CI/CD Project: Automating Terraform Deployments with AWS CodeBuild

Overview

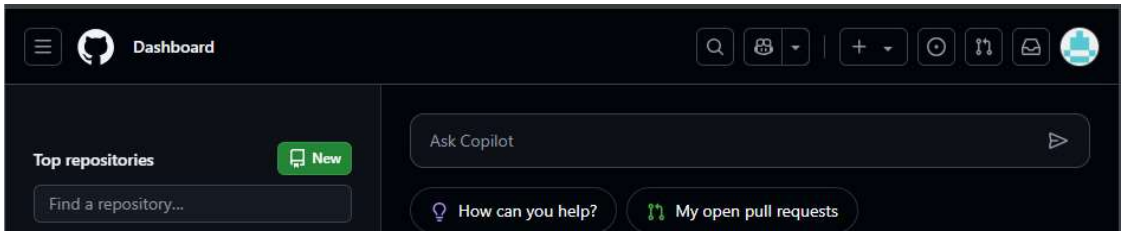
This project sets up a CI/CD pipeline using AWS CodeBuild to automatically apply a Terraform script whenever a change is committed to a GitHub repository. The key steps include:

- Cloning a private GitHub repository using SSH
- Creating an IAM user with programmatic access
- Writing shell scripts and a buildspec.yml file
- Creating a personal access token
- Storing the Terraform state file in an S3 bucket
- Setting up an AWS CodeBuild job

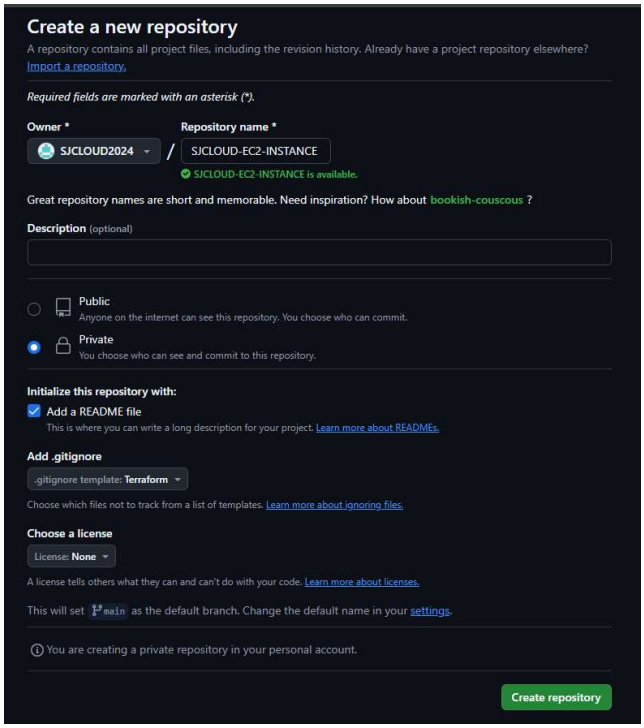
1. Clone a Private GitHub Repository Using SSH

Steps:

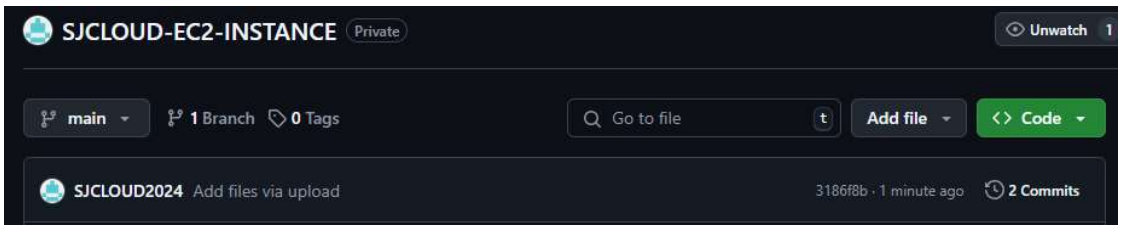
1. Create a GitHub Repository (if not already done):
    - Go to GitHub and create a new private repository.



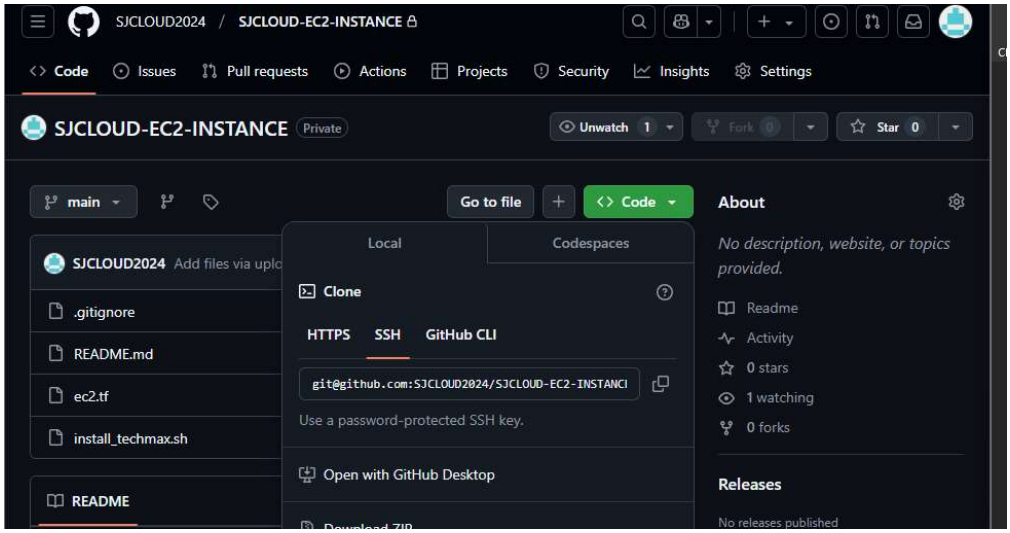
    - Name it (e.g., SJCLOUD-EC2-INSTANCE).
    - Add a README file and select "Terraform" from the .gitignore dropdown.
    - Click "Create Repository."



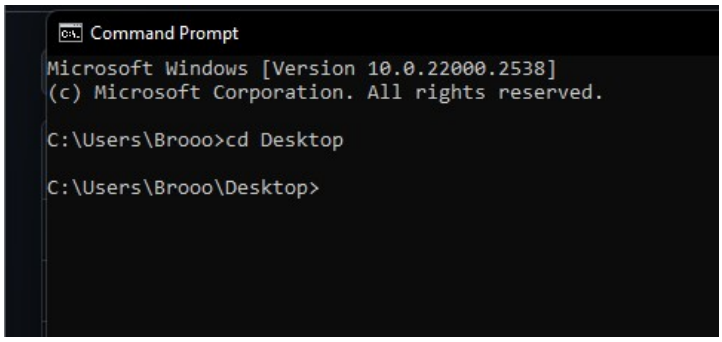
2. Upload Required Files:
    - Click "Add File" > "Upload Files," then drag and drop the necessary files. (Files can be found in my repo)
    - Click "Commit Changes."



