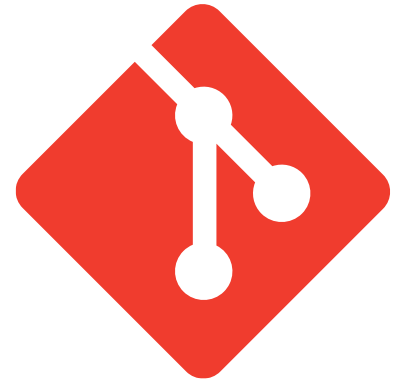


Git and CI/CD Platforms

Mohammed-Ammar Hassnou

Seminar "StayInSync" –
16.05.2025 StuPro - Universität Stuttgart

Agenda

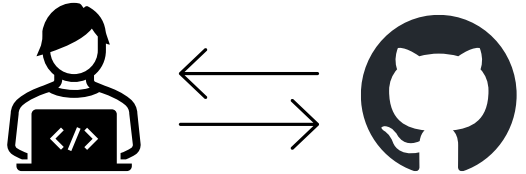
Presentation Outline:

1. Introduction to Git and Hosting Platforms
2. Understanding CI/CD Concepts
3. Git Hosting Platforms and CI/CD Integration
4. Comparison: GitHub, GitLab, Bitbucket
5. Configuring Workflows: Branch strategies and Triggers
6. Commit Templates, and Best Practices
7. CI/CD Demo: GitHub Actions
8. Platform Landscape & Reflection
9. Summary and Platform Recommendations
10. Q&A

1. Intro to Git Hosting Platforms

1. Intro to Git Hosting Platforms

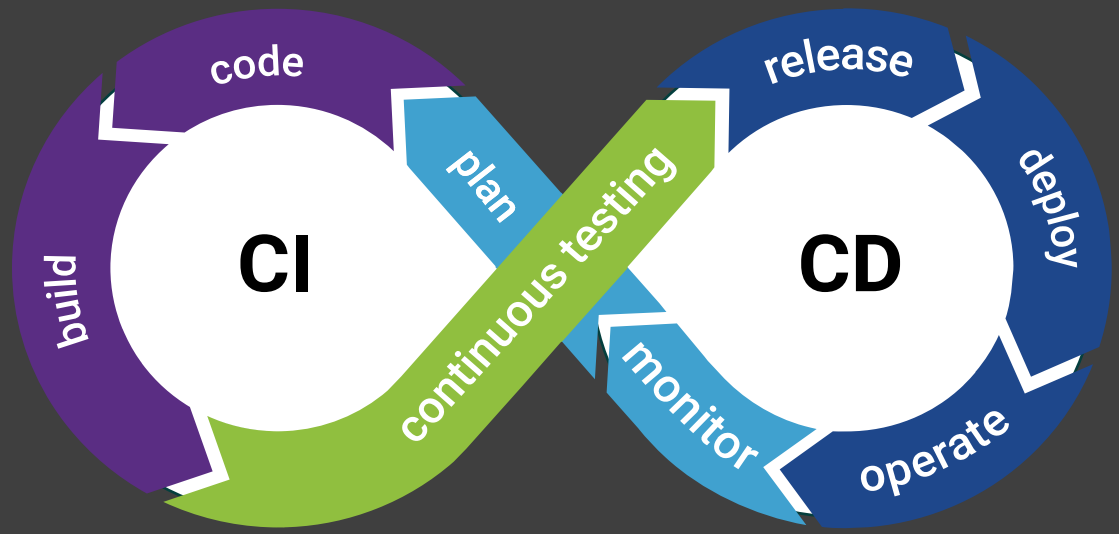
- **Git** is a distributed **version control system** for tracking code changes
- **Git** platforms provide remote repositories, collaboration tools, and CI/CD features
- Most popular **Git** platforms:
  - GitHub (by Microsoft)
  - GitLab (Open-core platform)
  - Bitbucket (by Atlassian)
- Key features of **Git** hosting platforms:
  - ***Repository management*** (branches, forks, merges)
  - ***Pull/Merge Requests*** for team collaborations
  - ***Issue tracking***, project boards, wikis
  - ***CI/CD pipelines*** integration for automation



