CI/CD WITH GITHUB ACTIONS

ROHAN THAPA
thaparohan2019@gmail.com

Introduction to CI/CD

Continuous Integration (CI):

- **Goal:** Automatically integrate and test code changes frequently.
- **Purpose:** Developers commit code frequently to the shared repository. Automated tests are run to ensure new changes don't break existing functionality.

Continuous Deployment (CD):

- **Goal:** Automate the release of code changes.
- **Purpose:** After the code passes tests and is approved, it is automatically deployed to production or staging environments without manual intervention.

Key Benefits of CI/CD:

- **Faster Delivery:** Automates testing and deployment, speeding up the release process.

- **Higher Code Quality:** Automated tests ensure the code works as expected.

- **Reduced Errors:** Less manual intervention means fewer mistakes in production.

- **Continuous Feedback:** Developers can identify bugs and issues quickly.

GitHub Actions Overview

GitHub Actions is a **CI/CD** tool integrated within **GitHub**. It allows you to create custom workflows to **build, test,** and **deploy** code directly from your **GitHub repository**.

Workflows are defined as **YAML files** inside your repository.

GitHub Actions Overview

Key Features:

- **Automation of workflows:** Build, test, and deploy your code automatically.

- **Event-driven workflows:** Trigger workflows based on specific events (e.g., push, pull_request, release, schedule).

- **Support for multiple languages:** JavaScript, Python, Go, Java, and more.

- **Custom actions:** Reusable code that can be shared across workflows or even other repositories.

Key Concepts in GitHub Actions

Workflows:

- A workflow is an automated process defined by a **YAML** file. It runs one or more jobs.
- Workflows can be triggered by events like **code pushes, pull requests**, or on a schedule.
- Location: **.github/workflows/ directory**.

Events:

- Events are activities that **trigger the workflow**. Example events include **push, pull_request, release**, or **scheduled cron jobs**.

Key Concepts in GitHub Actions

Jobs:

- A job is a **set of steps** that run on the same runner.
- Jobs can run **sequentially** or **in parallel**.
- Each job runs on its **own virtual machine** (runner).

Steps:

- Steps are **individual tasks** that make up a job. These steps can **run commands**, **scripts, or actions**.
- You can either write your own commands or use existing pre-defined actions from the GitHub Actions marketplace.

Key Concepts in GitHub Actions

Runners:

- Runners are servers that run the jobs. **GitHub provides free hosted runners**, or you can use self-hosted runners.
- Hosted runners come with predefined environments (e.g., Linux, Windows, macOS).

Anatomy of a GitHub Actions Workflow

Here's an example of a simple **CI workflow** for a Spring Boot project using Maven:

```yaml
name: CI Pipeline

on:
  push:
    branches:
      - master
  pull_request:
    branches:
      - master

jobs:
  build:

    runs-on: ubuntu-latest # Specifies the runner environment (Ubuntu)

    steps:
    - name: Checkout repository
      uses: actions/checkout@v2

    - name: Set up JDK
      uses: actions/setup-java@v4
      with:
        java-version: '17' # Define the JDK version

    - name: Cache Maven repository
      uses: actions/cache@v4
      with:
        path: ~/.m2/repository
        key: ${{ runner.os }}-maven-${{ hashFiles('**/pom.xml') }}
        restore-keys: ${{ runner.os }}-maven

    - name: Build with Maven
      run: mvn clean package -DskipTests # Compile the project

    - name: Run tests
      run: mvn test # Run unit tests
```

Explanation

1. **name:** Name of the workflow.
2. **on:** Triggers the workflow on push and pull_request to the main branch.
3. **jobs:** Contains the build job.
    - **runs-on:** Specifies that the job will run on an ubuntu-latest environment.
    - **steps:** Defines the steps of the job:
        - **Checkout code:** Uses the actions/checkout action to pull the code from the repository.
        - **Setup JDK:** Uses the actions/setup-java action to install JDK 17.
        - **Cache Maven dependencies:** Uses actions/cache to cache the .m2 folder for faster builds.
        - **Build and test:** Executes Maven commands to compile the code and run tests.

