



Sardar Patel Institute of Technology

Bhavan's Campus, Munshi Nagar, Andheri (W), Mumbai : 400058, India

(Autonomous College of Affiliated to University of Mumbai)

Team Member Name:	Team Member UID:
Adwait Purao	2021300101

EXPERIMENT NO.:	9
AIM:	Continuous Integration using DevOps

THEORY:

DevOps Model:

DevOps is a set of practices, tools, and a cultural philosophy that automates and integrates the processes between software development and IT teams. It emphasizes team empowerment, cross-team communication and collaboration, and technology automation.

How does DevOps work?

A DevOps team includes developers and IT operations working collaboratively throughout the product lifecycle, in order to increase the speed and quality of software deployment. It's a new way of working, a cultural shift, that has significant implications for teams and the organizations they work for.

Under a DevOps model, development and operations teams are no longer “siloeed.” Sometimes, these two teams merge into a single team where the engineers work across the entire application lifecycle — from development and test to deployment and operations — and have a range of multidisciplinary skills.

DevOps teams use tools to automate and accelerate processes, which helps to increase reliability. A DevOps toolchain helps teams tackle important DevOps fundamentals including continuous integration, continuous delivery, automation, and collaboration.

DevOps values are sometimes applied to teams other than development. When security teams adopt a DevOps approach, security is an active and integrated part of the development process. This is called DevSecOps.



Sardar Patel Institute of Technology

Bhavan's Campus, Munshi Nagar, Andheri (W), Mumbai : 400058, India

(Autonomous College of Affiliated to University of Mumbai)

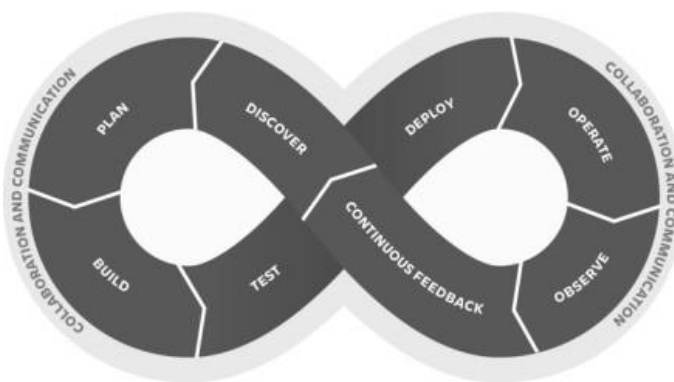
Benefits of DevOps:

Some benefits of DevOps include:

- Faster, better product delivery
- Faster issue resolution and reduced complexity
- Greater scalability and availability
- More stable operating environments
- Better resource utilization
- Greater automation
- Greater visibility into system outcomes
- Greater innovation

The DevOps lifecycle:

DevOps lifecycle is a combination of different phases of continuous software development, integration, testing, deployment, and monitoring. A competent DevOps lifecycle is necessary to build superior quality software through the system.



Here are some important DevOps Lifecycle phases / Key components of DevOps:

- 1. Continuous Development:** The planning and coding of the software are involved in this phase. It is the planning phase that decides the vision of the project.
- 2. Continuous Integration:** The new feature code is continuously integrated with the existing code. It is therefore a continuous development of software. The updated code is then integrated continuously and smoothly with the systems to reflect changes to the end users.
- 3. Continuous Testing:** In this phase, the developed software is continuously tested for bugs. For continuous testing, test automation tools such as TestNG, JUnit, Selenium, etc are used.
- 4. Continuous Deployment:** In this phase, the code is deployed to the production servers. It is essential to make sure that the code is correctly used on all the servers.



Sardar Patel Institute of Technology

Bhavan's Campus, Munshi Nagar, Andheri (W), Mumbai : 400058, India

(Autonomous College of Affiliated to University of Mumbai)

5. Continuous Monitoring: This is a very crucial stage of the DevOps life cycle where you continuously monitor the performance of your application. The system errors such as low memory, server not reachable, etc are resolved in this phase.

6. Continuous Feedback: The application development is consistently improved by analyzing the results from the operations of the software. During this phase, customer behavior is evaluated regularly on each release to improve future releases and deployments.

7. Continuous Operations: The last phase in the DevOps lifecycle is crucial for reducing planned downtime, such as scheduled maintenance. Continuous operation automates the process of launching the app and its updates. It uses container management systems like Kubernetes and Docker to eliminate downtime.

