**Complete Jenkins Python Integration Guide**

**Goal:** Set up GitHub Actions workflow to handle the entire CI/CD process, including the build, tests, and deployments, independent of Jenkins. Also perform tests local on Jenkins.

About: Jenkins is an open-source automation server that is widely used for continuous integration and continuous delivery (CI/CD). It supports building, deploying, and automating software development processes. Jenkins provides hundreds of plugins to support building, deploying, and automating any project. With Jenkins, developers can automatically build and test their code, making it easier to integrate changes and ensure software quality. It is highly extensible and can be configured to meet the specific needs of projects.

**Prerequisites**

**Required Software**

* Python 3.9+ (3.11 recommended)
* Jenkins LTS (2.426.1+)
* Git
* Visual Studio Code (or preferred Python IDE)
* pytest, pytest-cov for testing

**System Requirements**

* Memory: 4GB minimum (8GB recommended)
* Disk Space: 10GB minimum
* Processor: Dual-core (quad-core recommended)

**1. Installation Steps**

**Python Installation**

**Windows:**

winget install Python.Python.3.11

python --version

pip --version

**macOS:**

brew install python@3.11

echo 'export PATH="/usr/local/opt/python@3.11/bin:$PATH"' >> ~/.zshrc

source ~/.zshrc

**Linux (Ubuntu/Debian):**

sudo apt update

sudo apt install python3.11 python3-pip

python3.11 --version

pip3 –version

**Jenkins Configuration**

Install OpenJDK 17

**Windows**

Install OpenJDK 17:

powershell

winget install Microsoft.OpenJDK.17

Download Jenkins MSI installer from the official Jenkins website.

Run the MSI installer and follow the on-screen instructions.

**Ubuntu (x86\_64 and ARM)**

Install OpenJDK 17:

sudo apt update

sudo apt install openjdk-17-jdk -y

Add Jenkins repository:

curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null

echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian-stable binary/ | sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null

Install Jenkins:

sudo apt update

sudo apt install jenkins -y

Start Jenkins:

sudo systemctl start jenkins

**macOS**

Install Homebrew if not already installed:

bash

/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

Install OpenJDK 17:

brew install openjdk@17

Install Jenkins:

brew install jenkins-lts

Start Jenkins:

brew services start jenkins-lts

**Red Hat / Fedora**

Install OpenJDK 17:

sudo dnf install java-17-openjdk-devel -y

Add Jenkins repository:

sudo wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo

sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key

Install Jenkins:

sudo dnf install jenkins -y

Start Jenkins:

sudo systemctl start jenkins

After installation, Jenkins will typically be accessible at **http://localhost:8080**. If you'll need to retrieve the initial admin password from /var/lib/jenkins/secrets/initialAdminPassword (Linux/macOS) or C:\Program Files\Jenkins\secrets\initialAdminPassword (Windows) to complete the setup.

**2. Exposing Jenkins to External Network**

To expose Jenkins to an external network for GitHub webhooks:

**npm install -g localtunnel**

**lt --port 8080**

This will provide a public URL like https://yourname.loca.lt. Use this URL in your Jenkins settings.

**3. Python Project Setup**

**Project Structure**

bank\_system/

├── src/bank/account.py

├── tests/test\_account.py

├── .github/workflows/jenkins.yml

├── requirements.txt

├── pytest.ini

├── Jenkinsfile (optional for pipeline tests)

└── venv/

**Source Code Example (src/bank/account.py)**

from dataclasses import dataclass

from decimal import Decimal

from datetime import datetime

@dataclass

class Transaction:

amount: Decimal

timestamp: datetime

description: str

transaction\_type: str # 'deposit' or 'withdrawal'

class BankAccount:

def \_\_init\_\_(self, account\_number: str, owner\_name: str, initial\_balance: Decimal = Decimal('0.00')):

self.account\_number = account\_number

self.owner\_name = owner\_name

self.\_balance = initial\_balance

self.\_transactions = []

if initial\_balance > 0:

self.\_add\_transaction(initial\_balance, "Initial deposit")

def deposit(self, amount: Decimal) -> bool:

if amount <= 0:

raise ValueError("Deposit must be positive")

self.\_balance += amount

self.\_add\_transaction(amount, "Deposit")

return True

**requirements.txt**

pytest

pytest-cov

requirements breakdown:

Each of the Python modules mentioned above serves a unique purpose in the development process:

-pytest: A testing framework that makes it easy to write simple and scalable test cases for Python code. It's used for running unit tests, supporting fixtures, parameterized testing, and more.

- pytest-cov: An extension to pytest that provides coverage reports. It measures code coverage of Python programs, helping developers identify untested parts of a codebase by running the tests and analyzing the results.