1.Understanding CI/CD Concepts

2. Understanding CI/CD Concepts

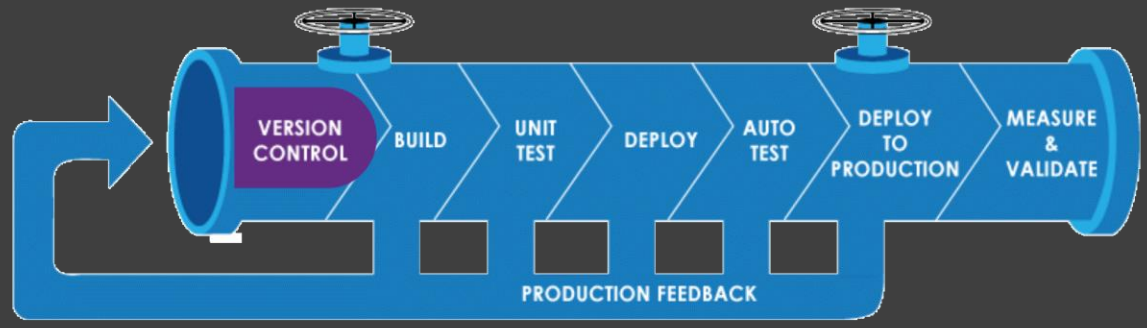
- **CI (Continues Integration)**
  - Automatically builds and tests code after every commit or push
  - Detects issues early --> better code quality and integration
- **CD (Continues Delivery/Deployment)**
  - Delivery = automatically preparing code for release.
  - Deployment = automatically releasing code to production
- **Benefits of CI/CD**
  - Faster development cycles
  - Early bug detection
  - Consistent and repeatable deployment
  - DevOps culture support



2. How does CI/CD work technically ?

- **Workflow file:**
  - CI/CD is defined in a YAML file in the repository
- **Trigger:**
  - Defines when the pipeline starts: e.g push, pull_request, or manual
- **Stages:**
  - High-level phases like build, test, deploy
  - Run in order, can be skipped or conditional
- **Jobs:**
  - Each stage contains one or more jobs
  - A job = unit of work executed on a runner
- **Steps (GitHub only)**
  - A job is made of steps → (commands like install, run tests, deploy etc..)

- **Runners:**
  - Virtual machines or Docker containers that run the jobs
- **Image:**
  - Specifies the base environment (eg. Python: 3.11)



3. Git Hosting Platforms and CI/CD Integration

3. What is GitHub ?

- The most widely used Git hosting Platform, owned by Microsoft
- It enables developers to host, share, and collaborate on code via Git repositories
- Widely adopted by:
  - Open-source communities
  - Enterprise and startups
  - Individual developers
- Core features:
  - Repositories (public/private)
  - Branching, Pull Requests
  - Issues, Labels, Project Boards, Milestones
  - Wikis and Discussions
- Integration:
  - Deep integration with Visual Studio Code, Copilot, Azure
  - GitHub Marketplace for tools, bots, and extensions

3. CI/CD with GitHub

- **GitHub Actions** is GitHub's **native CI/CD platform**.
- It lets you **automate workflows** like:
  - Build → Test → Deploy
- Workflows are defined in **.yml files** inside:
  - .github/workflows/your-workflow.yml
- **Key Concepts:**
  - **Trigger:** defines when a workflow runs (e.g., push, pull_request)
  - **Jobs:** logical steps grouped together (e.g., test, deploy)
  - **Steps:** individual commands/scripts within a job
  - **Runners:** virtual machines (Ubuntu, Windows, macOS)

- Example Workflow:

```yaml
name: Build and Test
on: [push]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - run: npm install && npm test
```

3. What is GitLab ?

- **GitLab** is an **open-core DevOps platform** that combines Git hosting with CI/CD, security, and monitoring tools.
- Available as:
  - **GitLab.com**
  - **Self-managed / self-hosted version**
- **Core Features:**
  - Repositories, Merge Requests (MRs), Issues, Boards, Milestones
  - Built-in Container Registry and Helm Charts support
  - Wiki and built-in Documentation
- **All-in-One DevOps Toolchain:**
  - Git hosting + CI/CD + Testing + Code Review + Monitoring in a single platform
- **Popular in companies** that want:
  - Full DevSecOps lifecycle in one place
  - Full control via self-hosting

3. CI/CD with GitLab

- **GitLab CI/CD** is natively integrated – no external setup required.
- **Pipelines are defined in .gitlab-ci.yml** in the project root.
- **Key Components:**
  - **Stages**: e.g., build, test, deploy
  - **Jobs**: defined per stage, executed in runners
  - **Runners**: GitLab's execution agents (can be shared or self-hosted)
- **Execution Logic:**
  - Each commit can trigger the pipeline
  - Jobs run based on stage order, parallelized where possible

Example:

```yaml
stages:
  - build
  - test

build:
  stage: build
  script:
    - make build

test:
  stage: test
  script:
    - make test
```

3. What is Bitbucket ?

- **Bitbucket** is a Git repository management platform developed by **Atlassian**, the makers of Jira, Trello, and Confluence.
- It is tightly integrated into the **Atlassian ecosystem**, making it ideal for agile development teams using Jira for issue tracking.
- **Core Features:**
  - Git repository hosting
  - Pull Requests with inline code review and comments
  - Branch permissions, pipelines, and deployment tracking
  - Snippet sharing and built-in issue tracking
- **Best suited for:**
  - Teams already using **Jira/Confluence**
  - Agile workflows with strong project management integration

