**Complete Step-by-Step Documentation with Code**

**Step 1: Create Node.js app (index.js)**

js

CopyEdit

// index.js

const express = require('express');

const app = express();

app.get('/', (req, res) => {

res.send('Hello from Node.js app running in Docker!');

});

const PORT = process.env.PORT || 3000;

app.listen(PORT, () => {

console.log(`Server running on port ${PORT}`);

});

**Step 2: Create package.json**

json

CopyEdit

{

"name": "nodejs-demo-app",

"version": "1.0.0",

"main": "index.js",

"scripts": {

"start": "node index.js"

},

"dependencies": {

"express": "^4.18.2"

}

}

Run to install dependencies:

bash

CopyEdit

npm install express

**Step 3: Create Dockerfile**

Dockerfile

CopyEdit

# Use Node official image

FROM node:18-alpine

# Set working directory

WORKDIR /app

# Copy package files and install dependencies

COPY package\*.json ./

RUN npm install

# Copy rest of app source code

COPY . .

# Expose port 3000

EXPOSE 3000

# Command to run the app

CMD ["npm", "start"]

**Step 4: Build and run Docker image locally**

bash

CopyEdit

# Build Docker image (run in root folder containing Dockerfile)

docker build -t nodejs-demo-app .

# Run container mapping port 3000

docker run -p 3000:3000 nodejs-demo-app

Check in browser: http://localhost:3000 should show your message.

**Step 5: Create Git repository and push to GitHub**

bash

CopyEdit

git init

git add .

git commit -m "Initial commit with Node.js app and Dockerfile"

git branch -M main

git remote add origin https://github.com/yourusername/nodejs-demo-app.git

git push -u origin main

**Step 6: Push Docker image to Docker Hub**

Login to Docker Hub from terminal:

bash

CopyEdit

docker login

# Enter your Docker Hub username and password (or Personal Access Token)

Tag your image:

bash

CopyEdit

docker tag nodejs-demo-app yourdockerhubusername/nodejs-demo-app:latest

Push image:

bash

CopyEdit

docker push yourdockerhubusername/nodejs-demo-app:latest

**Step 7: Setup GitHub Actions for CI/CD**

Create .github/workflows/ci-cd.yml:

yaml

CopyEdit

name: CI/CD Pipeline

on:

push:

branches: [ "main" ]

jobs:

build:

runs-on: ubuntu-latest

steps:

- name: Checkout code

uses: actions/checkout@v3

- name: Set up Docker Buildx

uses: docker/setup-buildx-action@v3

- name: Log in to DockerHub

uses: docker/login-action@v3

with:

username: ${{ secrets.DOCKER\_USERNAME }}

password: ${{ secrets.DOCKER\_PASSWORD }}

- name: Build and push Docker image

uses: docker/build-push-action@v5

with:

context: .

file: ./Dockerfile

push: true

tags: yourdockerhubusername/nodejs-demo-app:latest

**Step 8: Add secrets in GitHub repository settings**

* DOCKER\_USERNAME: Your Docker Hub username
* DOCKER\_PASSWORD: Your Docker Hub password or Personal Access Token

**Step 9: Trigger CI/CD pipeline**

* Push any change to main branch, e.g.

bash

CopyEdit

git commit --allow-empty -m "Trigger CI/CD pipeline"

