**DEPLOY AN EC2 INSTANCE INSIDE A CUSTOM VPC USING TERRAFORM**

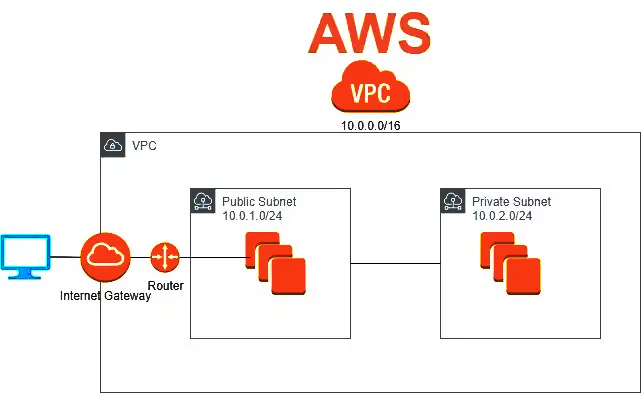
A picture containing text, clipart

Description automatically generated

**Project Description**

You have a cloud management team, and they are in charge of deploying. You deploy an Amazon EC2 instance inside a custom VPC just for your team and you are to add this instance inside a public subnet making it accessible to the public using IaC (Terraform).﻿

**Architecture Diagram**



**What is Terraform?**

Terraform is an infrastructure-as-code (IaC) tool through which you define both cloud and on-prem resources in human-readable configuration files that you can version, reuse and share across various cloud platforms as well as on-premises.

Terraform allows you to quickly automate and deploy resources on-prem and in the cloud. In my first project, I provisioned an Amazon EC2 instance in a custom VPC manually via the AWS console. This was time consuming and when I deleted all the resources deployed in the project, I would have to start all over again. But, with Terraform, it is all about automation. By creating a Terraform a module with configuration files, we can create reproducible infrastructure. Even when the resources are deleted, they can be reproduced easily and quickly because the codes are reusable.

Terraform code is stored in the **.tf** or **.tfjson** file extension and the syntax is written in Hashicorp Configuration Language.

**Prerequisites**

1. AWS account

2. AWS CLI

3. VS Code

4. Terraform Installed

5. Git account

To begin, launch VScode and create a terraform module with the following configuration files needed by Terraform to create our resources; **main.tf, variable.tf and output.tf**

