

```

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

```

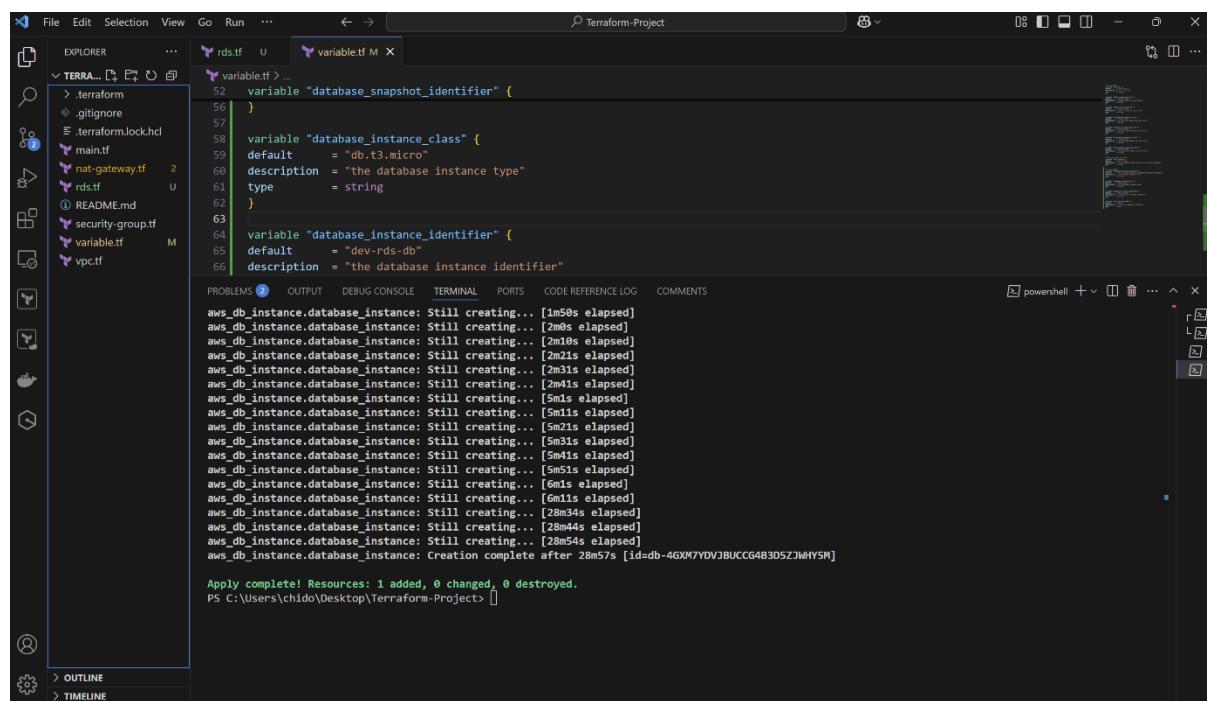
Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

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

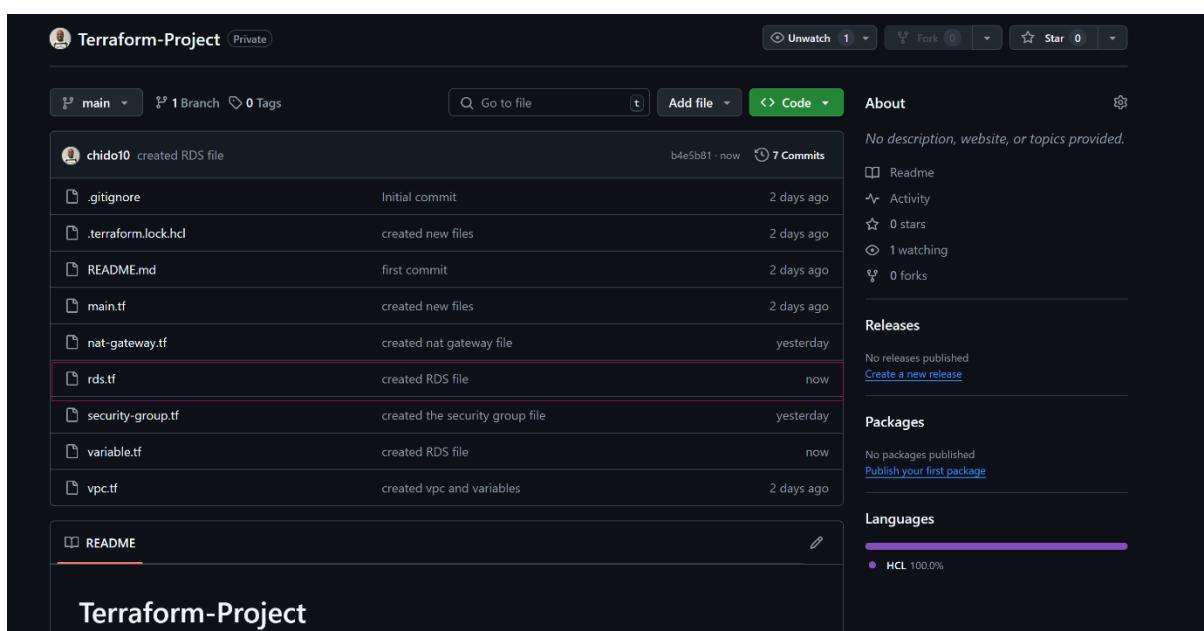
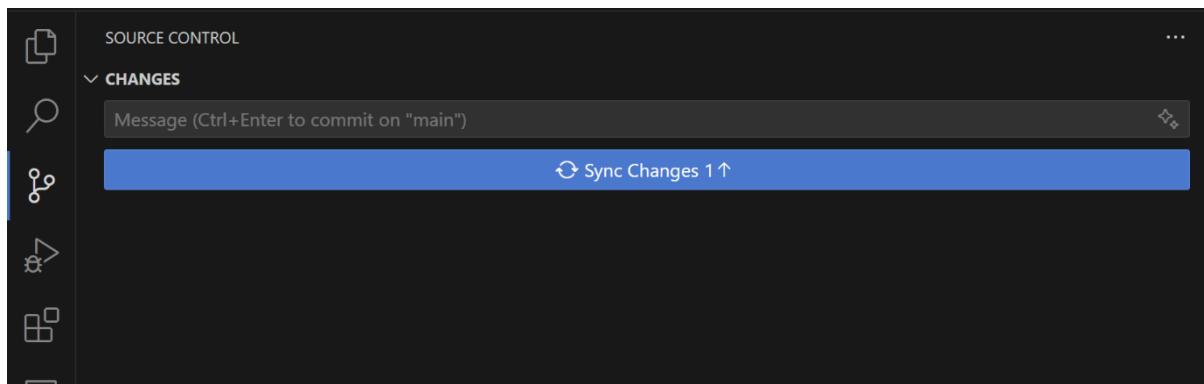
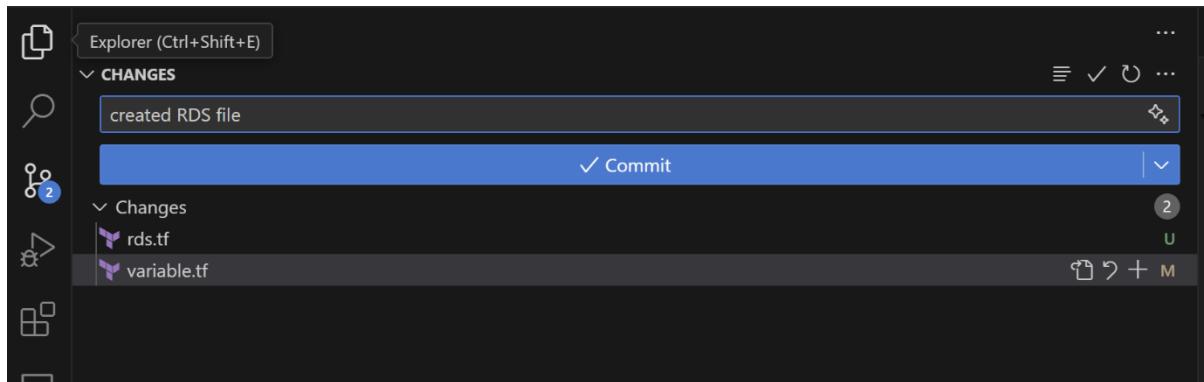
Once the apply is finished without errors, go back to the AWS RDS console:

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.