What is CI/CD?

“CI/CD” stands for the combined practices of Continuous Integration (CI) and Continuous Delivery (CD). It falls under DevOps (the joining of development and operations) and combines the practices of continuous integration and continuous delivery. CI/CD automates much or all of the manual human intervention traditionally needed to get new code from a commit into production such as build, test, and deploy, as well as infrastructure provisioning. With a CI/CD pipeline, developers can make changes to code that are then automatically tested and pushed out for delivery and deployment.

Steps involved in CI/CD:

Each change on the master Git branch performs the following steps:

- Build code
- Run unit tests
- If the tests pass, a Deploy block updates the production code that runs in the cloud

PROCEDURE:

Netlify Setup:

- Setup netlify account and add your project files
- Generate a new access token
- Go to Site Overview => Site Settings => General => Copy site ID

GitHub Setup:

- Commit and push project files to GitHub remote repository branch
- Go to settings => ci/cd => add these two variables: NETLIFY_AUTH_TOKEN, NETLIFY_SITE_ID
- Clone the project on your local machine



Machine Setup And Changes:

Checkout your branch and make the changes in your project

- Commit and push the changes to remote repository and wait for pipeline to succeed
- Merge the created branch to the main branch and wait for pipeline to get succeed

IMPLEMENTATION:

```
name: Node.js CI

on:
  push:
    branches: [master]
  pull_request:
    branches: [master]

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v2
      - name: Setup Node.js
        uses: actions/setup-node@v1
        with:
          node-version: '16.x'
      - run: npm install
      - run: npm run build
      - uses: actions/upload-artifact@v2
        with:
          name: build
          path: build/

  test:
    needs: build
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v2
      - name: Setup Node.js
        uses: actions/setup-node@v1
        with:
          node-version: '16.x'
      - run: npm install
      - uses: actions/download-artifact@v2
        with:
          name: build
          path: build
      - run: npm run test
```

```

deploy:
  needs: test
  runs-on: ubuntu-latest
  if: github.ref == 'refs/heads/master'

steps:
  - name: Checkout Repository
    uses: actions/checkout@v2

  - name: Setup Node.js
    uses: actions/setup-node@v1
    with:
      node-version: '16.x'

  - name: Download Build Artifacts
    uses: actions/download-artifact@v2
    with:
      name: build
      path: build

  - name: Install Netlify CLI
    run: npm install -g netlify-cli

  - name: Deploy to Netlify
    run: npx netlify deploy --dir=build --site ${ secrets.NETLIFY_SITE_ID } --auth ${ secrets.NETLIFY_AUTH_TOKEN } --prod

```

Netlify:

Site Overview:


Configuration for adwait-purao-cicd

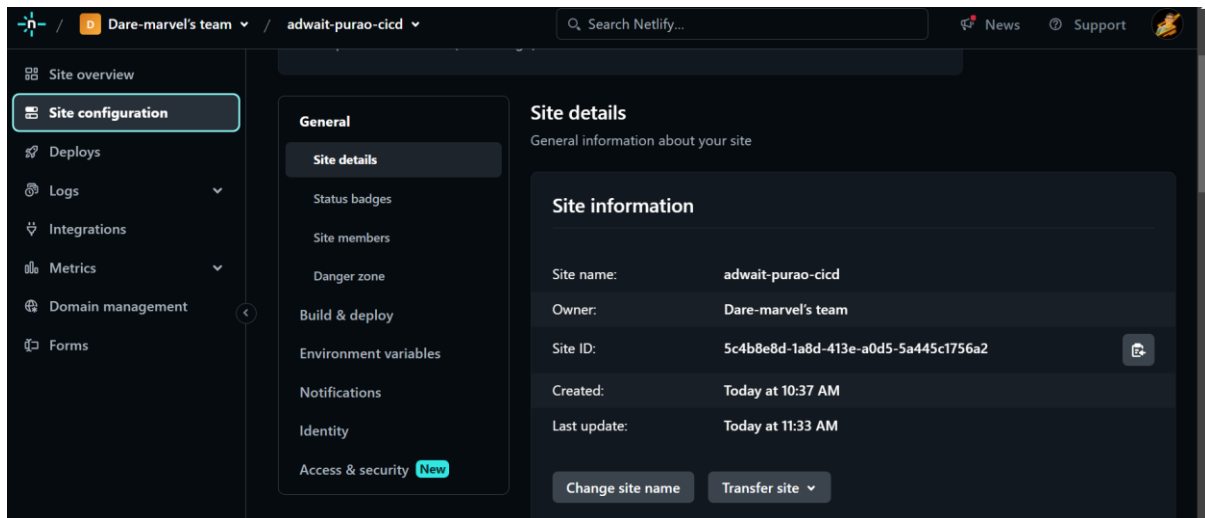
[adwait-purao-cicd.netlify.app](#)

Manual deploys.

