# Continuous Integration
# Continuous Deployment

Group 05 - Topic I

# Members

**22125041**

Mai Đăng Khoa

**22125042**

Lê Mai Khôi

**22125075**

Nguyễn Duy Phúc

**22125084**

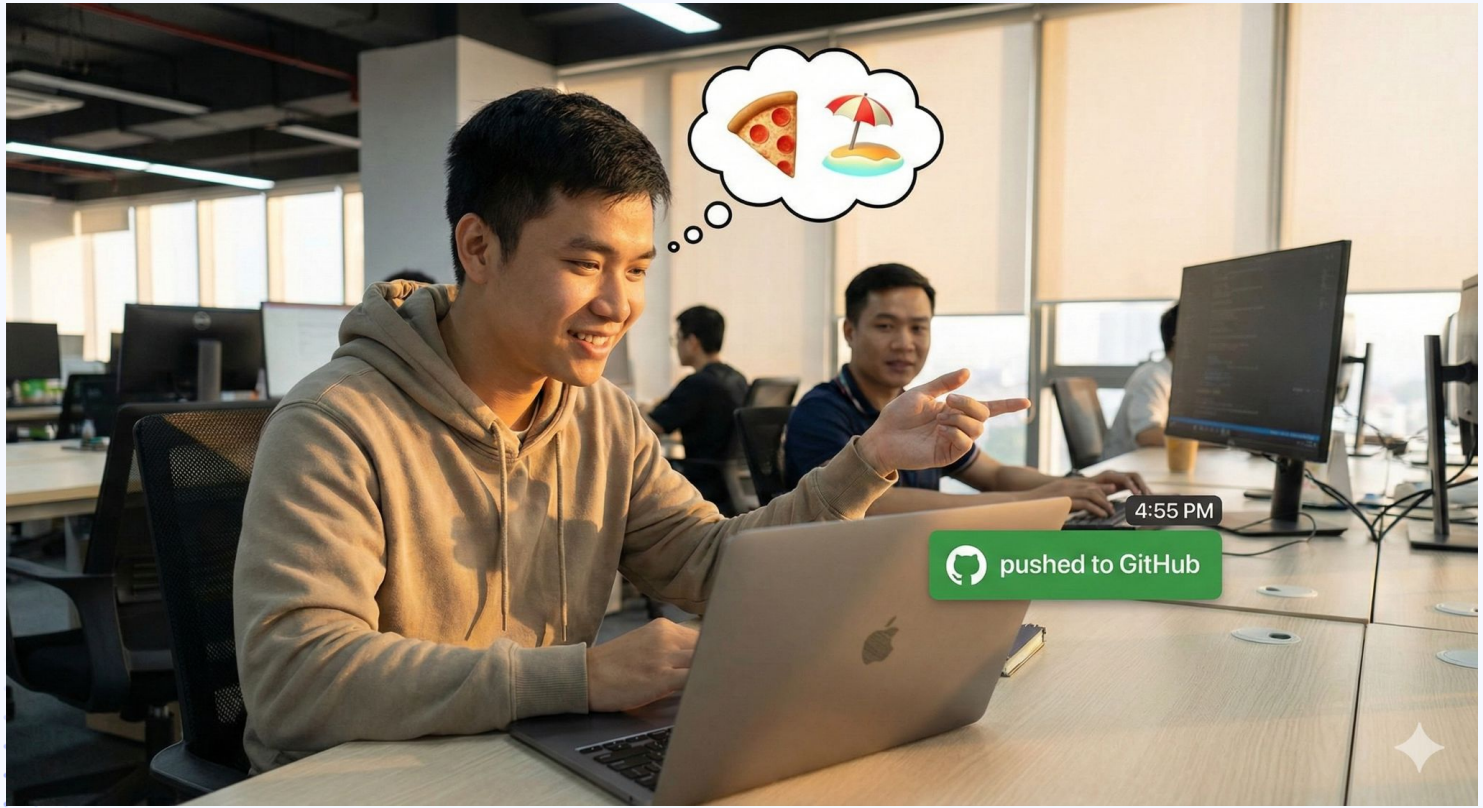
Nguyễn Trọng Quý

# Table of contents

# 01

# Introduction

# The Friday Afternoon Nightmare



# The Friday Afternoon Nightmare

## However, with no CI/CD:

- The application breaks as soon as it runs on the shared server because Quy's laptop had different dependencies installed, and he forgot to commit a configuration change.
- The problem is: nobody noticed. The team assumes everything on main is fine, because there is no automated system rebuilding or retesting the project after each push. And at that time, several more developers push their own changes on top of Quy's commit.

# The Friday Afternoon Nightmare

# The Friday Afternoon Nightmare

## With no CI/CD:

- Because the testing isn't triggered right when the code was pushed, and many developers push code at the same time, it was hard to track the commit that caused failure and delay the whole process.
- Additionally, without CD, there is no pipeline controlling releases, a broken build can be shipped to production by accident, damaging the company's reputation and profits.