The following tasks will be performed

1. Clone the GitHub repo to push the configuration files to

2. Create a custom VPC

3. Create a public subnet

4. Configure and launch an ec2 instance inside the custom VPC.

5. Add user data

6. Attach a security group to the ec2 instance created

7. Initialize Terraform and provider plugins

8. Check the plan – the changes required by the configuration

9. Launch the EC2 instance

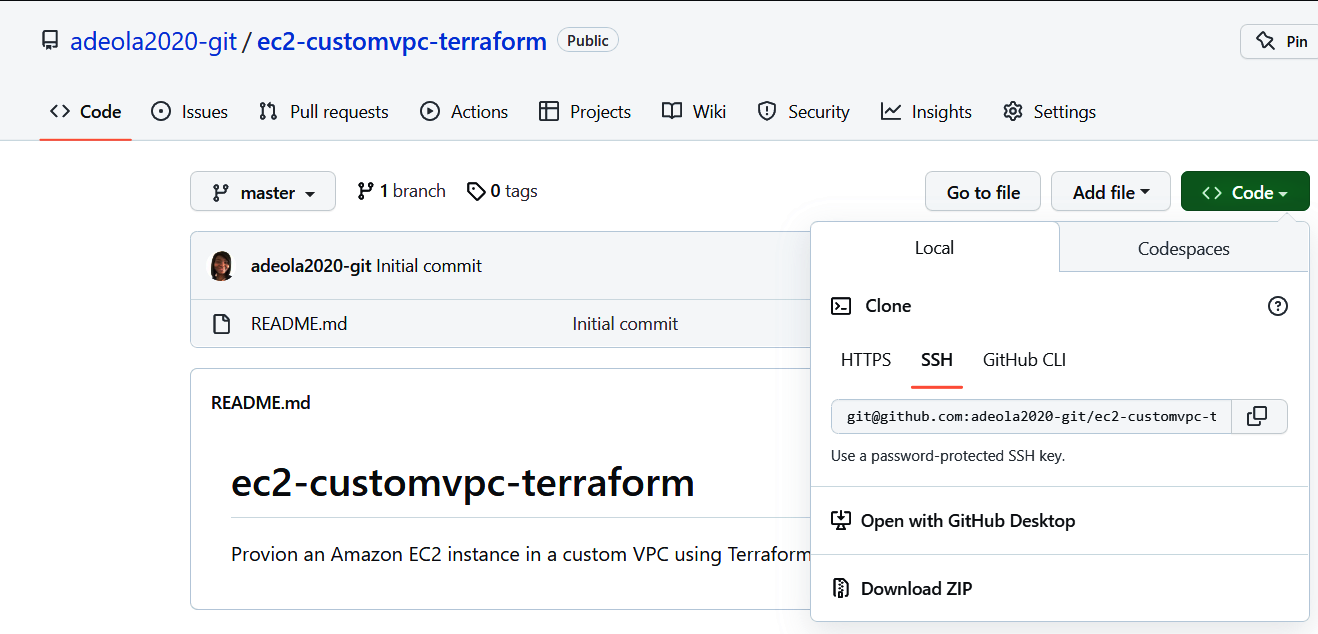
10. View the website and other resources Terraform created

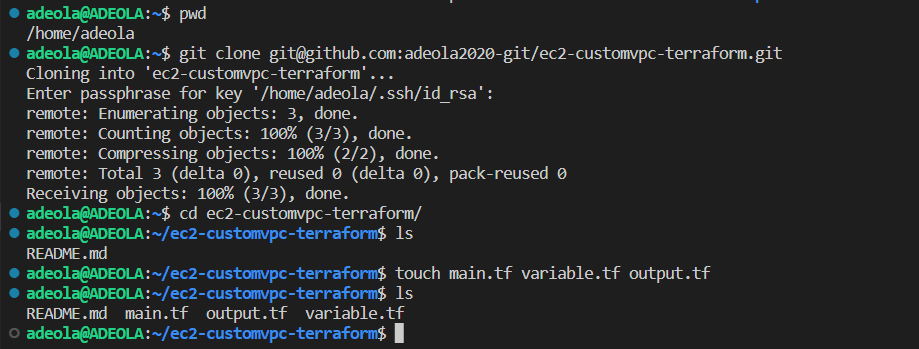
11. Delete the provisioned resources

12. Add configuration files to the remote repository

**Task 1: Clone the GitHub repo**

Clone the GitHub repo and change to the initialized directory. Then, create the **main.tf, variable.tf** and **output.tf** configuration files and start writing configuration codes to provision the resources needed to build our infrastructure.