Other (optional)

- pylint: A static code analysis tool that checks for errors in Python code, enforces a coding standard, and looks for code smells. It helps maintain a consistent coding style and improve code quality by suggesting improvements.

- black: An opinionated code formatter for Python. It automatically formats code to ensure a uniform style across a project, reducing the need for discussions about programming style in code reviews. Black is known for its speed and robust formatting abilities.

-mypy is a static type checker for Python. It is used for adding optional type hints to Python code, allowing developers to catch type-related errors before runtime. By analyzing type annotations, mypy helps ensure that the functions are used with the correct types, enhancing code reliability and maintainability. It is particularly useful in large codebases where type errors can be hard to track down.

**4. Jenkins Configuration**

Establish login username

When first installing Jenkins and opening localhost:8080, you need to establish a username and password. Here's the process:

1. When you first access Jenkins at **localhost:8080**, you'll be presented with an "Unlock Jenkins" page.
2. Jenkins generates an initial administrator password automatically during installation. You'll need to retrieve this password to proceed.
3. The initial admin password can typically be found in a file called initialAdminPassword located in the Jenkins installation directory. For example:
   * On Windows: C:\Program Files\Jenkins\secrets\initialAdminPassword
   * On Linux/Mac: /var/lib/jenkins/secrets/initialAdminPassword
4. After entering this initial password, you'll be guided through the setup process.
5. During setup, you'll have two options.
   * Install suggested plugins (recommended for beginners)
   * Select specific plugins to install (Junit choices for sure)
6. After plugin installation, you'll be prompted to create the first admin user. This is where you'll establish your own username and password for future logins.
7. Alternatively, you can choose to continue as admin using the initial password, but it's recommended to create a new admin user for security reasons.

Establish API token (necessary)

To generate a token for a Jenkins user via the dashboard, follow these steps:

-Log in to your Jenkins instance using the credentials of the user for whom you want to generate the token.

-Navigate to the user's profile page by clicking on the username in the top-right corner of the Jenkins dashboard.

-Click on "Configure" in the left sidebar to access the user configuration page.

-Scroll down to the "API Token" section on the configuration page.

-Click on the "Add new Token" button.

-Give your token a descriptive name to help you identify its purpose later.

-Click on the "Generate" button next to the token name field.

Jenkins will generate a new API token. **Important:** Copy this token immediately and store it securely, as it will not be displayed again

-Click on the "Save" button at the bottom of the page to confirm the changes.

Remember, it's crucial to securely store your Jenkins credentials once you've set them up. If you forget your password later, there are procedures to reset it, but it's best to avoid that situation by keeping your login information safe

Configure a job name

Manually Create the Job in Jenkins Dashboard: Before you can trigger a job via a script or YAML configuration, you need to set it up in Jenkins. To do this:

- Open Jenkins in your web browser.

- Click on "New Item" in the top left menu.

- Enter a name for the new job (e.g., job1).

- Select the job type, such as "Freestyle project".

- Click "OK".

- Configure the job as needed and save.

Configure Build

Easiest via dashboard:

1. Go to your job configuration page **http://localhost:8080/job/job1/configure**

On the left hand side you will find the following panel of options for various configurations. Snapshot follows.

A screenshot of a computer

AI-generated content may be incorrect.

Click on Source Code Management as shown below, to start some configuration entries. Example follows.

A screenshot of a computer

AI-generated content may be incorrect.

Note Credentials above are your Jenkins name/password.

Follow thru with Triggers- include your Auth Token

A screenshot of a computer

AI-generated content may be incorrect.

For Build Steps select Add Build Steps and choose Execute Shell. Enter script as follows

# Define the virtual environment path explicitly

VENV\_PATH="/home/parallels/270/venv"

# Activate the virtual environment

. $VENV\_PATH/bin/activate

# Add the project root to PYTHONPATH

export PYTHONPATH="/home/parallels/270:$PYTHONPATH"

# Manually install the necessary dependencies

pip install pytest

# Run the tests from the correct directory

pytest /home/parallels/270/bank\_system/tests --maxfail=1 --disable-warnings --tb=short --junitxml=result.xml

For Post-build Actions choose \*\*/result.xml in the field. This file will be built starting from your root path.

Snapshot follows.

A screenshot of a chat

AI-generated content may be incorrect.

Click "Save" to commit all your entries.

Run local test

This allows testing Jenkinsfiles locally without committing to a repository.

To manually trigger a build from the Jenkins web interface, follow these steps:

1. Log in to Jenkins: Open your web browser and navigate to the URL where your Jenkins server is hosted. Log in with your credentials.

2. Navigate to the Job: Once logged in, you will see the Jenkins dashboard. Find the job you would like to trigger in the list of jobs displayed on the dashboard.