3. CI/CD With Bitbucket

- Bitbucket Pipelines is Bitbucket's **native CI/CD service**, tightly integrated into its interface.
- Requires a single file: **.bitbucket-pipelines.yml** in the root of your repository.
- **Key Concepts:**
  - Uses **Docker containers** to run builds and scripts
  - Supports multiple steps, parallel builds, and deployment environments
- **Workflow:**
  - Triggered by events like push, pull request, or manual trigger
  - Each step runs in a clean Docker container
  - Predefined templates for Node.js, Python, Java, etc.
- **Integrations:**
  - **Jira integration**: link commits and deployments to issues
  - Deployments can be tracked across environments

Example:

```yaml
pipelines:
  default:
    - step:
        name: Build and Test
        image: node:16
        script:
          - npm install
          - npm test
```

4. GitHub vs. GitLab vs. Bitbucket

GitHub vs. GitLab vs. Bitbucket

Feature	GitHub	GitLab	Bitbucket
CI/CD Integration	GitHub Actions	GitLab CI/CD	Bitbucket pipelines
Hosting Model	Cloud only	Cloud + Self-host	Cloud only
Merge Workflow	Pull Requests	Merge Requests	Pull Requests
Best For	Open-source, teams	Full DevOps flow	Jira integration
Pricing	Free tier w/limits	Free + premium tiers	Free tier w/limits

Key Takeaways:

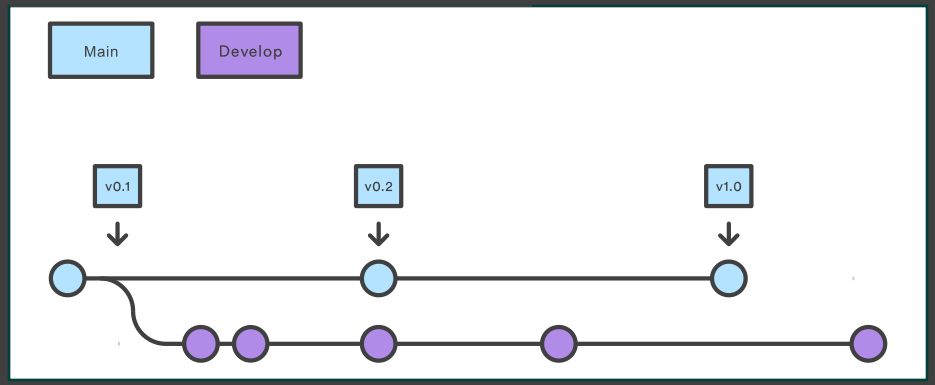
- **GitHub** is ideal for open-source and community projects (Massive open-source community, largest ecosystem of actions, integration with VS Code and Copilot)
- **GitLab** offers a more integrated DevOps experience (Native DevOps lifecycle in one tool (SCM + CI/CD + Monitoring, strong self-hosting options, Auto DevOps)
- **Bitbucket** is preferred in Atlassian ecosystems (Tight integration with Atlassian tools (Jira, Trello), good for agile workflows in enterprise teams)

5. Configuring Workflows: Branch Strategies and Triggers

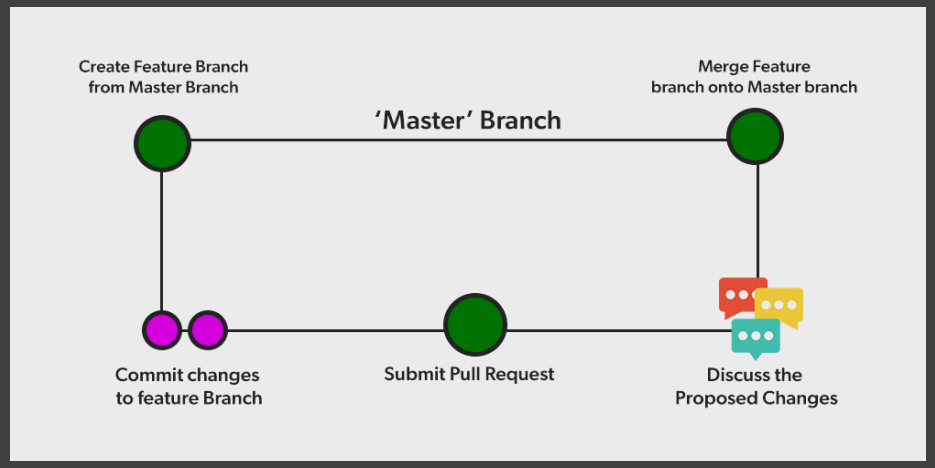
Configuring Workflows: Branch Strategies

- In collaborative development, using a **branching strategy** is essential for organizing work and managing changes.
- A **branch strategy** defines:
  - How branches are named
  - When branches are created/merged
  - Who merges into main branches
- Most CI/CD workflows rely on a clear **branching model** to trigger pipelines and releases.

