# Assignment 1

Objective: In this assignment, you will create a simple GitHub Actions workflow that builds a custom Docker image with a basic NGINX configuration, deploys it to an Amazon EC2 instance, and ensures that the container remains running even after an EC2 instance restart.

Evaluation Criteria:

Your assignment will be evaluated based on the following criteria:

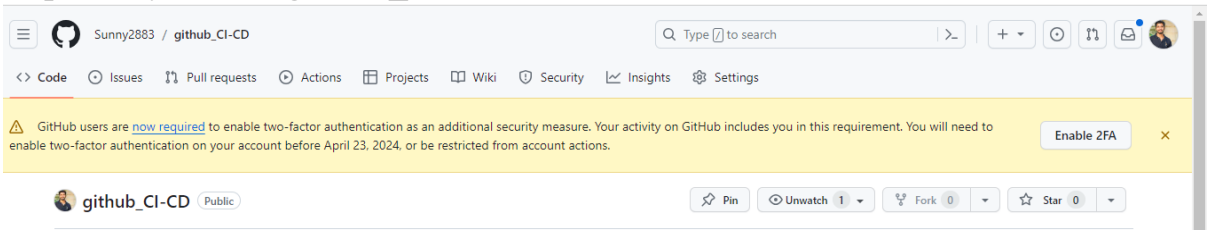
- Successful setup of GitHub Actions workflow. - 10 points
- Secure handling of sensitive information using GitHub Secrets. - 15 points
- Successful building and deploying of the custom Docker image to your EC2 instance. - 20 points
- Proper handling of the Docker container to ensure it remains running after an EC2 instance restart with documentation. -15 points
- Create another branch and push it. Pipeline should not trigger from the any other branch instead of dev. -10 points.

**"Automated Deployment of NGINX with Terraform and GitHub Actions"**

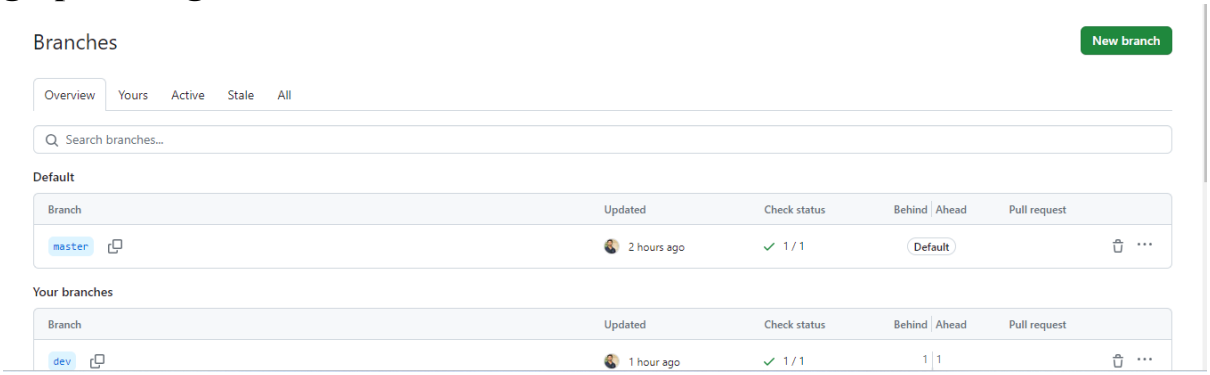
This assignment involves creating a Dockerfile for an NGINX image with basic configuration, utilizing Terraform to provision AWS resources, and integrating Terraform Cloud with GitHub Actions for automated deployment. GitHub Actions workflows are configured to build the NGINX image, deploy it to EC2 instances provisioned by Terraform, ensuring seamless container availability.