git push

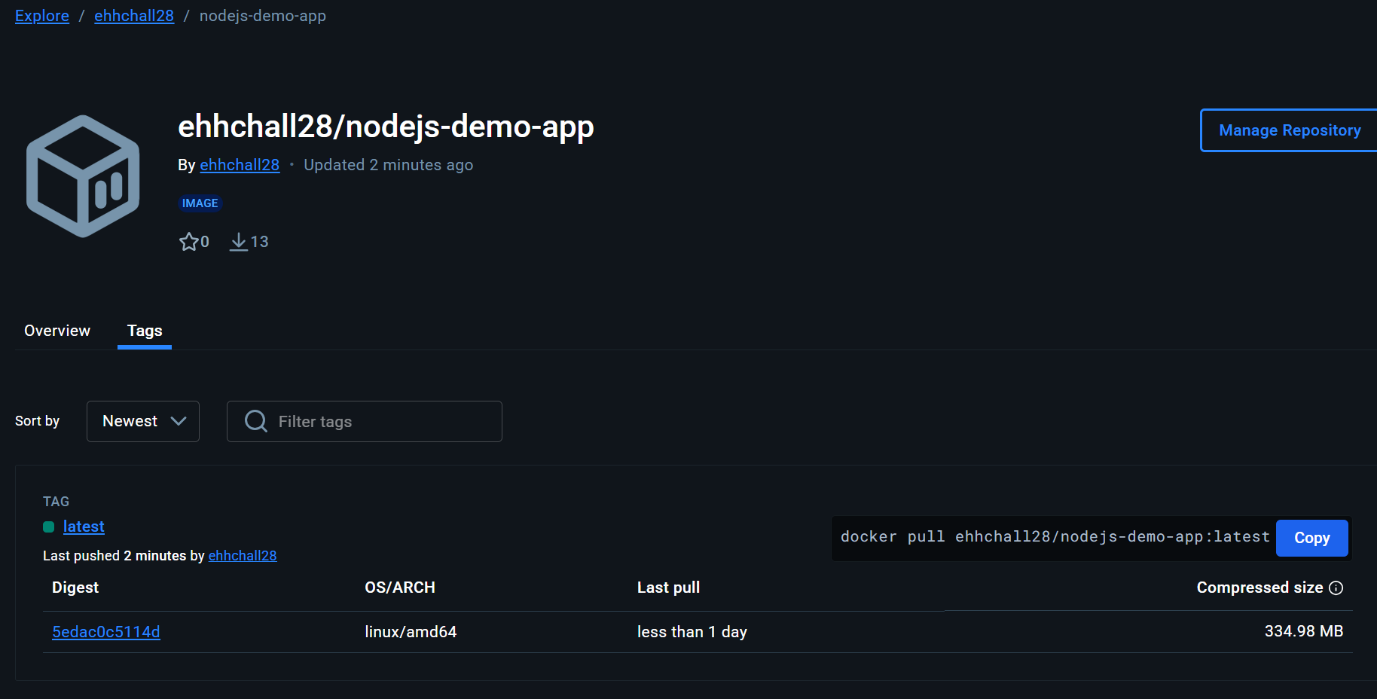
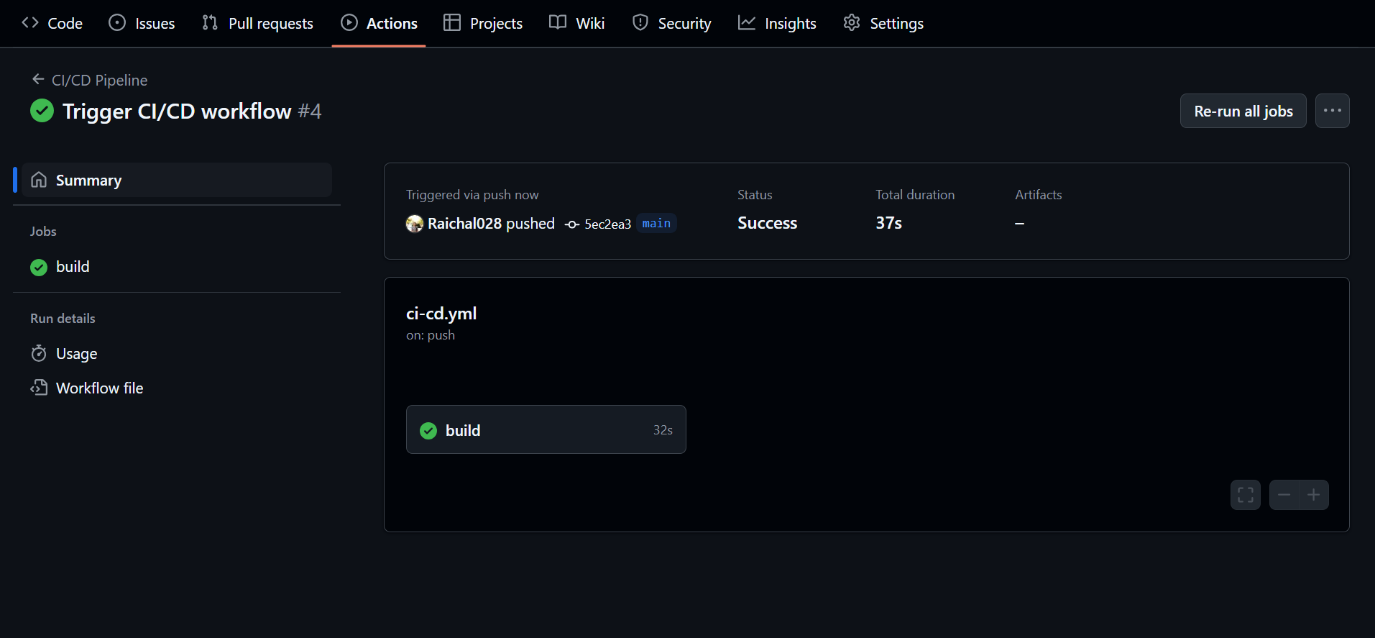
* Check **Actions** tab in GitHub repo to see pipeline status.
* Confirm image gets built and pushed to Docker Hub automatically.