Example: 1. Git flow:

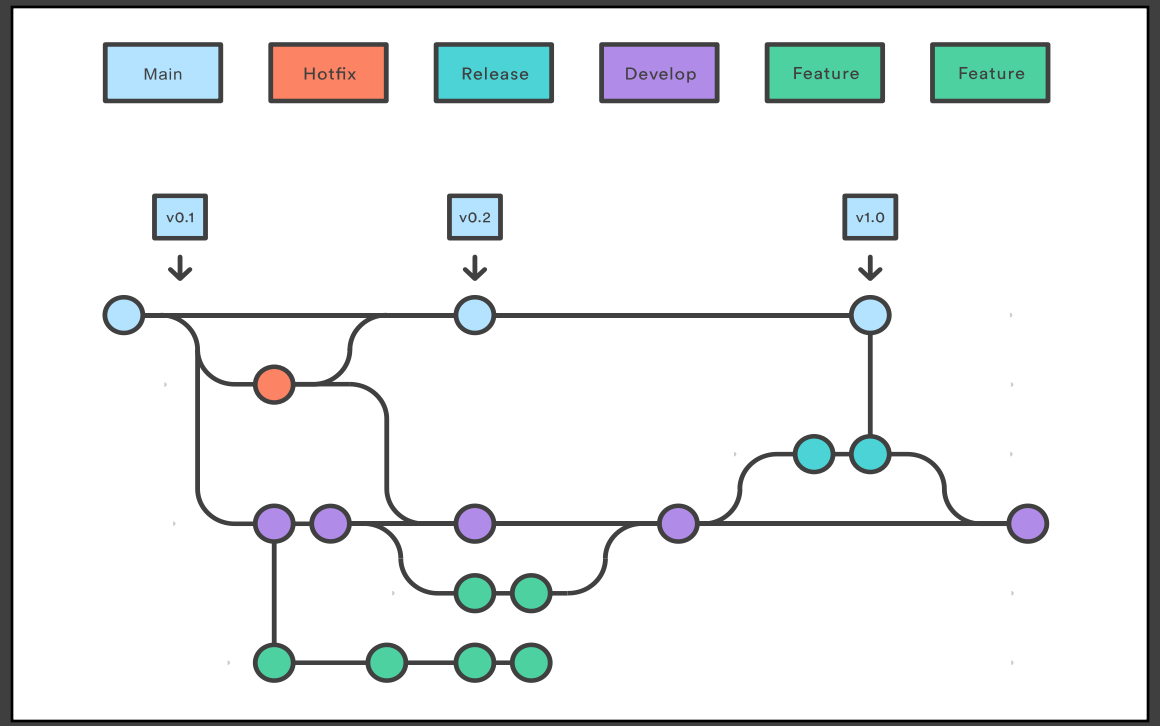


Example: 2. GitHub flow:



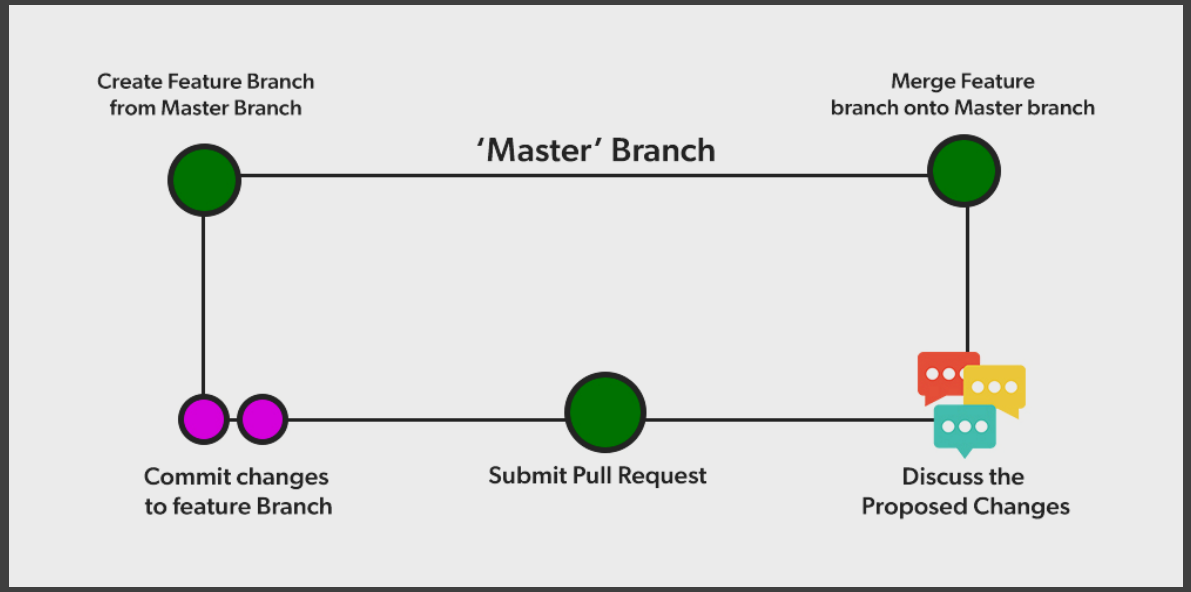
Branch Strategies: Git Flow

- **Structure:**
  - **main**: production-ready code only – always stable
  - **develop**: active development – base for new features
  - **feature**: for new features (branched from develop, merged back when done)
  - **release**: used to prepare a new release (e.g., release/1.2)
    - Bug fixes, version bumps, etc.
  - **hotfix**: urgent fixes to main (e.g., hotfix/1.2.1)
    - Also merged into develop to stay in sync
- **Advantages:**
  - Stable release process
  - Good for applications with scheduled releases
  - Clearly separates feature development from production
- **Challenges:**
  - Heavy structure, more merges
  - Slower iteration for small teams
- **Best for:**
  - Large teams
  - Long-term Projects



Branch Strategies: GitHub Flow

- **Structure:**
  - Single main branch (always deployable)
  - Create short-lived feature branches from main
  - Pull request → Code review → Merge to main → Auto-deploy
- **Advantages:**
  - Simple, fast, and easy to automate
  - Great with GitHub Actions
  - Ideal for modern CI/CD pipelines
- **Challenges:**
  - Less structured for versioning or large releases
  - Needs discipline to avoid breaking production
- **Best for:**
  - Teams using Continuous Delivery (CD), fast iterations



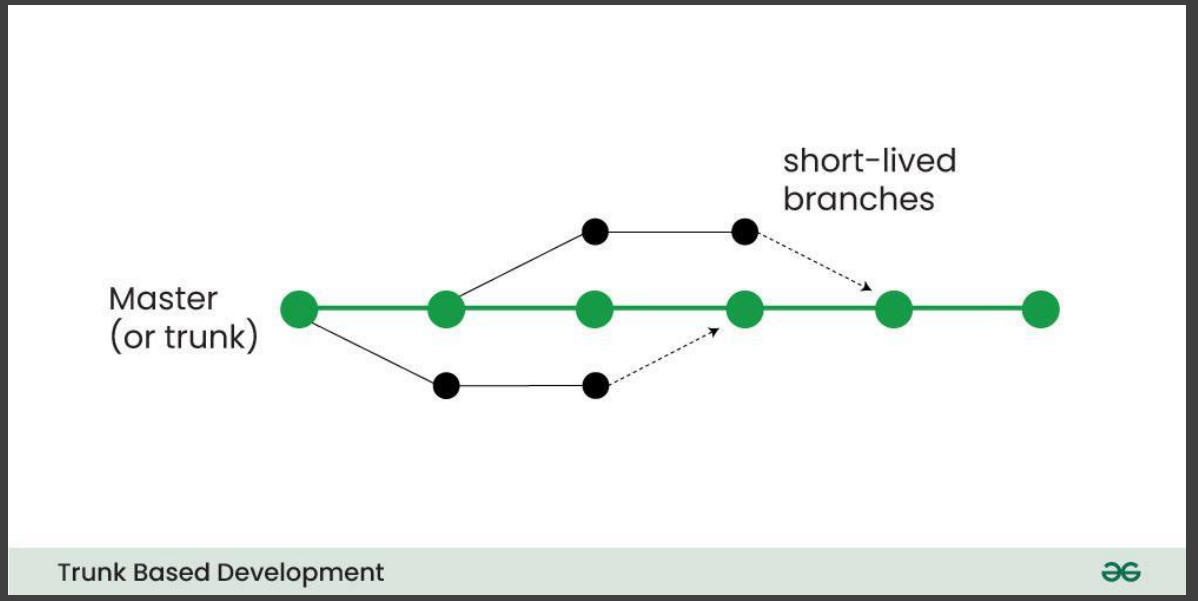
Branch Strategies: Trunk-Based Development

- Structure:
  - One central main (or trunk) branch
  - Developers push changes frequently – often multiple times per day
  - Short-lived branches allowed (max 1-2 days)

- Advantages:
  - Encourages rapid integration
  - Reduces merge conflicts
  - Enables frequent releases

- Challenges:
  - Requires strong testing + automation
  - Riskier without discipline