The screenshot shows a code editor with two tabs: 'alb.tf' and 'variable.tf'. The 'alb.tf' tab is active and displays the following Terraform code:

```
1 # create application load balancer
2 # terraform aws create application load balancer
3 resource "aws_lb" "application_load_balancer" {
4   name          = "dev-alb"
5   internal      = false
6   load_balancer_type = "application"
7   security_groups = [aws_security_group.alb_security_group.id]
8
9   subnet_mapping {
10     subnet_id = aws_subnet.public_subnet_az1.id
11   }
12
13   subnet_mapping {
14     subnet_id = aws_subnet.public_subnet_az2.id
15   }
16
17   enable_deletion_protection = false
18
19   tags = {
20     Name = "dev-alb"
21   }
22 }
23
24 # create target group
25 # terraform aws create target group
26 resource "aws_lb_target_group" "alb_target_group" {
27   name          = "dev-tg"
28   target_type   = "instance"
29   port          = 80
30   protocol      = "HTTP"
31   vpc_id        = aws_vpc.vpc.id
32
33   health_check {
34     interval = 30
35     path     = "/health"
36     timeout = 10
37     unhealthy_threshold = 3
38     healthy_threshold = 2
39   }
40 }
```

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.).

```
alb.tf > resource "aws_lb_listener" "alb_https_listener"
22 }
23
24 # create target group
25 # terraform aws create target group
26 resource "aws_lb_target_group" "alb_target_group" {
27   name          = "dev-tg"
28   target_type   = "instance"
29   port          = 80
30   protocol      = "HTTP"
31   vpc_id        = aws_vpc.vpc.id
32
33   health_check {
34     healthy_threshold  = 5
35     interval           = 30
36     matcher             = "200,301,302"
37     path                = "/"
38     port                = "traffic-port"
39     protocol            = "HTTP"
40     timeout             = 5
41     unhealthy_threshold = 2
42   }
43 }
44
45 # create a listener on port 80 with redirect action
46 # terraform aws create listener
47 resource "aws_lb_listener" "alb_http_listener" {
48   load_balancer_arn = aws_lb.application_load_balancer.arn
49   port              = 80
50   protocol          = "HTTP"
51
52   default_action {
53     type = "redirect"
```

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.

The screenshot shows a code editor interface with two tabs: 'alb.tf' and 'variable.tf'. The 'alb.tf' tab contains the following Terraform code:

```
46 # terraform aws create listener
47 resource "aws_lb_listener" "alb_https_listener" {
48   load_balancer_arn = aws_lb.application_load_balancer.arn
49   port              = 80
50   protocol          = "HTTP"
51
52   default_action {
53     type = "redirect"
54
55     redirect {
56       host    = "#{host}"
57       path    = "/#{path}"
58       port    = 443
59       protocol = "HTTPS"
60       status_code = "HTTP_301"
61     }
62   }
63 }
64
65 # create a listener on port 443 with forward action
66 # terraform aws create listener
67 resource "aws_lb_listener" "alb_https_listener" {
68   load_balancer_arn = aws_lb.application_load_balancer.arn
69   port              = 443
70   protocol          = "HTTPS"
71   ssl_policy        = "ELBSecurityPolicy-2016-08"
72   certificate_arn   = var.ssl_certificate_arn
73
74   default_action {
75     type      = "forward"
76     target_group_arn = aws_lb_target_group.alb_target_group.arn
77   }
}
```

The screenshot shows a code editor interface with two tabs: 'alb.tf' and 'variable.tf'. The 'variable.tf' tab contains the following Terraform code:

```
58 variable "database_instance_class" {
59   default    = "db.t3.micro"
60   description = "the database instance type"
61   type       = string
62 }
63
64 variable "database_instance_identifier" {
65   default    = "dev-rds-db"
66   description = "the database instance identifier"
67   type       = string
68 }
69
70 variable "multi_az_deployment" {
71   default    = false
72   description = "create a standby db instance"
73   type       = bool
74 }
75
76 # application load balancer variables
77 variable "ssl_certificate_arn" {
78   default    = "arn:aws:acm:us-east-1:375630861224:certificate/3f768c7c-c470-4cd1-8add-56714f8a5c49"
79   description = "ssl certificate arn"
80   type       = string
81 }
```

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-02c07d8bb8e648f098]
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-040fc8cb3be547dc]
aws_security_group.ssh_security_group: Refreshing state... [id=sg-0d31c157b6a1fb1b]
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-06a2ffffce49954f88]
aws_route_table_association.private_data_subnet_az2_route_table_az2_association: Refreshing state... [id=rtbassoc-01d7257cfed2783]
aws_route_table_association.private_app_subnet_az2_route_table_az2_association: Refreshing state... [id=rtbassoc-01dd1209bc8bbb90]
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-0ec84a573ffedf08]
```

Name	DNS name	State	VPC ID	Availability Zones	Type
dev-alb	dev-alb-1745379396.us-east-1.elb.amazonaws.com	Active	vpc-0e2536827512bafa6	2 Availability Zones	application

Protocol:Port	Default action	Rules	ARN	Security policy	Default
HTTP:80	Redirect to HTTPS://#{host}:443/#{path}#?	1 rule		Not applicable	Not a
HTTPS:443	Forward to target group • dev-tg (100%) • Target group stickiness: Off	1 rule		ELBSecurityPolicy-2016-08	chido

EC2 > Target groups

Target groups (1) Info

Name	ARN	Port	Protocol	Target type	Load balancer
dev-tg	arn:aws:elasticloadbalancing:us-east-1:123456789012:targetgroup/dev-tg/5678901234567890	80	HTTP	Instance	dev-alb

0 target groups selected

Select a target group above.

Settings

EC2 > Load balancers

Load balancers (1/1)

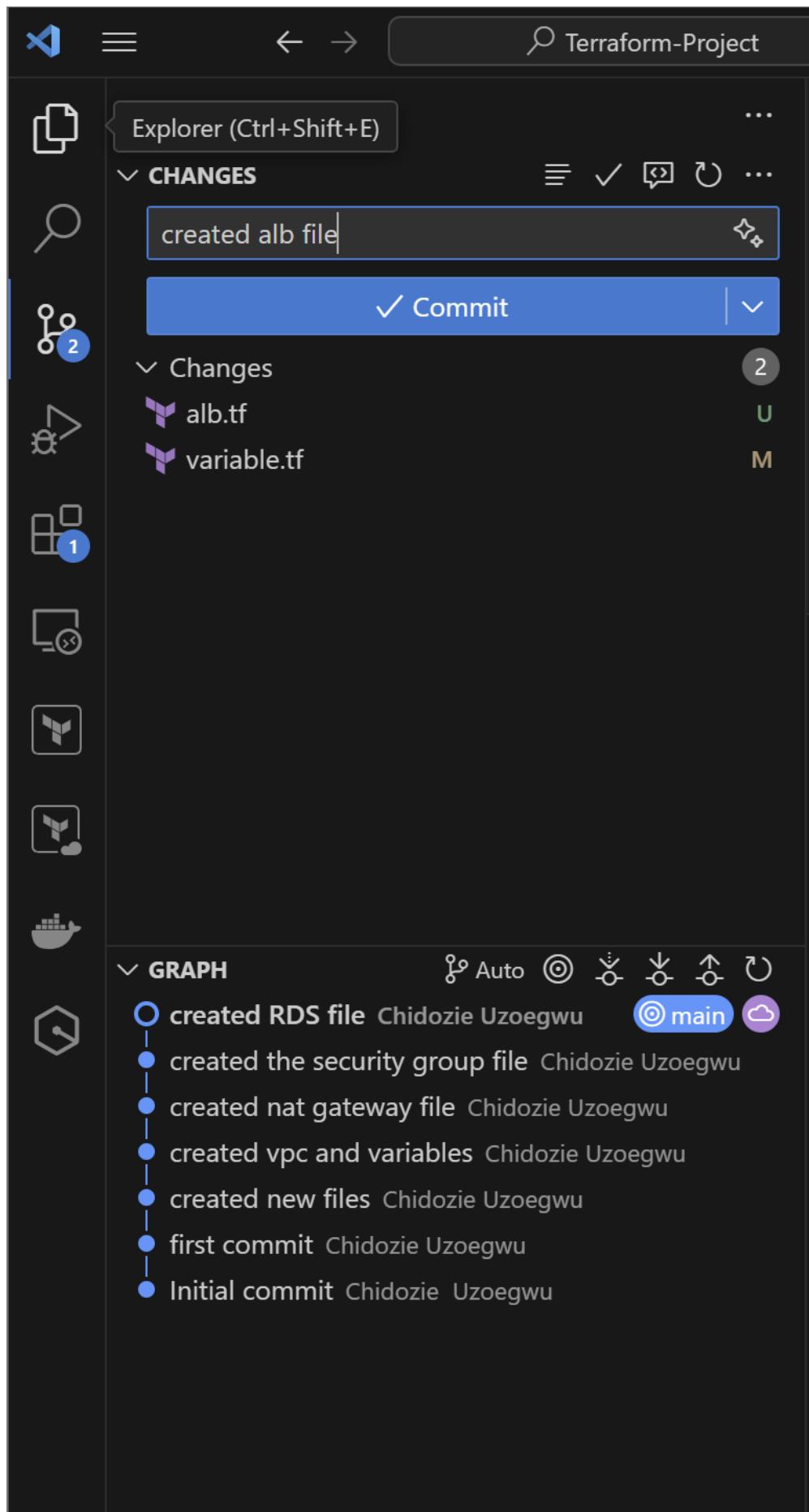
Name	DNS name	State	VPC ID	Availability Zones	Type	Date created
dev-alb	dev-alb-1745579596.us-east-1.elb.amazonaws.com	Active	vpc-de2536827512baf46	2 Availability Zones	application	March 30, 2025, 12:55 (UTC+01:00)

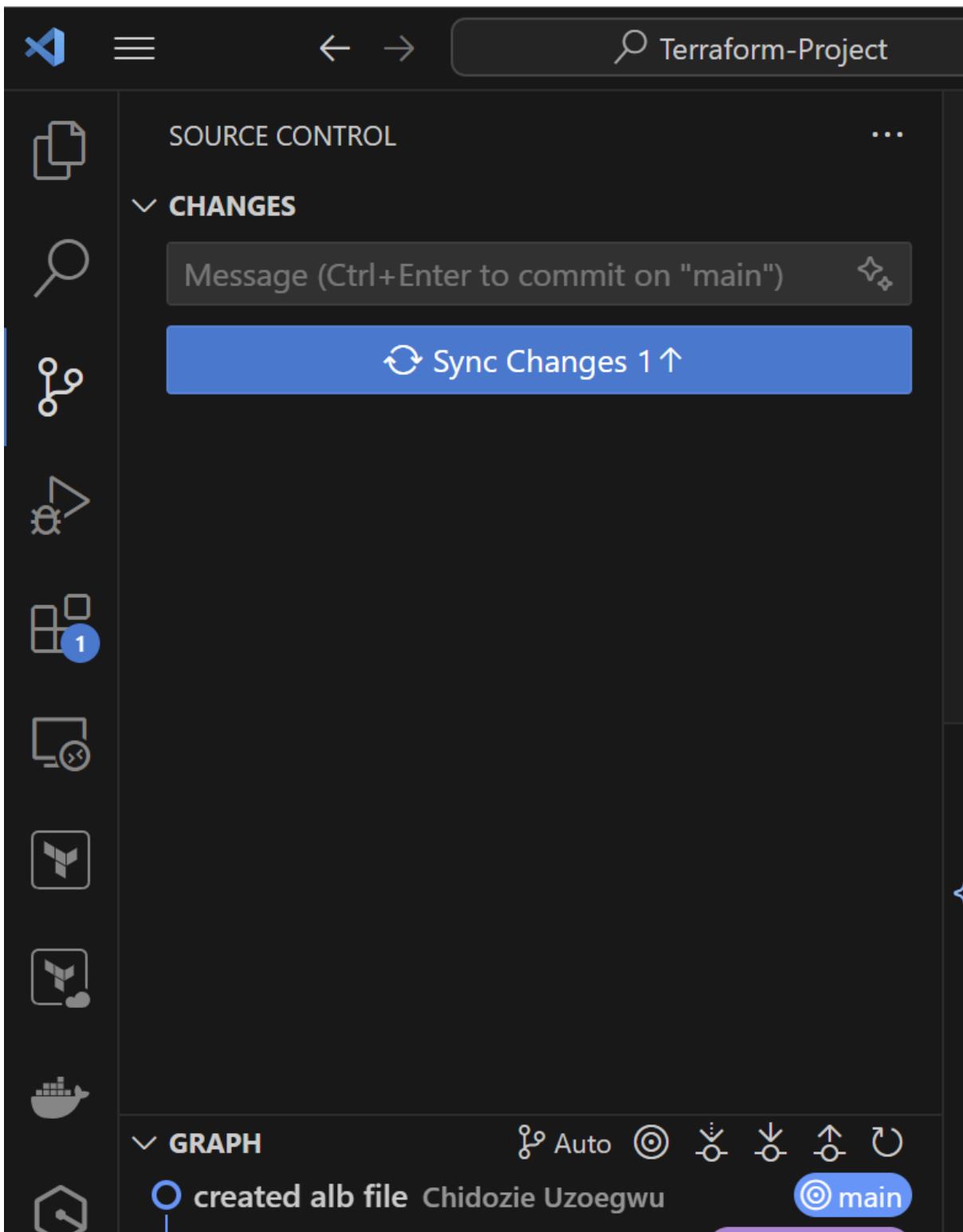
Load balancer: dev-alb

Listeners and rules (2) Info

Protocol/Port	Default action	Rules	ARN	Security policy	Default SSL/TLS certificate	mTLS	Trust store	Trust store association
HTTP:80	Redirect to HTTPS://#[Host].443/#[path]# (HTTP)	1 rule	Not applicable	Not applicable	Not applicable	Not applicable	Not applicable	Not applicable
HTTPS:443	Forward to target group (HTTP)	1 rule	ELBSecurityPolicy-2016-08	childishop.click (Certificate)	Off	Not applicable	Not applicable	Not applicable

Afterwards, we can push our code to GitHub.





