CI/CD Pipeline Setup using GitHub Actions

1. Aim

The primary aim of this experiment is to establish a **Continuous Integration (CI)** and **Continuous Deployment (CD)** pipeline using **GitHub Actions**. This pipeline will automate the processes of dependency installation, building, testing, and simulated deployment of a sample web application upon every code push to the main branch.

2. Objective

Upon completion of this procedure, the user will be able to:

1. Define an event-driven workflow in GitHub Actions using YAML.
2. Configure a multi-step job to run on a hosted runner (e.g., ubuntu-latest).
3. Implement a CI stage to install dependencies, run build scripts, and execute tests.
4. Utilize actions/upload-artifact and actions/download-artifact to transfer build outputs between separate jobs.
5. Implement a conditional CD stage that executes only upon successful CI completion and for specific branch merges.

3. Theory (CI/CD and GitHub Actions)

Continuous Integration (CI)

CI is a development practice where developers merge their code changes into a central repository frequently (often several times a day). Automated builds and tests are run after each integration to quickly detect integration errors. This practice leads to fewer integration problems and allows teams to develop cohesive software more rapidly.

Continuous Deployment (CD)

CD is the next logical step after CI. It automates the deployment of the application to production or staging environments after all tests have passed. The goal is to ensure that a fully verified version of the application is delivered to users as quickly and reliably as possible.

GitHub Actions Components

Component	Description
Workflow	A configurable automated process defined

	in a YAML file, placed in the .github/workflows directory.
Event	The specific activity that triggers the workflow (e.g., push to a branch, pull_request, scheduled time).
Job	A set of steps that execute on the same runner. Jobs run in parallel by default but can be configured to run sequentially using needs.
Step	An individual task within a job. A step can execute a shell command (run) or use an existing Action (uses).
Runner	A server that executes the workflow. GitHub provides hosted runners (ubuntu-latest, windows-latest, etc.), or users can host their own.
Action	A packaged piece of code (like a reusable function) that performs a specific task, such as checking out code or setting up a Node.js environment.

4. Procedure

The following steps outline how to set up the CI/CD pipeline using the code provided in the next section:

1. **Project Setup (Local):** Create a simple Node.js project directory. Include the following minimal files:
   - package.json: Containing a start, build, and test script (e.g., "build": "echo 'Building application...' && mkdir -p dist", "test": "echo 'Tests passed!'").
   - index.js: A simple placeholder file.
2. **Repository Initialization:** Initialize a Git repository, commit the files, and push them to a new repository on GitHub.
3. **Workflow Directory Creation:** Inside the repository, create the necessary folder structure: .github/workflows.
4. **Workflow File Creation:** Place the provided GitHub Actions code (ci_cd_pipeline.yml) inside the .github/workflows directory.
5. **Commit and Push:** Commit the workflow file and push it to the remote repository's main

branch.
6. **Pipeline Execution:** Navigate to the "Actions" tab on the GitHub repository. The push event will automatically trigger the workflow, and the user can monitor the status of the build_and_test and deploy_staging jobs in real-time.

5. Code

5.1. GitHub Actions Workflow (ci_cd_pipeline.yml)

The complete, multi-stage workflow is provided in the file block below.

5.2. Simulated Node.js Context

To make the workflow runnable, the project root is assumed to contain a package.json file with scripts that match the CI steps:

```
{
  "name": "ci-cd-demo",
  "version": "1.0.0",
  "scripts": {
    "build": "echo 'Building static assets...' && mkdir -p dist && echo 'App content' > dist/index.html",
    "test": "echo 'Running unit and integration tests... All tests passed!'"
  }
}
```

6. Output (Simulated Execution Log)

The GitHub Actions log for a successful run on the main branch would show the following sequence:

Job/Step	**Status**	**Log Snippet**
Job: build_and_test	**Success**	
Step: Set up Node.js	Success	Successfully set up version 20 of Node.js
Step: Install dependencies	Success	npm WARN deprecated... 0 vulnerabilities found
Step: Run build and tests	Success	Running unit and integration tests... All tests

		passed!
Step: Upload Artifacts	Success	Uploaded artifact 'web-app-dist' (1.2KB)
Job: deploy_staging	**Success**	
Step: Download Artifacts	Success	Downloaded artifact 'web-app-dist' to './app_to_deploy'
Step: Simulate Deployment	Success	Deployment to Staging Environment Started...
		Artifacts ready for deployment at: ./app_to_deploy
		Deployed successfully on Tue Oct 14...

7. Learning Outcomes

1. **Orchestration of Multi-Stage Workflows:** The ability to design complex, multi-job pipelines where data (artifacts) is passed between stages and jobs are sequenced using the needs dependency (e.g., deploy_staging depends on build_and_test).
2. **Declarative Automation with YAML:** Proficiency in defining infrastructure and processes using the declarative YAML syntax required by modern automation tools, focusing on events, runners, and custom shell commands.
3. **CI/CD Logic Implementation:** Understanding how to use conditional statements (if: github.ref == 'refs/heads/main') to implement continuous deployment logic, ensuring that deployment only occurs after specific conditions (like merging to the main branch) are met.