**Summary of Mistakes & Fixes**

| **Mistake** | **Fix / Explanation** |
| --- | --- |
| No start script in package.json | Added "start": "node index.js" for npm start to work |
| Dockerfile not found / wrong path in workflow | Made sure Dockerfile is in root and path in workflow YAML is correct |
| Port 3000 in use error when running Docker | Stopped other apps or changed port mapping |
| Docker login failure in GitHub Actions | Used Docker Hub Personal Access Token stored in GitHub secrets |
| Git push rejected due to branch mismatch | Renamed branch to main and pulled remote changes first |
| GitHub Actions workflow not triggered | Named workflow .yml correctly in .github/workflows and pushed to main branch |

**What we achieved?**

* Wrote a Node.js app.
* Dockerized the app.
* Built and ran locally.
* Pushed code to GitHub.
* Pushed Docker image to Docker Hub.
* Automated build and push on every code push using GitHub Actions CI/CD.



**What is this whole process? And why is it cool?**

**Imagine this:**

You made a really cool toy robot at home. You want to show it to all your friends. But every time you change something in your robot — like its color or how it moves — you have to rebuild it and tell all your friends again. That’s a lot of work, right?

Now, what if you had a magical machine that:

1. **Sees when you make changes to your robot**
2. **Automatically builds the robot for you**
3. **Sends the new robot to all your friends without you lifting a finger**

Wouldn’t that be awesome?

**What we did today is like making that magical machine for your computer program (the Node.js app) and its box (Docker image)!**

**Here’s the simple story:**

1. **You built your app (robot).**  
   This is your Node.js app that runs on your computer.
2. **You put your app in a special box (Docker).**  
   This box makes sure your app can run anywhere — on your friend's computer or on a big computer in the cloud — exactly the same way.
3. **You created a magic recipe (CI/CD pipeline) using GitHub Actions.**  
   This recipe watches your app’s code and whenever you change it, the magic starts:
   * It builds the app box for you automatically.
   * It sends that box to a place called Docker Hub — like a big toy store where anyone can get your robot.
   * It does all this without you needing to do anything extra!

**Why is this useful?**

* **Saves time and effort:** You don’t have to rebuild and send the app yourself every time you change something. The magic recipe does it for you!
* **Keeps things consistent:** Your app will always work the same on your computer, your friend’s computer, or any server.
* **Makes teamwork easy:** If you work with friends on your app, everyone gets the latest, working version automatically.
* **Helps in real jobs:** Big companies use this process to deliver software quickly and safely.

**2. What is CI/CD?**

* **CI** stands for **Continuous Integration**: It’s the process of automatically building and testing code every time a team member pushes changes to version control (like GitHub).
* **CD** stands for:
  + **Continuous Delivery**: Automatically prepares code to be released to production (e.g., builds and pushes Docker images).
  + **Continuous Deployment**: Automatically deploys code to production once tests pass.

✅ CI/CD ensures **fast, consistent, and reliable software delivery**.

**3. How do GitHub Actions work?**

* GitHub Actions lets you **automate tasks** like testing, building, and deploying your code.
* It runs based on **triggers**, such as push, pull\_request, or a scheduled time.
* You write a **workflow file** (.yml) inside .github/workflows/ to define:
* **YAML** originally stood for:
* **Y**et **A**nother **M**arkup **L**anguage
* A .yml file is a human-readable data serialization file, commonly used for configuration files, CI/CD pipelines (like in Jenkins, GitHub Actions, GitLab CI, Azure DevOps), Docker Compose, Kubernetes manifests, etc.
  + What actions to take
  + When to take them (event trigger)
  + What environments (runners) to run on

✅ It runs the defined jobs inside GitHub’s virtual machines.