Full CI/CD Pipeline for a Spring Boot Project

In a complete **CI/CD** pipeline for **Spring Boot**, we'll typically perform these stages:

1. Build:

- The project is built using **Maven or Gradle.**

```
- name: Build the project
  run: mvn clean package
```

2. Run Unit Tests:

- Run automated unit tests to ensure the application behaves as expected.

```
- name: Run unit tests
  run: mvn test
```

Full CI/CD Pipeline for a Spring Boot Project

3. Run Integration Tests:

- Optionally, run integration tests to verify how different parts of the application work together.

```
- name: Run integration tests
  run: mvn verify
```

4. **Deploy to Staging/Production:**
   - Once **tests pass**, the application can be deployed to a **staging** or **production** environment.

Advanced Features of GitHub Actions

1. Parallel Jobs:

- You can run different jobs in parallel to speed up the workflow. For instance, you can run tests in parallel across different environments.

```yaml
jobs:
  test-java:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        java-version: [8, 11, 17] # Test with multiple JDK versions
    steps:
      - name: Set up JDK
        uses: actions/setup-java@v4
        with:
          java-version: ${{ matrix.java-version }}
      - run: mvn test
```

Advanced Features of GitHub Actions

2. Caching Dependencies:

- Caching Maven or Gradle dependencies to reduce build times.

```yaml
- name: Cache Maven repository
  uses: actions/cache@v4
  with:
    path: ~/.m2/repository
    key: ${{ runner.os }}-maven-${{ hashFiles('**/pom.xml') }}
```

Advanced Features of GitHub Actions

3. Notifications:

- You can send notifications to team members via **Slack, email**, or other tools if a build or deployment **succeeds or fails.**

```yaml
- name: Send Slack notification
  uses: 8398a7/action-slack@v3
  with:
    status: success
  env:
    SLACK_WEBHOOK_URL: ${{ secrets.SLACK_WEBHOOK_URL }}
```

Continuous Deployment (CD) with GitHub Actions

Once the **CI pipeline passes**, we can set up **Continuous Deployment** to automatically deploy the application. You can deploy to services like **AWS, Heroku**, or **DigitalOcean**.

Example: Deploy to Docker Hub:

```yaml
- name: Log in to Docker Hub
  run: echo "${{ secrets.DOCKER_PASSWORD }}" | docker login -u "${{ secrets.DOCKER_USERNAME }}" --password-stdin

- name: Build Docker Image
  run: docker build -t your-dockerhub-username/spring-boot-app .

- name: Push Docker Image
  run: docker push your-dockerhub-username/spring-boot-app
```

Continuous Deployment (CD) with GitHub Actions

Example: Deploy to Heroku:

```yaml
- name: Deploy to Heroku
  uses: akhileshns/heroku-deploy@v3.12.12
  with:
    heroku_api_key: ${{ secrets.HEROKU_API_KEY }}
    heroku_app_name: "spring-boot-heroku-app"
    heroku_email: "your-email@example.com"
```

Deploy to AWS Elastic Beanstalk:

```yaml
- name: Deploy to AWS Elastic Beanstalk
  run: /
    eb init -p java-17 spring-boot-app --region us-west-2
    eb deploy
```

Couldn’t try these

Complete Example with CI/CD

Docker file

```
FROM maven:3.9.7 AS build
WORKDIR /app

COPY pom.xml .
RUN mvn dependency:go-offline -B

COPY src ./src
RUN mvn clean package -DskipTests

FROM openjdk:21
WORKDIR /app

COPY --from=build /app/target/Notes-Management-System.jar app.jar

EXPOSE 8080
ENTRYPOINT ["java" , "-jar" , "app.jar"]
```