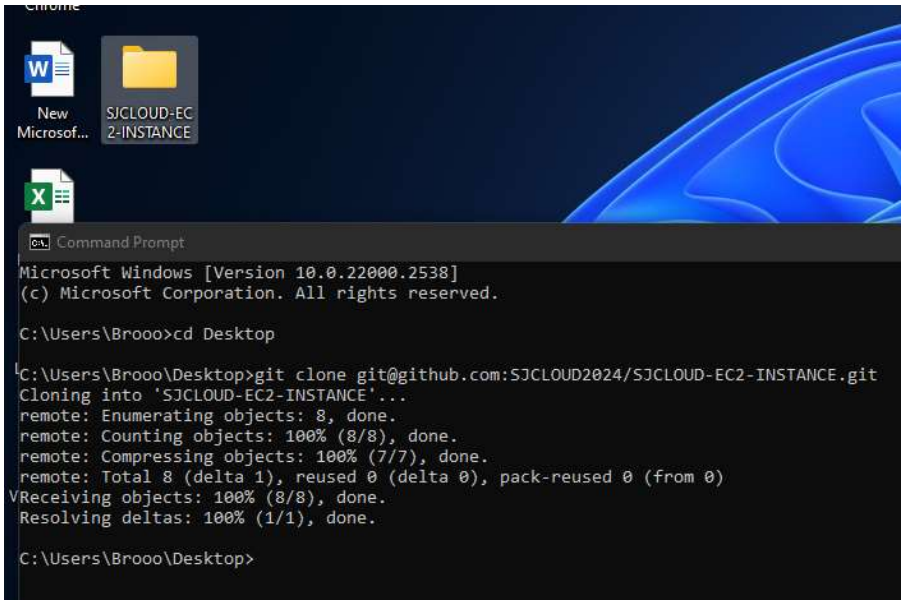
3. Clone the Repository Locally:
    - Copy the SSH URL from the "Code" section (ensure the "SSH" tab is selected).



- Open a terminal and navigate to the desired directory (cd Desktop if cloning to Desktop).

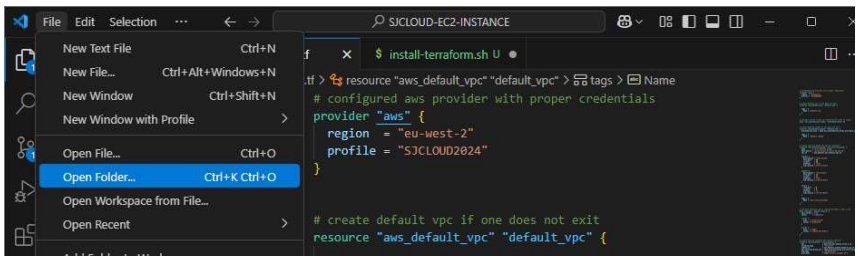


- Run: git clone <SSH-URL>



2. Test Your Terraform Script Locally

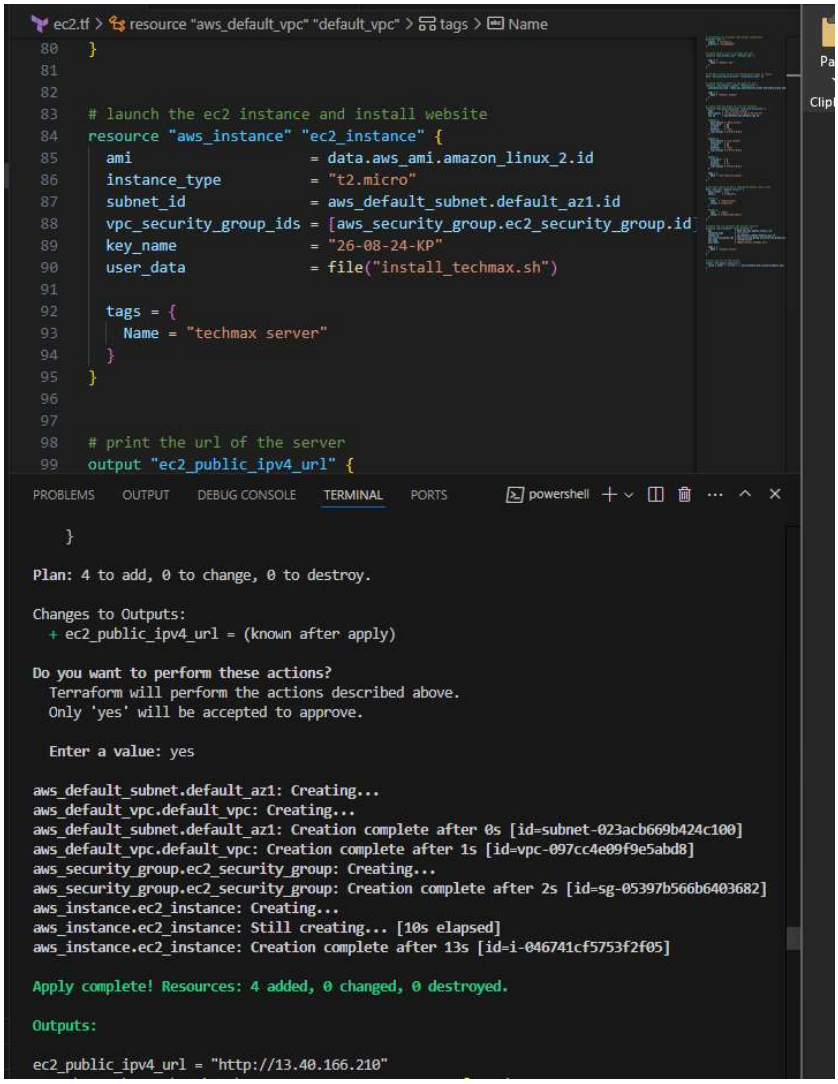
1. Open **Visual Studio Code (VS Code)** and load the Terraform project folder.



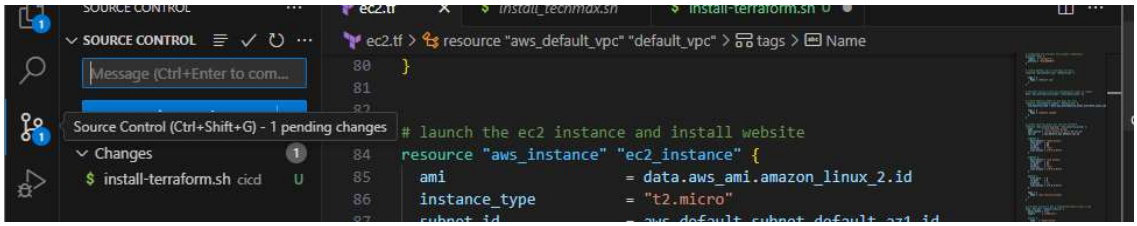
2. Modify the ec2.tf file:

- Update the profile on line 4 and key_name on line 89.