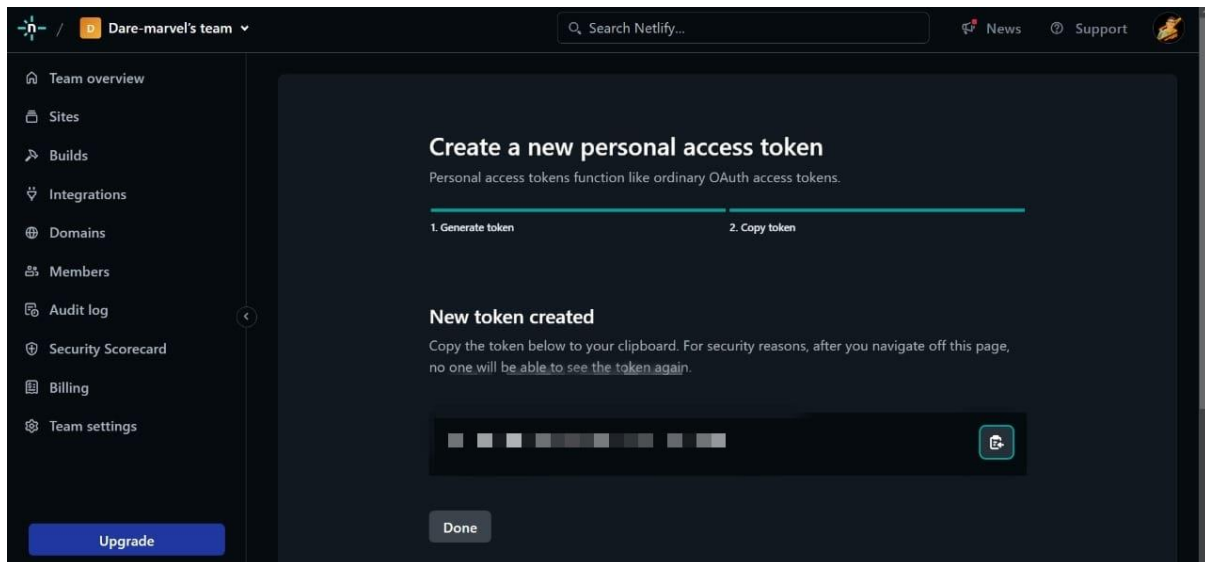
Owned by [Dare-marvel's team](#).

Last update at 11:33 AM (5 hours ago)