```

Terraform-Project / alb.tf

chid010 created alb file

Code Blame 78 lines (67 loc) · 1.92 KB

1  # create application load balancer
2  # terraform aus create application load balancer
3  resource "aws_lb" "application_load_balancer" {
4      name            = "dev-alb"
5      internal        = false
6      load_balancer_type = "application"
7      security_groups = [aws_security_group.alb_security_group.id]
8
9      subnet_mapping {
10         subnet_id = aws_subnet.public_subnet_az1.id
11     }
12
13     subnet_mapping {
14         subnet_id = aws_subnet.public_subnet_az2.id
15     }
16
17     enable_deletion_protection = false
18
19     tags = [
20         {Name = "dev-alb"}
21     ]
22   }
23
24   # create target group
25   # terraform aus create target group
26   resource "aws_lb_target_group" "alb_target_group" {
27       name            = "dev-tg"
28       target_type    = "instance"
29       port           = 80
30       protocol       = "HTTP"
31       vpc_id         = aws_vpc.vpc.id
32
33       health_check {
34           healthy_threshold = 5
35           interval        = 30
36           matcher         = "*200,301,302"
37           path             = "/"
38           port             = "traffic-port"
39           protocol        = "HTTP"
40           timeout         = 5
41           unhealthy_threshold = 2
42       }
43   }
44
45   # create a listener on port 80 with redirect action
46   # terraform aus create listener

```

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:

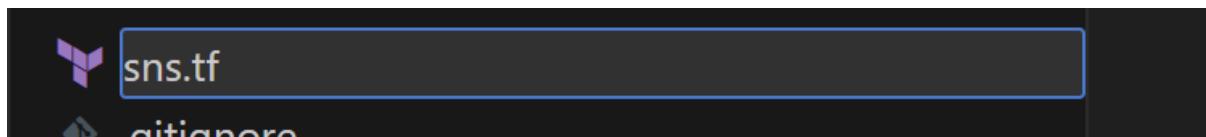
1. Create an SNS topic named dev-sns-topic

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

```
sns.tf > resource "aws_sns_topic_subscription" "notification_topic" > endpoint
1 # create an sns topic
2 # terraform aws create sns topic
3 resource "aws_sns_topic" "user_updates" {
4   name      = "dev-sns-topic"
5 }
6
7 # create an sns topic subscription
8 # terraform aws sns topic subscription
9 resource "aws_sns_topic_subscription" "notification_topic" [
10   topic_arn = aws_sns_topic.user_updates.arn
11   protocol  = "email"
12   endpoint   = var.operator_email
13 }
```

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.