# Software Under Test

A simple to-do app developed by our group specifically for this presentation/demo with the assistance of generative AI ✨

2 repositories:

- **Backend** (https://github.com/mdkhoat2/simple-todo-app)
  - Java + Spring Boot
  - PostgreSQL
  - Deployed through Railway
- **Frontend** (https://github.com/duyphuc0701/simple-todo-app-frontend)
  - React
  - Deployed through Cloudflare Pages

# 02 Continuous Integration

All you need to know about CI

# What is Continuous Integration?

**Continuous Integration (CI)** is a development practice where:

- Developers merge changes **frequently**, often several times per day.
- Every merge triggers **automatic builds and tests** to validate the code.
- The goal is to keep the **main branch always stable, functional, and deployable.**
- CI provides **fast feedback**, helping teams detect integration issues early.

# Why do we need CI?

## Integration Conflict

Multiple developers merging at the same time

## Slow manual testing

Bugs detected late, increase cost and effort

**Before CI**

## “It works on my machine”

Runs fine locally but not on other environments

## Unclear Responsibility

Hard to know which commit has bugs

# The 3 pillars of CI

## Source Code Management

Effective source code management using Git

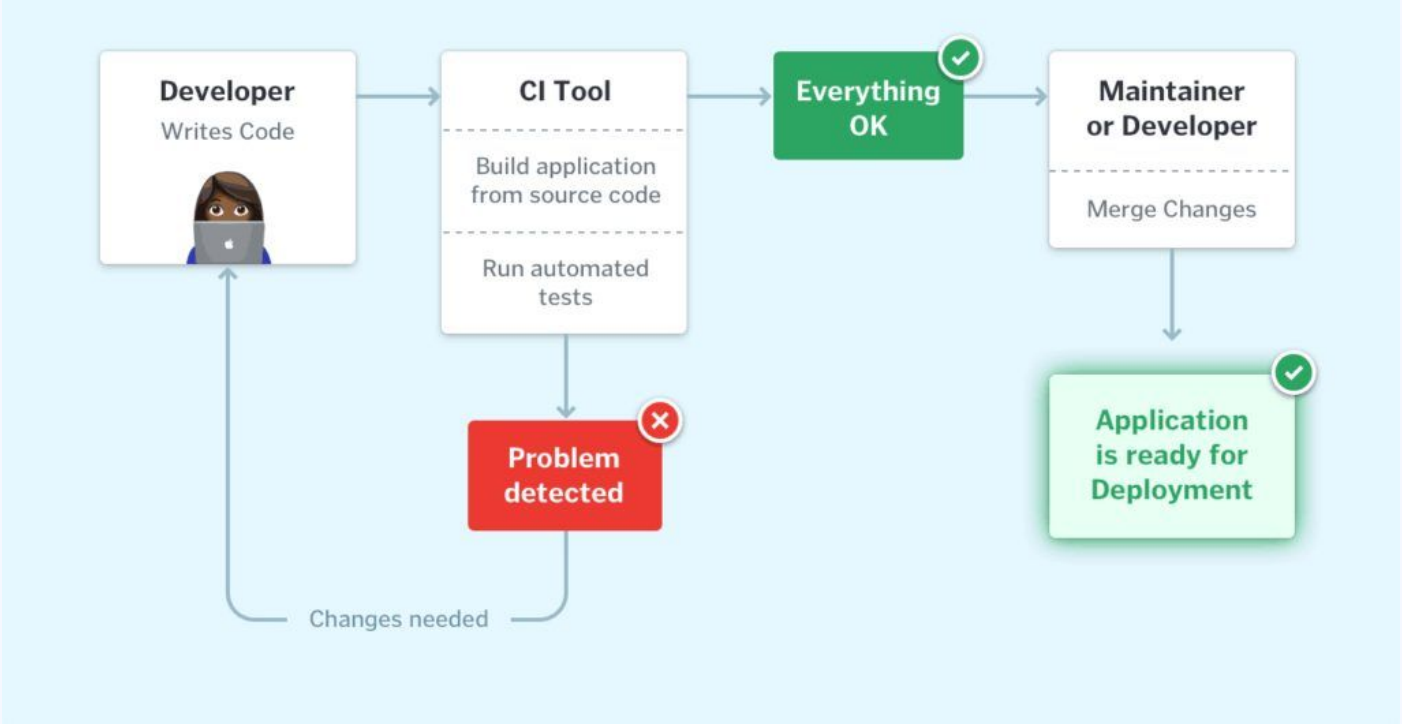

## Automated Build

Builds run automatically on every commit


## Automated Testing

Verifying code automatically

# Complete CI Flow



# Popular CI Tools

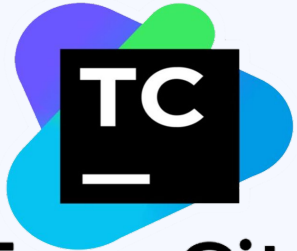