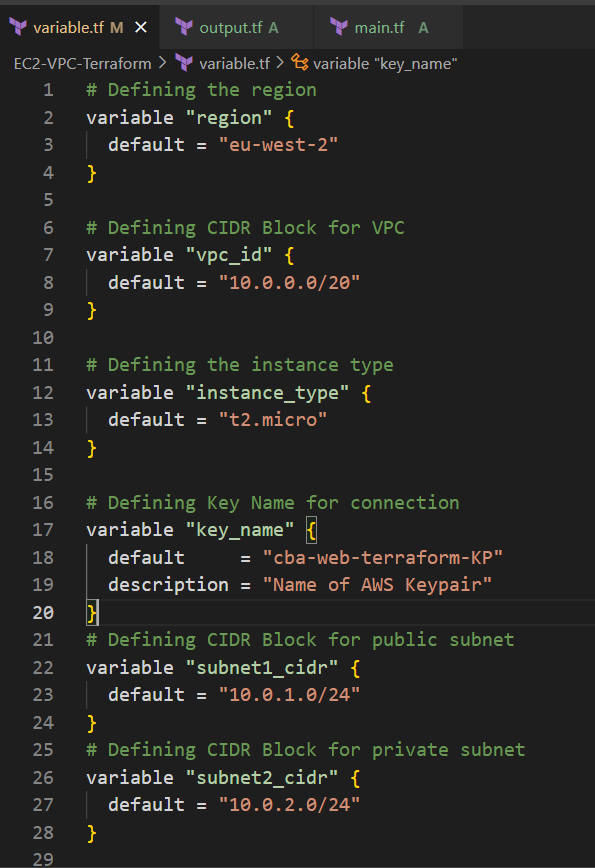




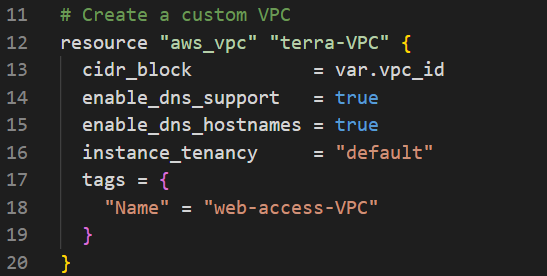
**Task 2: Create a custom VPC**

The **variable.tf** file has the configuration for our variables used in the **main.tf** file. And on the **main.tf** file, we have the resource block for the custom VPC as shown below;

**- variable.tf**



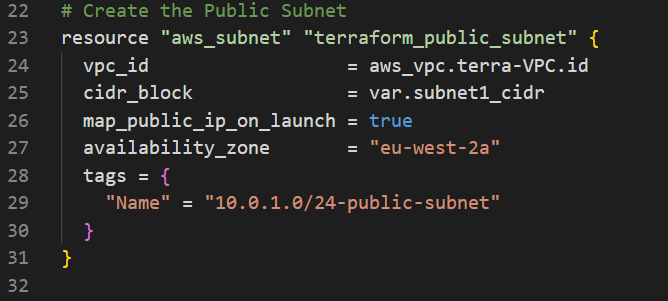
**- main.tf**



**Task 3: Create a Public subnet**

A public subnet on the VPC is where our EC2 instance will be created to make it accessible the public.

**- main.tf**



**Tasks 4, 5 and 6: Configure the ec2 instance**

Configure the ec2 instance inside the public subnet of the custom VPC, attach a Security Group to the instance and add userdata.

**- main.tf**

Text

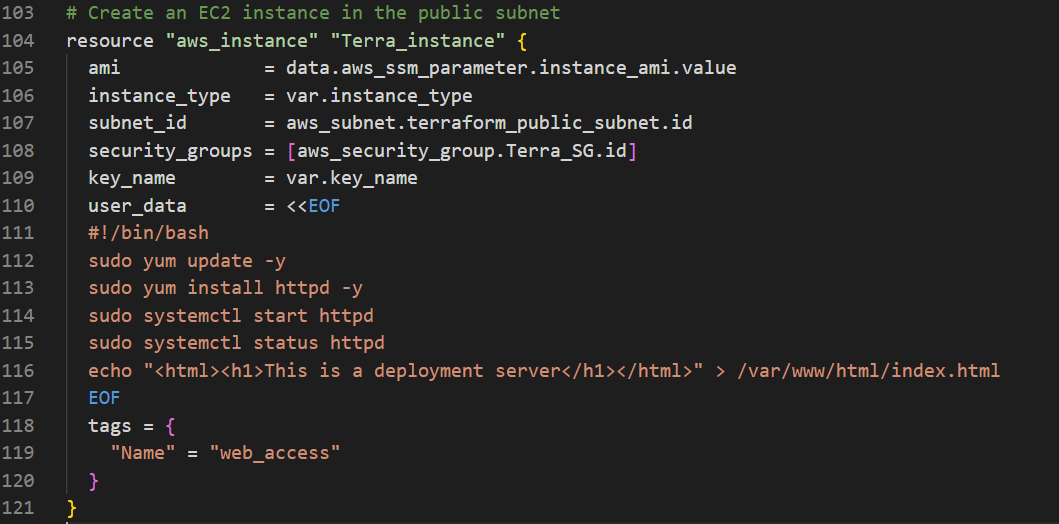
Description automatically generated

Text

Description automatically generated

Text

Description automatically generated



From the instance resource block in the **main.tf** file, we added a **user\_data** with a bashscript to install an Apache webserver on the instance at launch and we will be able to view our webpage with the contents in the index.html file configured.