```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS CODE REFERENCE LOG COMMENTS
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_az1: Refreshing state... [id=ipalloc-035a7e76907b2048]
aws_eip.eip_for_nat_gateway_az2: Refreshing state... [id=ipalloc-0a3c649035e2ef090]
data.aws_db_snapshot.latest_db_snapshot: Read complete after 0s [id=shopwise]
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_subnet_private_data_subnet_az1: Refreshing state... [id=subnet-0909acc2b7b6418c7]
aws_internet_gateway.internet_gateway: Refreshing state... [id=igw-0550600374ce71ab9]
aws_subnet_private_app_subnet_az1: Refreshing state... [id=subnet-0124b4873d7181605]
aws_security_group.ssh_security_group: Refreshing state... [id=sg-0031c157b6a1af1b1]
aws_subnet_public_subnet_az1: Refreshing state... [id=subnet-040cfc8cb3be547dc]
aws_lb_target_group.alb_target_group: Refreshing state... [id=arn:aws:elasticloadbalancing:us-east-1:375630861224:targetgroup/dev-tg/206605fb9957c641]
aws_security_group.alb_security_group: Refreshing state... [id=sg-0300d75a203b30103]
aws_nat_gateway.nat_gateway_az2: Refreshing state... [id=nat-0b90246fb7c4095a]
aws_route_table.public_route_table: Refreshing state... [id=rtb-0aad8ffe33ba8d07d]
aws_db_subnet_group.database_subnets: Refreshing state... [id=database_subnets]
aws_nat_gateway.nat_gateway_az1: Refreshing state... [id=nat-02c2aa8fe11beed3]
aws_lb.application_load_balancer: Refreshing state... [id=arn:aws:elasticloadbalancing:us-east-1:375630861224:loadbalancer/app/dev-alb/8d0b30127038db01]
aws_route_table.private_route_table_az2: Refreshing state... [id=rtb-0906b7387a0adefa4e]
aws_security_group.webserver_security_group: Refreshing state... [id=sg-01f1d1d714b18a486]
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]
aws_route_table_association.private_data_subnet_az2_route_table_association: Refreshing state... [id=rtbassoc-01d7257cfed278331]
aws_route_table_association.private_app_subnet_az2_route_table_association: Refreshing state... [id=rtbassoc-01dd1209bc8bb9c1]

```

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

The screenshot shows the AWS SNS Topics page. At the top, there is a blue banner with the text "New Feature: Amazon SNS now supports High Throughput FIFO topics. Learn more". Below the banner, the title "dev-sns-topic" is displayed. On the left, a sidebar shows "Mobile" sections for Push notifications and Text messaging (SMS). The main area has tabs for Subscriptions, Access policy, Data protection policy, Delivery policy (HTTP/S), Delivery status logging, Encryption, Tags, and Integrations. Under the Subscriptions tab, there is a table with one row. The row contains the ID "87b35508-4715-4e04-8a84-a...", the Endpoint "louddoodoo@gmail.com", the Status "Confirmed", and the Protocol "EMAIL". There are buttons for Edit, Delete, Request confirmation, Confirm subscription, and Create subscription.

The screenshot shows the AWS SNS Topics page. At the top, there is a blue banner with the text "New Feature: Amazon SNS now supports High Throughput FIFO topics. Learn more". Below the banner, the title "Topics (2)" is displayed. The main area has tabs for Subscriptions, Access policy, Data protection policy, Delivery policy (HTTP/S), Delivery status logging, Encryption, Tags, and Integrations. Under the Subscriptions tab, there is a table with two rows. The first row has the Name "dev-sns-topic", Type "Standard", and ARN "arn:aws:sns:us-east-1:375630861224:dev-sns-t...". The second row has the Name "Dev-Topic", Type "Standard", and ARN "arn:aws:sns:us-east-1:375630861224:Dev-Topic". There are buttons for Edit, Delete, Publish message, and Create topic.

6. Version Control Confirmation

In Visual Studio Code:

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

The screenshot shows a dark-themed interface for managing a Terraform project. The top bar includes a back/forward navigation, a search field labeled "Terraform-Project", and a menu icon.

SOURCE CONTROL

CHANGES

- created SNS file

Commit

Changes

- sns.tf
- variable.tf

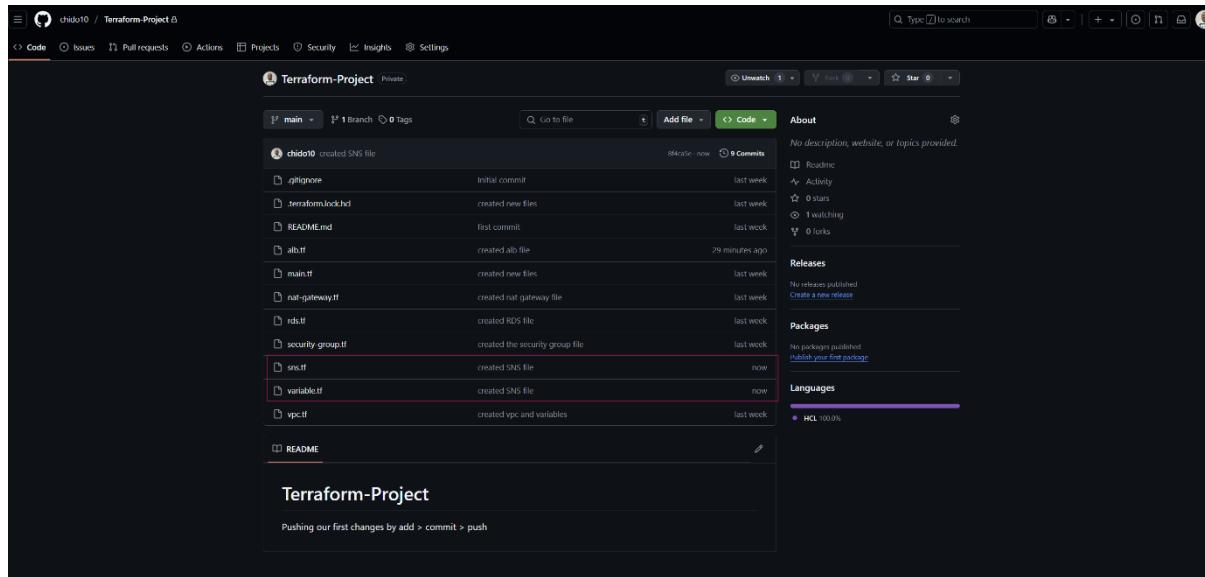
GRAPH

- 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

Icons on the left side represent various project components: file, search, deployment, cloud, database, network, and server.

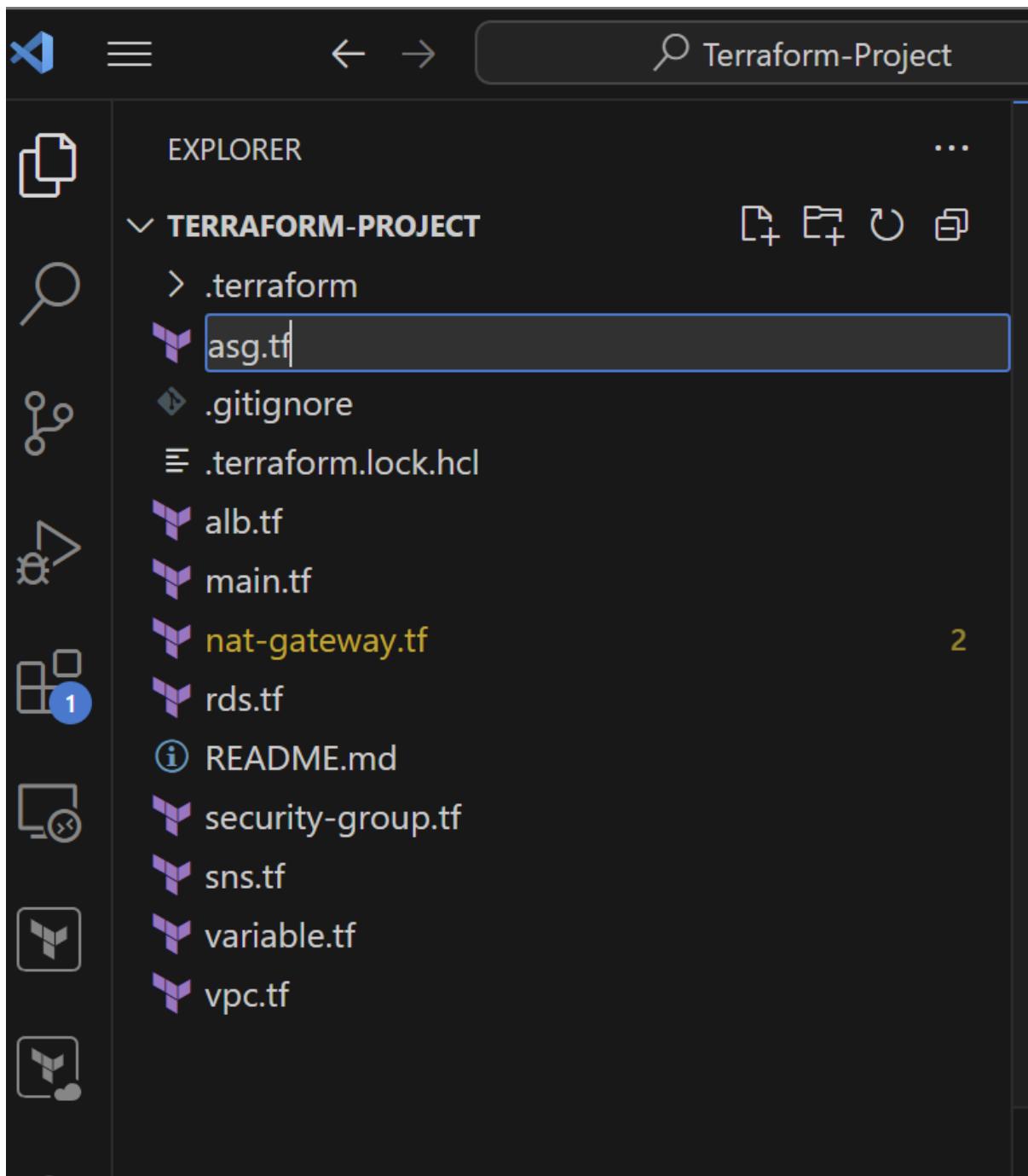
7. Pushed to GitHub

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 | vpc.tf | variable.tf M
asg.tf > resource "aws_autoscaling_notification" "webserver_asg_notifications" > [ ] notifications
  1 # create a launch template
  2 # terraform aws launch template
  3 resource "aws_launch_template" "webserver_launch_template" {
  4   name        = var.launch_template_name
  5   image_id    = var.ec2_image_id
  6   instance_type = var.ec2_instance_type
  7   key_name    = var.ec2_key_pair_name
  8   description  = "launch template for asg"
  9
 10  monitoring {
 11    enabled = true
 12  }
 13
 14  vpc_security_group_ids = [aws_security_group.webserver_security_group.id]
 15 }
 16
 17 # create auto scaling group
 18 # terraform aws autoscaling group
 19 resource "aws_autoscaling_group" "auto_scaling_group" {
 20   vpc_zone_identifier = [aws_subnet.private_app_subnet_az1.id, aws_subnet.private_app_subnet_az2.id]
 21   desired_capacity     = 2
 22   max_size             = 4
 23   min_size             = 1
 24   name                 = "dev-asg"
 25   health_check_type    = "ELB"
 26
 27   launch_template {
 28     name      = aws_launch_template.webserver_launch_template.name
 29     version   = "$Latest"
 30   }
 31
 32   tag {
 33     key       = "Name"
 34     value     = "asg-webserver"
 35     propagate_at_launch = true
 36   }
 37
 38   lifecycle {
 39     ignore_changes = [target_group_arns]
```

```

resource "aws_autoscaling_group" "auto_scaling_group" {
  tag {
    value          = "asg-webserver"
    propagate_at_launch = true
  }
  lifecycle {
    ignore_changes = [target_group_arns]
  }
}

# attach auto scaling group to alb target group
# terraform aws autoscaling attachment
resource "aws_autoscaling_attachment" "asp_alb_target_group_attachment" {
  autoscaling_group_name = aws_autoscaling_group.auto_scaling_group.id
  lb_target_group_arn   = aws_lb_target_group.alb_target_group.arn
}

# create an auto scaling group notification
# terraform aws autoscaling notification
resource "aws_autoscaling_notification" "webserver_asg_notifications" {
  group_names = [aws_autoscaling_group.auto_scaling_group.name]

  notifications = [
    "autoscaling:EC2_INSTANCE_LAUNCH",
    "autoscaling:EC2_INSTANCE_TERMINATE",
    "autoscaling:EC2_INSTANCE_LAUNCH_ERROR",
    "autoscaling:EC2_INSTANCE_TERMINATE_ERROR",
  ]
}

topic_arn = aws sns topic.user_updates.arn
}