## Cloud-based tools




# Popular CI Tools

## Self-hosted/Enterprise

Jenkins




TeamCity

Bamboo

# GitHub Actions

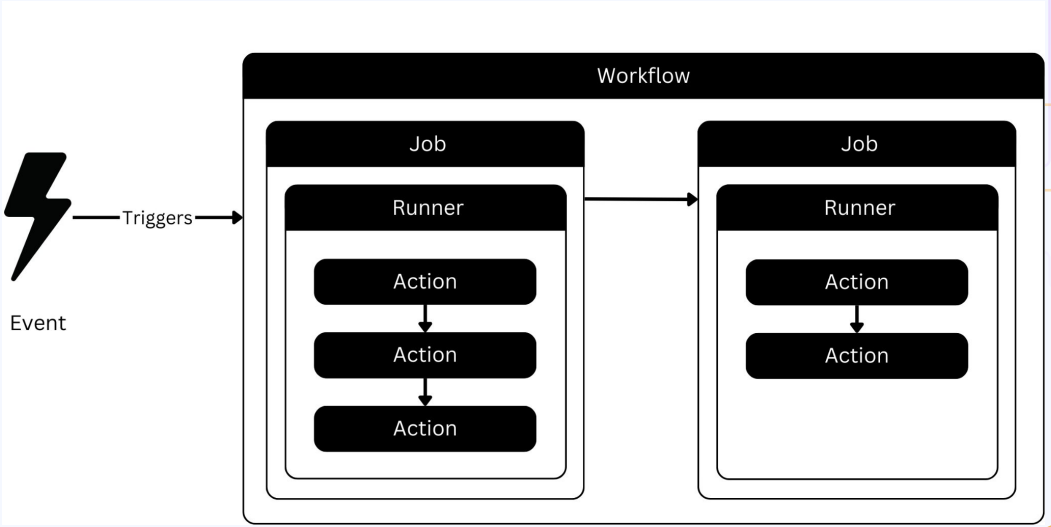
In-depth tutorial about using GitHub Actions

# Github Actions

- Native CI/CD built into GitHub
- YAML-based workflow files
- Triggered by events: Push, Pull Request, Cron
- Provides Linux, Windows, macOS runners
- Zero setup, free for public repos
- Strong ecosystem and many templates

# Github Actions Components

- GitHub Actions automatically scans all YAML files in `.github/workflows/` folder
- Each YAML file describes a WORKFLOW.
- Each WORKFLOW can be broken down into JOBS.
- Each JOB is only triggered if certain conditions are met.



# Structure of a yaml

```yaml
name: Backend CI Pipeline  # 1. Workflow Name

on: ...                    # 2. TRIGGERS (When does it run?)

jobs:                      # 3. JOBS (What to do?)
  build-job:
    runs-on: ...           #    - Where does it run? (Runner)
    steps:                 #    - Sequence of tasks
      - name: ...
      - uses: ...
      - run: ...
```

# The Trigger (on)

```yaml
on:
  # Option 1: Trigger on push to specific branches
  push:
    branches: [ "main", "dev" ]
    paths-ignore: [ "**.md" ]  # Optimization: Ignore if only docs changed

  # Option 2: Trigger when a Pull Request is opened
  pull_request:
    branches: [ "main" ]

  # Option 3: Run on schedule (Cron Syntax)
  schedule:
    - cron: '0 0 * * *'        # Runs at 00:00 every day
```