Step1: GitHub Repository Setup:

- Create a new GitHub repository for your project.
- Repository Name: **github_CI-CD**



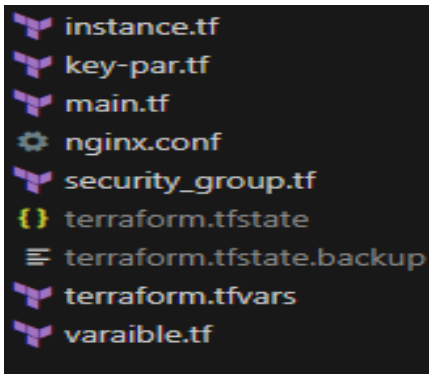

- Create two branches, dev and master, using the following commands:
- **git checkout -b dev**
- **git push origin dev**
- **git checkout -b master**
- **git push origin master**




**** Clone the repository to your local machine. ****

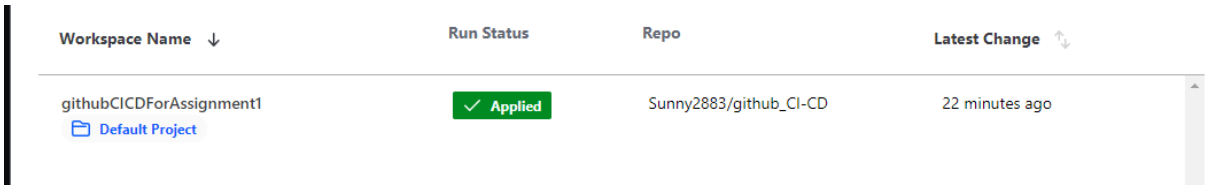
Step 2: Terraform Configuration:

- Write Terraform configuration files (.tf) to define your infrastructure resources, such as EC2 instances, security groups, Variables etc.

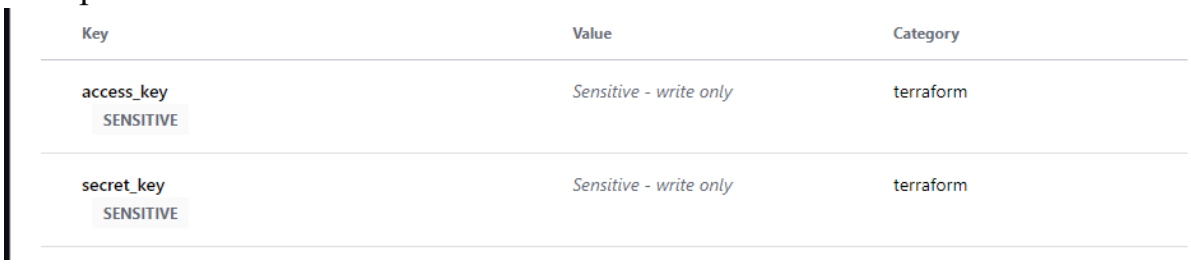


Step 3: Terraform Cloud Setup:

- Sign up for Terraform cloud and Create a new workspace in Terraform Cloud for your project.
- Link your GitHub repository to the Terraform Cloud workspace.
- Workspace name: **githubCICDForAssignment1**
- Repository: **Sunny2883/github_CI-CD**

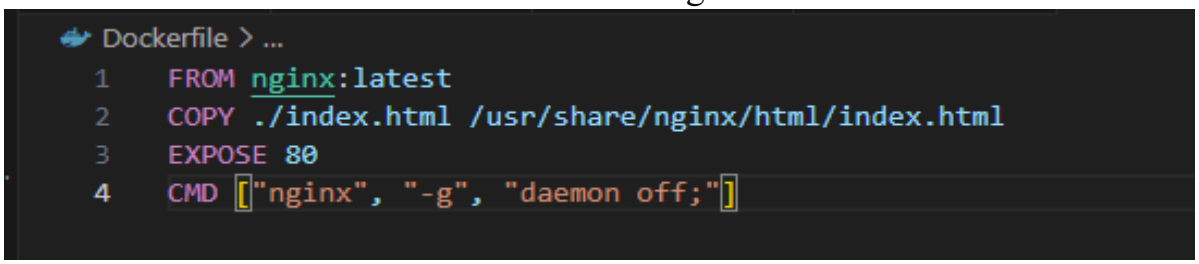



- Set up environment variables or sensitive values in Terraform Cloud.