**Task 7: Initialize Terraform and provider plugins**

Run **terraform init** to initialize our Terraform infrastructure and provider plugin

Text

Description automatically generated

**Task 8: Check the changes required by the current configuration**

Run **terraform validate** to check if the configuration is valid, then, **terraform plan** to show changes required by the current configuration.

Text

Description automatically generated

**$ terraform plan**

Graphical user interface

Description automatically generated

**Task 9: Launch the EC2 instance**

To launch the EC2 instance, we run **terraform apply** and select **yes** when prompted. This creates and launches our EC2 instance and all the other resources in the configuration plan.

Also, we created an **output.tf** file which displays our **website url** after instance is launched with **terraform apply.**

- **output.tf**

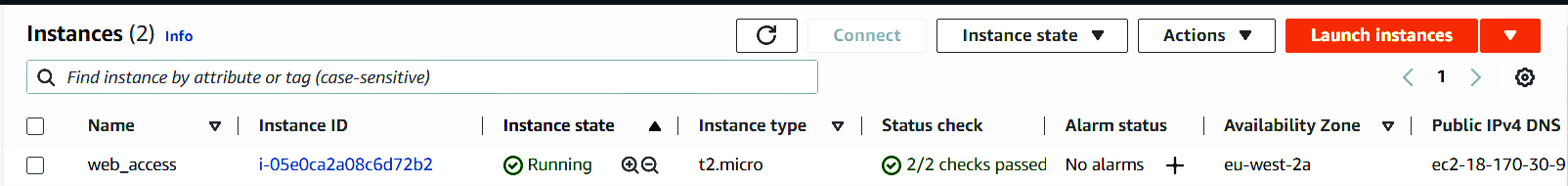
A screenshot of a computer

Description automatically generated with medium confidence

Result after running **terraform apply**

Text

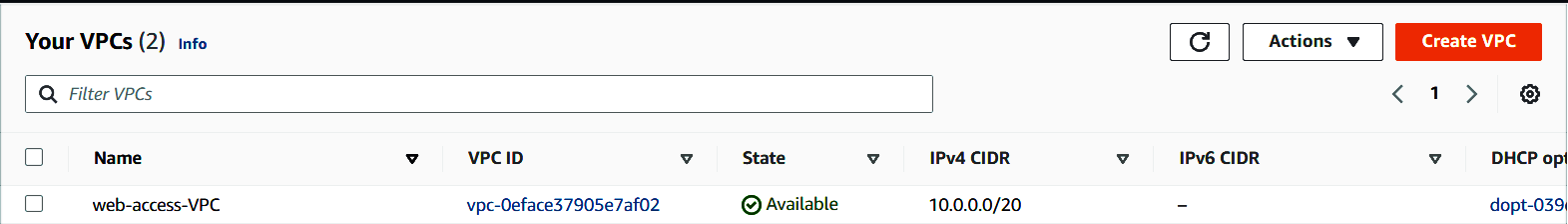
Description automatically generated



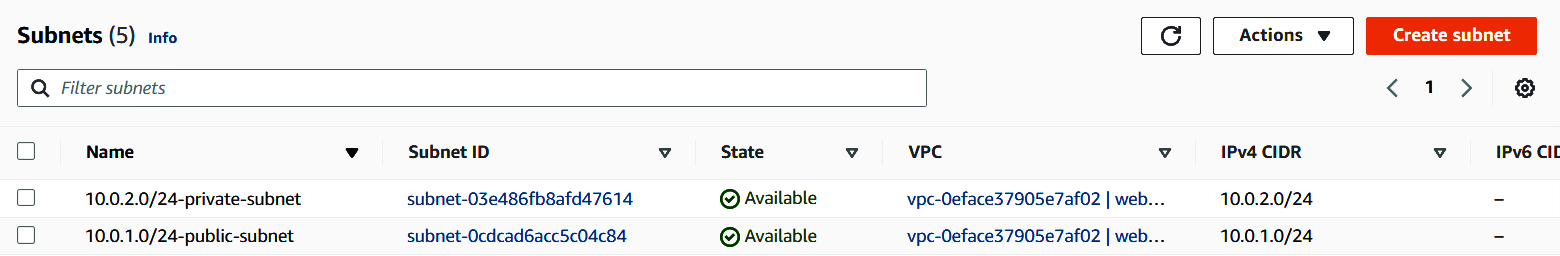
**Task 10: View the website and other resources Terraform created**

Verify the other resources created via the AWS Management console.