# The Runner (runs-on)

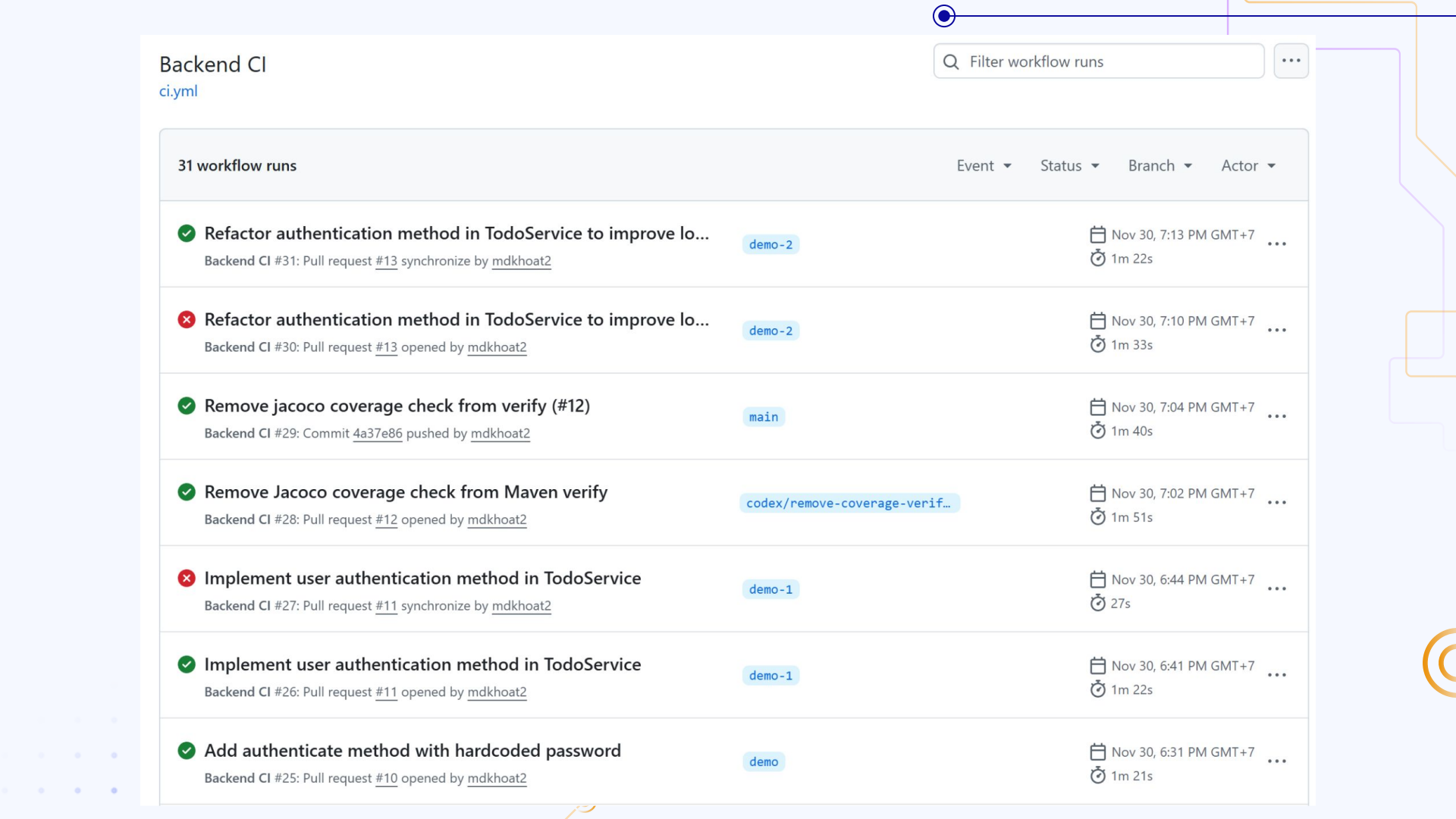
```yaml
jobs:
  test-backend:
    # Choose the Operating System for the Virtual Machine
    runs-on: ubuntu-latest

    # Other available options:
    # runs-on: windows-latest
    # runs-on: macos-latest
```

# Actions vs. Commands (uses vs run)

```yaml
steps:
      # OPTION 1: Use a pre-built Action (Borrowing tools)
      - name: Checkout Code
        uses: actions/checkout@v4  # Action Name @ Version

      - name: Setup Java
        uses: actions/setup-java@v4
        with:                      # Pass parameters (Inputs)
          java-version: '21'
          distribution: 'temurin'

      # OPTION 2: Run Shell Command (Do it yourself)
      - name: Run Maven Test
        run: mvn test              # Standard terminal command
```

# Conditional Execution (if)

```yaml
# Step 3: Run the tests
      - name: Run Tests
        run: mvn test

      # Step 4: Upload Report (Smart Control)
      - name: Upload Test Report
        # --- EXPLANATION OF CONDITIONS ---
        # if: success() -> Default. Runs only if previous step PASSED.
        # if: failure() -> Runs only if previous step FAILED.
        # if: always()  -> Runs REGARDLESS of pass/fail status.
        # --------------------------------
        if: always()
        uses: actions/upload-artifact@v4
        with:
          name: junit-report
          path: target/surefire-reports
```


Backend CI
ci.yml
Filter workflow runs
31 workflow runs
Event
Status
Branch
Actor
Refactor authentication method in TodoService to improve lo...
Backend CI #31: Pull request #13 synchronize by mdkhoat2
demo-2
Nov 30, 7:13 PM GMT+7
1m 22s
Refactor authentication method in TodoService to improve lo...
Backend CI #30: Pull request #13 opened by mdkhoat2
demo-2
Nov 30, 7:10 PM GMT+7
1m 33s
Remove jacoco coverage check from verify (#12)
Backend CI #29: Commit 4a37e86 pushed by mdkhoat2
main
Nov 30, 7:04 PM GMT+7
1m 40s
Remove Jacoco coverage check from Maven verify
Backend CI #28: Pull request #12 opened by mdkhoat2
codex/remove-coverage-verif…
Nov 30, 7:02 PM GMT+7
1m 51s
Implement user authentication method in TodoService
Backend CI #27: Pull request #11 synchronize by mdkhoat2
demo-1
Nov 30, 6:44 PM GMT+7
27s
Implement user authentication method in TodoService
Backend CI #26: Pull request #11 opened by mdkhoat2
demo-1
Nov 30, 6:41 PM GMT+7
1m 22s
Add authenticate method with hardcoded password
Backend CI #25: Pull request #10 opened by mdkhoat2
demo
Nov 30, 6:31 PM GMT+7
1m 21s

# Scheduled Workflows in GitHub Actions

- GitHub Actions workflows do not have to run only on push / pull_request.
- We can also run workflows at a specific time using schedule.
- The schedule uses a cron expression to define:
  - time of day
  - days of week
  - days of month, etc.

```
<minute> <hour> <day-of-month> <month> <day-of-week> <command>
```