3. Run the following commands in the terminal:
    - Terraform init
    - Terraform plan
    - Terraform apply



4. Verify that the EC2 instance is created and the install_techmax.sh script is executed.
5. Once confirmed, destroy the resources by running the following command:
    - Terraform destroy
6. Finally remember to push the changes to your GitHub repo



3. Create Shell Scripts for CodeBuild

Steps:

1. In VS Code, create a new folder named cicd inside your project.
2. Inside cicd, create three shell scripts:
    - **install-terraform.sh** (Installs Terraform in the CodeBuild container)
    - **configure-named-profile.sh** (Configures an AWS named profile)
    - **apply-terraform.sh** (Runs Terraform commands)

Install-terraform.sh:

cicd > $ install-terraform.sh

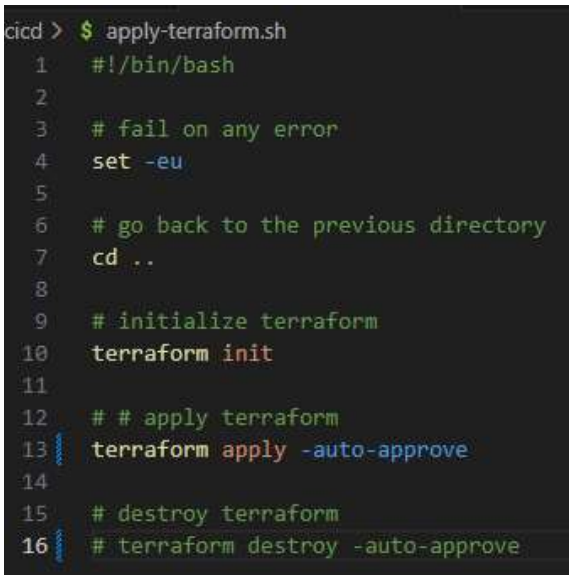
```
#!/bin/bash

# fail on any error
set -eu

# install yum-config-manager to manage your repositories
sudo yum install -y yum-utils

# use yum-config-manager to add the official HashiCorp Linux repository
sudo yum-config-manager --add-repo https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo

# install terraform
sudo yum -y install terraform

# verify terraform is installed
terraform --version
```

Configure-name-profile.sh:

cicd > $ configure-named-profile.sh

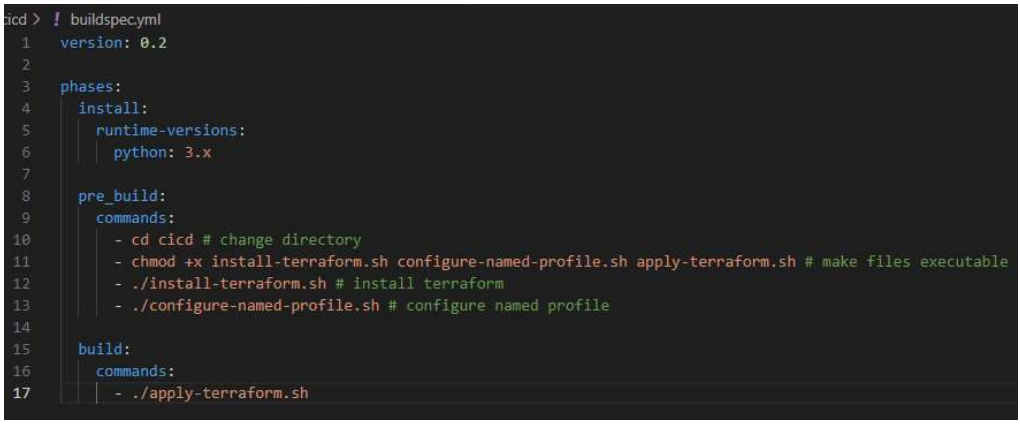
```
#!/bin/bash

# fail on any error
set -eu

# configure named profile
aws configure set aws_access_key_id $AWS_ACCESS_KEY_ID --profile $PROFILE_NAME
aws configure set aws_secret_access_key $AWS_SECRET_ACCESS_KEY --profile $PROFILE_NAME
aws configure set region $AWS_REGION --profile $PROFILE_NAME

# verify that profile is configured
aws configure list --profile $PROFILE_NAME
```

apply-terraform.sh:



4. Create a buildspec.yml File for CodeBuild

1. In the cicd folder, create a file named buildspec.yml
2. Insert the following content:

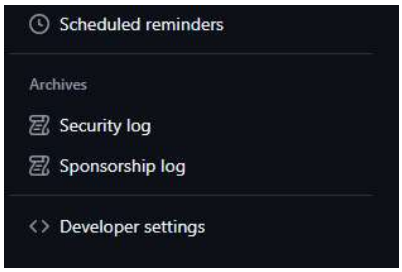


3. Save and push all changes to GitHub.

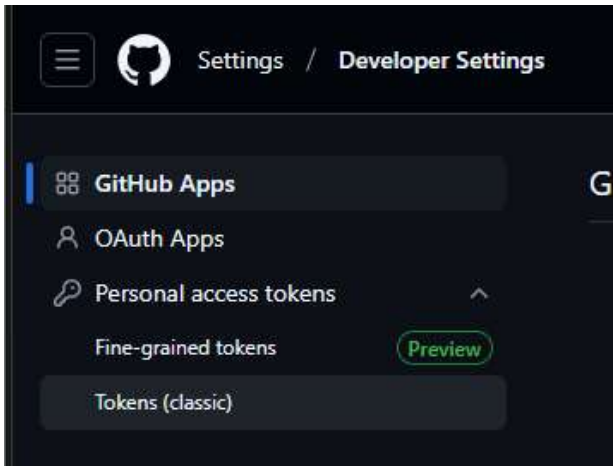
5. Create a Personal Access Token in GitHub

Steps:

1. Navigate to GitHub Settings > Developer Settings.

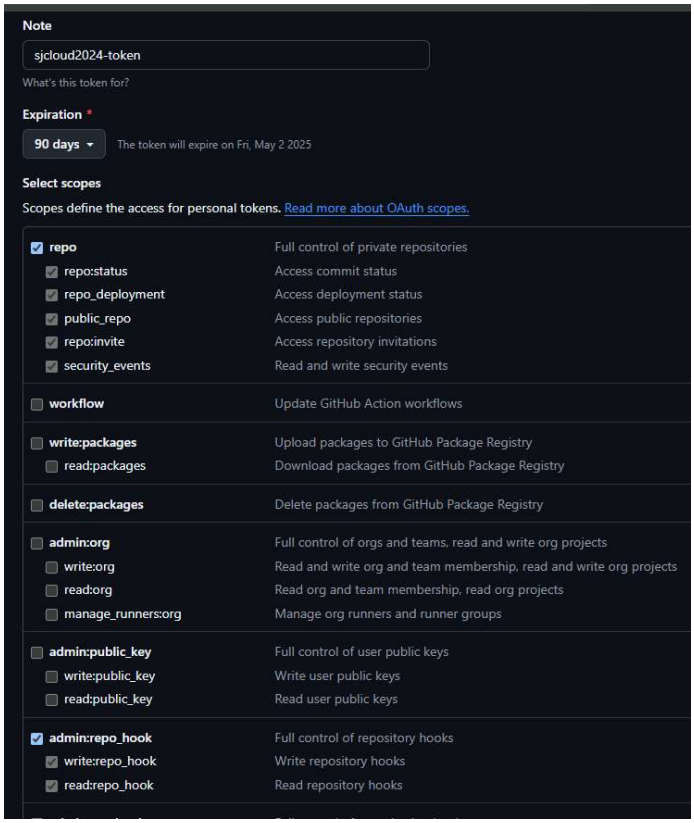


2. Go to Personal Access Tokens > Tokens (Classic).



3. Click Generate New Token (Classic).

4. Set a name (e.g., sjcloud-token) and choose an expiration date.
5. Select **repo** and **admin:repo_hook** scopes.
6. Generate and securely store the token.



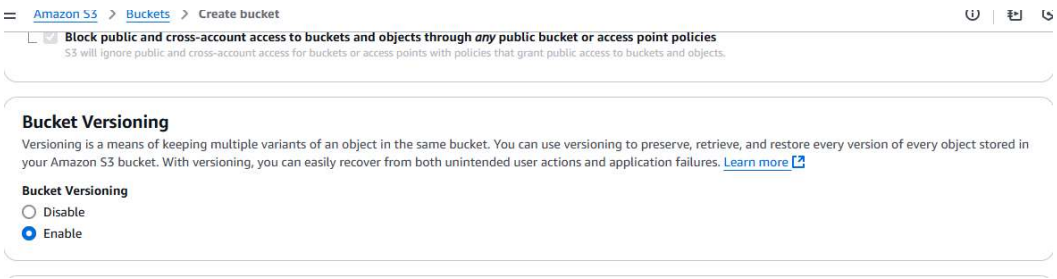
Once the personal access token has been created, make sure to save your token on your computer because you will only be able to see to the code token once.

6. Store Terraform State in an S3 Bucket

Steps:

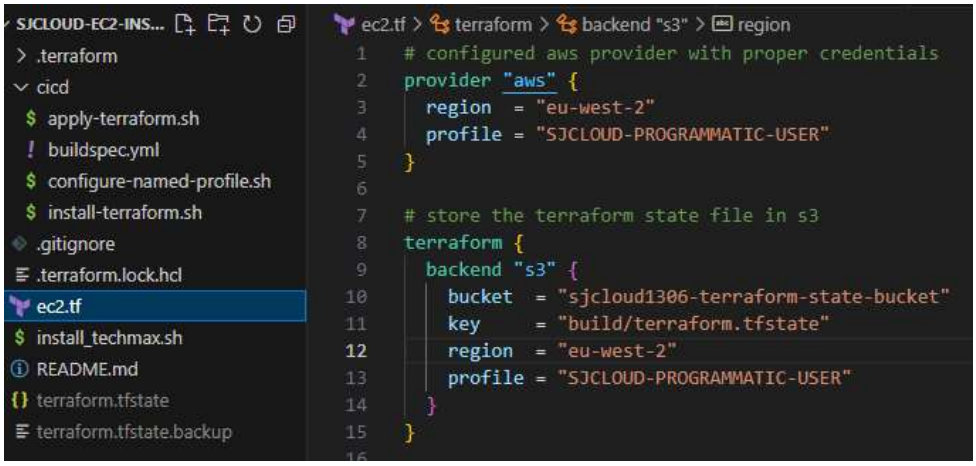
1. In AWS, navigate to S3 and create a bucket:
    - Choose a unique name.

- Enable Versioning.

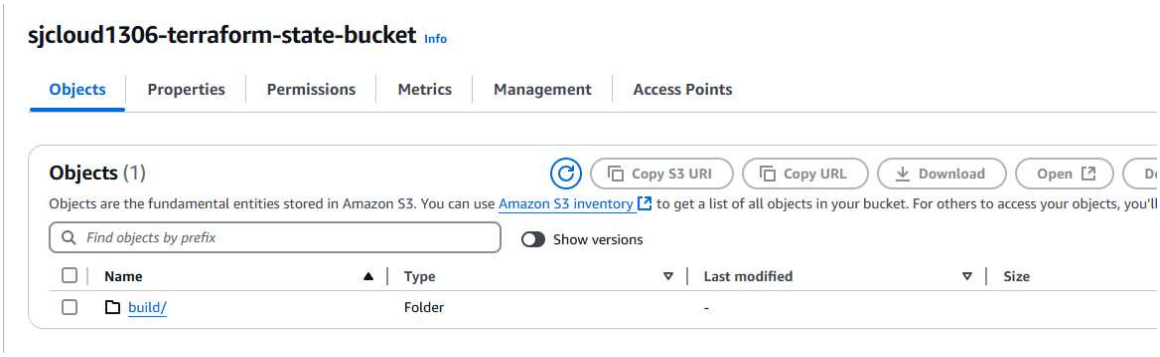


- Leave other settings as default and create the bucket.

2. Modify the Terraform provider file to include lines 7 to 15:



3. Run **terraform init** to configure the backend. VSCode will output a message confirming the successful configuration of the backend S3 bucket which will store the terraform state file. refer back to S3 in the management console. You should be able to see your terraform.tfstate saved in your bucket as below.

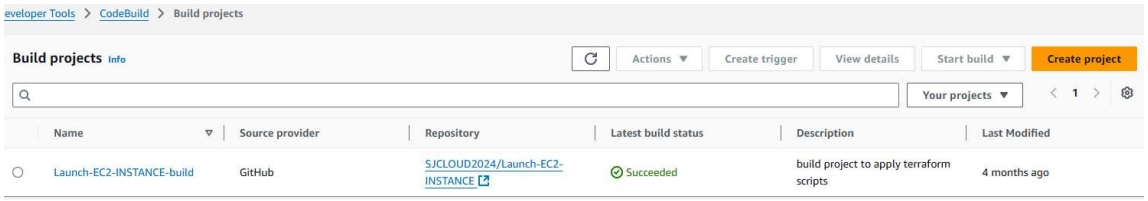


Finally make sure to run **terraform destroy**!