1. The custom VPC



b). Public and private subnets

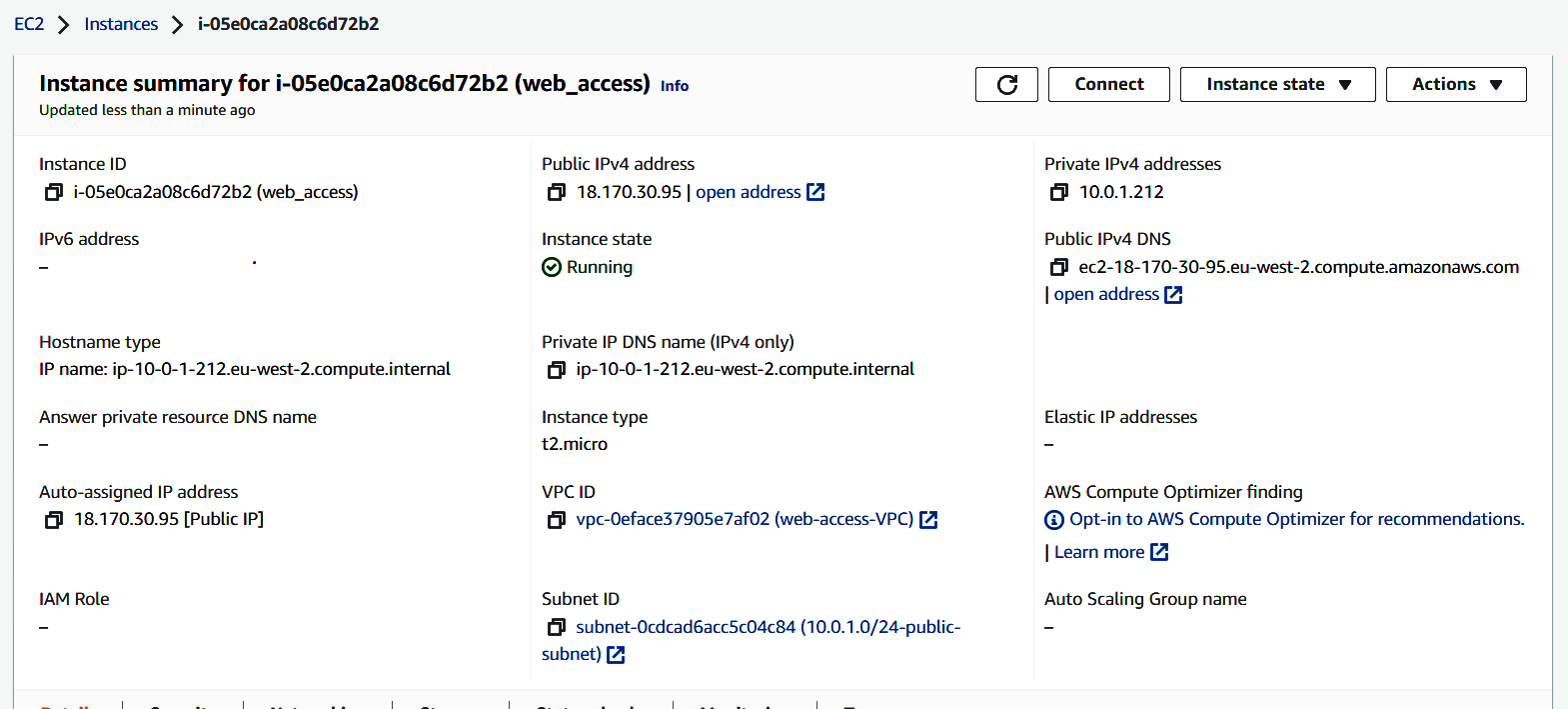


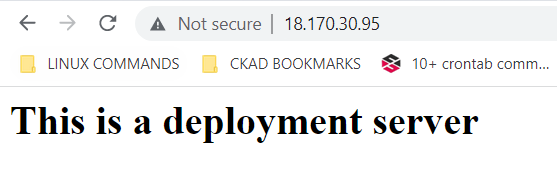
c). Security group

Graphical user interface, text, application, email

Description automatically generated

Copy the displayed url from the terminal on VScode or navigate to the **AWS console 🡪 Services 🡪 EC2 🡪 Instances New.** Select the instance, copy the displayed **Public ipv4 address** and place in your browser to view the configured webpage.



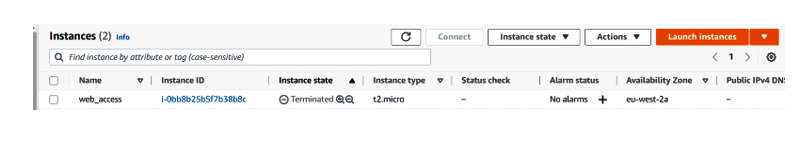