Find out more about cron expression at:
https://www.baeldung.com/cron-expressions

```yaml
# Every day at 03:00 (3 AM)
schedule:
  - cron: '0 3 * * *'

# Every Monday at 01:30
schedule:
  - cron: '30 1 * * 1'

# Every 6 hours
schedule:
  - cron: '0 */6 * * *'

# Every Sunday at 00:00
schedule:
  - cron: '0 0 * * 0'
```

# Testing Integration

Integrate many types of software testing into CI workflow

# Linting


- Lint: a tool that checks your code for errors, bad patterns, and style issues.
- ESLint: Lint tool for Javascript/Typescript

How to use it:

- Install ESLint by running: *npm install eslint --save-dev*
- Create config file: *npx eslint --init*
- Run ESLint in workflow: *npx eslint .* (or add a script in package.json as below)

```
{
  "name": "frontend",
  "scripts": {
    "lint": "eslint .",
  },
}
```

```yaml
name: ESLint Check

on:
  push:
  pull_request:

jobs:
  lint:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3

      - uses: actions/setup-node@v3
        with:
          node-version: 18

      - run: npm install
      - run: npm run lint
```

# Functional Testing & Code Coverage

- Functional testing can be integrated and executed at different levels: unit, integration, …
- Testing libraries usually allows measuring code coverage
- Here we show a sample test case using Java and JUnit

```java
class UpdateTodoRequestTest {

    @Test
    void gettersAndSettersWork() {
        UpdateTodoRequest req = new UpdateTodoRequest();
        req.setTitle("New Title");
        req.setCompleted(Boolean.TRUE);
        req.setDueDate("2025-12-31");
        req.setPriority("HIGH");

        assertEquals("New Title", req.getTitle());
        assertEquals(Boolean.TRUE, req.getCompleted());
        assertEquals("2025-12-31", req.getDueDate());
        assertEquals("HIGH", req.getPriority());
    }
}
```

# Functional Testing & Code Coverage

- To integrate functional testing to GitHub Action workflow:
  - Define a job with steps: Checkout, Setup Java, Build, Run tests, Upload report
  - Specify actions, conditions, commands, ... for each step
- To view the code coverage report, download the artifact file from GitHub Actions

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

    - name: Set up JDK 24
      uses: actions/setup-java@v4
      with:
        distribution: temurin
        java-version: "24"
        cache: maven

    - name: Build and run tests with Maven
      run: mvn -B -DskipTests=false verify

    - name: Upload JaCoCo HTML report
      if: always()
      uses: actions/upload-artifact@v4
      with:
        name: jacoco-report
        path: target/site/jacoco
```

# UI Testing

```
import { test, expect, enterUserName } from './fixtures';

test.describe('Todo App - Name Entry', () => {
  test('should display name entry on initial load', async ({ page, mockApiServer }) => {
    await page.goto('/login');
    await expect(page.locator('text=Welcome to Todo App')).toBeVisible();
    await expect(page.locator('input[placeholder="Your name..."]')).toBeVisible();
    await expect(page.locator('button:has-text("Start")')).toBeVisible();
  });
  ...
});
```

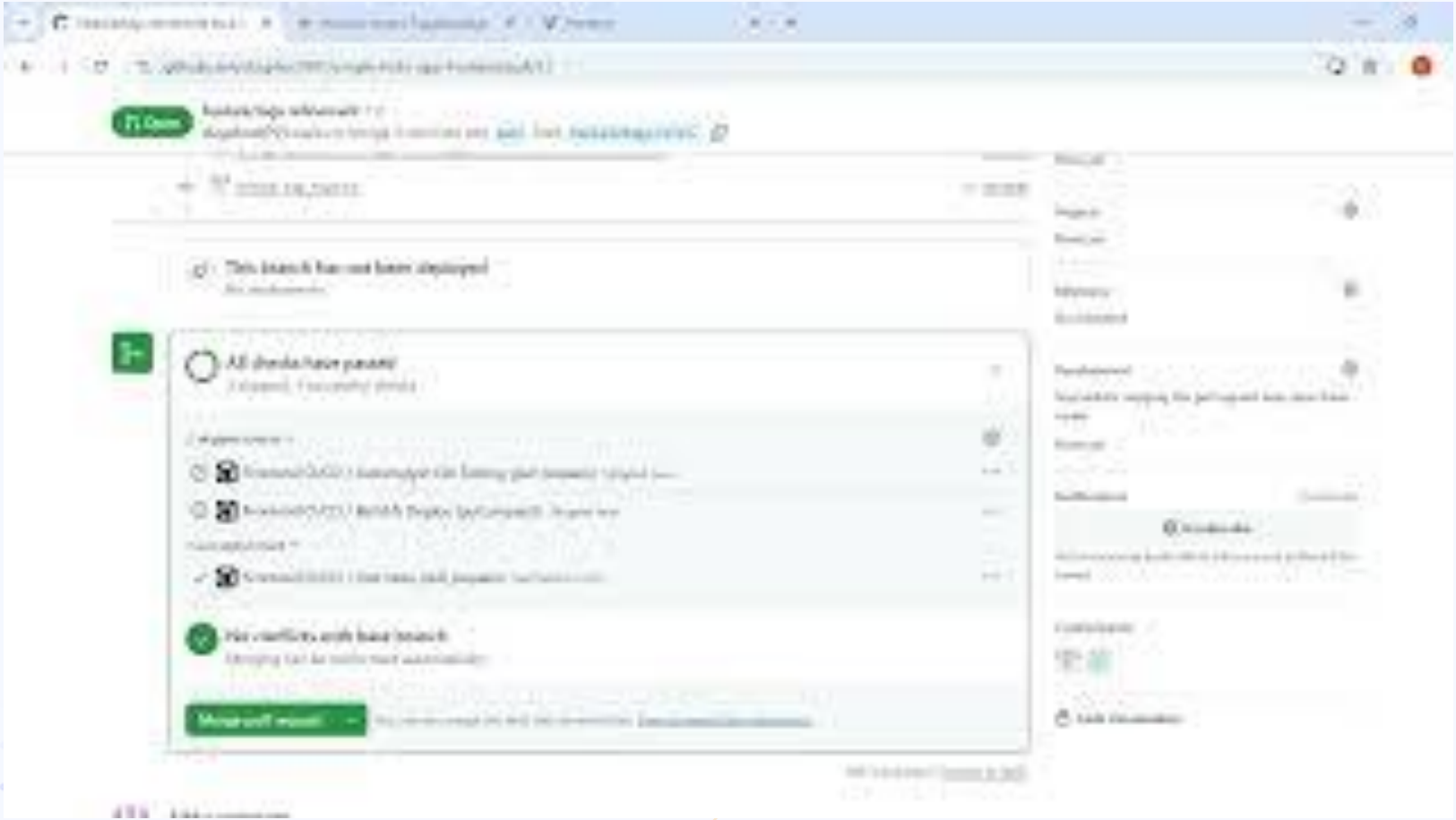
This is a test case for UI testing using Playwright. You can also use other libraries such as Selenium, Cypress, ...

# UI Testing

- To integrate UI testing to GitHub Action workflow:
  - Define a job with steps: Checkout, Setup Node, Install dependencies, Install Playwright, Run tests, Upload report
  - Specify actions, conditions, commands, … for each step
- To view the Playwright report, download the artifact file from GitHub Actions