- Best for:
  - High performance teams
  - Continuous integration



Configuring Workflows: Triggers in CI/CD

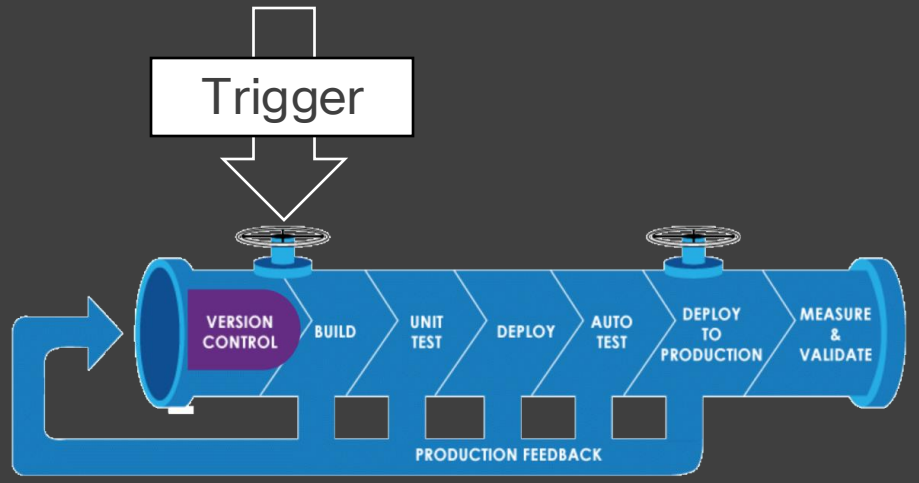
- A **trigger** is an event that **starts a pipeline**.
- **Triggers** define **when** a CI/CD workflow should run (build, test, deploy).
- Supported in all major CI/CD systems (GitHub Actions, GitLab CI, Bitbucket Pipelines).

Common Trigger Types:

1. **Push**
   - Triggered when code is pushed to a branch.
   - Most common trigger for testing changes automatically.
2. **Pull/Merge Request**
   - Triggered when a pull/merge request is opened, updated, or merged.
   - Used to validate code before merging (e.g., run tests, check style).
3. **Schedule**
   - Time-based trigger (e.g., daily build at 02:00 AM).
   - Useful for maintenance jobs, backups, or monitoring.
4. **Manual**
   - Triggered by a developer manually (via UI or API).
   - Used for sensitive steps like production deployment.
5. **Tag/Release**
   - Triggered when a Git tag is pushed (e.g., v1.0.0).
   - Used to publish releases, create packages, deploy to production.
6. **Webhook/API**
   - Triggered via external service or event (e.g., after a successful build elsewhere).

Best Practice:

✓ Combine triggers with conditions (e.g. only run on main or develop) to control pipeline flow.



6. Commit Templates and Best Practices

6. Commit Templates and Best Practices

- Clear and structured **commit messages** are essential for collaboration, CI/CD workflows, and release automation.
- **Commit templates** help enforce consistency and improve readability.
- Popular format: **Conventional Commits**

Why use Commit Templates ?

- Easier **code reviews**
- Better **automated changelogs**
- Clean **Git History**
- Required by some **CI/CD tools** (GitHub Actions workflows)

Structure of Conventional Commits:

<type>[optional scope]: <description>

[optional body]

[optional footer(s)]

Examples:

- Feat(login): add OAuth2 login option
- Fix(api): handle 500 errors correctly
- Docs(readme): update usage instructions

Common Types:

Type	Meaning
Feat	New feature
Fix	Bug fix
Docs	Documentation only change
Style	Code formatting only change
Refactor	Code refactor
Test	Adding or fixing tests
Chore	Minor changes, tooling, config

6. Commit Templates and Best Practices

- Git allows using a **template file** to guide commit messages.
- This helps enforce a consistent structure across teams.
- Templates are pre-loaded in the editor when running git commit.

Benefits:

- ✓ Enforces clear & structured messages
- ✓ Useful with **Conventional Commits**
- ✓ Improves readability & changelogs

How to Setup:

1. **Create a template file** (e.g. ~/.gitmessage.txt):

```
type(scope): short summary

# Details:
# write the details here

Refs: #
```

2. **Configure Git globally:**

```
git config --global commit.template ~/.gitmessage.txt
```

3. **Running git commit will look something like this:**

```
type(scope): short summary


# Details:
# write the details here


Refs: #


# ------------------------ >8 ------------------------
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch main
# Changes to be committed:
#   modified:   src/app.js
```

6. Commit Templates and Best Practices

Best Practices for Commit Messages:

- **Use the imperative mood**
  → *"Add login validation"* instead of *"Added login validation"*
- **Keep the subject line under 50 characters**
  → Concise summary helps with readability
- **Separate subject from body with a blank line**
  → Follows the conventional commit format
- **Explain the "why" and not just the "what"**
  → Makes history more meaningful
- **Use present tense**
  → Aligns with most Git tools (e.g., *"Fix bug"*, not *"Fixed bug"*)
- **Avoid generic messages** like "update", "fix bug"
  → Be specific about the change and its context

A Good Commit:

feat(api): add GET /users endpoint with pagination

Implemented GET /users with ?page & ?limit query params.
Returns paginated list of users from the database.