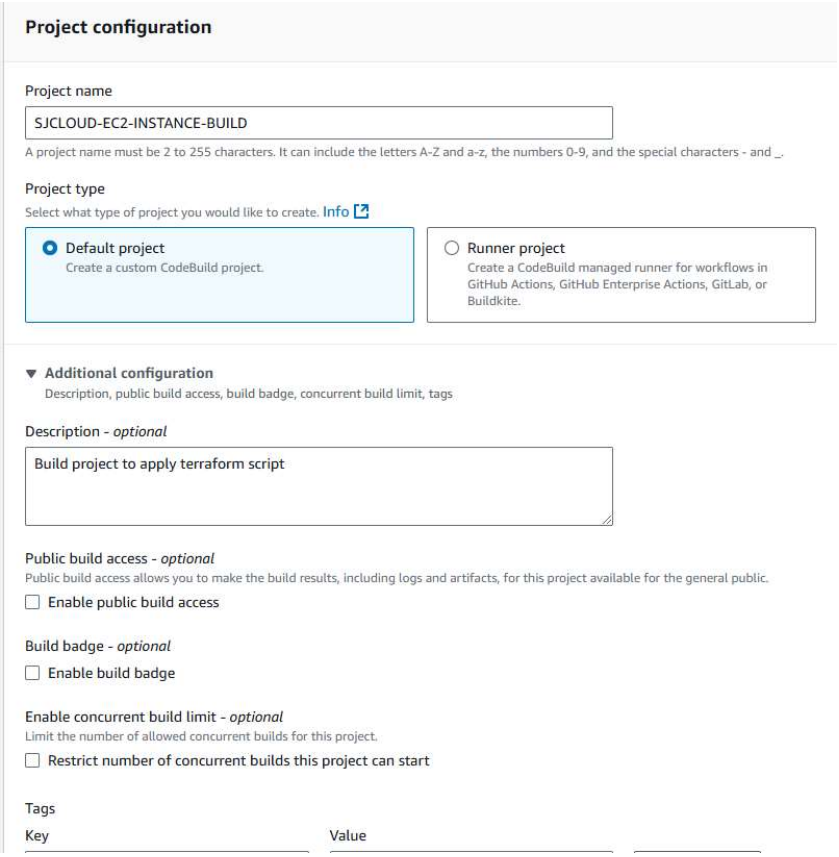
7. Create a Build Project in AWS CodeBuild

Steps:

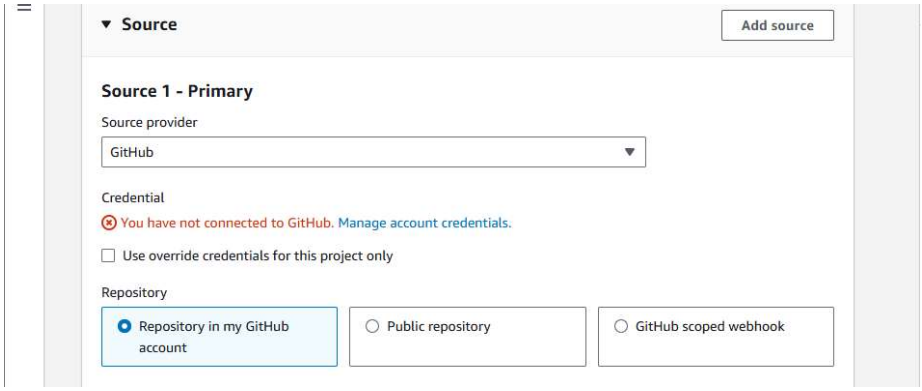
1. Navigate to AWS CodeBuild and click Create Project.

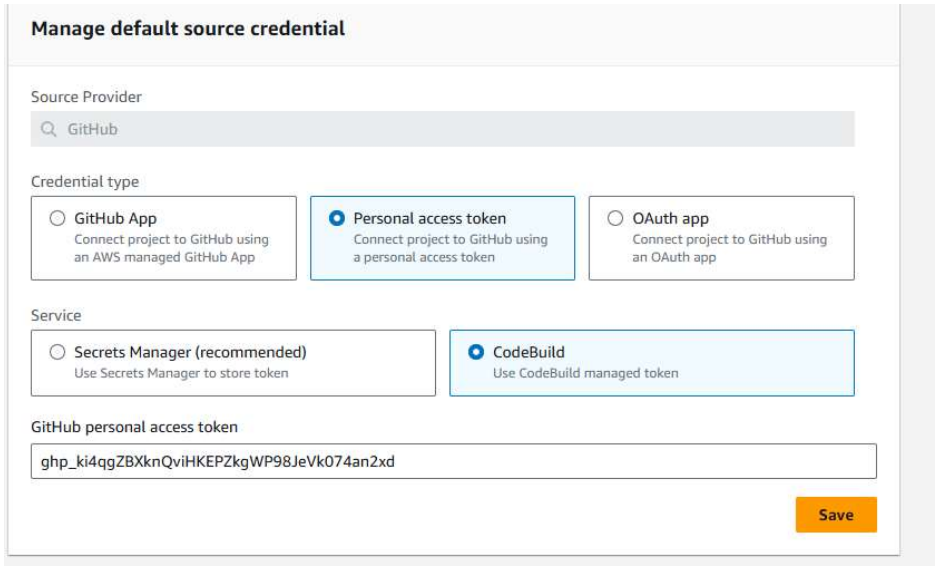


2. Configure:
    - Name: Use the same name as your GitHub repo.

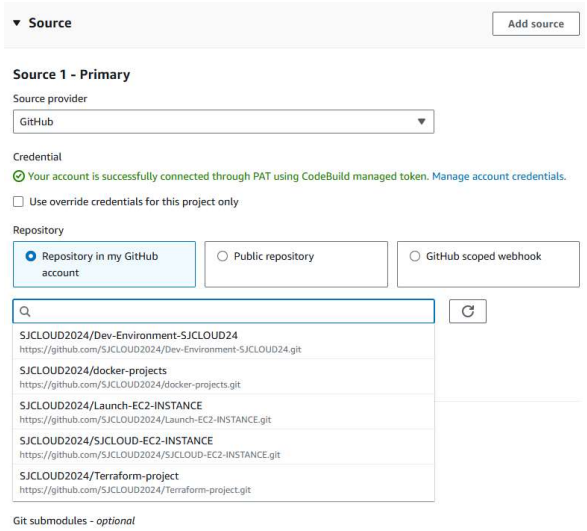


- Source Provider: GitHub. You many get the following error notice. In this case click manage account credentials. Within manage account credentials be sure to select ‘personal access token’ and under ‘service’ select ‘CodeBuild’. The following section will require you to input your personal access token. Please refer to above sections for reference.