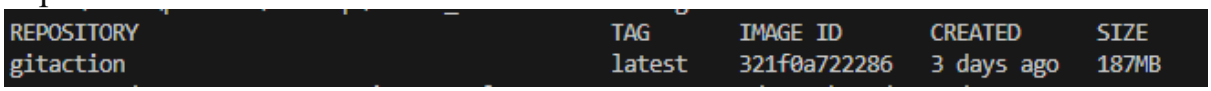



Step 4: Docker Image Creation:

- Write a Dockerfile with a basic NGINX configuration.

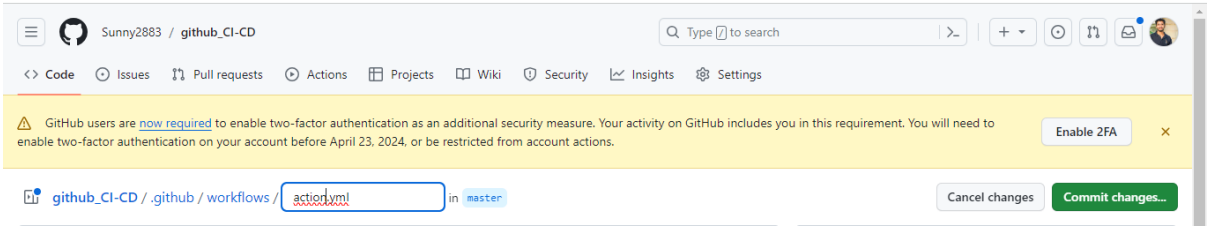



- Build the Docker image locally and test it to ensure it functions as expected.

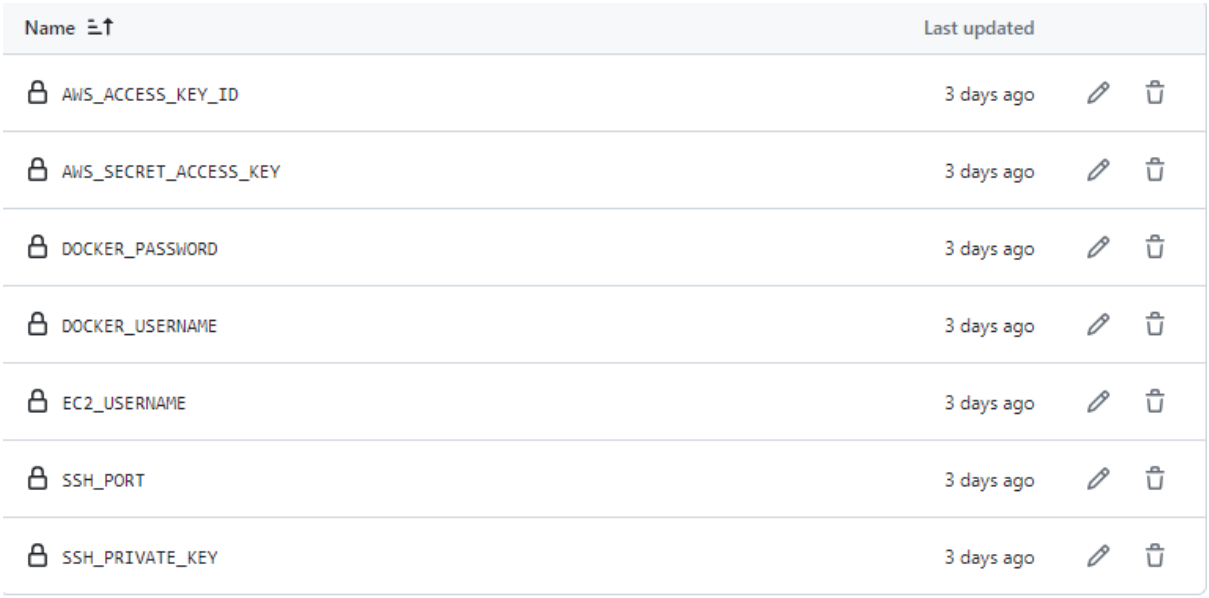


Step 5: GitHub Actions Workflow Setup:

- Create a .github/workflows directory in your project repository and Inside this directory, create a YAML file (action.yml) to define your GitHub Actions workflow.