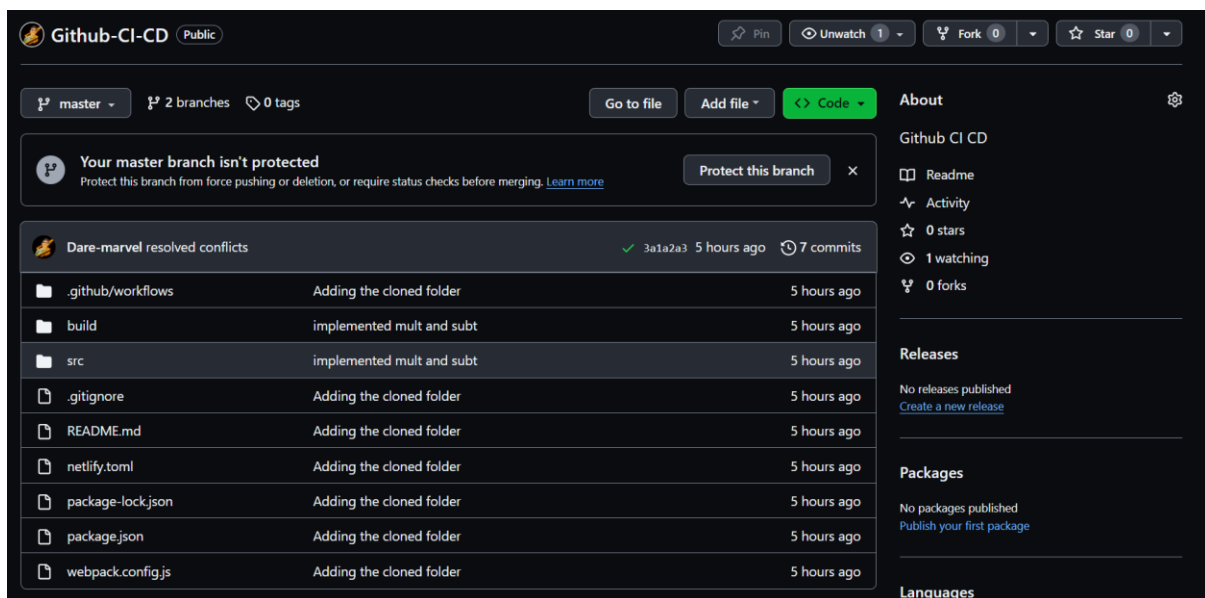


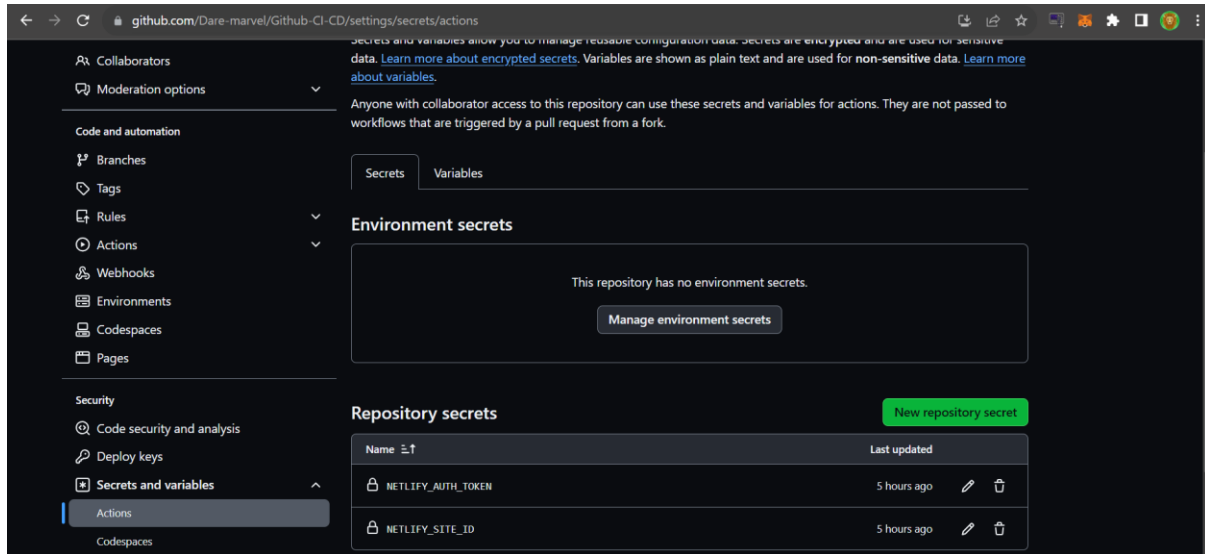


Creating an access token:



Github Repository Setup:





Local Machine Setup:

```
aspur@LAPTOP-LG4IQEFB MINGW64 /b/My CI CD/Github-CI-CD (master|MERGING)
$ git add .

aspur@LAPTOP-LG4IQEFB MINGW64 /b/My CI CD/Github-CI-CD (master|MERGING)
$ git commit -m "resolved conflicts"
[master 3a1a2a3] resolved conflicts

aspur@LAPTOP-LG4IQEFB MINGW64 /b/My CI CD/Github-CI-CD (master)
$ git push origin master
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 4 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 730 bytes | 730.00 KiB/s, done.
Total 6 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To https://github.com/Dare-marvel/Github-CI-CD.git
ca60019..3a1a2a3 master -> master
```

```
aspur@LAPTOP-LG4IQEFB MINGW64 /b/My CI CD/Github-CI-CD (master)
$ npm run build

> build
> webpack

asset main.bundle.js 972 bytes [compared for emit] [minimized] (name: main)
asset index.html 807 bytes [compared for emit]
./src/index.js 1.2 KiB [built] [code generated]
./src/calculator/add.js 112 bytes [built] [code generated]
./src/calculator/subtract.js 125 bytes [built] [code generated]
./src/calculator/multiply.js 128 bytes [built] [code generated]
./src/calculator/divide.js 174 bytes [built] [code generated]
webpack 5.74.0 compiled successfully in 871 ms
```



```
aspur@LAPTOP-LG4IQEFB MINGW64 /b/My CI CD/Github-CI-CD (master)
$ npm run test
```

```
> test
> jest
```

```
PASS src/index.test.js
```