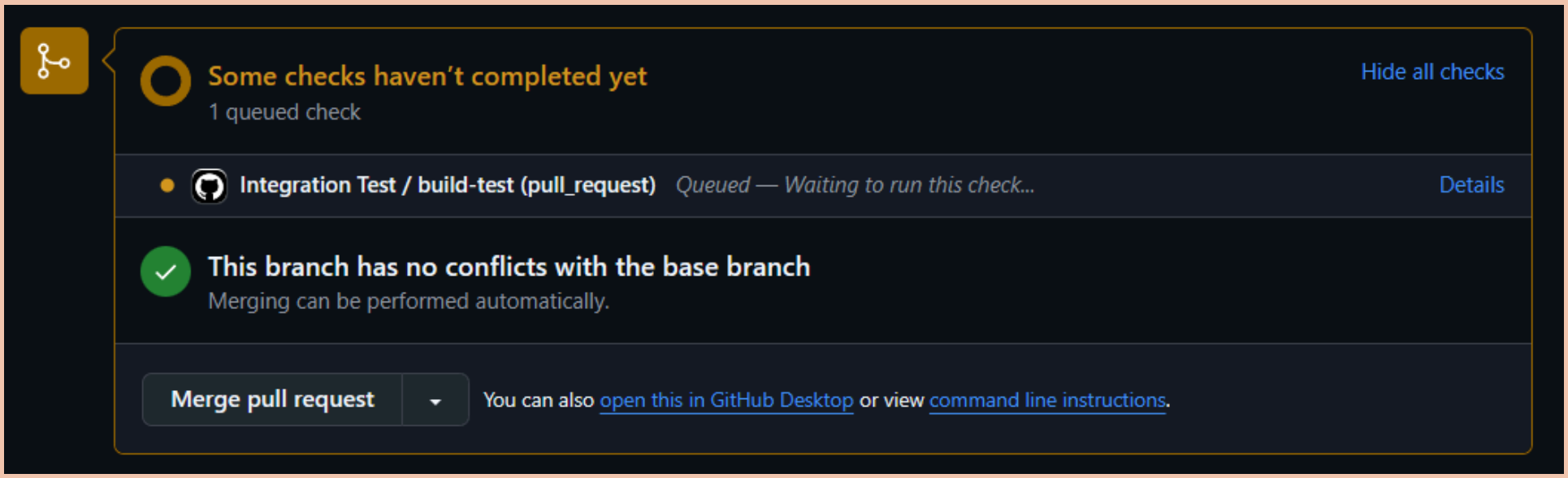
Complete Example with CI/CD

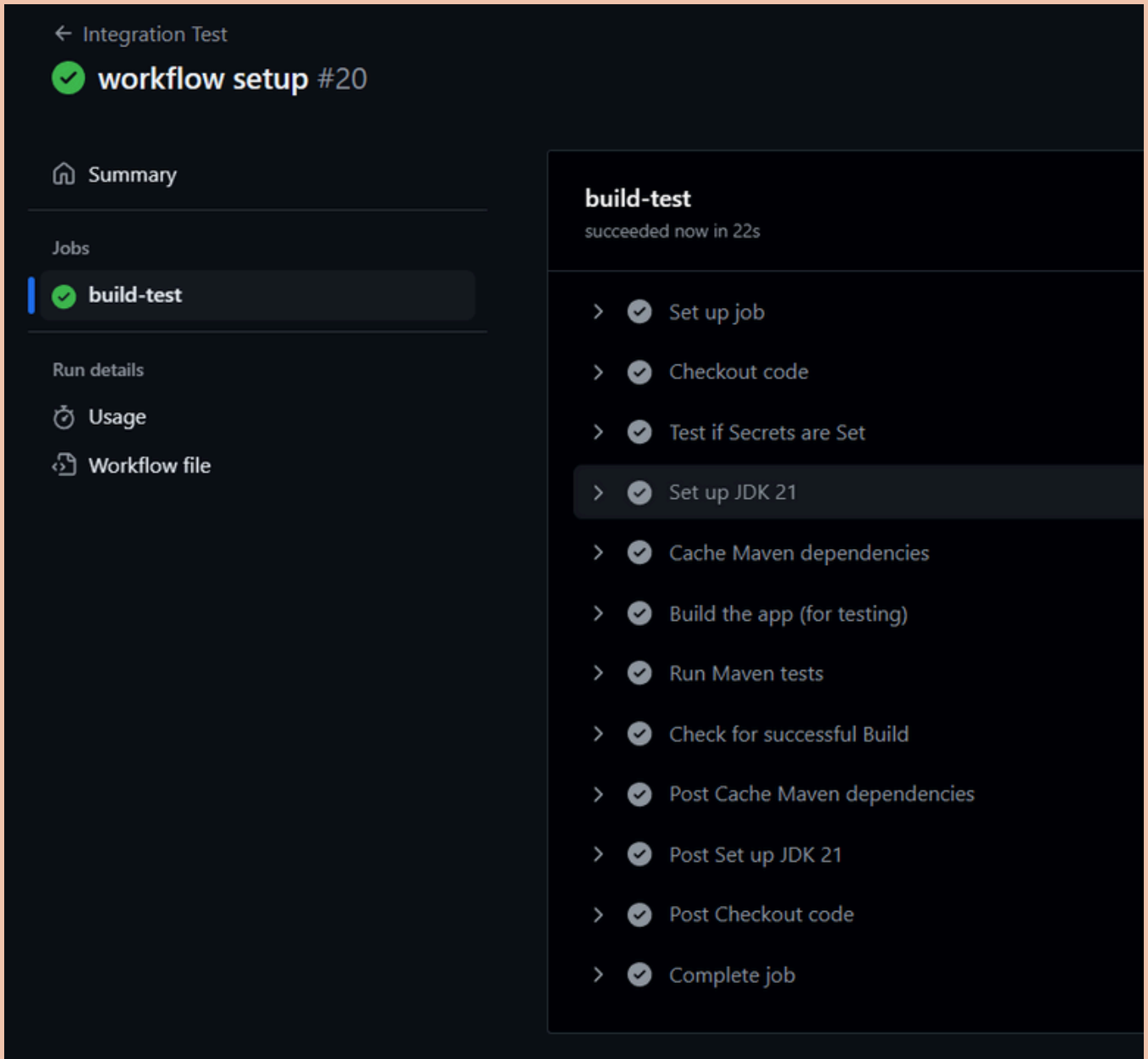
Continuous Integration

```yaml
name: Integration Test

on:
  pull_request:
    branches:
      - master

jobs:
  build-test:
    runs-on: ubuntu-latest

    env:
      BASE_URL: ${{ secrets.BASE_URL }}
      MONGO_DB: ${{ secrets.MONGO_DB }}
      JWT_SECRETS: ${{ secrets.JWT_SECRETS }}
      MONGO_URI: ${{ secrets.MONGO_URI }}
      GOOGLE_CREDENTIALS: ${{ secrets.GOOGLE_CREDENTIALS }}

    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Test if Secrets are Set
        run: |
          echo "Checking if BASE_URL is set: ${{ secrets.BASE_URL }}"
          echo "Checking if MONGO_URI is set: ${{ secrets.MONGO_URI }}"
          echo "Checking if JWT_SECRETS is set: ${{ secrets.JWT_SECRETS }}"

      - name: Set up JDK 21
        uses: actions/setup-java@v4
        with:
          java-version: '21'
          distribution: 'temurin'

      - name: Cache Maven dependencies
        uses: actions/cache@v4
        with:
          path: ~/.m2
          key: ${{ runner.os }}-maven-${{ hashFiles('**/pom.xml') }}
          restore-keys: ${{ runner.os }}-maven

      - name: Build the app (for testing)
        run: mvn clean package -DskipTests

      - name: Run Maven tests
        run: mvn clean test

      - name: Check for successful Build
        run: echo "Integration test passed"
```

