|  |  |
| --- | --- |
| **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 “siloed.” 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.

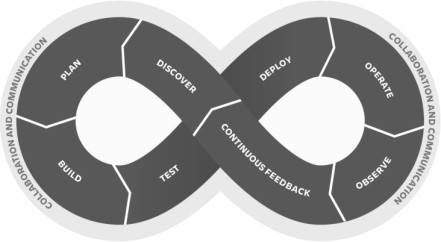
## Benefits of DevOps:

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.
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



### Local 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:

## A black background with white text Description automatically generated

## A screenshot of a computer Description automatically generated

## Creating an access token:

## A screenshot of a computer Description automatically generated

## Github Repository Setup:

## A screenshot of a computer Description automatically generated

## A screenshot of a computer Description automatically generated

## Local Machine Setup:

## A screenshot of a computer program Description automatically generated

## A screen shot of a computer program Description automatically generated

## A computer screen shot of a computer program Description automatically generated

## A screenshot of a computer Description automatically generated

## Deployment Screenshot:

## A screenshot of a computer Description automatically generated

## 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.

## 