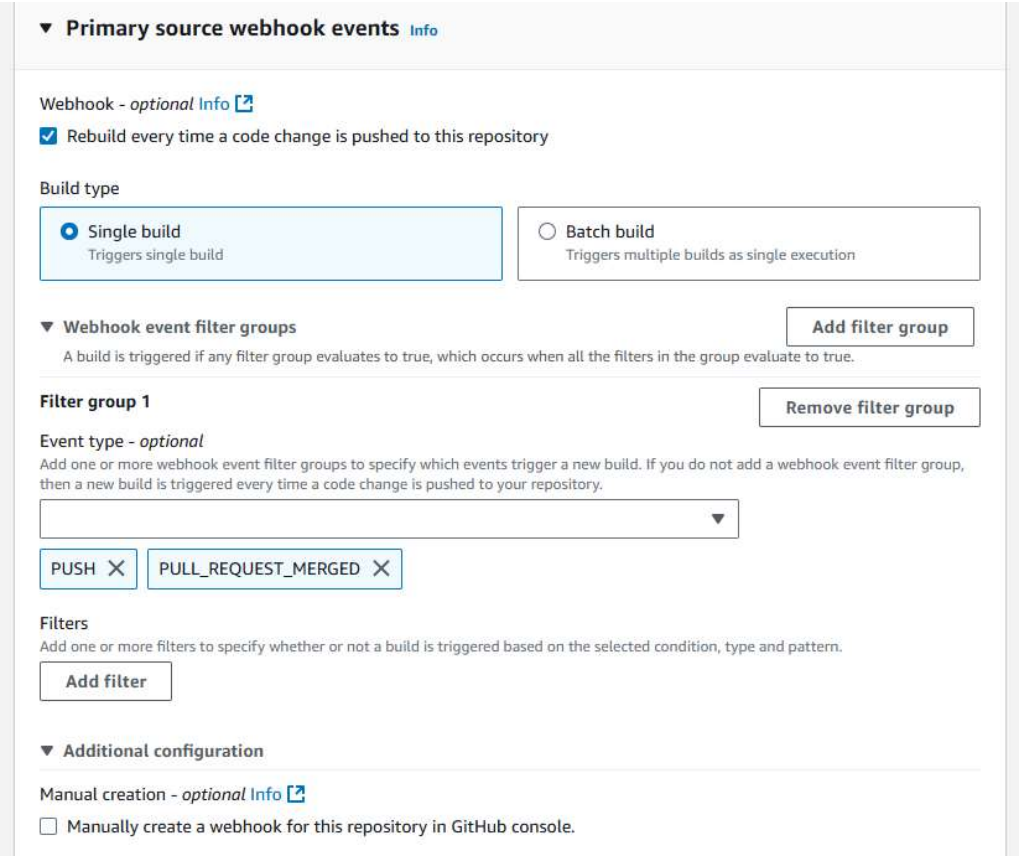


Once the above settings have been saved, you should be taken back to the main build page. Now when you select the ‘repository in my GitHub account’ you should be presented with the below:

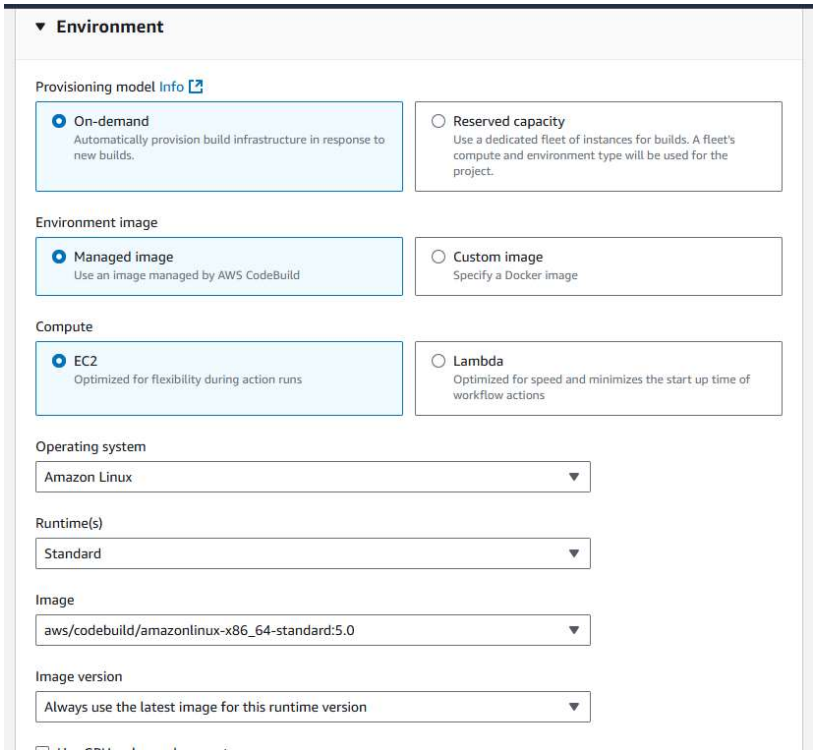


Make sure to select the correct repository from the dropdown list.

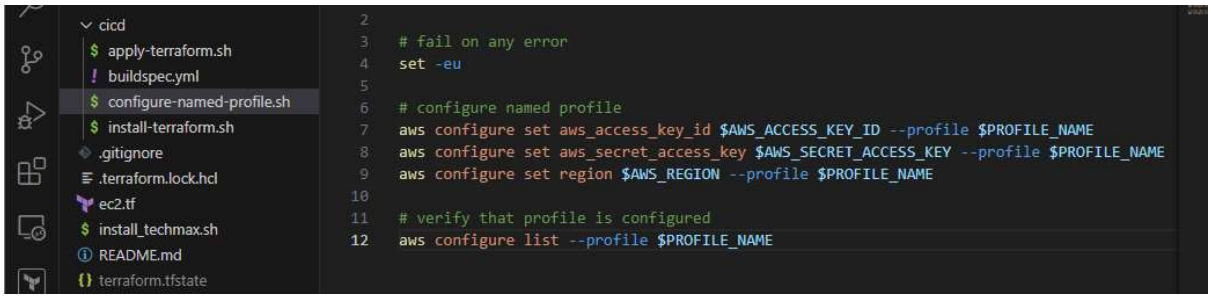
- Webhook Events: Enable rebuild on code changes and below select ‘single build’. As the name suggests this will rebuild our code any time changes are pushed to our GitHub repository. Within the webhook event filter groups select ‘push’ and ‘pull_request_merged’. Selecting ‘PUSH’ and ‘PULL_REQUEST_MERGED’ means any time we push or merge a pull request in our GitHubrepository it will rebuild our project.