Complete Example with CI/CD

In Action





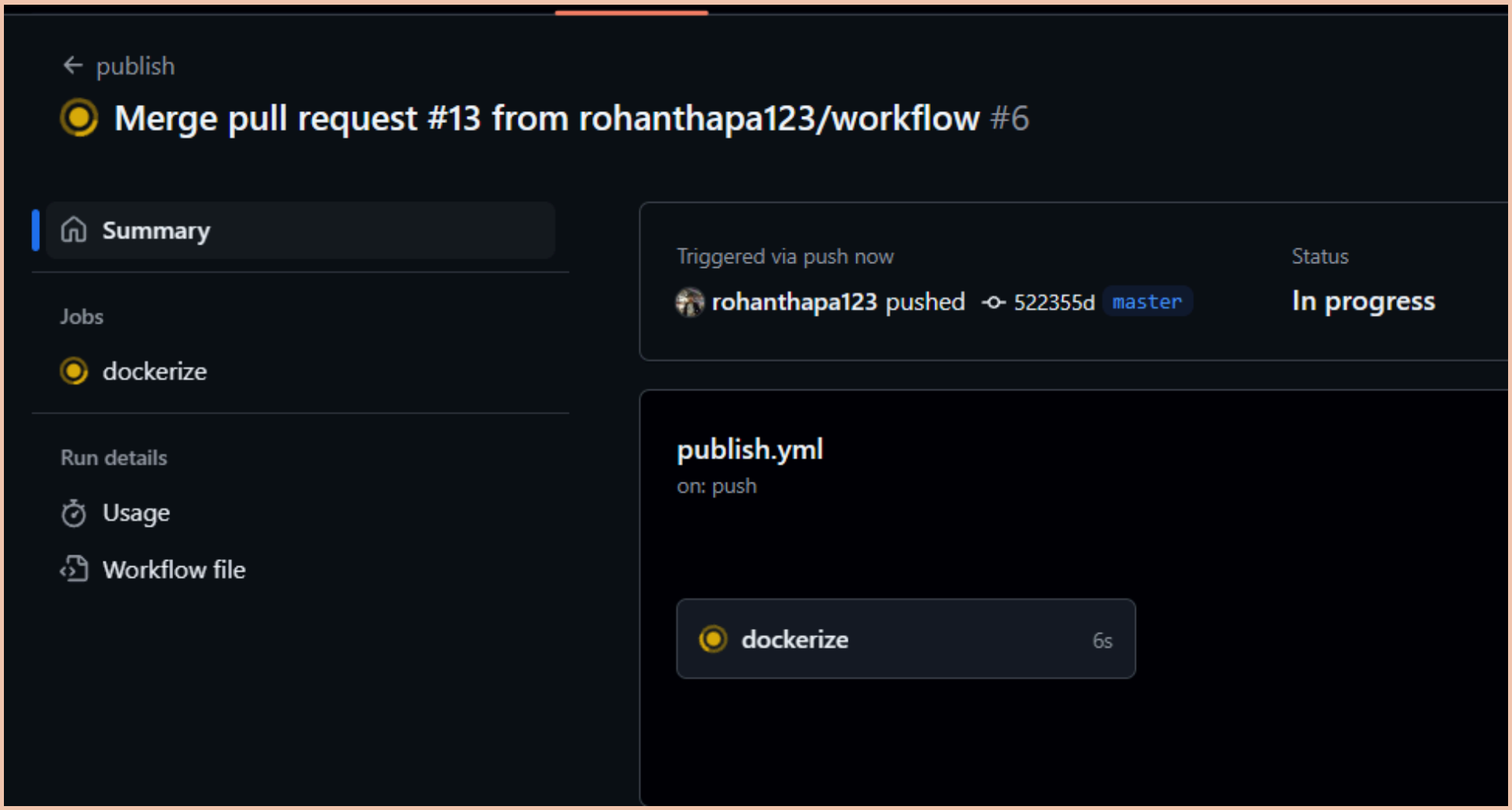
Complete Example with CI/CD

Deployment

```yaml
name: publish

on:
  push:
    branches:
      - master
jobs:
  dockerize:
    runs-on: ubuntu-latest

    env:
      BASE_URL: ${{ secrets.BASE_URL }}
      MONGO_DB: ${{ secrets.MONGO_DB }}
      JWT_SECRETS: ${{ secrets.JWT_SECRETS }}
      MONGO_URI: ${{ secrets.MONGO_URI }}
      GOOGLE_CREDENTIALS: ${{ secrets.GOOGLE_CREDENTIALS }}

    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Build Docker image
        run: docker build -t ${{ secrets.DOCKER_USERNAME }}/notes-management-system .

      - name: Login to dockerhub
        run: echo "${{ secrets.DOCKER_PASSWORD }}" | docker login -u ${{ secrets.DOCKER_USERNAME }} --password-stdin

      - name: Push Docker image
        run: docker push ${{ secrets.DOCKER_USERNAME }}/notes-management-system
```