3. Trigger the Build: Click on the job name to open the job details page. On the left side of this page, you should see a "Build Now" link. Click "Build Now" to trigger a new build.

4. View the Build Progress: After clicking "Build Now", a new build will be queued and you can see it in the 'Build History' on the left-hand side. You can click on the build number to view the build console output and other details.

Triggering a build this way is straightforward and useful for quick manual   
 interventions or testing changes immediately after configuration updates.

To trigger a Jenkins job using the command line, you typically need the following:

1. Jenkins URL: The base URL where your Jenkins instance is hosted.

2. Jenkins API token or username/password for authentication.

3. The name of the job you want to trigger.

Below is a step-by-step process using curl to trigger a Jenkins job:

1. Generate an API token for your Jenkins user if you haven't already.

2. Use the curl command to trigger the job:

Example: curl -X POST "http://JENKINS\_URL/job/JOB\_NAME/build" \

--user "USERNAME:API\_TOKEN"

Replace `JENKINS\_URL` with your Jenkins server URL, `JOB\_NAME` with the name of your job, and `USERNAME:API\_TOKEN` with your Jenkins username and generated API token.

Note: Using API tokens is recommended over using plain passwords for security reasons.

You might also need to install `curl` if it's not already available in your system.

Make sure that your Jenkins instance is configured to allow builds triggered via curl or API calls, and necessary permissions are set for the user whose credentials are being used.

**Jenkinsfile (optional)**

pipeline {

agent any

environment {

PYTHON\_VERSION = '3.11'

VENV\_NAME = 'venv'

}

stages {

stage('Setup') {

steps {

script {

sh 'python3 -m venv ${VENV\_NAME} && source ${VENV\_NAME}/bin/activate && pip install -r requirements.txt'

}

}

}

stage('Test') {

steps {

script {

sh 'source ${VENV\_NAME}/bin/activate && pytest tests/'

}

}

}

}

}

**5. GitHub Actions Workflow (.github/workflows/jenkins.yml) – *triggered via a push***

name: Trigger Jenkins Pipeline

on:

push:

branches:

- main

jobs:

trigger:

runs-on: ubuntu-latest

steps:

- name: Install jq and curl

run: sudo apt-get update && sudo apt-get install -y jq curl

- name: Trigger Jenkins Build

env:

JENKINS\_USER: jpapadem

JENKINS\_TOKEN: 11e4a4fb90eddbdeef7f24ed4cfe835924

JENKINS\_URL: "https://yellow-turkeys-turn.loca.lt"

JOB\_NAME: "job1"

run: |

# Check Jenkins connection

RESPONSE=$(curl -s -o /dev/null -w "%{http\_code}" -k "${JENKINS\_URL}/api/json")

if [ "$RESPONSE" != "200" ]; then

echo "Cannot connect to Jenkins server"

exit 1

fi

# Try to trigger the job directly

RESPONSE=$(curl -s -k -X POST -u ${JENKINS\_USER}:${JENKINS\_TOKEN} "${JENKINS\_URL}/job/${JOB\_NAME}/build?token=${JENKINS\_TOKEN}")

if [ $? -ne 0 ]; then

echo "Failed to trigger Jenkins job directly"

exit 1

fi

# Wait for build completion

for i in {1..12}; do

echo "Checking build status..."

BUILD\_STATUS=$(curl -s -k -u ${JENKINS\_USER}:${JENKINS\_TOKEN} "${JENKINS\_URL}/job/${JOB\_NAME}/lastBuild/api/json")

if [ "$(echo "$BUILD\_STATUS" | jq -r '.building')" == "false" ]; then

RESULT=$(echo "$BUILD\_STATUS" | jq -r '.result')

if [ "$RESULT" == "SUCCESS" ]; then

echo "Build completed successfully!"

exit 0

else

echo "Build failed with result: $RESULT"

exit 1

fi

fi

echo "Waiting 5 seconds..."

sleep 5

done

echo "Build did not complete within the timeout period"

exit 1

**Set up a webhook in GitHub with these steps:**

Repository Webhook Setup

1. Navigate to your GitHub repository.
2. Click on "Settings" in the repository menu.
3. Select "Webhooks" from the left sidebar.
4. Click the "Add webhook" button.

Configuring the Webhook

1. Enter the Payload URL where you want to receive webhook events.

Ex. https://yourname.loca.lt /github-webhook/

1. Choose the Content type (usually "application/json").
2. Optionally, set a Secret for added security.
3. Select the events you want to trigger the webhook:
   * Choose "**Just the push event**" for push notifications only.
   * Select "Send me everything" for all events.
   * Pick "Let me select individual events" to customize.
4. Ensure the "Active" checkbox is ticked.
5. Click "Add webhook" to complete the setup.