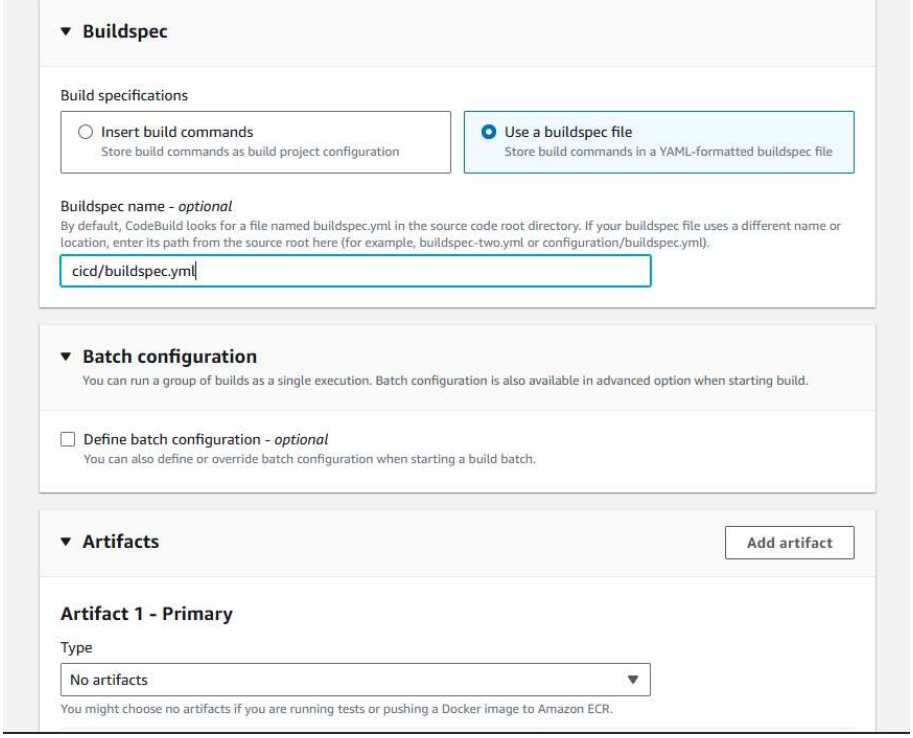
- Environment:
    - Compute: EC2
    - OS: Amazon Linux
    - Runtime: Standard, latest image



    - Environment Variables: Add
        - AWS_ACCESS_KEY_ID - Value (access key from when your programmatic user was created)
        - AWS_SECRET_ACCESS_KEY – Value (secret key from your programmatic user)
        - AWS_REGION – Value (as per the region of your EC2 instance)
        - PROFILE_NAME – Value (IAM user name)

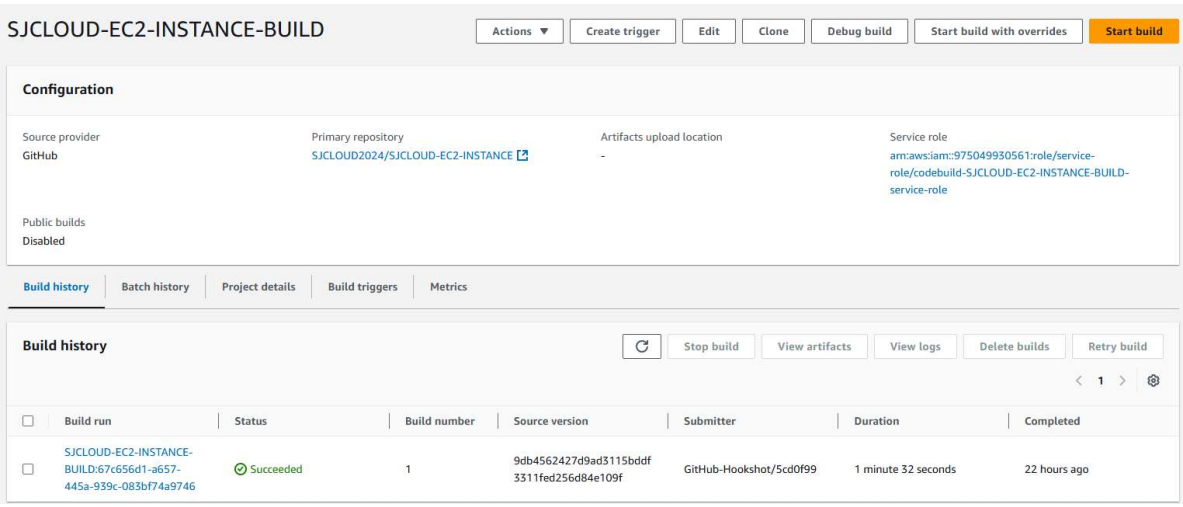


3. Specify the **Buildspec File Location**: **cicd/buildspec.yml**



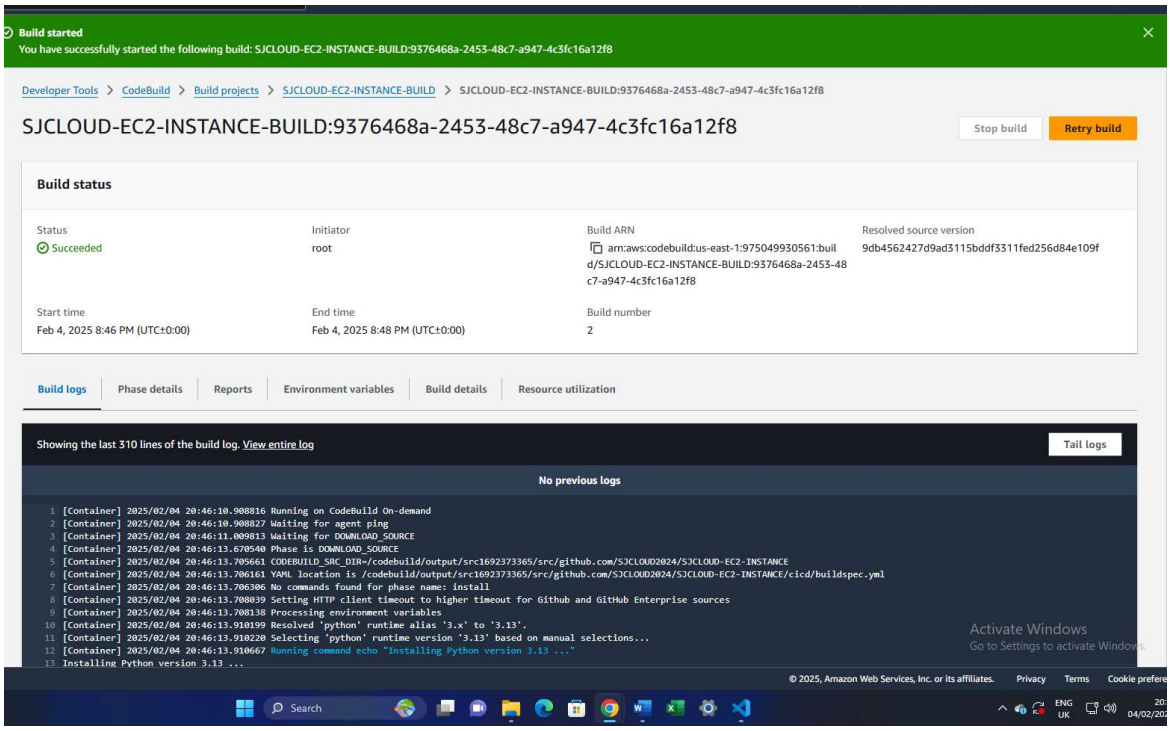
4. Click Create Build Project.