```

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-0e2536827512bafaf6]
aws_eip.eip_for_nat_gateway_az2: Refreshing state... [id=eilalloc-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=eilalloc-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-040cfcc8cb3be547dc]
aws_subnet.private_app_subnet_az2: Refreshing state... [id=subnet-08bd016021a840cf6]
aws_subnet.public_subnet_az2: Refreshing state... [id=subnet-02c07d8bb6e648f098]
aws_subnet.private_data_subnet_az2: Refreshing state... [id=subnet-0f84f7c3f42127c47]
aws_security_group.ssh_security_group: Refreshing state... [id=sg-0d31c157b6a1af1b1]
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-0aad8fe33ba8d07d]
aws_nat_gateway.nat_gateway_az1: Refreshing state... [id=nat-02c22a8fe11beed3]

```

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

Name	Launch template/configuration	Instances	Status	Desired capacity	Min	Max	Availability Zones
dev-asg	dev-launch-template Version Latest	2	-	2	1	4	us-east-1a, us-east-1b

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

The screenshot shows the AWS EC2 Launch Templates page. On the left, there's a sidebar with navigation links for EC2 (Dashboard, Global View, Events), Instances (Instances, Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations), and Images (AMIs, AMI Catalog). The main content area has a title "Launch Templates (1) info". A search bar is at the top. Below it is a table with one row:

Launch Template ID	Launch Template Name	Default Version	Latest Version	Create Time	Created By	Managed	Operator
lt-07e973188947c78c	dev-launch-template	1	1	2025-03-30T13:18:18.000Z	arn:aws:iam::375630861224:us...	false	-

Below the table, a message says "Select a launch template".

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

Check Target Group health in ALB

The screenshot shows the AWS Application Load Balancer (ALB) Target Groups page. The target group is named `dev-tg`. The "Details" section includes:

- Target type: Instance
- Protocol: Port
- Port: 80
- Protocol version: HTTP1
- VPC: `vpc-0e2536827512bafaf6`
- Total targets: 2
- Health status: 2 Healthy, 0 Unhealthy, 0 Unused, 0 Initial, 0 Draining

A table titled "Distribution of targets by Availability Zone (AZ)" shows the distribution of targets across zones. Below the table, a note says "Select values in this table to see corresponding filters applied to the Registered targets table below."

The "Targets" tab is selected, showing a table of registered targets:

Instance ID	Name	Port	Zone	Health status	Health status details	Administrative ...	Override details	Launch...	Anomaly de...
i-0a2499ca1ec7d53b0	asg-webserver	80	us-east-1b (us...)	Healthy	-	No override	No override is curr...	March 30...	Normal
i-04fc27b23d9252231	asg-webserver	80	us-east-1a (us...)	Healthy	-	No override	No override is curr...	March 30...	Normal

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

Verify running EC2 instances

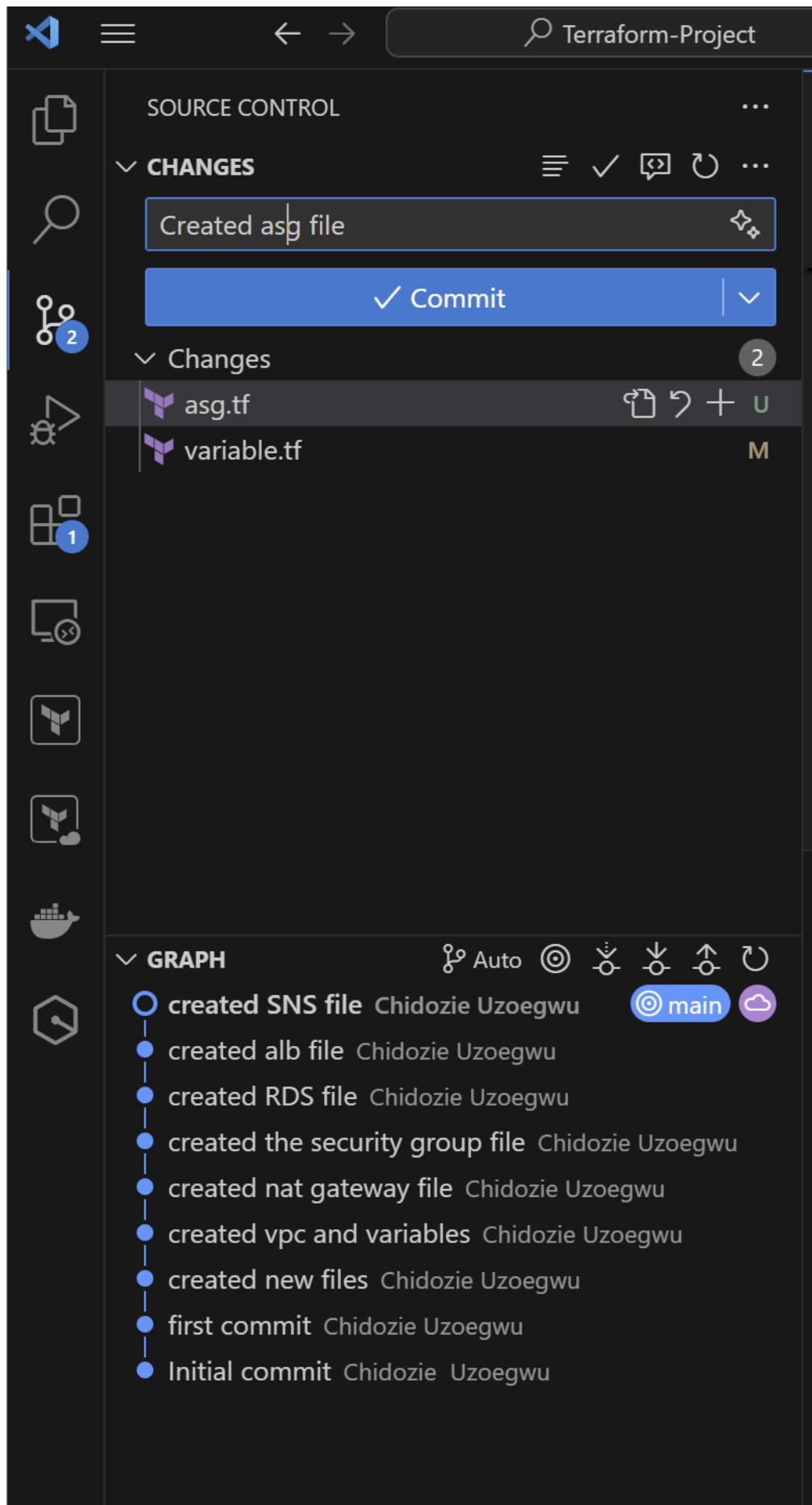
The screenshot shows the AWS EC2 Instances page with the following details:

- Instances (2) Info**: Shows there are 2 instances.
- Last updated**: less than a minute ago.
- Connect**: A blue button.
- Instance state**: A dropdown set to "All states".
- Actions**: A dropdown.
- Launch instances**: An orange button.
- Filter**: "Find Instance by attribute or tag (case-sensitive)".
- Table Headers**: Name, Instance ID, Instance state, Instance type, Status check, Alarm status, Availability Zone, Public IPv4 DNS, Public IPv4 IP, Elastic IP.
- Table Data**:

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 IP	Elastic IP
asg-webserver	i-0a2499ca1ec7d53b0	Running	t2.micro	2/2 checks passed	View alarms	us-east-1b	-	-	-
asg-webserver	i-04fc27b23d9252231	Running	t2.micro	2/2 checks passed	View alarms	us-east-1a	-	-	-

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



 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			