**Task 11: Delete the resources**

On the VScode terminal, run **terraform destroy** to delete the created resources, type yes when prompted. All the created resources are deleted automatically.

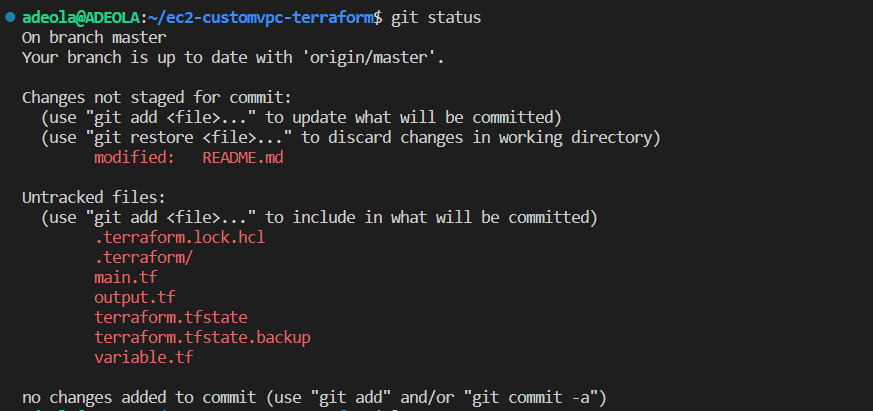
Text

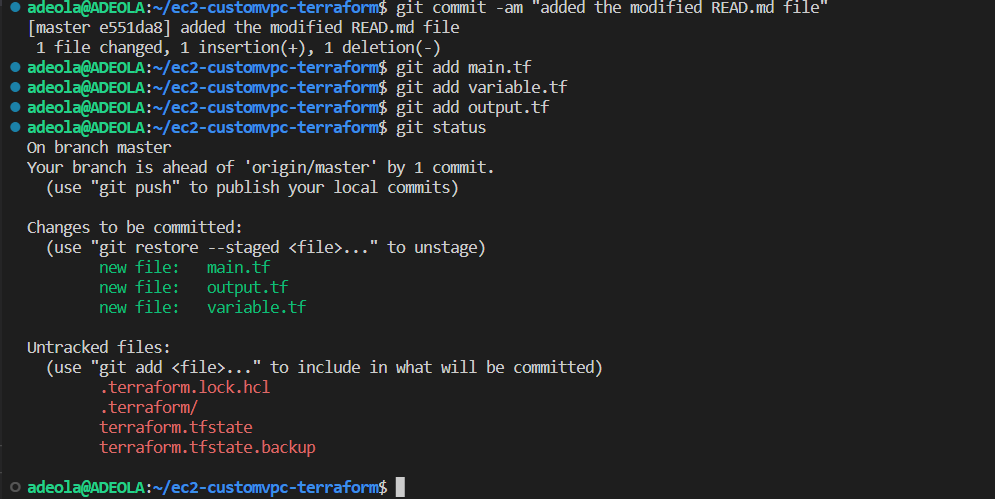
Description automatically generated



**Task 12: Add the configuration files and push to GitHub**

Add the **main.tf, output.tf** and **variable.tf** configurations files to the staging area and commit to the local repository. Then push to the remote repository on GitHub.