5. Click Start Build to test deployment.

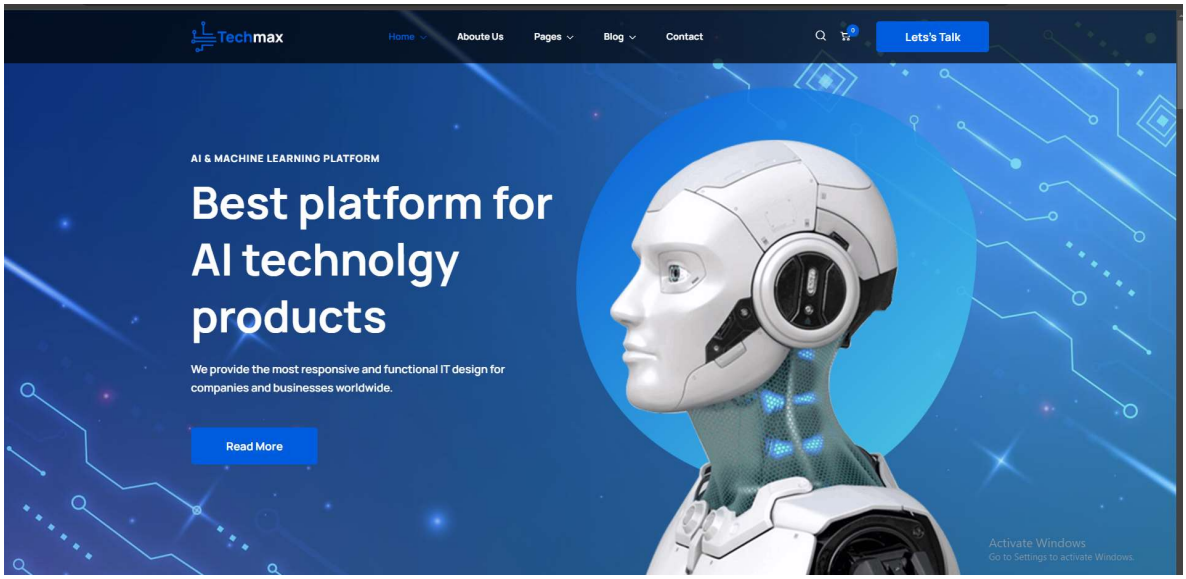


When we click ‘start build’ CodeBuild will clone our repository where our script is located which is in GitHub in our container. Then CodeBuild will use our build spec file to run all the commands we have specified.

After a few minutes your build project should have been build and the status should be ‘succeeded’.



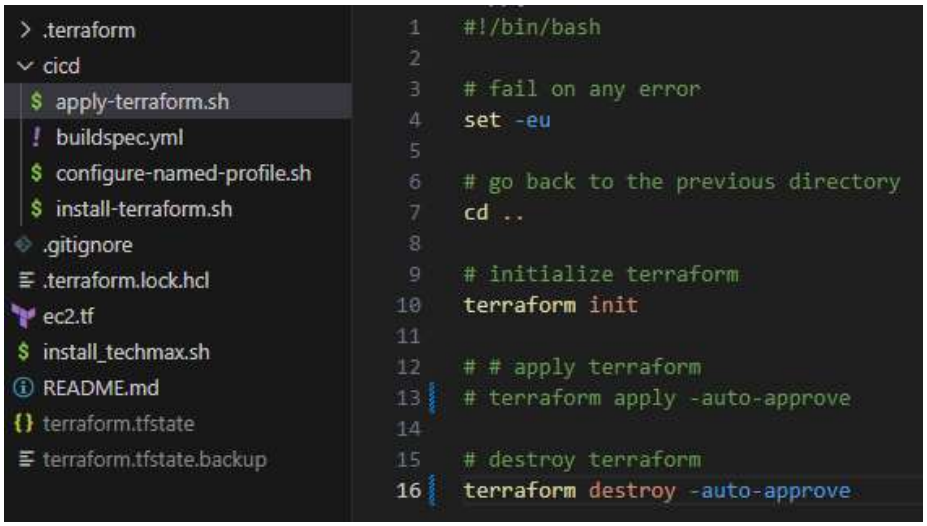
If you go through the build logs you will see what CodeBuild has done from start to finish. At this point you should be able to access your website via the link at the end of the CodeBuild log.



8. Destroy Resources Using CodeBuild

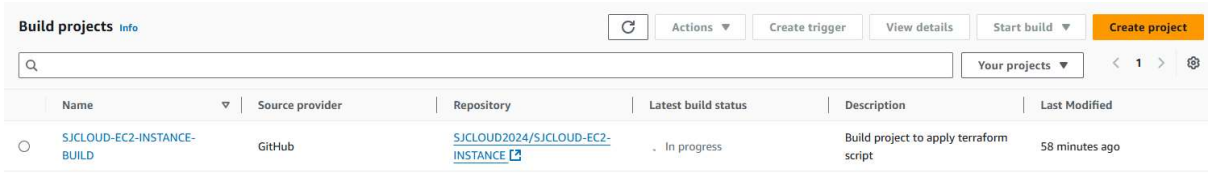
Steps:

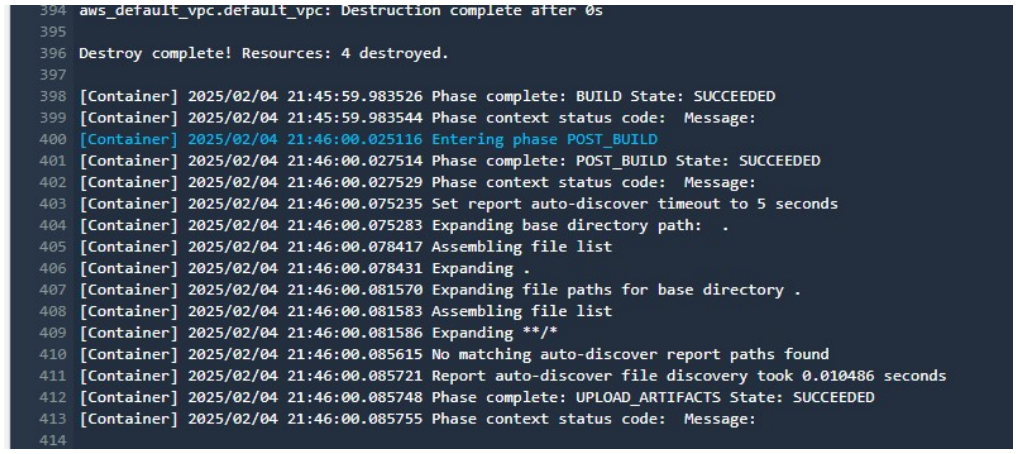
1. In **apply-terraform.sh** file in VSCode, comment out **terraform apply -auto-approve** and uncomment **terraform destroy -auto-approve**.



2. Push changes to GitHub.

3. CodeBuild will automatically run and destroy the resources





Conclusion

This project automates Terraform deployments using AWS CodeBuild, GitHub, and S3 for state management. Each commit triggers a build, ensuring infrastructure changes are consistently applied. The final step allows for automated resource teardown, making the workflow fully CI/CD compliant