```

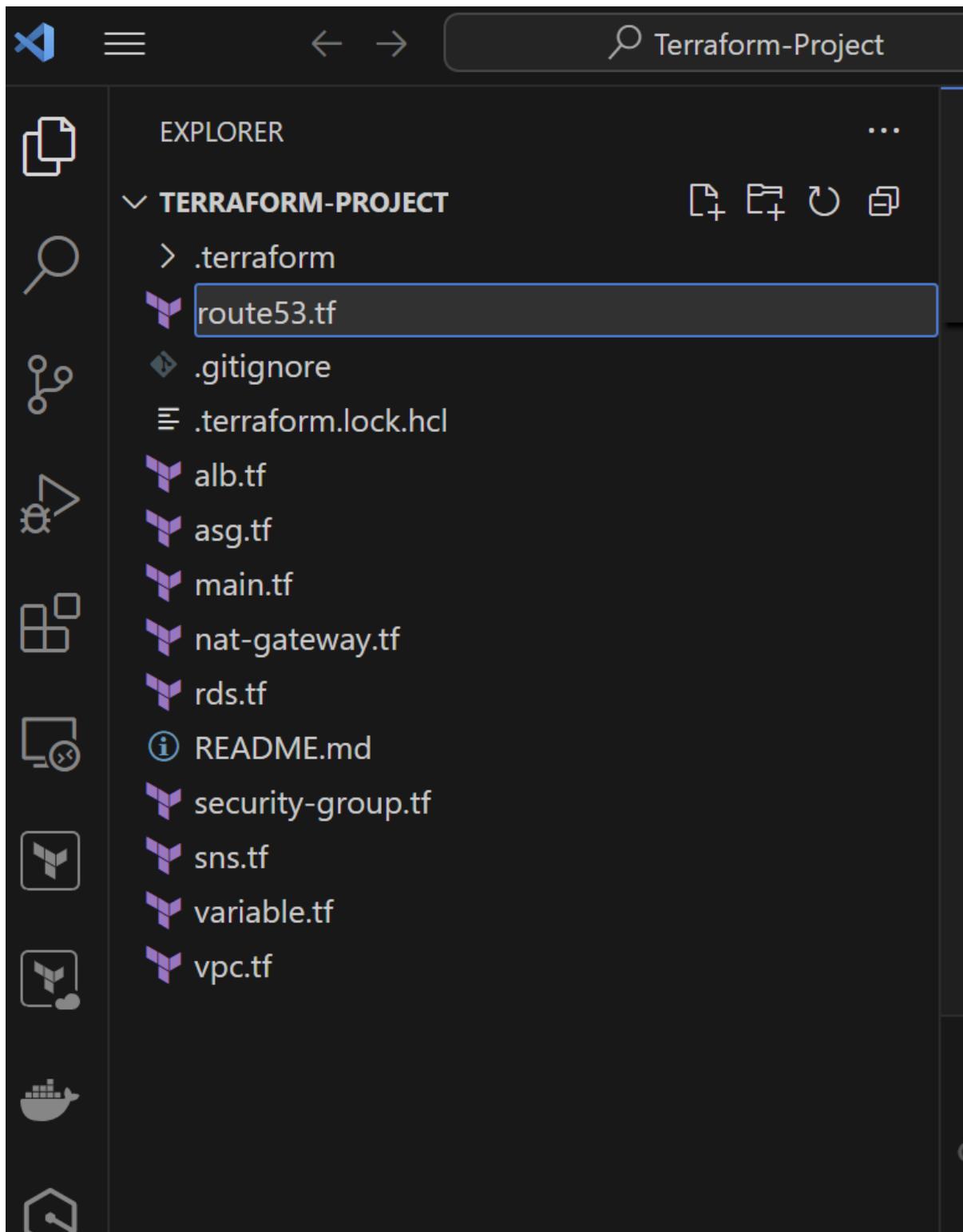
Terraform-Project / asg.tf

Code Blame 63 lines (53 sec) · 1:43 60

1 # create a launch template
2 # terraform aws lambda template
3 resource "aws_lambda_function" "webserver_launch_template" {
4   name          = var.launch_template_name
5   image_id      = var.lambda_image_id
6   instance_type = var.lambda_instance_type
7   memory_size   = var.lambda_memory_size
8   role          = var.lambda_role_arn
9   description   = "Launch template for asg"
10
11   monitoring {
12     enabled = true
13   }
14
15   vpc_security_group_ids = [aws_security_group.webserver_security_group.id]
16
17 # create auto scaling group
18 # terraform aws autoscaling group
19 resource "aws_autoscaling_group" "auto_scaling_group" {
20   vpc_zone_identifier = [aws_subnet.private_app_subnet.id, aws_subnet.private_app_subnet_a.id]
21   desired_capacity    = 1
22   max_size            = 4
23   min_size            = 1
24   launch_template     = aws_lambda_function.webserver_launch_template.arn
25   health_check_type   = "ELB"
26
27   launch_template {
28     name       = aws_lambda_function.webserver_launch_template.name
29     version    = "latest"
30   }
31
32   tag {
33     key        = "Name"
34     value      = "asg webserver"
35     propagate_to_launch = true
36   }
37
38   lifecycle {
39     ignore_changes = [target_group_arns]
40   }
41
42 # attach auto scaling group to alb target group
43 # terraform aws autoscaling attachment
44 resource "aws_autoscaling_attachment" "tag_asg_target_group_attachment" {
45   auto_scaling_group_name = aws_autoscaling_group.auto_scaling_group.name
46   lb_target_group_arn    = aws_lb_target_group.alb_target_group.arn
47 }
48
49 # create an auto scaling group notification
50 # terraform aws autoscaling notification
51 resource "aws_autoscaling_notification" "webserver_asg_notifications" {
52   group_name   = aws_autoscaling_group.auto_scaling_group.name
53   notifications = [
54     "autoscaling:EC2_INSTANCE_LAUNCH",
55     "autoscaling:EC2_INSTANCE_TERMINATE",
56     "autoscaling:EC2_INSTANCE_LAUNCH_ERROR",
57     "autoscaling:EC2_INSTANCE_TERMINATE_ERROR",
58   ]
59
60   topic_arn    = aws sns topic.user_updates.arn
61 }
62

```

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.

The screenshot shows the AWS Route 53 Registered domains page. On the left, there's a navigation sidebar with sections like Dashboard, Hosted zones, Health checks, Profiles, IP-based routing, Traffic flow, Domains, Resolver, DNS Firewall, and Application Recovery Controller. The main area is titled "Registered domains" and shows a table with two rows. The first row has "chidzieshop.click" in the Domain name column, "May 03, 2025" in the Expiration date column, "Off" in both Auto-renew and Transfer lock columns. The second row has "chidzieuozoqwu.click" in the Domain name column, "April 02, 2025" in the Expiration date column, "Off" in both Auto-renew and Transfer lock columns. There are buttons for Download billing report, Transfer in, Register domains, and navigation arrows.

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