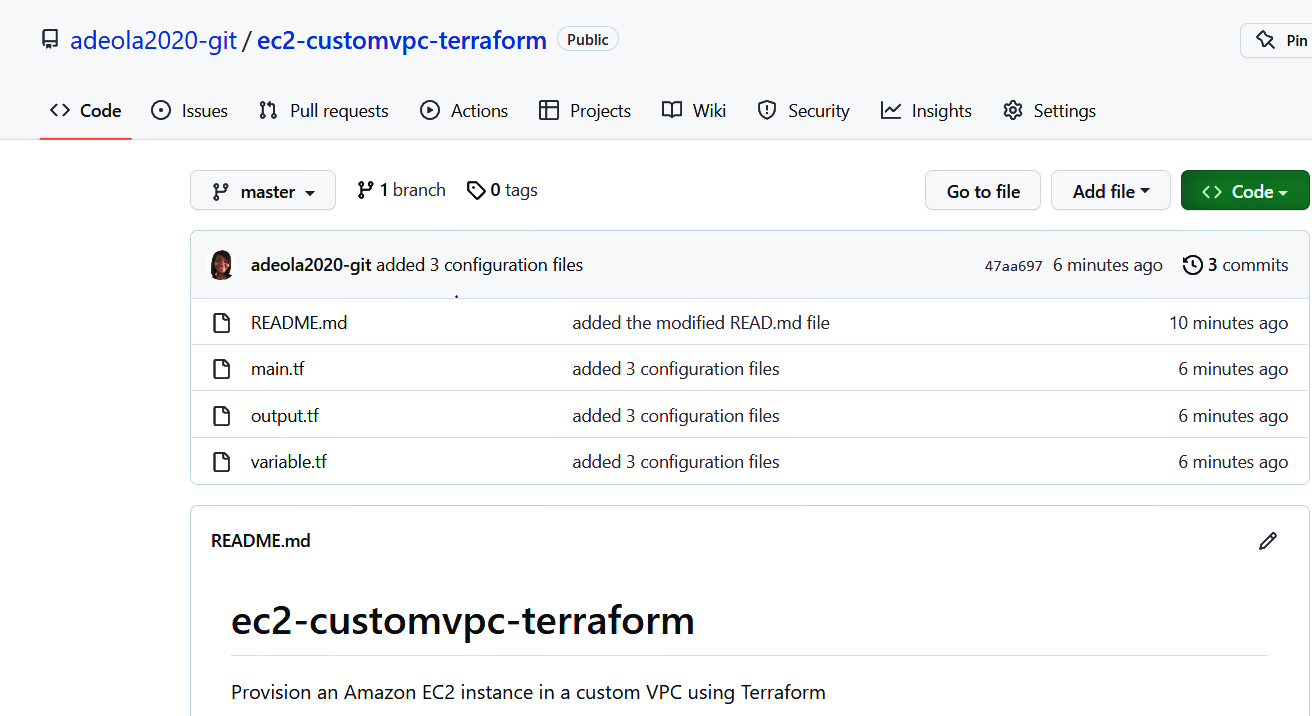




Text

Description automatically generated

Confirm that the configuration files have been added to the remote repository on GitHub.



**Conclusion**

We have been able to provision an Amazon EC2 instance in a custom VPC using Terraform. We also installed an Apache web server on our instance and made it accessible to the public. In the previous project where we provisioned the instance via the AWS console, the deployment process was slow – we spent time manually deploying each resource needed to complete the project. But with Terraform, we were able to automate the deployment of our infrastructure very quickly and easily simply by using codes in our configuration files. These files are version-controlled and reusable. So, we can at any time tear down the infrastructure and re-build it again whenever we make changes to the files.

Thanks for reading!