Complete Example with CI/CD

In Action

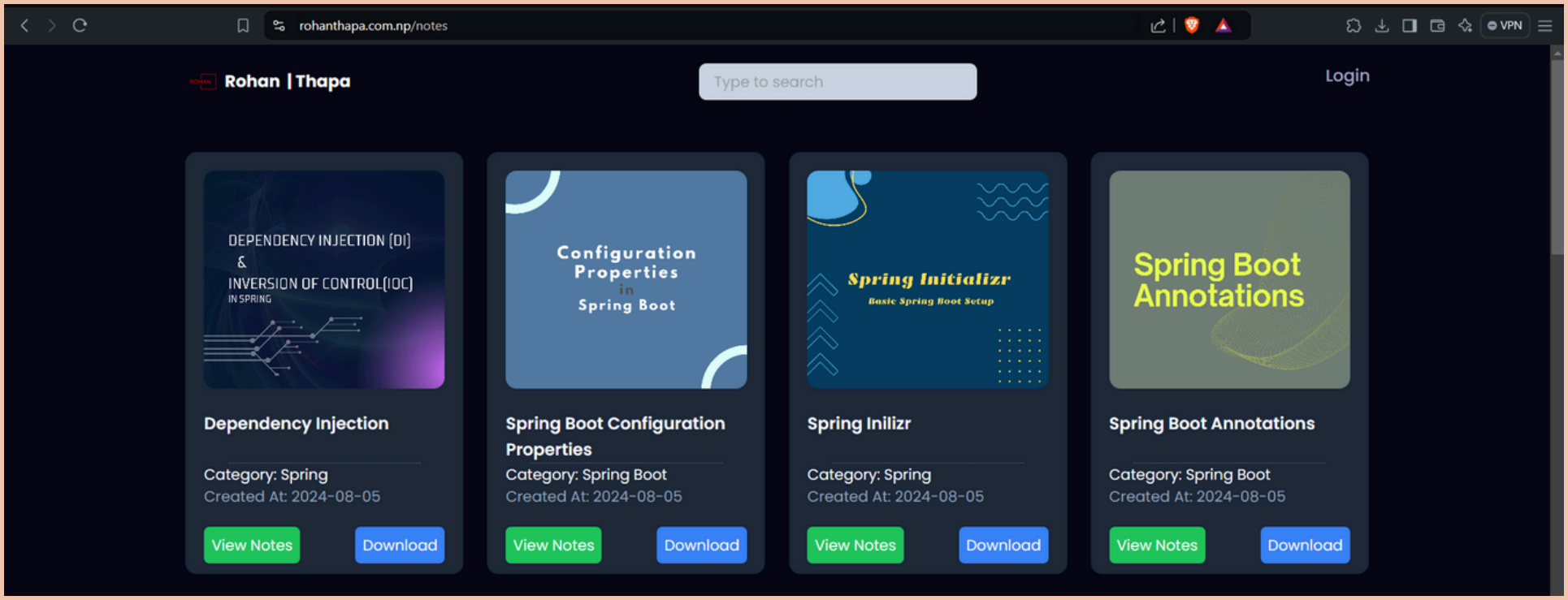


Complete Example with CI/CD

Uploaded to Dockerhub

Tag	OS	Type	Pulled	Pushed
latest		Image	5 minutes ago	5 minutes ago

And like So,
Finally the project is live <3



Best Practices for CI/CD with Github Action

Keep Workflows Modular:

- Break down complex workflows into smaller, reusable steps.

Use Secrets for Sensitive Data:

- Store sensitive information such as API keys or passwords using GitHub Secrets, rather than hard-coding them in YAML files.

Enable Caching:

- Leverage caching to speed up build times for dependencies and artifacts.

Best Practices for CI/CD with Github Action

Monitor & Debug:

- Use GitHub's detailed logs to identify bottlenecks, issues, and failed steps.

Parallelize Jobs:

- Run independent jobs in parallel to speed up the overall workflow.

Advantages of GitHub Actions for CI/CD

- **Deep GitHub Integration:** Native support and tight integration with GitHub repositories.

- **Customizability:** You can write highly customized workflows using YAML.

- **Pre-built Actions:** Thousands of pre-built actions are available in the GitHub Actions marketplace, which can speed up your development.

- **Free Usage Limits:** GitHub provides generous free usage for public and private repositories.

Advantages of GitHub Actions for Spring Boot Projects

- **Easy Integration**: Seamlessly integrates with Spring Boot projects hosted on GitHub.
- **Scalability:** Can handle large-scale projects with multiple jobs running in parallel.
- **Modular Workflows:** Flexibility to create customized CI/CD pipelines.
- **Secure:** GitHub Secrets can store sensitive information safely.
- **Community Support:** Many pre-built actions are available for testing, building, and deploying Spring Boot projects.

Conclusion

GitHub Actions is a versatile tool for automating your **CI/CD** pipeline, simplifying the process from integration to deployment.

Whether you're **building**, **testing**, or **deploying** applications, it provides an easy-to-configure, highly customizable solution for developers.

By leveraging the modularity, event-driven architecture, and integration with GitHub, teams can deliver higher quality code faster and with fewer manual steps.

Thank You

ROHAN THAPA
thaparohan2019@gmail.com