Includes basic input validation and error handling.

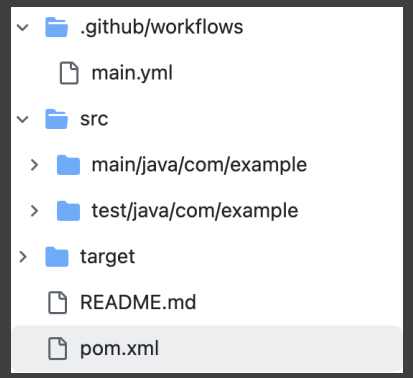
A Bad Commit:

fix login

7. CI/CD Demo
GitHub Actions

7. CI/CD Demo Setup

- Project structure:



- Pipeline Stages:
  - Build → Test → Deploy
- Triggers:
  - **On push** to the main branch
  - **On Pull** Request to the main branch
  - **Manually** via workflow_dispatch

- GitHub Actions environment:
  - Runner: ubuntu-latest
  - Java version: 17 (Temurin)
- Preview of the pipeline:

```yaml
# Name of the GitHub Actions workflow (appears in Actions tab)
name: CI Pipeline

# Global environment variables (available in all jobs)
env:
  APP_ENV: production # Example environment indicator (dev/stage/prod)

# Triggers: define when this workflow should run
on:
  push: # Trigger on push to main branch
    branches: [main]
  pull_request: # Trigger on PRs to main
    branches: [main]
  workflow_dispatch: # Enables manual triggering via GitHub UI

# Workflow jobs
jobs:
  # First job: Build stage
  build:
    runs-on: ubuntu-latest # GitHub-hosted Ubuntu VM
    env:
      BUILD_DIR: target # Job-specific variable (e.g., for Maven output)

    steps:
      - name: Checkout code
        uses: actions/checkout@v3 # Clone the repository code

      - name: Set up JDK 17
        uses: actions/setup-java@v3
        with:
          java-version: "17"
          distribution: "temurin" # Use Temurin (OpenJDK)

      - name: Build with Maven
        run: mvn compile # Compile Java code
```

```yaml
  # Second job: Test stage
  test:
    runs-on: ubuntu-latest
    needs: build # Run this only after 'build' job is successful

    steps:
      - uses: actions/checkout@v3

      - name: Set up JDK 17
        uses: actions/setup-java@v3
        with:
          java-version: "17"
          distribution: "temurin"

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

  # Final job: Deploy stage
  deploy:
    runs-on: ubuntu-latest
    needs: test # Run only after 'test' is successful

    env:
      DEPLOY_KEY: ${{ secrets.MY_SECRET_API_KEY }} # Inject secret via GitHub

    steps:
      - name: Fake Deploy Step
        run: echo "Deploying to $APP_ENV with key $DEPLOY_KEY ✅" # Fake deploy command
```

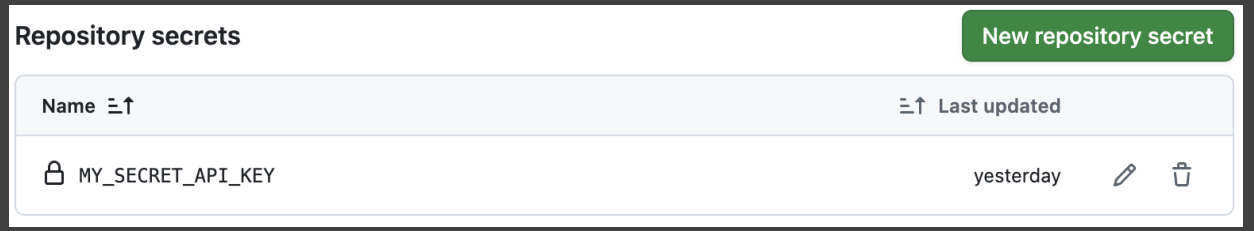
7. What Are Environment Variables in GitHub Actions ?

- Key-value pairs used to configure workflows
- Defined in the env: block or using *secrets*
- *Example:*

```yaml
# Global environment variables (available in all jobs)
env:
  APP_ENV: production # Example environment indicator (dev/stage/prod)
```

```yaml
env:
  DEPLOY_KEY: ${{ secrets.MY_SECRET_API_KEY }} # Inject secret via GitHub
```

- Why use them ?
  - Keep sensitive data out of your code
  - Reusable across steps
  - Cleaner and more secure workflows
- Define secrets in GitHub under:
  - → Repository Settings
  - → Secrets and variables
  - → Actions



Live Demo

CI/CD Pipeline Demo

8. Platform Landscape & Reflection

8. Platform Landscape & Reflection

Are there any new rising stars or reinventions in CI/CD?

