Hands-On DevOps: CI/CD | LSPP 2025

Slides Link: Link

Github Repository: Link

The goal of these assignments is not to test you, but to challenge you to think like a **real** DevOps engineer. The most important skill in this field is not knowing all the answers, but knowing how to find them. You will need to read documentation, analyze logs, and solve problems—and you'll build something amazing in the process.

General Instructions

- **Submission Guideline:** Create a single .SUBMISSION.md file in the root of your repository. In this file, provide links to all your deliverables (e.g., a link to your final README.md , a link to a specific workflow run, etc.). Submit the URL to your public GitHub repository.

- **Plagiarism Policy**: Your journey is your own. Please write your own code and analysis. Any instances of plagiarism will result in a score of 0.

Assignment 1: The Build Verifier 🛠️ (Compulsory)

The Goal: The first step in any reliable process is verification. This assignment focuses on the "CI" part of CI/CD: proving that your code is high-quality and can be reliably built into a standardized package (a Docker image) every single time.

Your Task:

1. **Choose Your Application:** You can use any simple application you like.
    - **Option A:** Use a simple project you have already built.

    - **Option B:** Create a new "Hello World" app in any language you're comfortable with (see examples below).
2. **Create a Dockerfile:** Write a standard Dockerfile for your chosen application.
3. **Create a "Build & Test" Workflow:**
    - Create a GitHub Actions workflow file (e.g., .github/workflows/build.yml).
    - This workflow must trigger on every push to your main branch.
    - It must contain two separate jobs:
        - **test-job:** This job should run your application's tests. If you don't have any, a simple "test" can be a script that just starts your server to ensure it doesn't crash. The entire workflow must fail if this job fails.
        - **build-job:** This job **must depend on test-job** (using the needs: keyword). It will only run if testing succeeds. Its purpose is to run the docker build command, proving your Dockerfile is valid. You **do not** need to push the image anywhere for this assignment.

What You Will Learn:

- How to write a Dockerfile from scratch for any application.
- How to create a multi-job workflow in GitHub Actions.
- How to use needs: to create dependencies between jobs (a core CI concept).

What to Submit:

- A link to your GitHub repository containing the Dockerfile and workflow file.
- A link to the log of a successful workflow run in your "Actions" tab.

Assignment 2: The Debugging Detective 🕵️ (Compulsory)

The Goal: A DevOps engineer spends 50% of their time fixing broken pipelines. This assignment teaches you that crucial skill: how to read error logs, understand the problem, and implement a fix.

The Task:

1. **Part A: Analyze the Working Pipeline**
    - In a new file called DEBUGGING.md, briefly explain how your working pipeline from Assignment 1 works. Describe the jobs, the trigger, and what the needs: keyword does.
2. **Part B: The "Break and Fix" Challenge**
    - **Break It:** Intentionally introduce a bug into your Dockerfile. A classic error is to change the first line (FROM ...) to refer to a base image that doesn't exist. For example: FROM node:18-this-is-a-fake-tag.
    - **Push and Observe:** Commit and push this broken Dockerfile. Go to the Actions tab and watch the pipeline fail.
    - **Analyze:** Click into the failed build-job. Read the error logs carefully. What message does Docker give you? Take a screenshot of the specific error message.
    - **Fix It:** Correct the Dockerfile, commit the fix, and push again. Watch the pipeline turn green.
3. **Document Your Work:** In your DEBUGGING.md file, add a new section for Part B. Include your screenshot of the error, explain what the error meant, and describe how you fixed it.

What You Will Learn:

- How to read and understand CI/CD error logs.
- How to troubleshoot common Docker and workflow errors.

- The critical DevOps practice of analyzing failures to prevent them in the future.

What to Submit:

- Your completed DEBUGGING.md file containing your analysis and screenshots.
- A link to the failed workflow run and the final successful (fixed) workflow run.

Assignment 3: The Release Architect 🚀 (Optional & Challenging)

The Goal: This is where you go from a student to a pro. Go beyond a simple build and create a professional, automated software release pipeline, just like major open-source projects do.

The Task:

1. **Research Git Tags & Versioning:** First, understand how Git tags (e.g., v1.0.0) are used to mark specific versions of your code.
2. **Research Release Automation:** Find a GitHub Action that can automatically create a "Release" on your repository's release page. A very popular and powerful one to research is **softprops/action-gh-release**.
3. **Create a "Release" Workflow:**
    - This new workflow must trigger **only when you push a tag** that starts with v (e.g., v1.0.0, v1.1.0-beta). The trigger will be on: push: tags: - 'v*'.
    - The workflow must perform these steps in order:
        i. **Multi-Stage Build:** Rewrite your Dockerfile to use a **multi-stage build** to create a smaller, more secure production image.

ii. **Security Scan:** Add a step to scan your final Docker image for vulnerabilities using a tool like **Trivy**. The pipeline should fail if it finds critical vulnerabilities. c. **Push to Registry:** Push the validated image to the **GitHub Packages Container Registry (ghcr.io)**. The image tag **must match the Git tag** you just pushed. d. **Create Release:** Use the release action you researched to create a new GitHub Release. It should automatically use your Git tag as the version and generate a "changelog" of commits.

4. **Bonus Challenge:** Monitoring: Add a Status Badge to your README.md file that shows the live status (passing/failing) of your main build workflow.

What You Will Learn to Research:

- Industry-standard Git tagging strategies like Semantic Versioning.
- The professional practice of release automation and automated changelog generation.
- How to create powerful, event-driven workflows that respond to different Git actions.

What to Submit:

- A README.md file that explains your release workflow and the research you did.
- A link to the automatically created release in your repository's "Releases" section.
- (If you did the bonus) A README.md proudly displaying your live status badge!