**4. What are runners?**

* **Runners** are the servers that execute your GitHub Actions.
* Types of runners:
  + **GitHub-hosted runners**: Managed by GitHub. You use OS like ubuntu-latest.
  + **Self-hosted runners**: Your own machines that you configure to run workflows. (eg be a PC, cloud VM like AWS EC2, Azure VM, or an on-premises server)

✅ Runners run the **jobs** defined in your workflow.

**5. Difference between Jobs and Steps**

| **Feature** | **Jobs** | **Steps** |
| --- | --- | --- |
| What it does | Defines a unit of work | Defines actions within a job |
| Runs on | Its own runner | Inside the same runner |
| Isolation | Each job is independent | Steps share the same environment |
| Example | Build job, Test job | Checkout code, install dependencies |

✅ Think of **jobs** as different **phases**, and **steps** as the **tasks** inside each phase.

**6. How to secure secrets in GitHub Actions?**

* Store secrets (like API keys, DockerHub tokens) in:
  + **Settings > Secrets and variables > Actions**
* Use them like this in workflows:

yaml

CopyEdit

with:

username: ${{ secrets.DOCKER\_USERNAME }}

password: ${{ secrets.DOCKER\_PASSWORD }}

✅ This keeps sensitive data **encrypted** and **out of your code**.

**7. How to handle deployment errors?**

* Use these strategies:
  + ✅ Add logging to know what failed
  + ✅ Use continue-on-error: false to stop on failure
  + ✅ Use if: conditions to control flow
  + ✅ Break deployment into steps to pinpoint failure
  + ✅ Re-run failed jobs from GitHub UI
  + ✅ Add try-catch or error handling in scripts

✅ Always test locally and log clearly.

.

**📖 8. Docker Build-Push Workflow Explained**

This workflow is used to **build a Docker image from your code, tag it, and push it to DockerHub (or any registry)** — typically automated via CI/CD tools like GitHub Actions.

**📌 Step-by-Step Process:**

1️⃣ **Checkout the source code** from the GitHub repository.  
2️⃣ **Login to DockerHub** using your DockerHub credentials (username and token).  
3️⃣ **Build the Docker image** from the Dockerfile in your repo.

bash

CopyEdit

docker build -t username/image-name .

4️⃣ **Tag the image** to assign a version or a latest tag.

bash

CopyEdit

docker tag image-name username/image-name:tag

5️⃣ **Push the image** to your DockerHub repository.

bash

CopyEdit

docker push username/image-name:tag

**📌 Automated Workflow in GitHub Actions (YAML)**

yml

CopyEdit

name: Build and Push Docker Image

on: [push]

jobs:

build-and-push:

runs-on: ubuntu-latest

steps:

# Step 1: Checkout code

- name: Checkout code

uses: actions/checkout@v4

# Step 2: Log in to DockerHub

- name: Log in to DockerHub

uses: docker/login-action@v3

with:

username: ${{ secrets.DOCKER\_USERNAME }}

password: ${{ secrets.DOCKER\_PASSWORD }}

# Step 3-5: Build, tag, and push the Docker image

- name: Build and push Docker image

uses: docker/build-push-action@v5

with:

push: true

tags: username/image-name:latest

**📌 Note:**

* You must add your **DockerHub credentials as GitHub Secrets**:
  + DOCKER\_USERNAME
  + DOCKER\_PASSWORD

**Go to:**  
GitHub repo → Settings → Secrets and variables → Actions → New repository secret

**✅ Summary:**

* **Checkout code**
* **Login to DockerHub**
* **Build image**
* **Tag image**
* **Push image**

All automated within GitHub Actions using **official Docker actions**.

**9. How can you test a CI/CD pipeline locally?**

You can't fully test GitHub Actions **as-is** locally, but:

**Alternatives:**

* ✅ Use [act](https://github.com/nektos/act) — a tool to run GitHub Actions locally:

bash

CopyEdit

brew install act # or use on Linux/Windows

act # runs your workflow as if it's on GitHub

* ✅ Use Docker to test builds locally:

bash

CopyEdit

docker build -t my-app .

docker run -p 3000:3000 my-app

✅ Local testing reduces errors **before pushing to GitHub**.