```yaml
- name: Install dependencies
  run: npm ci

- name: Install Playwright Chromium
  run: npx playwright install chromium

- name: Run Playwright tests
  run: npx playwright test --project=chromium

- name: Upload Playwright report
  if: always()
  uses: actions/upload-artifact@v4
  with:
    name: playwright-report
    path: playwright-report/**
    if-no-files-found: ignore
    retention-days: 7
```

# CI Demo Video



# What CI Automates

**CI automates many tasks and different types of testing:**

- Build and compile
- Functional Testing at different levels: unit, integration, ...
- Linting, formatting
- Security & Dependency scanning
- Code coverage
- UI Testing
- Reporting + notifications
- Artifact packaging (JAR, Docker image, ZIP, etc.)
- Documentation generation

# Benefits of CI

## Faster Feedback

Devs know within minutes if code breaks

## Higher Quality

Tests + lint + scans run on every commit

## Reduced Risk

Frequent integration prevents large conflicts


## Better Traceability

Clear history of who changed what


## Enhanced Collaboration

Devs can merge code change frequently


## Stable main branch

Always deployable, ready for CD

# 03

# Continuous Delivery & Deployment

All you need to know about CD

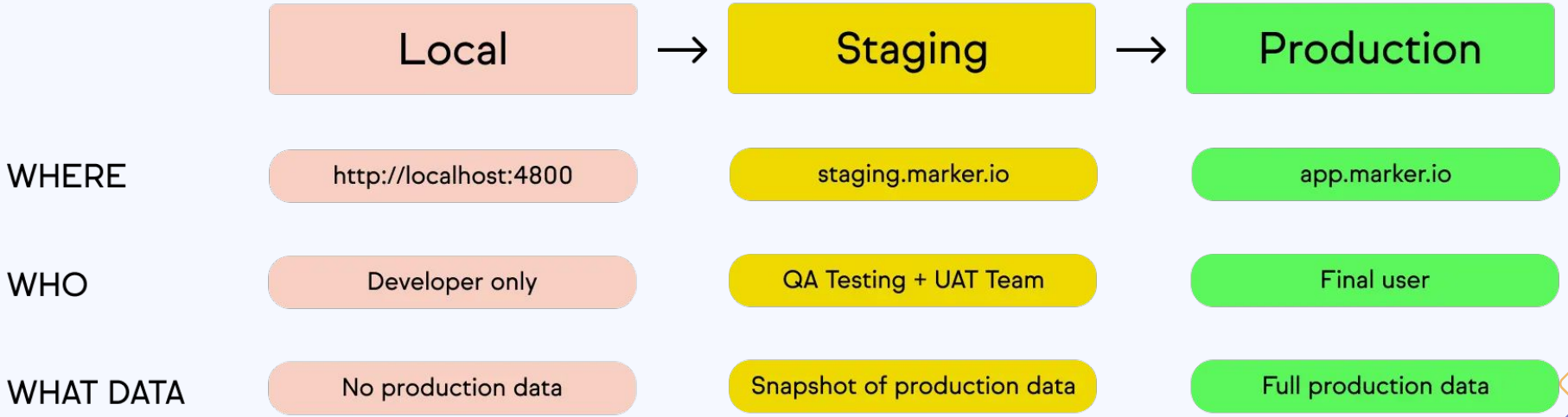
# What is Continuous Delivery?

- **Definition:** a software engineering practice in which code changes are prepared to be released to production
- **Manual Control:** there is a GUI control that requires human to manually approve and trigger the deployment to production
- **Purpose:** Acts as a final "failsafe" to prevent the team from deploying faulty code or incorrect versions to actual users

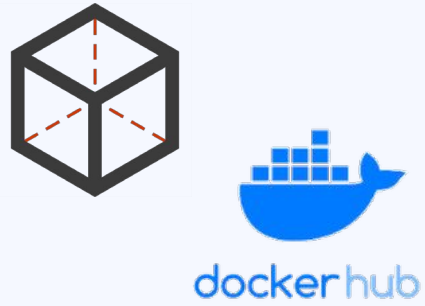
# What is Continuous Deployment?

- **Definition:** a software engineering practice that ensures code changes are continuously released into the production environment
- **Key Difference:** It removes the final manual failsafe and the human element found in Continuous Delivery
- **Prerequisites:** Requires a mature, trusted CI pipeline to ensure code is good enough for direct customer release
- **Benefit:** Save time for companies that have established a high level of trust in their automated testing

# Environments



# Deployment Platforms




Container Registries (GitHub Container Registry, Docker Hub, ...)


Public Cloud Providers Services (AWS, Azure, GCP, ...)

On Premise Servers

# Deploy Docker image

- To build and push Docker image to GitHub Container Registry:
  - Define a job with steps: Login to GitHub, Extract metadata, Build and push image
  - Specify actions, conditions, commands, … for each step
- Go to the Registry section in GitHub to view the Docker images

