1. Objective

By the end of this 2-hour lab, you will have built a complete Continuous Integration (CI) pipeline for a Node.js application. You will automate the process of testing and containerizing code, moving from a manual workflow to a fully automated one.

2. Prerequisites

- A personal GitHub Account.
- Git installed on your local machine.
- Node.js (LTS version) installed.
- Docker Desktop installed and running.
- A code editor like VS Code.

Action Required: Create Your Own Starter Project

Please follow these steps to create the necessary files. This will be your starting point for the lab.

1. Create a Project Directory On your local machine, create a new folder for the lab and navigate into it.

Bash

```
mkdir nodejs-ci-lab
cd nodejs-ci-lab
```

2. Create the Project Files Create the following five files inside the `nodejs-ci-lab` directory with the exact content provided below.

File: `package.json`

JSON

```
{
  "name": "nodejs-ci-lab",
  "version": "1.0.0",
  "description": "A Node.js app for the CI lab.",
  "main": "index.js",
  "scripts": {
    "start": "node index.js",
    "lint": "eslint ."
  },
  "dependencies": {
    "express": "^4.18.2"
  },
  "devDependencies": {
    "eslint": "^8.35.0"
  }
}
```

File: `index.js`

JavaScript

```
const express = require('express');
const app = express();
const PORT = 3000;

app.get('/', (req, res) => {
  res.send('Hello from Node.js App!');
});

app.get('/health', (req, res) => {
  res.status(200).json({ status: 'ok' });
});

app.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});
```

File: `.eslintrc.json`

JSON

```
{
    "env": {
        "commonjs": true,
        "es2021": true,
        "node": true
    },
    "extends": "eslint:recommended",
    "parserOptions": {
        "ecmaVersion": "latest"
    },
    "rules": {}
}
```

File: `Dockerfile`

Dockerfile

```
# Use an official Node.js runtime as a parent image
FROM node:18-alpine

# Set the working directory in the container
WORKDIR /usr/src/app

# Copy package.json and package-lock.json
COPY package*.json ./

# Install app dependencies
RUN npm ci --only=production

# Bundle app source
COPY . .

# Your app binds to port 3000 so you'll use this port
EXPOSE 3000

# Define the command to run your app
CMD [ "node", "index.js" ]
```

File: `.gitignore`

```
# Dependencies
/node_modules

# Logs
npm-debug.log*
yarn-debug.log*
yarn-error.log*
lerna-debug.log*

# Misc
.DS_Store
```

3. Set up the Git Repository Now, turn this local folder into a Git repository and push it to your own new repository on GitHub.

Bash

```
# Initialize a new git repository
git init -b main

# Add all the files to staging
git add .

# Make your first commit
git commit -m "Initial project setup for CI lab"

# Go to GitHub.com and create a new, empty repository (e.g., "nodejs-ci-
lab")
# DO NOT initialize it with a README or .gitignore on GitHub.

# Link your local repository to the one on GitHub
# Replace <YOUR_USERNAME> and <YOUR_REPONAME> with your actual details
git remote add origin
https://github.com/<YOUR_USERNAME>/<YOUR_REPONAME>.git

# Push your code to GitHub
git push -u origin main
```

3. Lab Environment Setup

1. **Explore the Project:**
    - Open the project in VS Code and familiarize yourself with the files (`index.js`, `package.json`, `Dockerfile`, etc.). The project is a fully functional Node.js application but lacks automation.

---

4. Lab Tasks

Task 1: The Manual Way - Local Verification

Before we automate, let's understand the manual process.

1. **Install Dependencies:** Open your terminal in the project root and run:

   Bash

   ```
   npm install
   ```

2. **Run the Linter Manually:** Execute the test script.

   Bash

   ```
   npm run lint
   ```

   - **Checkpoint:** You should see no output, meaning no errors were found.
3. **Run the Application Locally:**

   Bash

   ```
   npm start
   ```

   - **Checkpoint:** Open `http://localhost:3000` in a browser. You should see "Hello from Node.js App!". Stop the server with `Ctrl+C`.
4. **Build & Run the Docker Container Manually:**

   Bash

   ```
   docker build -t my-manual-app .
   docker run -p 3000:3000 -d my-manual-app
   ```

   - **Checkpoint:** Refresh `http://localhost:3000`. It should still work.
   - **Cleanup:** Stop the container using its ID from `docker ps`
   - `docker stop <CONTAINER_ID>`

Task 2: Your First CI Workflow - Automated Testing

1. **Create Workflow Files:** In your project root, create the directory structure `.github/workflows/`. Inside that, create a new file named `ci.yml`.
2. **Write the Workflow:** Paste the following content into `ci.yml`:

   YAML

   ```
   name: Node.js CI

   on:
     push:
       branches: [ "main" ]

   jobs:
     build-and-test:
       runs-on: ubuntu-latest
       steps:
         - name: Checkout Repository
           uses: actions/checkout@v4
   ```

```
      - name: Set up Node.js
        uses: actions/setup-node@v4
        with:
          node-version: '18'
          cache: 'npm'

      - name: Install Dependencies
        run: npm ci

      - name: Test with ESLint
        run: npm run lint
```

3. **Trigger the Action:** Commit and push your changes to GitHub.

   Bash

   ```
   git add .
   git commit -m "feat: Add initial CI workflow for testing"
   git push origin main
   ```

4. **Observe the Workflow Run:** Go to the "**Actions**" tab on your GitHub repository to see your workflow run successfully.
5. **Test the Failure Case:**
   - In `index.js`, add `var x = 5;` at the end of the file to introduce a linting error.
   - Commit and push this change.
   - **Checkpoint:** Go to the "Actions" tab and observe that the new run **fails**. Investigate the logs for the "Test with ESLint" step to see the error.
   - **Fix the error**, commit, and push again to make the pipeline pass.

Task 3: Automating Containerization

1. **Update `ci.yml`:** Modify your workflow to grant permissions and add steps for logging into Docker and pushing the image.

   YAML

   ```
   name: Node.js CI

   on:
     push:
       branches: [ "main" ]

   jobs:
     build-and-test:
       runs-on: ubuntu-latest
       permissions:
         contents: read
         packages: write
       steps:
         - name: Checkout Repository
           uses: actions/checkout@v4

         - name: Set up Node.js
           uses: actions/setup-node@v4
           with:
             node-version: '18'
   ```

```
          cache: 'npm'

      - name: Install Dependencies
        run: npm ci

      - name: Test with ESLint
        run: npm run lint

      - name: Log in to GitHub Container Registry
        uses: docker/login-action@v3
        with:
          registry: ghcr.io
          username: ${{ github.actor }}
          password: ${{ secrets.GITHUB_TOKEN }}

      - name: Build and Push Docker Image
        uses: docker/build-push-action@v5
        with:
          context: .
          push: true
          tags: ghcr.io/${{ github.repository_owner }}/nodejs-ci-
lab:${{ github.sha }}
```

2. **Commit and Push** the final workflow changes.
3. **Verify the Published Package:**
    - Watch the Action run successfully.
    - **Checkpoint:** On your repository's main page, go to the **"Packages"** section on the right-hand side. You should see your published `nodejs-ci-lab` container image.