- Securely handle sensitive information using GitHub Secrets and Terraform Cloud variables.



Step 6: Define the workflow steps to build the Docker image, push it to a Docker registry (e.g., Docker Hub ), deploy it to your EC2 instance, and apply Terraform configuration using Terraform Cloud.

.github > workflows > action.yml

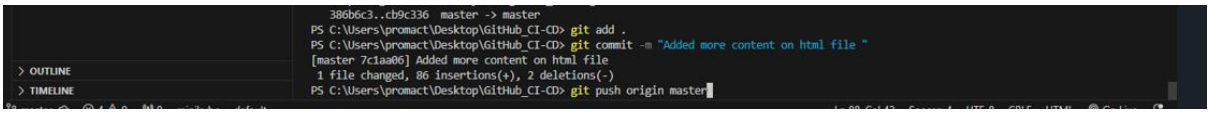
```yaml
name: Terraform EC2 Deployment

on:
  push:
    branches: ["dev"]
  pull_request:
    branches: ["dev"]
  workflow_dispatch:

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v3



      - name: Build and push Docker image
        env:
          DOCKER_USERNAME: ${{ secrets.DOCKER_USERNAME }}
          DOCKER_PASSWORD: ${{ secrets.DOCKER_PASSWORD }}
        run: |

          docker build -t mynginx .

          docker login -u $DOCKER_USERNAME -p $DOCKER_PASSWORD

          docker tag mynginx $DOCKER_USERNAME/mynginx:latest

          docker push $DOCKER_USERNAME/mynginx:latest
```

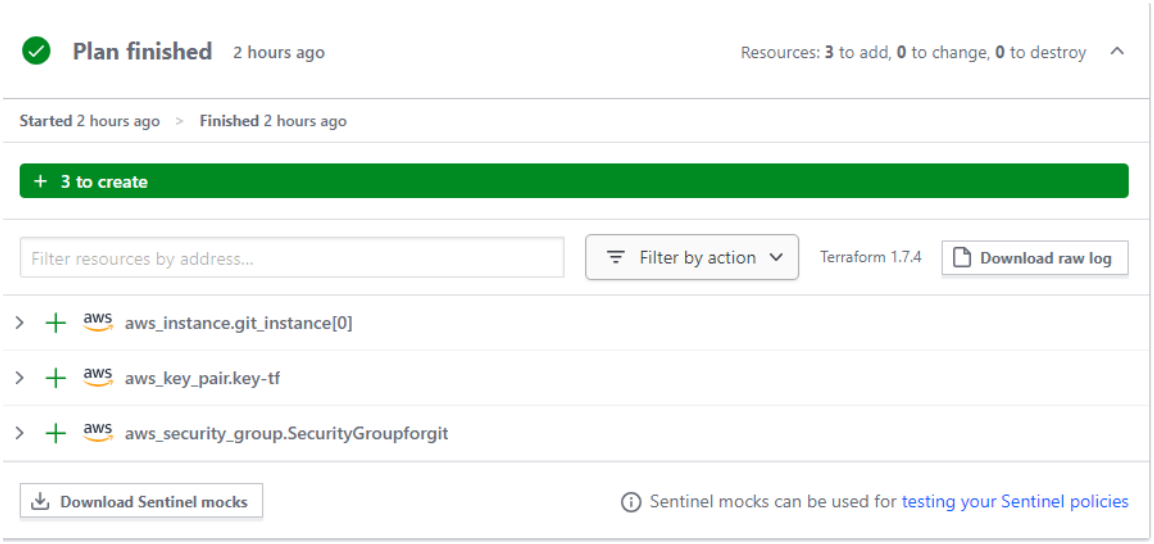
.github > workflows > action.yml

```yaml
jobs:
  deploy:
    steps:
      - name: Build and push Docker image
        run: |
          docker tag mynginx $DOCKER_USERNAME/mynginx:latest

          docker push $DOCKER_USERNAME/mynginx:latest

      - name: AWS login
        uses: aws-actions/configure-aws-credentials@v4
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: us-east-2

      - name: SSH into EC2 instance and deploy container
        uses: appleboy/ssh-action@master
        with:
          host: "44.212.58.158"
          username: ${{ secrets.EC2_USERNAME }}
          key: ${{ secrets.SSH_PRIVATE_KEY }}
          port: ${{ secrets.SSH_PORT }}
          script: |
            sudo docker pull "${{ secrets.DOCKER_USERNAME }}/mynginx:latest"
            sudo docker run -d --restart=always -p 80:80 "${{ secrets.DOCKER_USERNAME }}/mynginx:latest"
```

