PHASE 1

PROJECT TITLE :- Setting up a CI/CD pipeline for automated deployment

COLLEGE NAME:-Dr. SMCE

GROUP MEMBERS:-

1. Amith C Y [USN:- 1CC21CS004]
2. Charan K M [USN:- 1CC21CS017]
3. Nitin Teja N[USN:-1CC21CS040]
4. Sunil C M [USN:-1CC21CS057]

1. <u>Setting up a CI/CD pipeline for automated deployment:-</u>

In modern **DevOps practices**, setting up a **CI/CD (Continuous Integration and Continuous Deployment)** pipeline is essential for **automating software delivery**. The goal is to reduce manual effort, catch bugs early, and deploy code changes quickly and reliably.

What is CI/CD?

- **CI (Continuous Integration):** Automatically tests and integrates new code into a shared repository as soon as it's committed.
- **CD (Continuous Deployment/Delivery):** Automatically builds, tests, and deploys the application to staging or production environments.

Why Set Up a CI/CD Pipeline?

- Faster development cycles
- Consistent and error-free deployments
- Immediate feedback for developers
- Improved collaboration between Dev and Ops teams

What Happens in a CI/CD Pipeline?

1. **Code Commit** – Developer pushes code to GitHub/GitLab.
2. **Automated Build** – Tool like Jenkins or GitHub Actions compiles the code.
3. **Automated Testing** – Unit and integration tests run.
4. **Build Artifacts** – Docker image is created.
5. **Deployment** – Image is deployed to a cloud server.
6. **Notifications** – Success or failure messages are sent via email/Slack.

2. <u>Core Functionalities of the Application:-</u>

1. **Automated Code Integration (CI):**
   - Automatically pulls the latest code when changes are pushed to the repository (e.g., GitHub).
   - Merges code from multiple developers and checks for conflicts.
2. **Automated Testing:**

    - Runs unit, integration, and functional tests.
    - Ensures the new code doesn't break existing features.
3. **Build Automation:**
    - Compiles the code or builds Docker images.
    - Prepares the application for deployment.
4. **Artifact Storage:**
    - Stores build outputs (e.g., Docker images or compiled binaries) in a registry or storage system like Docker Hub or GitHub Container Registry.
5. **Automated Deployment (CD):**
    - Deploys the application to staging or production environments.
    - Can be configured for **manual approval** or **fully automated release**.
6. **Monitoring and Notifications:**
    - Sends alerts (via Slack, email, etc.) for build success/failure or deployment status.
    - Integrates with monitoring tools to track deployment performance.
7. **Rollback Mechanism (Optional):**
    - Allows reverting to the previous version in case of deployment failure

.

3. Services And Tools Used for Setting Up a CI/CD Pipeline for Automated Deployment:-

To set up a robust CI/CD pipeline, you'll need a combination of **DevOps tools** that help in coding, building, testing, deploying, and monitoring the application.

Tool	**Purpose**
Git	Version control system to manage code changes
GitHub / GitLab	Code repository and CI/CD integration platform
GitHub Actions / Jenkins / GitLab CI	Automate build, test, and deploy pipelines
Docker	Containerizes the application for consistent deployment
Docker Hub	Stores and shares Docker images
VS Code / IntelliJ	Code editor for software development
Postman	API testing to validate endpoints before and after deployment
Slack / Email	Sends notifications on build or deployment events
SonarQube	Analyzes code quality, bugs, and security vulnerabilities
AWS	Cloud platforms to host and deploy applications

4. Project Setup: Setting Up a CI/CD Pipeline for Automated Deployment:-

Project: Setting Up a CI/CD Pipeline for Automated Deployment

Here's a step-by-step approach to complete your DevOps project:

Step-by-Step Execution Plan:

1. Plan and Prepare
Choose the tech stack (e.g., Python, Node.js, etc.)
Set up the GitHub repository
Define pipeline stages: Build → Test → Deploy

2. Write and Push Code
Write your application code in a local IDE (e.g., VS Code)
Create a .gitignore and necessary config files
Push the code to GitHub/GitLab

3. Containerize the Application
Create a Dockerfile in your project rootZ
Build and test the Docker image locally
Push the image to Docker Hub or GitHub Container Registry

4. Set Up CI Pipeline (Continuous Integration)
Use GitHub Actions or Jenkins to:
Install dependencies
Run automated tests
Build the Docker image
Store the build artifact or push to container registry
Example: Create .github/workflows/main.yml for GitHub Actions

5. Set Up CD Pipeline (Continuous Deployment)
Configure deployment to a server or cloud (e.g., AWS EC2, Heroku, Kubernetes)
Use CI tool to:
SSH into the server or trigger Helm/K8s deployment
Deploy the latest build automatically

6. Add Notification and Monitoring
Integrate Slack or Email to receive notifications for each pipeline stage (Optional) Set up Prometheus + Grafana for performance monitoring

7. Test and Improve
Make code changes and push again
Observe automated build, test, and deployment
Debug and refine as needed

Project Flow:-

Code Commit → CI Trigger → Test → Build Docker Image → Push Image → Deploy to Server → Notify Team

5. Explanation About the Tools & Services Used in Setting Up a CI/CD Pipeline for Automated Deployment:-

In this project, you will use a combination of **DevOps tools and services** that work together to automate the software delivery process—from code commit to deployment.

In this project, you will use a combination of DevOps tools and services that work together to automate the software delivery process—from code commit to deployment.

1. Git & GitHub / GitLab

Purpose: Source code management (SCM) and version control.

Use: Hosts your code and allows collaboration with other developers. Triggers the pipeline when new code is pushed.

Service: GitHub / GitLab

2. GitHub Actions / GitLab CI / Jenkins

Purpose: CI/CD pipeline orchestration.

Use: Automates the process of building, testing, and deploying your code.
Examples:
GitHub Actions: YAML-based workflow built into GitHub.
Jenkins: Open-source CI server installed and managed manually.
Service: GitHub Actions Docs / Jenkins Docs
3. Docker

Purpose: Containerization.

Use: Packages your application and its dependencies into a single, portable container image.
Why: Ensures consistent environments across development, testing, and production.
Service: Docker

4. Docker Hub / GitHub Container Registry Purpose: Container image storage and distribution. Use: Store and pull Docker images for deployment. Service: Docker Hub

5. AWS / Azure / GCP
Purpose: Cloud hosting for application deployment.

Use: Hosts your deployed application (e.g., via EC2, ECS, App Service, or Compute Engine).
Example: Use AWS EC2 to host your Docker container.
Service:
AWS
Azure
GCP
6. Slack / Email

Purpose: Notification system.
Use: Sends alerts when a build or deployment succeeds or fails.
Service: Slack API

7. Kubernetes
Purpose: Container orchestration.
Use: Deploy and manage multiple containers at scale.
Service: Kubernetes

8. Terraform
Purpose: Infrastructure as Code (IaC).
Use: Automate infrastructure provisioning (e.g., setting up cloud servers).
Services:
Terraform
Ansible

6. Reference Links:

- GitHub Actions
- Jenkins
- Docker
- Kubernetes
- Terraform
- Ansible
- AWS
- Azure
- GCP
- Prometheus
- Grafana