Hands-On Lab: Building a Full CI/CD Pipeline with Jenkins

Mini-Project: Personal Banking API Service

Target Audience: Java (SpringBoot) & Python (FastAPI) Developers

python_Mini Project_ci_cd_lab.zip

🎯 The Mission

Your Goal

Build a complete, automated CI/CD pipeline for a simple banking application. In the next 60 minutes, you will automate the entire process from a code commit to a production release.

1. **Commit Pipeline (CI):** Automatically build, test, and package our application on every code change.
2. **Acceptance Pipeline (Staging):** Deploy the application to a 'staging' environment and run validation tests.
3. **Release Pipeline (Production):** Promote the tested application to 'production' after a manual approval step.

Learning Outcome: This lab will teach you the fundamentals of creating robust, multi-stage, and multi-language pipelines in Jenkins.

🏦 Mini-Project Requirement

We will work on a **Personal Banking API Service**. To keep it achievable in 60 minutes, the scope is very focused.

The Application

A simple microservice with a single API endpoint:

```
GET /balance/{account_id}
```

Focus

This simple service allows us to focus on the pipeline logic, not complex application code.

Functionality

When a user queries the endpoint, it returns a hardcoded JSON response.

Example Request:

```
GET /balance/12345
```

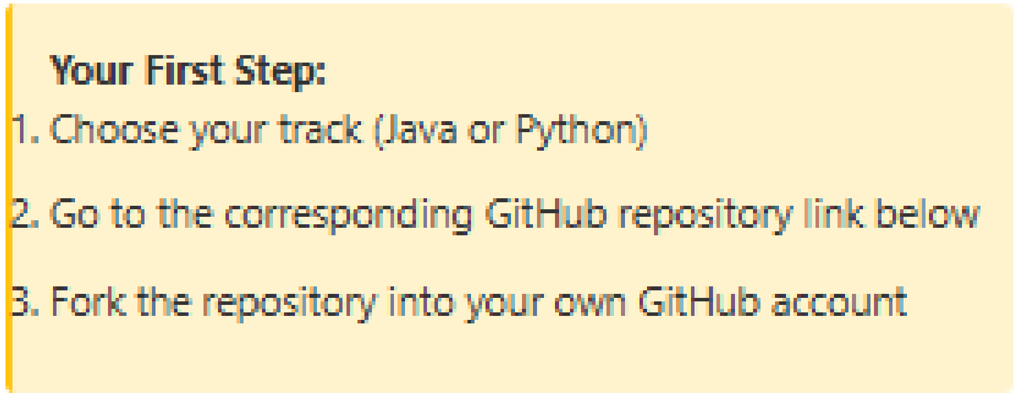
Example Response:

```
{"account_id": "12345", "balance": 1500.75, "currency": "USD"}
```

This simple service allows us to focus on the pipeline logic, not complex application code.

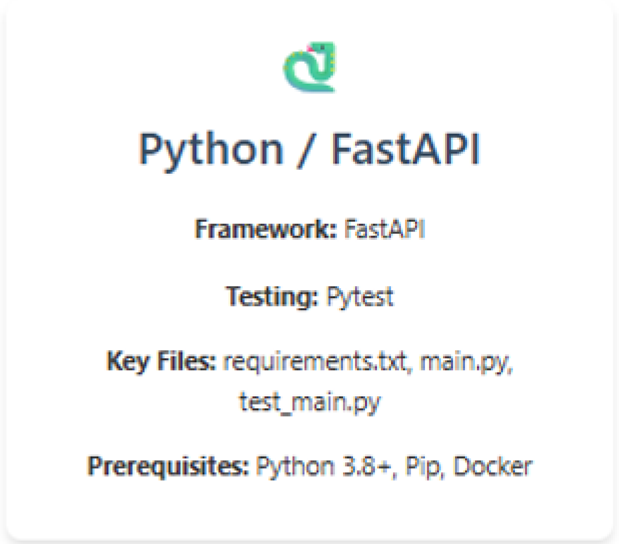
Project Setup & Prerequisites

Before we begin, you need a starting point. We have created two template repositories for you.



Your First Step:

1. Choose your track (Java or Python)
2. Go to the corresponding GitHub repository link below
3. Fork the repository into your own GitHub account