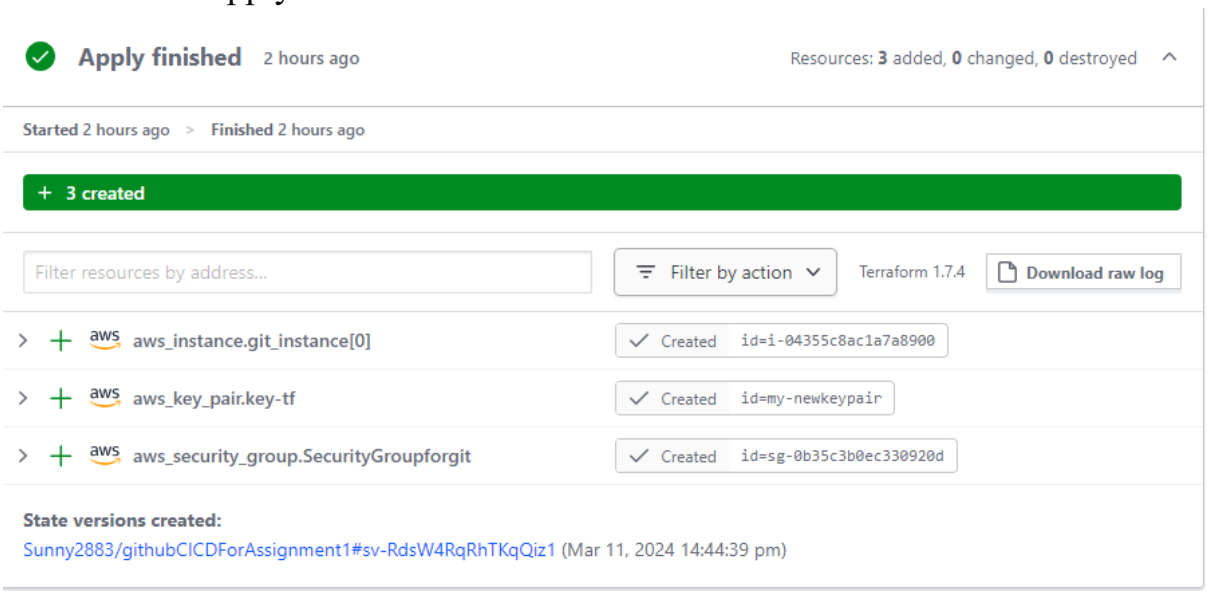
Step 7: Deployment and Testing:

- Trigger the command git push origin master and Verify the Changes on terraform cloud.




