BMS INSTITUTE OF TECHNOLOGY AND MANAGEMENT

Autonomous Institute under VTU, Belagavi, Karnataka - 590 018

Yelahanka, Bengaluru, Karnataka - 560 119

Devops (BCS601)

CCA Report

On

“Implementing CI/CD pi[peline for Git Usage of Mini Project”

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

by

Shreyas CG 1BY22CS170

Under the Guidance of

Dr. Ravi Hosur

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Avalahalli, Yelahanka, Bengaluru, Karnataka -560119

May 2025

Evaluation sheet

Name	USN	Environment Setup)	Application Implementation	Documentation	Understanding and Analysis	Innovation/Extra Work	Report	Total
		3	2	5	2	3	5	20
Shreyas CG	1BY22CS170							

	Name	Signature with date
Course Coordinator	Dr Ravi Hosur	

TABLE OF CONTENTS

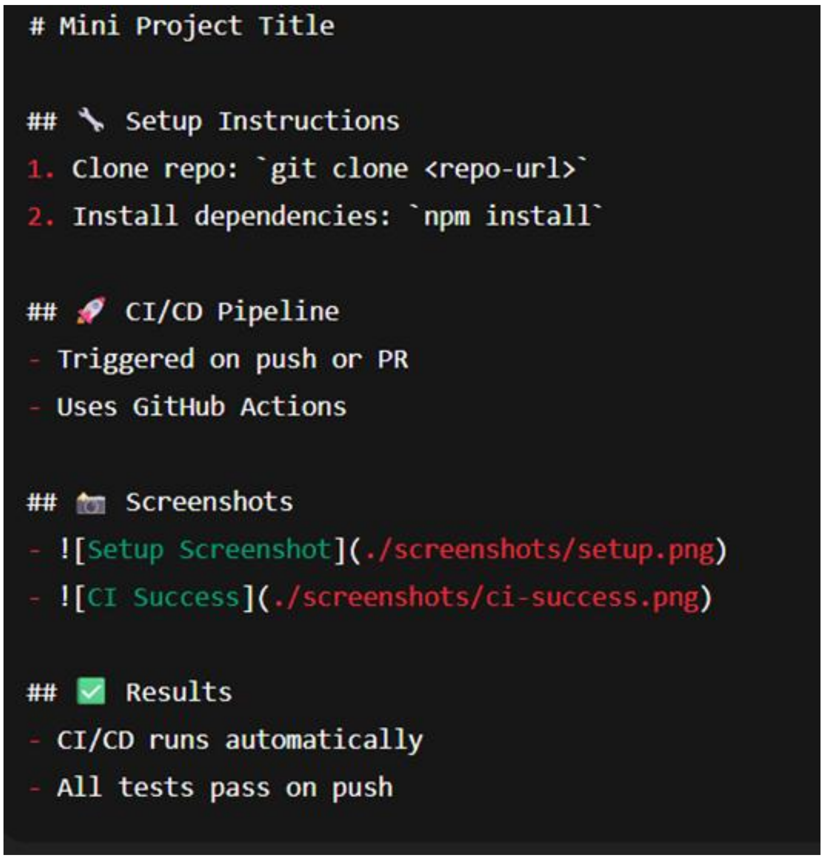
1. Environment Setup and Git Usage

- Objective:
    1. To set up the development environment and demonstrate the use of Git for version control including initialization, commits, branches, and pushing code to a remote repository.
- Tools Used:
    1. Git
    2. GitHub
    3. VS Code / Terminal / Git Bash
- Steps & Commands:
    1. Install Git (if not already installed):
        - sudo apt update
        - sudo apt install git -y
    2. Configure Git (Global Settings):
        - git config --global user.name "Your Name"
        - git config --global user.email "your.email@example.com"
    3. Create Project Folder & Initialize Git:\
        - mkdir mini-project
        - cd mini-project
        - git init
    4. Create a Sample File and Commit:
        - echo "# Mini Project" > README.md
        - git add README.md
        - git commit -m "Initial commit with README"
    5. Create a New Branch:
        - git checkout -b dev
    6. Push Project to Github:
        - git remote add origin https://github.com/your-username/mini-project.git
        - git branch -M main
        - git push -u origin main

2. CI/CD Pipeline implementation

- Objective:
    1. To create a functional CI/CD pipeline that builds and tests the project automatically upon code changes using GitHub Actions.
- Tools Used:
    1. - GitHub Actions
    2. - YAML
    3. - Node.js (for example project)
- Steps to Implement:
    1. Create Workflow File inside your repository:
    2. Add CI Configuration (Sample for Node.js):
        ```
        name: CI Pipeline
        on: [push, pull_request]
        jobs:
          build:
            runs-on: ubuntu-latest
            steps:
              - uses: actions/checkout@v2
              - name: Setup Node.js
                uses: actions/setup-node@v3
                with:
                  node-version: '16'
              - name: Install dependencies
                run: npm install
              - name: Run Tests
                run: npm test
        ```
    3. Commit and Push the Workflow:
        ```
        git add .github/workflows/ci.yml
        git commit -m "Add CI pipeline using GitHub Actions"
        git push
        ```
    4. Observe CI in GitHub:
        - - Go to Actions tab in the repo to see the pipeline in action.
        - - Verify successful runs on each push or PR.