```yaml
- name: Log in to GitHub Container Registry
  uses: docker/login-action@v3
  with:
    registry: ghcr.io
    username: ${{ github.actor }}
    password: ${{ secrets.GITHUB_TOKEN }}
- name: Extract metadata (tags, labels)
  id: meta
  uses: docker/metadata-action@v5
  with:
    images: ghcr.io/${{ github.repository }}
    tags: |
      type=sha
      type=raw,value=latest
- name: Build and push
  uses: docker/build-push-action@v6
  with:
    context: .
    push: true
    tags: ${{ steps.meta.outputs.tags }}
    labels: ${{ steps.meta.outputs.labels }}
```

# Deploy Frontend to Cloudflare

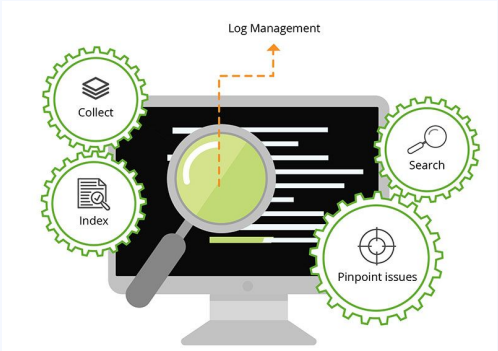
```yaml
- name: Install & Build
  env:
    VITE_API_BASE_URL: ${{ secrets.PROD_API_BASE_URL }}
  run: |
    npm ci
    npm run build