- Confirm and Apply:

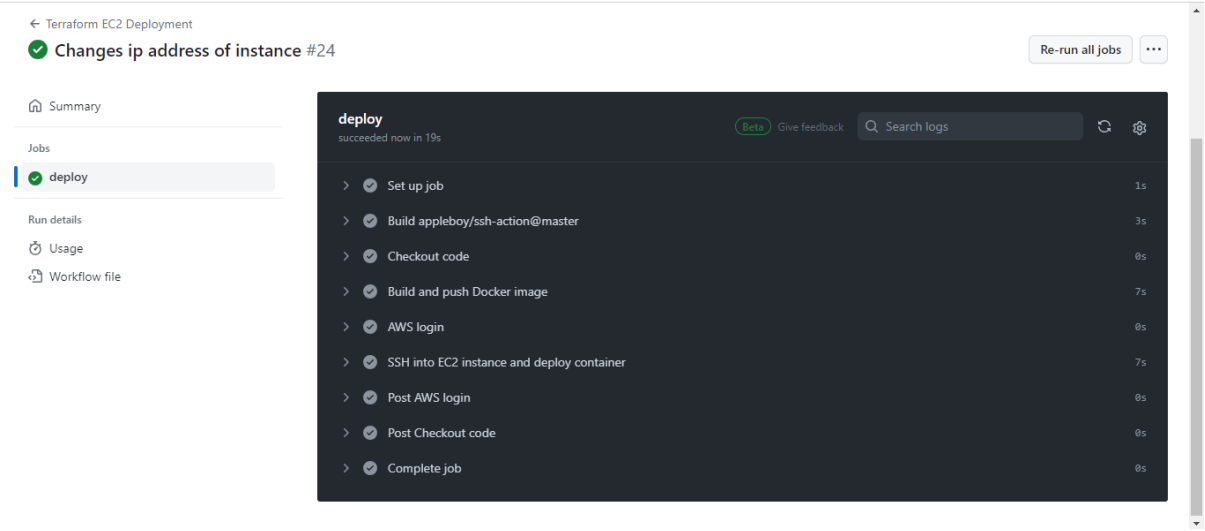


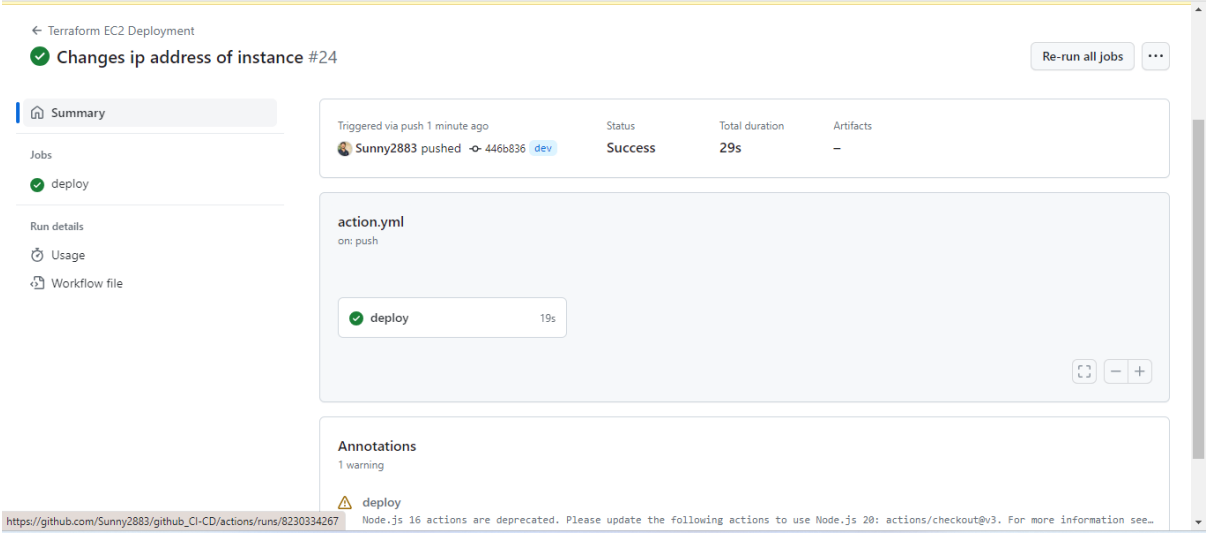

Resources Created on AWS:

- Instance name: gitactionInstance
- Instance ip: 44.202.92.11 (Changes Everytime)
- Instance id: i-04355c8ac1a7a8900
- Instance type: t2.micro
- Key-pair: my-newkeypair

- Security Group: SecurityGroupgit
- Security group id: sg-0b35c3b0ec330920d

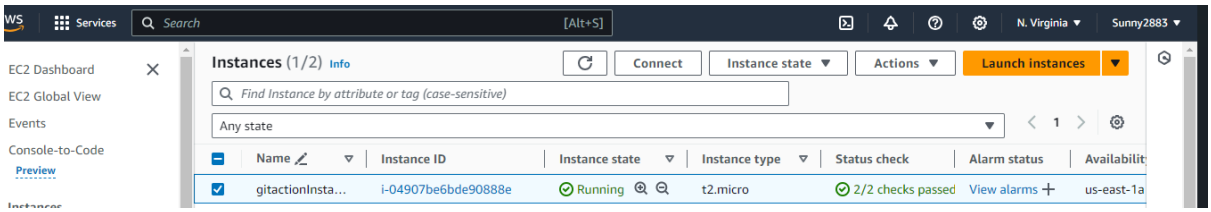
Step 8: Test the GitHub Actions workflow by pushing changes to the **dev** branch and observing the deployment process.



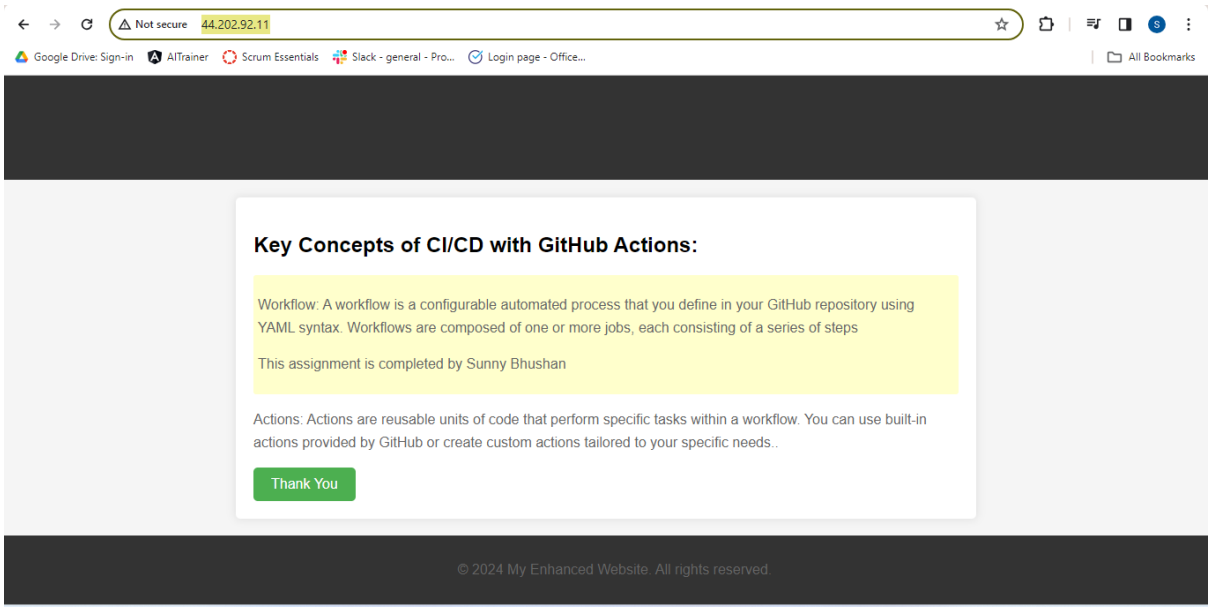
Step 9: Veirfy The Application on Port http://44.202.92.11/

**** I did not use Elastic ip address so everytime ports changes ****




## Final Output:

Home Page:




# Thank You