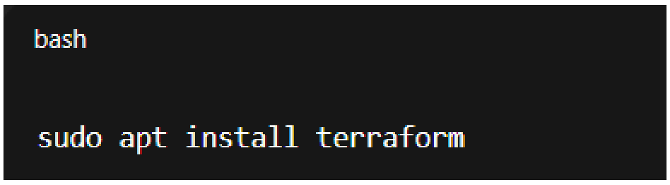
3. Documentation and Presentation

- Objective:
    - To provide a well-structured and clearly explained documentation of the project implementation process, supported by relevant screenshots, diagrams, and logs. The aim is to ensure the project is understandable, reproducible, and presentable for academic or professional review.
- Requirements:
    - Clear, step-by-step explanation
    - Logs/Screenshots
    - Well-formatted README.md
- README.md structure:
    - 

```
# Mini Project Title

## 🔧 Setup Instructions
1. Clone repo: `git clone <repo-url>`
2. Install dependencies: `npm install`

## 🚀 CI/CD Pipeline
- Triggered on push or PR
- Uses GitHub Actions

## 📷 Screenshots
- ![Setup Screenshot](./screenshots/setup.png)
- ![CI Success](./screenshots/ci-success.png)

## ✅ Results
- CI/CD runs automatically
- All tests pass on push
```

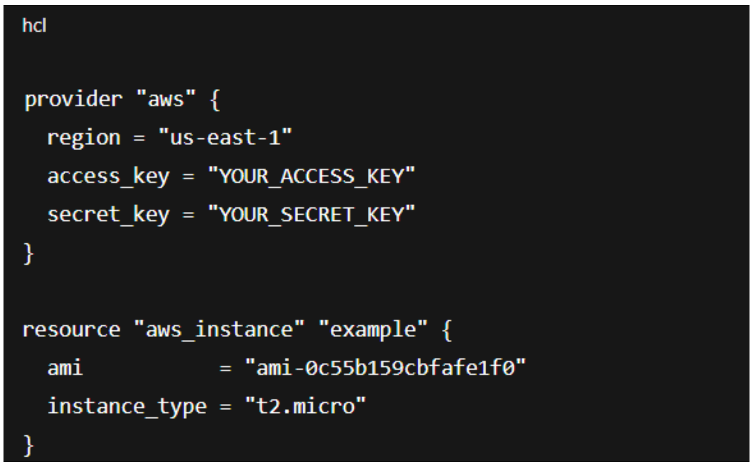
4. Infrastructure as Code (IaC)

- Tools:
    - Terraform / Ansible
- Example: Using Terraform to provision EC2 on AWS
    - Install Terraform



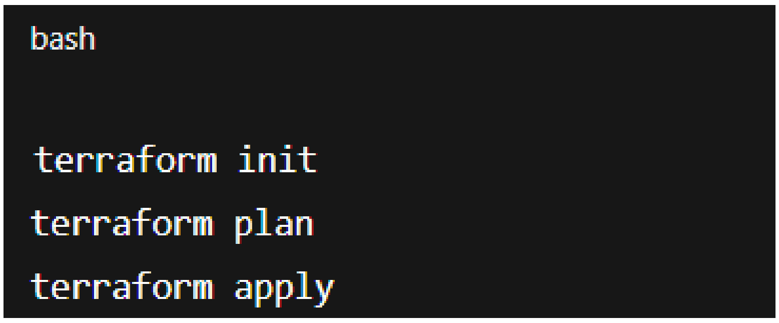
```bash
sudo apt install terraform
```

    - Create main.tf



```hcl
provider "aws" {
  region = "us-east-1"
  access_key = "YOUR_ACCESS_KEY"
  secret_key = "YOUR_SECRET_KEY"
}

resource "aws_instance" "example" {
  ami           = "ami-0c55b159cbfafe1f0"
  instance_type = "t2.micro"
}
```