Python / FastAPI

Framework: FastAPI

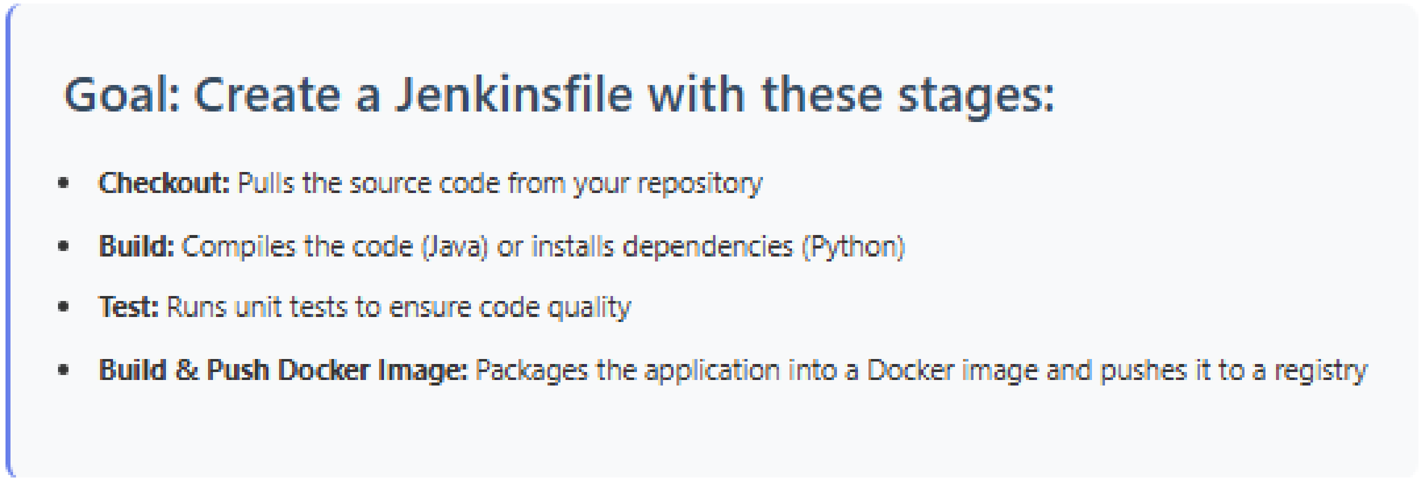
Testing: Pytest

Key Files: requirements.txt, main.py, test_main.py

Prerequisites: Python 3.8+, Pip, Docker

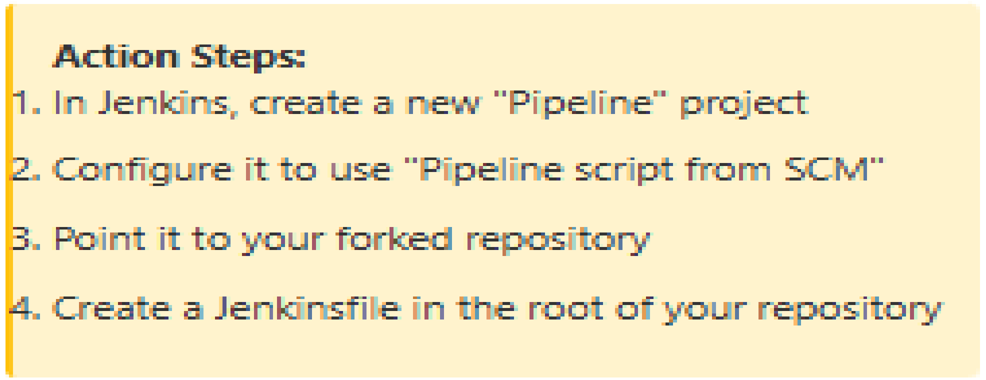
Lab Part 1 - Implementing the Commit Pipeline (CI)

The Commit Pipeline is our first line of defence. It validates that the new code integrates correctly.



Goal: Create a Jenkinsfile with these stages:

- **Checkout:** Pulls the source code from your repository
- **Build:** Compiles the code (Java) or installs dependencies (Python)
- **Test:** Runs unit tests to ensure code quality
- **Build & Push Docker Image:** Packages the application into a Docker image and pushes it to a registry



Action Steps:

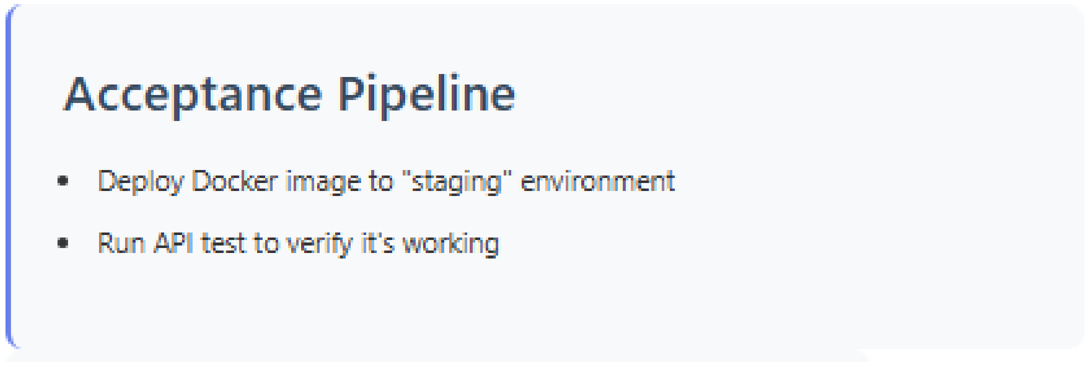
1. In Jenkins, create a new "Pipeline" project
2. Configure it to use "Pipeline script from SCM"
3. Point it to your forked repository
4. Create a Jenkinsfile in the root of your repository