```
route53.tf
1 # get hosted zone details
2 # terraform aws data hosted zone
3 data "aws_route53_zone" "hosted_zone" {
4   name = var.domain_name
5 }
6
7 # create a record set in route 53
8 # terraform aws route 53 record
9 resource "aws_route53_record" "site_domain" {
10   zone_id = data.aws_route53_zone.hosted_zone.zone_id
11   name    = var.record_name
12   type    = "A"
13
14   alias {
15     name          = aws_lb.application_load_balancer.dns_name
16     zone_id      = aws_lb.application_load_balancer.zone_id
17     evaluate_target_health = true
18   }
19 }
```

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

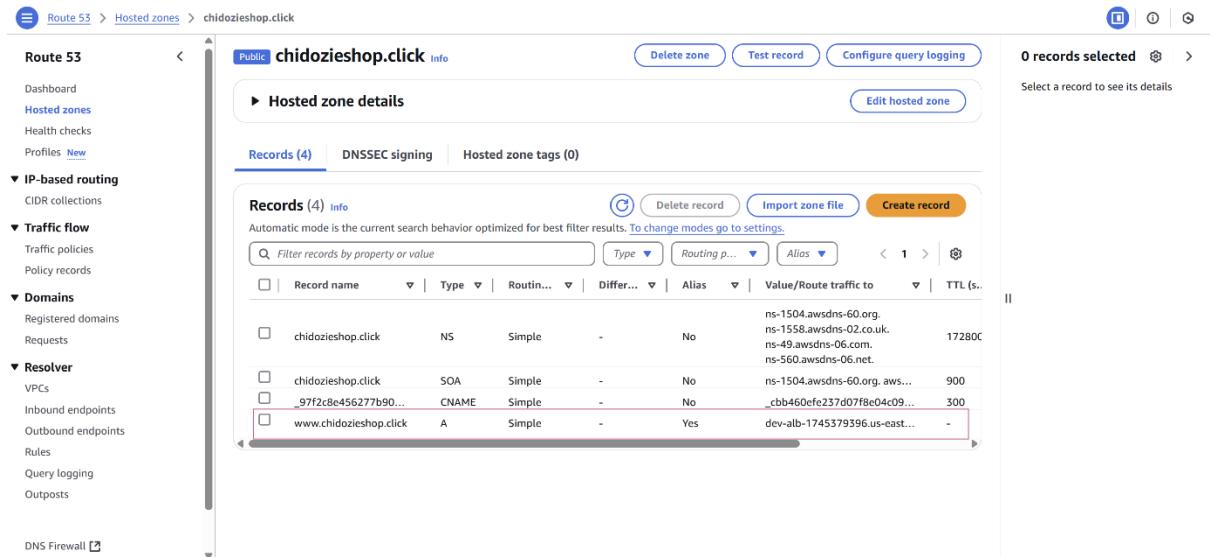
variable "record_name" {
  default      = "www"
  description   = "sub domain name"
  type         = string
}
```

Next, we terraform apply and confirm it successful

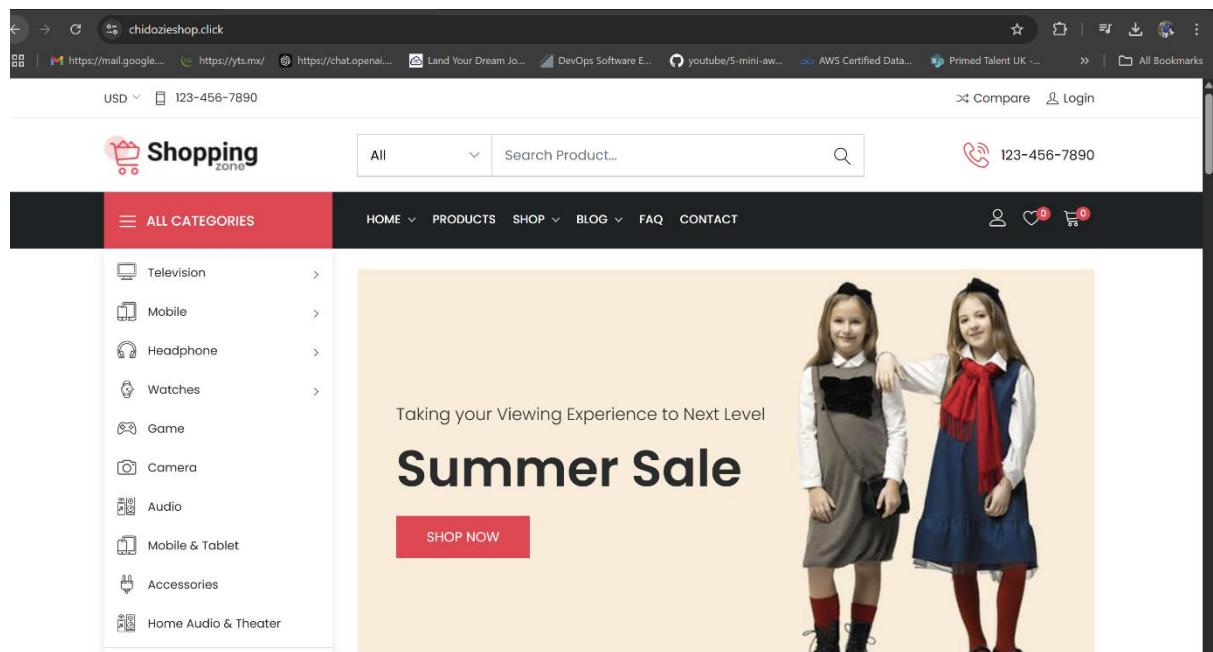
```
Apply complete! Resources: 2 added, 0 changed,
Manage: Users\chido\Desktop\Terraform-Project>
```

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

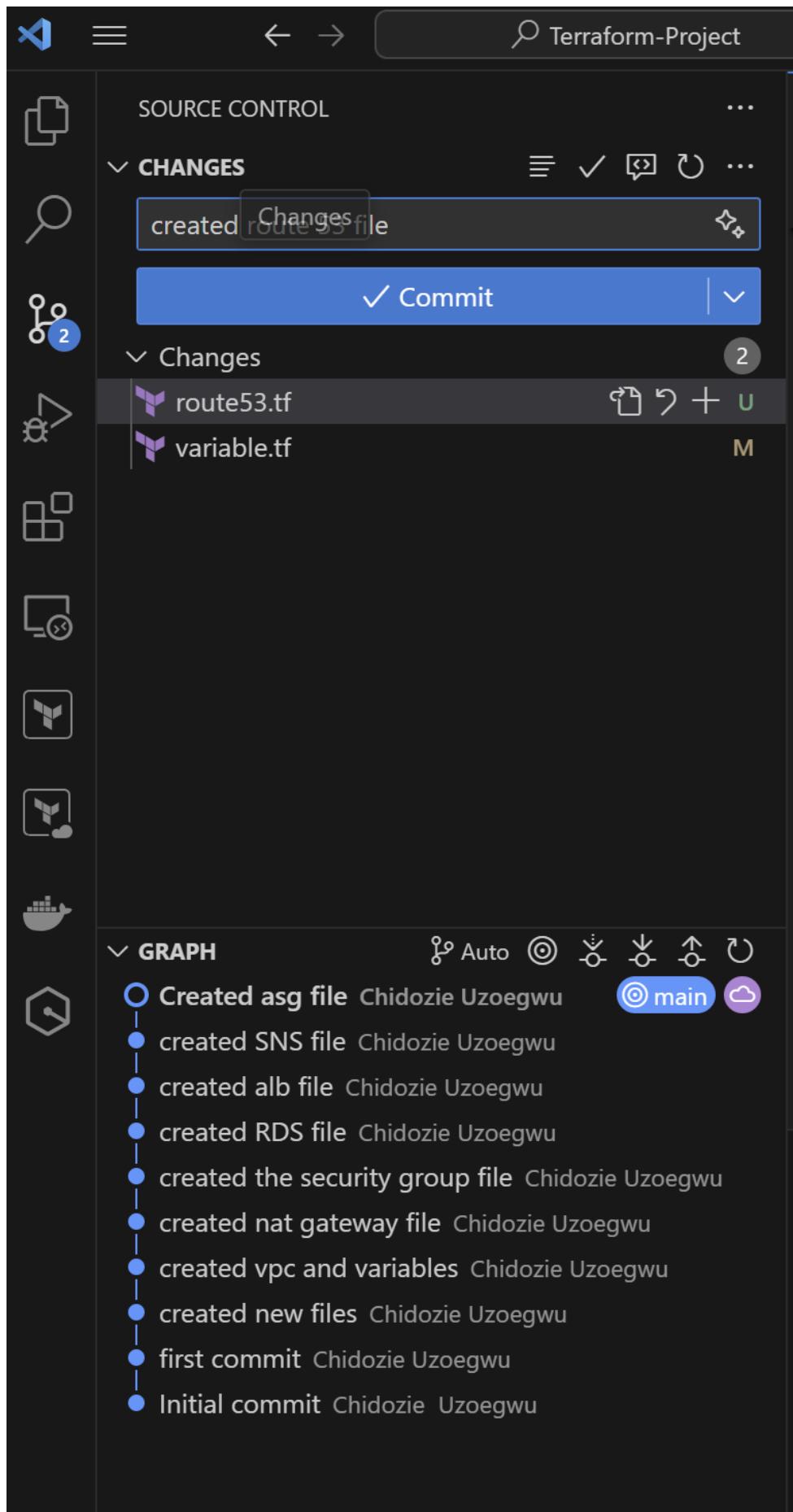


The screenshot shows the AWS Route 53 console. On the left, a sidebar menu includes 'Route 53' (selected), 'Dashboard', 'Hosted zones' (selected), 'Health checks', 'Profiles New', 'IP-based routing', 'Traffic flow', 'Domains', 'Resolver', and 'DNS Firewall'. The main area displays the 'chidozieshop.click' hosted zone details. It shows 4 records: an NS record for 'chidozieshop.click' pointing to AWS servers, an SOA record, a CNAME record for '_97f2c8e456277b90...', and an A record for 'www.chidozieshop.click' pointing to an IP address. Buttons for 'Delete zone', 'Test record', 'Configure query logging', and 'Edit hosted zone' are at the top right.