- name: Deploy to Cloudflare Pages
  uses: cloudflare/wrangler-action@v3
  with:
    apiToken: ${{ secrets.CLOUDFLARE_API_TOKEN }}
    accountId: ${{ secrets.CLOUDFLARE_ACCOUNT_ID }}
    command: pages deploy ./dist --project-name=simple-todo-app-frontend --branch=${{ github.ref_name }}
    gitHubToken: ${{ secrets.GITHUB_TOKEN }}
```

# CD Demo Video

# Continuous Monitoring

## Metrics

Measuring quantitative data (CPU usage, Memory, Latency) in real-time.




## Logs

Essential for troubleshooting "root cause" when bugs occur.

## Alerting

Automatically notifying (Slack, Email, or PagerDuty) when metrics exceed thresholds.

# Benefits of CD

## Frequent & Small Releases

Release often with fewer changes per update

## Increased Productivity

Not wasting time on manual deployments

## Lower Release Risk

Easier to fix or rollback than "Big Bang" releases


## Faster Value & Feedback

Validate ideas instantly with real user


## Boosted Confidence

Staging environments acts as a practice run


## Stress-Free Deployments

Routine event, not a "Friday nightmare"

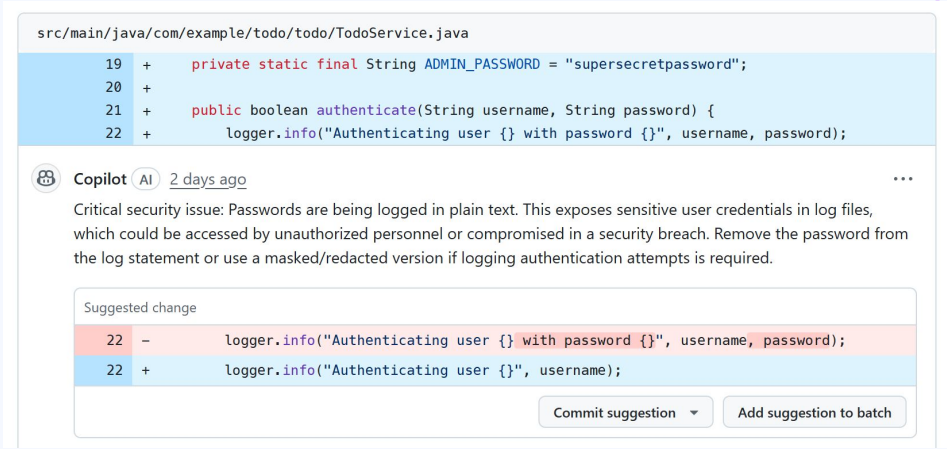
# 04

# The AI-First Approach

From “Automated” to “Intelligent”

# Copilot

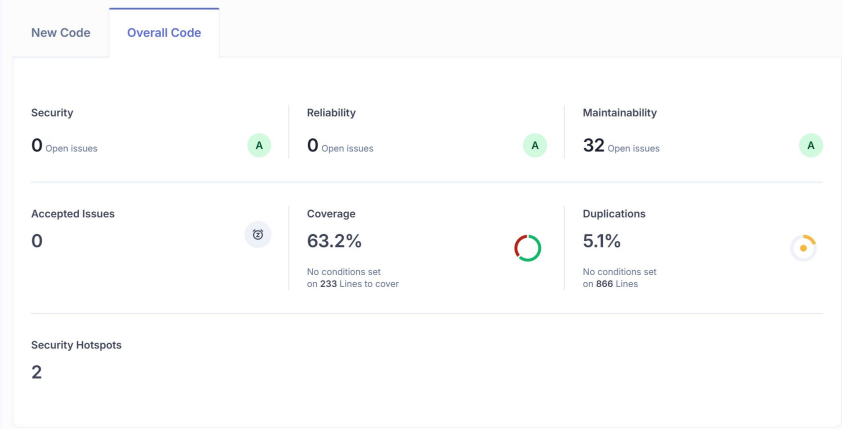
Copilot analyzes code intent, predicts logical errors, and suggests readability improvements

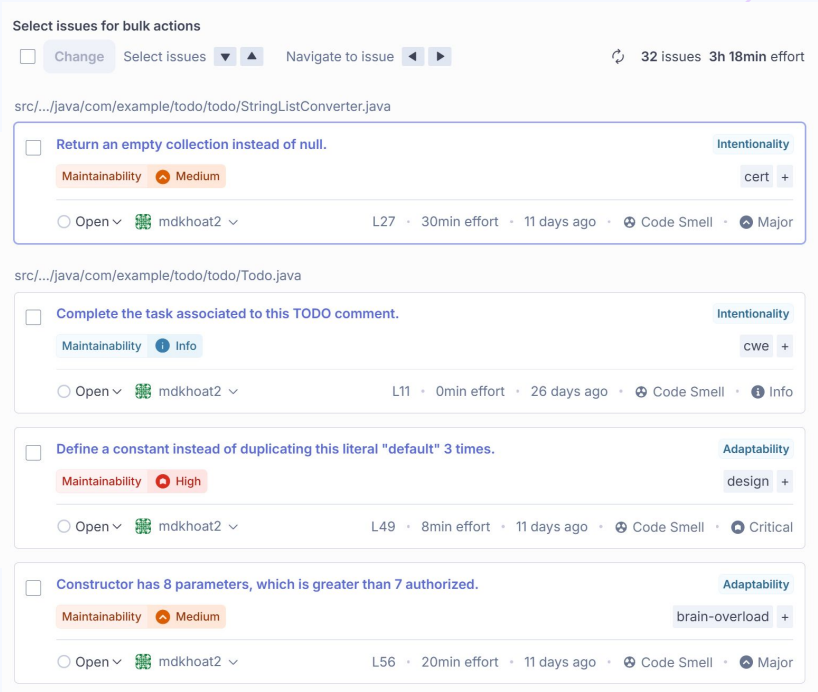



# SonarQube

SonarQube

Explaining vulnerabilities in plain language and suggesting code fixes instantly.




PR Summary
Failed
3
0
0.00%
0.0%

# 05

# Conclusion

Summarize key points of the presentation

# Key Takeaways

01


**CI/CD**

It’s a Culture,
Not Just Tools

02


**Work**

Automate to
Accelerate

03

**Pipeline**

Start Small,
Improve Daily

“The goal of CI/CD is not just to release software faster, but to sleep better at night.”

—**Someone** **(in our group)**

# Thanks!

## Do you have any questions?


nguyenquyking@gmail.com

+84 77 470 864

nguyenquyking.com