- **Top players:**
  1. **GitHub** > 100 million Estimated Users (2024)
  2. **GitLab** ~30 million Estimated Users (2024)
  3. **Bitbucket** ~10-15 million Estimated Users (2024)

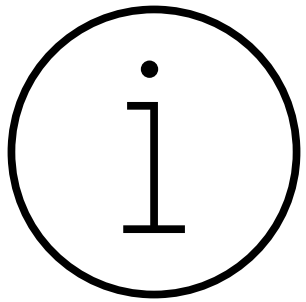
- **Market Trends**
  - No truly **disruptive newcomer** currently surpassing the major players
  - Most innovation happens **within GitHub/GitLab ecosystems**
  - Focus shifts to **workflow automation**, **AI assistance**, and **security**

9. Summary and Platform Recommendations

9. Summary and Platform Recommendations

- **Seminar Summary:**
  - Gained practical insights into **CI/CD concepts** and workflows
  - Explored multiple CI/CD platforms (e.g., **GitHub Actions**, **GitLab CI and Bitbucket Pipeline**)
  - Implemented **basic pipelines** with build, test, and deploy stages
  - Learned how to handle **secrets**, **triggers**, and **environment variables**

- Recommended Platform: **GitHub**
  - **Integrated version control** and CI/CD in one place
  - Widely adopted and supported in open source & industry
  - Great for **team collaboration**, **pull requests**, and **code reviews**
  - Secure by default with **Secrets**, **branch protection**, etc.
  - Easy setup and **ready-to-use CI/CD workflows**



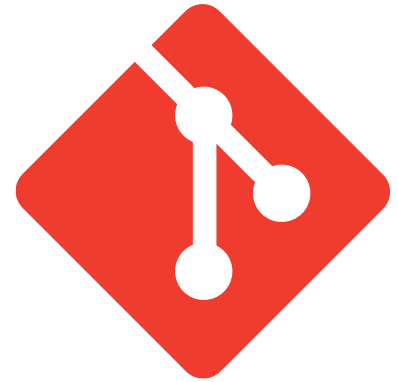
10. *Q&A ?*
Let's Discuss!

Sources & References (1)

- **Documentation:**
  - https://docs.github.com/actions
  - https://docs.gitlab.com/ee/ci/
  - https://support.atlassian.com/bitbucket-cloud/docs/pipelines/
  - https://git-scm.com/docs
- **Branching strategies:**
  - **Git Flow:** https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow
  - **GitHub Flow:** https://www.geeksforgeeks.org/git-flow-vs-github-flow/
- **Trunkbased Development:**
  - https://www.atlassian.com/continuous-delivery/continuous-integration/trunk-based-development
  - https://www.geeksforgeeks.org/trunk-based-development-in-software-development/
- **CI/CD:**
  - https://about.gitlab.com/topics/ci-cd/
  - https://github.com/resources/articles/devops/ci-cd
- **Git platforms comparison:**
  - https://marker.io/blog/github-vs-gitlab-vs-bitbucket
  - https://machine-learning-blog.de/2019/10/04/was-sind-die-unterschiede-github-gitlab-und-bitbucket-im-vergleich/
- **Commit templates and best practices:**
  - https://www.conventionalcommits.org/en/v1.0.0/
  - https://git-scm.com/docs/git-commit/2.10.5

Sources & References (2)

- **Demo:**
  - https://github.com/actions/starter-workflows/blob/main/ci/maven.yml
  - https://github.com/actions
  - https://medium.com/@pathirage/step-in-to-ci-cd-a-hands-on-guide-to-building-ci-cd-pipeline-with-github-actions-7490d6f7d8ff
  - https://docs.github.com/en/actions/writing-workflows/choosing-what-your-workflow-does/store-information-in-variables
- **Statistics and numbers:**
  - https://kinsta.com/blog/github-statistics/
  - https://about.gitlab.com/press/releases/2024-03-04-gitlab-reports-fourth-quarter-and-full-fiscal-year-2024-financial-results/
  - https://expandedramblings.com/index.php/bitbucket-statistics-and-facts/
- **Assets & Visuals**
  - GitHub logos: https://github.com/logos
  - GitHub octodex: https://octodex.github.com/
  - GitLab Logo: https://about.gitlab.com/press/press-kit/
  - Git Logo: https://git-scm.com/downloads/logos
  - CI/CD Visual: https://www.blackduck.com/glossary/what-is-cicd.html
  - CI/CD Pipeline Visual: https://medium.com/@pathirage/step-in-to-ci-cd-a-hands-on-guide-to-building-ci-cd-pipeline-with-github-actions-7490d6f7d8ff
  - All other visuals/screenshots: Created by the presenter



Thank you for listening

Mohammed-Ammar Hassnou