Additional Configuration Options

* **SSL verification**: You can enable or disable SSL verification for the webhook.
* **Events**: You can subscribe to specific events like pushes, pull requests, or releases.

Testing the Webhook

After setting up the webhook, GitHub will send a "ping" event to verify the configuration. You can view the delivery details in the webhook settings to ensure it's working correctly.Remember to keep your webhook URL and secret secure, as they provide access to potentially sensitive information about your repository.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A green graph with red dots

AI-generated content may be incorrect.

**6. \*Running Tests Locally**

python3 -m venv venv

source venv/bin/activate

export PYTHONPATH="/home/parallels/270:$PYTHONPATH"

pip install -r requirements.txt

From project root run following trigger

**PYTHONPATH=/home/parallels/270/bank\_system/src pytest**

**7. Troubleshooting**

**Common Issues**

# Check Python version

python --version

# Verify pip installation

pip --version

# List installed packages

pip list

**Jenkins Debugging**

stage('Debug Info') {

steps {

sh 'python --version'

sh 'pip list'

}

}

**8. Best Practices**

* Follow PEP 8 coding standards.
* Use virtual environments for dependency management.
* Automate testing in Jenkins.
* Use credentials management for security.
* Keep pipeline stages modular and efficient.

Refs:

Here are some excellent Jenkins tutorials and resources to help you get started or advance your knowledge:

<https://www.jenkins.io/doc/tutorials/>

<https://www.jenkins.io/doc/tutorials/tutorial-for-installing-jenkins-on-AWS/>

Beginner-Friendly Tutorials

1. **Jenkins Beginner Tutorial - Step by Step (Udemy)**  
   A comprehensive course designed for complete beginners to learn Jenkins from scratch, covering setup, job creation, and pipelines.  
   [Udemy Jenkins Beginner Tutorial8](https://www.udemy.com/course/jenkins-beginner-tutorial-step-by-step/)
2. **6-Step Jenkins Tutorial for Beginners (Codefresh)**  
   Learn to install Jenkins, create pipelines, manage plugins, and set up security in a beginner-friendly guide.  
   [Codefresh Jenkins Tutorial17](https://codefresh.io/learn/jenkins/6-step-jenkins-tutorial-for-beginners/)
3. **Jenkins Getting Started: Step-by-Step for Beginners (YouTube)**  
   A video tutorial that walks you through prerequisites, installation, setup, creating your first job, and running builds.  
   [YouTube - Jenkins Getting Started](https://www.youtube.com/watch?v=cy2257Y7I54)
4. **Jenkins Tutorial - TutorialsPoint**  
   A text-based guide explaining how to use Jenkins to build and test software projects continuously.  
   [TutorialsPoint Jenkins Tutorial14](https://www.tutorialspoint.com/jenkins/index.htm)

Intermediate to Advanced Tutorials

1. **Advanced Jenkins Course - DevOps Shack (YouTube)**  
   A 10-day advanced series covering pipelines, plugins, multi-branch pipelines, and CI/CD setups.  
   [YouTube - Advanced Jenkins Course12](https://www.youtube.com/playlist?list=PLAdTNzDIZj_8qXuqxFudMEiPY4ryVKMkI)
2. **Jenkins CICD - Advanced Lesson (Cloud Academy)**  
   Learn advanced CI/CD techniques with hands-on demonstrations of pipelines, Docker integration, and scaling Jenkins with agents.  
   [Cloud Academy Advanced Course3](https://cloudacademy.com/course/jenkins-cicd-advanced/jenkins-infrastructure/)
3. **Jenkins Pipeline Intermediate Course (CloudBees University)**  
   Focuses on creating shared libraries, advanced pipeline features, and best practices for automation workflows.  
   [CloudBees University Pipeline Course7](https://hackr.io/blog/jenkins-certification)
4. **Pluralsight Advanced Jenkins Training**  
   Covers building applications in an automated fashion with advanced techniques for CI/CD workflows.  
   [Pluralsight Advanced Course9](https://www.pluralsight.com/professional-services/it-ops/advanced-jenkins)

Official Documentation

1. **Jenkins Tutorials Overview**  
   The official Jenkins documentation provides a series of tutorials for both beginners and advanced users, including pipeline syntax and CI/CD concepts.  
   [Jenkins Official Tutorials11](https://www.jenkins.io/doc/tutorials/)

Community Recommendations

1. **Reddit Discussions on Learning Jenkins**  
   Explore community-suggested resources and tips for learning Jenkins effectively from experienced DevOps professionals.  
   [Reddit Discussions on Learning Jenkins1](https://www.reddit.com/r/devops/comments/ucg2yp/how_to_learn_jenkins/)

These resources cater to a range of skill levels and learning preferences, from video tutorials to detailed documentation and hands-on courses.