- ✓ Should add two numbers (3 ms)
- ✓ Should subtract two numbers
- ✓ Should multiply two numbers (1 ms)
- ✓ Should divide two numbers (1 ms)
- ✓ Should handle division by zero (1 ms)

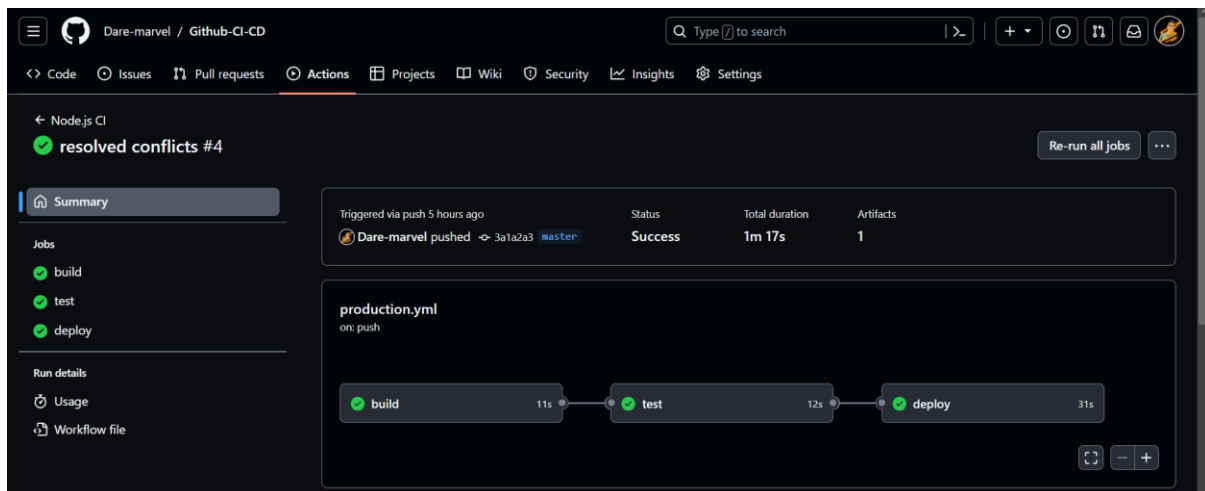
```
Test Suites: 1 passed, 1 total
```

```
Tests: 5 passed, 5 total
```

```
Snapshots: 0 total
```

```
Time: 0.478 s, estimated 1 s
```

```
Ran all test suites.
```



The screenshot shows the GitHub Actions interface for a workflow named 'production.yml' triggered by a push to the 'master' branch. The workflow consists of three jobs: 'build' (11s), 'test' (12s), and 'deploy' (31s), all of which completed successfully. The interface includes a sidebar with navigation options like Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The main area displays the workflow summary, including the trigger event, status, total duration, and artifacts. The 'production.yml' file content is also visible, showing the 'on: push' trigger and the sequence of jobs.

Node.js CI

resolved conflicts #4

Re-run all jobs

Summary

Jobs

- build
- test
- deploy

Run details

- Usage
- Workflow file

Triggered via push 5 hours ago

Dare-marvel pushed -> 3a1a2a3 master

Status: Success

Total duration: 1m 17s

Artifacts: 1

production.yml

on: push

build (11s) → test (12s) → deploy (31s)

Deployment Screenshot:



6562df9c09319d36c740ac96--adwait-purao-cicd.netlify.app

Calculator

65 x 63 Calculate

Result: 4095

Links :

Github: <https://github.com/Dare-marvel/Github-CI-CD>

Netlify: <https://6562df9c09319d36c740ac96--adwait-purao-cicd.netlify.app/>

Conclusion:

In this experiment, the focus was on studying DevOps concepts and lifecycle, leading to the successful implementation of a streamlined CI/CD pipeline using GitHub and Netlify. This integration facilitated automated code processes, including integration, testing, and deployment. GitHub supported collaborative coding as a version control system, while Netlify automated deployment and hosting. This approach efficiently identifies integration issues early and ensures a continuous, reliable deployment of tested code changes to production environments.