    - Run Commands:



```bash
terraform init
terraform plan
terraform apply
```

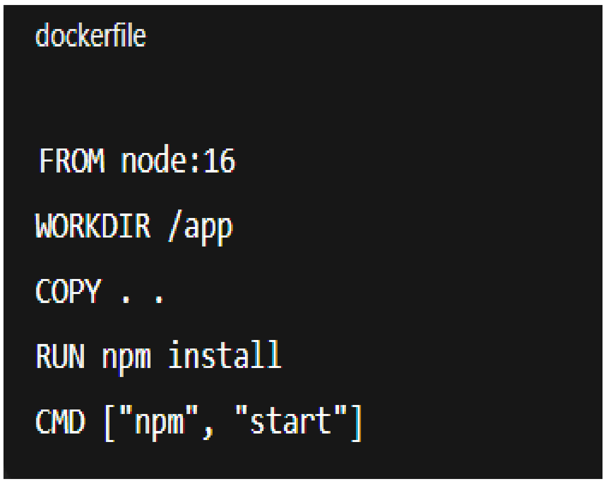
5. Innovation Extra Work or Capstone

Demonstrated by integrating:

- Git for version control
- GitHub Actions for CI/CD
- Docker to containerize the app
- Terraform for provisioning infrastructure
- Deploy a sample app to cloud (e.g., Heroku/AWS)

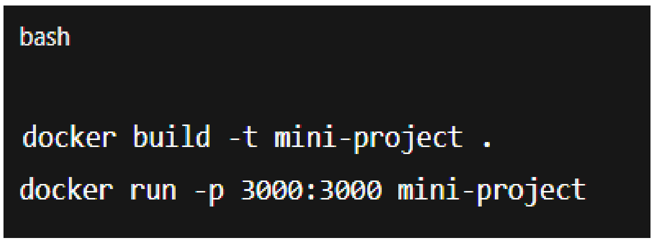
Example: Docker + GitHub Actions + Terraform

1. Dockerize App

   a. 

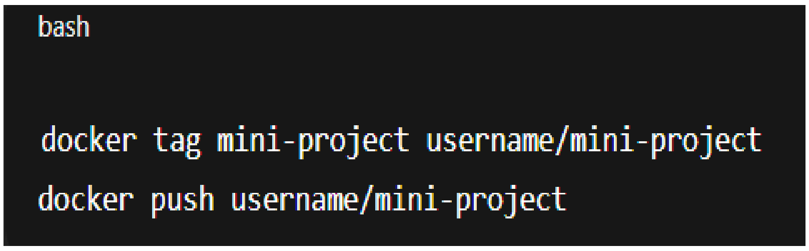
```
dockerfile

FROM node:16
WORKDIR /app
COPY . .
RUN npm install
CMD ["npm", "start"]
```

2. Build & Run Docker

   a. 

```
bash

docker build -t mini-project .
docker run -p 3000:3000 mini-project
```

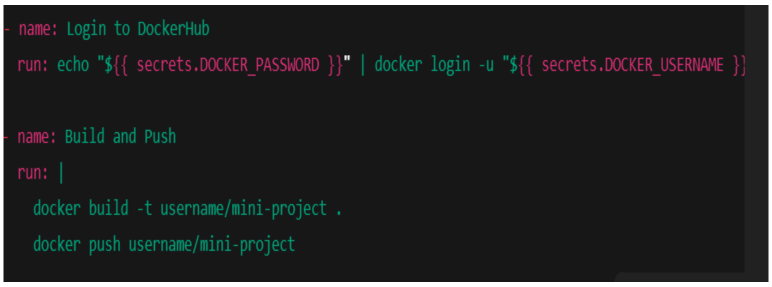
3. Push to Docker Hub

   a. 

```
bash

docker tag mini-project username/mini-project
docker push username/mini-project
```

4. Use GitHub Actions to Build & Push Docker Image

   a. Add to your GitHub Actions `.yml` file:

   b. 

```
- name: Login to DockerHub
  run: echo "${{ secrets.DOCKER_PASSWORD }}" | docker login -u "${{ secrets.DOCKER_USERNAME }}

- name: Build and Push
  run: |
    docker build -t username/mini-project .
    docker push username/mini-project
```

6. CONCLUSION

The implementation of this mini project allowed us to gain hands-on experience with two foundational aspects of modern software development: version control and continuous integration. Setting up the environment using Git enabled efficient tracking of changes, branch management, and collaborative development. We explored essential Git operations such as initializing a repository, committing changes, creating branches, and pushing code to a remote GitHub repository.

Furthermore, we implemented a CI/CD pipeline using GitHub Actions, which automatically builds and tests the codebase on every push or pull request. This helped establish an automated, repeatable, and error-free workflow that significantly reduces the time between code development and deployment. The CI/CD integration ensures code stability, catches bugs early, and enforces consistency in the development process.

By completing these tasks, we have laid a strong foundation for incorporating DevOps practices into future projects. It has also improved our understanding of automated workflows and version control, which are crucial skills in both academic projects and the software industry.

7. REFERENCES

[1] Git Documentation – Official guide for Git usage and configuration

[2] GitHub Docs – GitHub Actions – Workflow automation with GitHub Actions

[3] Node.js – Runtime used for the sample project in the CI/CD pipeline

[4] Markdown Guide – For formatting project documentation

[5] GitHub Repository (Your own repo URL here)