Goal

Create a Jenkinsfile that performs all stages without any errors or exceptions, without any manual interventions

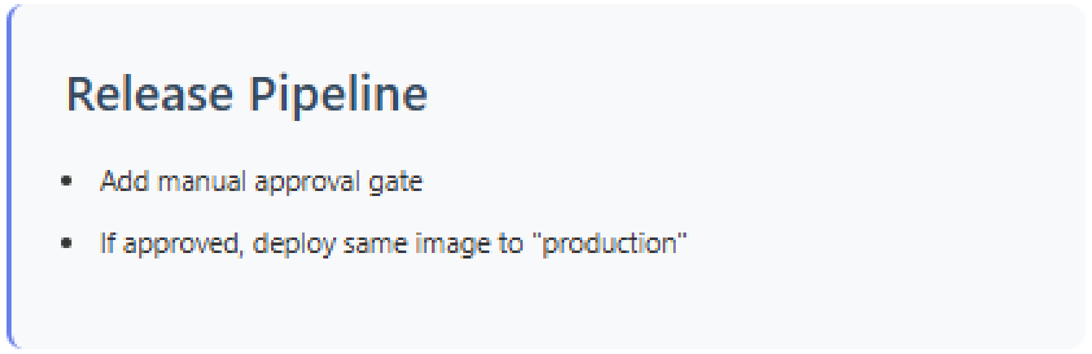
Lab Part 2 - Acceptance & Release Pipelines (CD)

Now let's extend our pipeline to handle deployments.



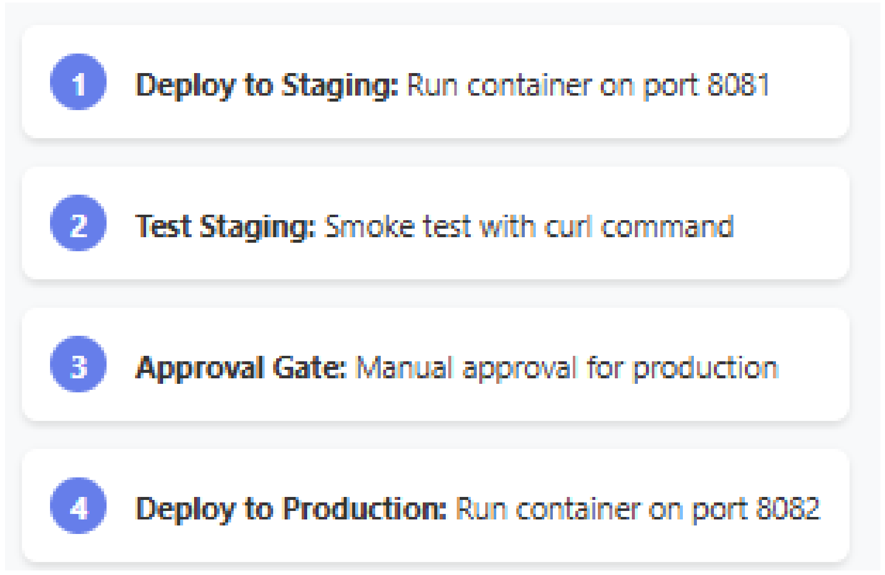
Acceptance Pipeline

- Deploy Docker image to "staging" environment
- Run API test to verify it's working



Release Pipeline

- Add manual approval gate
- If approved, deploy same image to "production"



1. **Deploy to Staging:** Run container on port 8081
2. **Test Staging:** Smoke test with curl command
3. **Approval Gate:** Manual approval for production
4. **Deploy to Production:** Run container on port 8082

Goal

Extend our pipeline to handle deployments with staging and production environments.

Key Principle

"Build Once, Deploy Many" - The same Docker image moves through all environments

✅ Conclusion & Key Learnings

Congratulations! You have successfully built a complete CI/CD pipeline that can build, test, and deploy both Java and Python applications.

Automated CI

Your code is now automatically built and tested on every commit

Infrastructure as Code

You defined your entire pipeline as code using a Jenkinsfile

Quality Gates

You implemented unit tests, API smoke tests, and manual approval steps

Containerization

You packaged your application with Docker, ensuring consistency across environments

Multi-Environment Deployment

You created a flow to deploy to Staging and Production environments

GOAL

- A single Jenkinsfile can handle different technologies by checking for characteristic files
- The "Build Once, Deploy Many" principle ensures consistency across environments
- CI/CD is about building a reliable, repeatable, and safe path to production 🚀

🚀 **Key Takeaway:** CI/CD is not just about automation; it's about building a reliable, repeatable, and safe path to production through the principle "Build Once, Deploy Many."

Questions?

Thank you for your attention