The screenshot shows a web browser displaying a shopping website. The URL is 'chidozieshop.click'. The page features a header with a phone icon and the number '123-456-7890'. A navigation bar includes 'All' and a search bar. The main content area has a sidebar with categories like 'Television', 'Mobile', 'Headphone', etc., and a large promotional banner for a 'Summer Sale' featuring two girls in school uniforms. A 'SHOP NOW' button is visible on the banner.

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

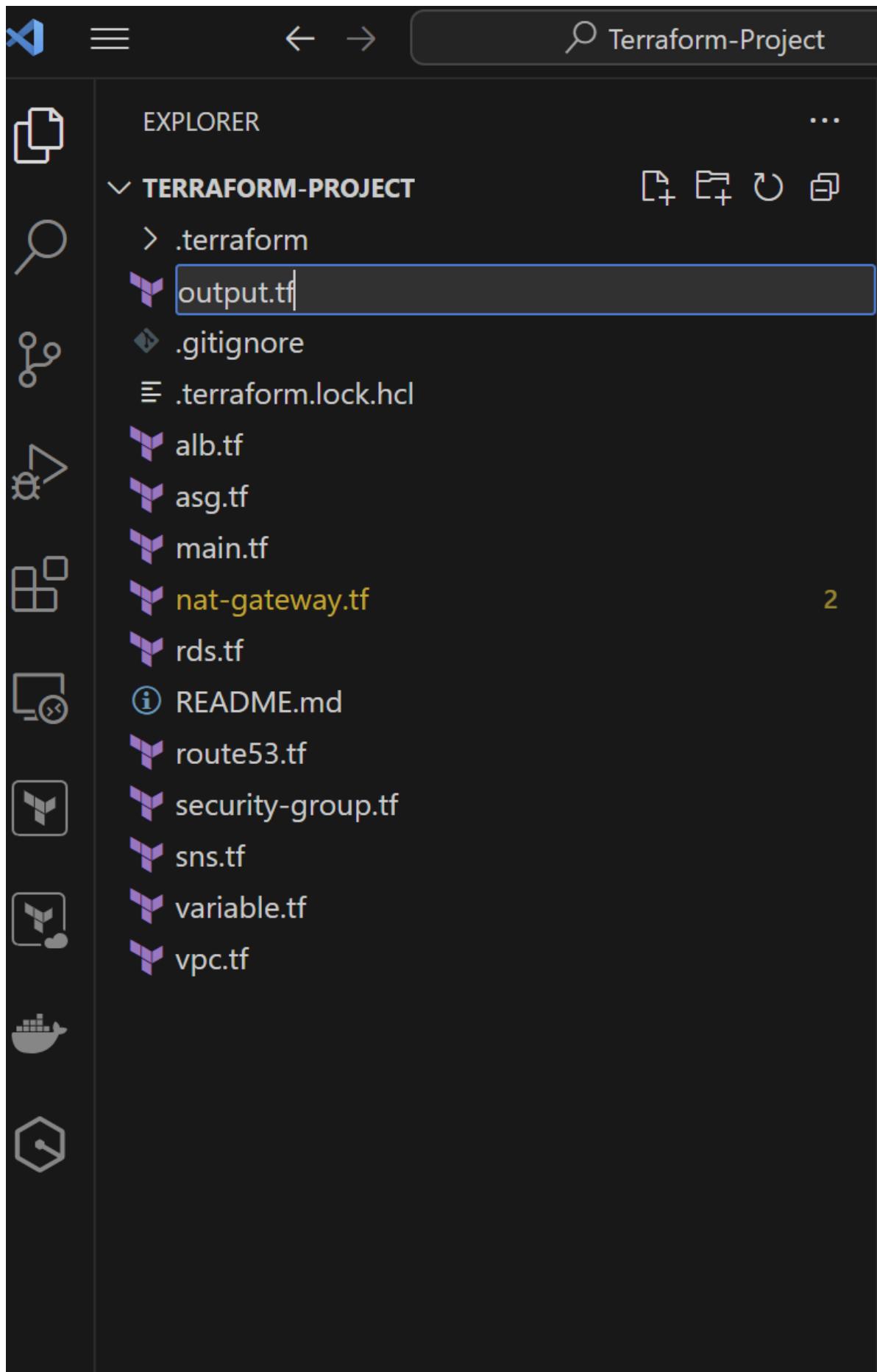


Afterwards, confirm in our repository

The screenshot shows a GitHub repository named "Terraform-Project". The repository is private, has 1 branch, and 0 tags. It contains 11 commits from user "chido10" over the last week. The commits include creating route 53 files, .gitignore, .terraform.lock.hcl, README.md, alb.tf, asg.tf, main.tf, nat-gateway.tf, rds.tf, route53.tf, security-group.tf, sns.tf, variable.tf, and vpc.tf. The "route53.tf" commit is highlighted with a red border. The repository has 0 stars, 1 watching, and 0 forks. It also has no releases published and no packages published. The Languages section shows HCL at 100.0%. The code editor on the right displays the "route53.tf" file, which contains Terraform configuration for creating a Route 53 record set.

```
1 # get hosted zone details
2 # terraform aws data hosted zone
3 data "aws_route53_zone" "hosted_zone" {
4   name = var.domain_name
5 }
6
7 # create a record set in route 53
8 # terraform aws route 53 record
9 resource "aws_route53_record" "site_domain" {
10   zone_id = data.aws_route53_zone.hosted_zone.zone_id
11   name   = var.record_name
12   type   = "A"
13
14   alias {
15     name      = aws_lb.application_load_balancer.dns_name
16     zone_id   = aws_lb.application_load_balancer.zone_id
17     evaluate_target_health = true
18   }
19 }
```

28. Terraform Outputs



```
output.tf
1  output "vpc_id" {
2    value = aws_vpc.vpc.id
3  }
4
5  output "public_subnet_az1_id" {
6    value = aws_subnet.public_subnet_az1.id
7  }
8
9  output "website_url" {
10   value = join("", [var.record_name, ".", var.domain_name])
11 }
```

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-040cf8cb3be547dc]
aws_subnet.private_subnet_az1: Refreshing state... [id=subnet-0124bf873d71816b5]
```

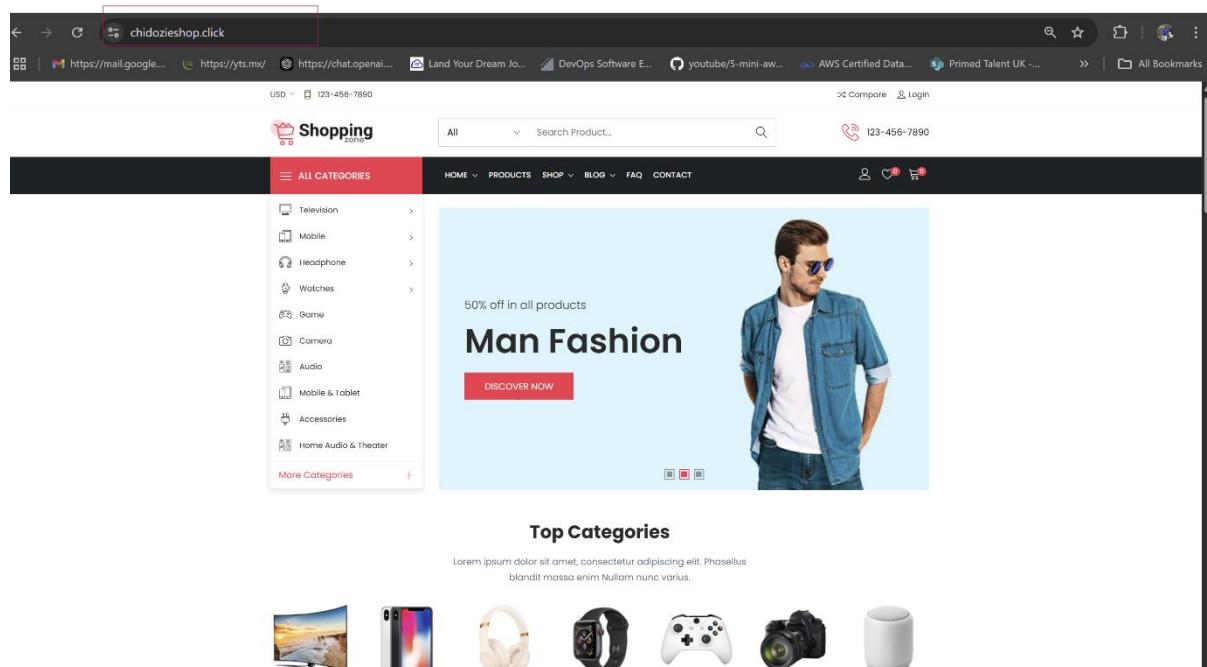
```
Changes to Outputs:
+ public_subnet_az1_id = "subnet-040cf8cb3be547dc"
+ vpc_id                = "vpc-0e2536827512bafa6"
+ website_url            = "www.chidozieshop.click"
```

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

Outputs:

```
public_subnet_az1_id = "subnet-040cf8cb3be547dc"
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.

```
Destroy complete! Resources: 38 destroyed.  
PS C:\Users\chido\Desktop\Terraform-Project>
```

- **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.