Use Case: Jenkins

Use Case Title: Setup and Run a Basic Jenkins CI Pipeline from Git Repository

Problem Statement:

User Story	Requirement	Deliverables
User Story 1: Install and Set Up Jenkins	**As** a DevOps beginner, **I want** to set up Jenkins, connect it to a GitHub repository, and execute a basic CI job for a Python project, **So that** I can automate code pulling, testing, and feedback generation using Jenkins and relevant plugins. Set up a local or cloud-based Jenkins instance to run jobs.	1. Install Jenkins (preferably via Docker or .war file) 2. Access Jenkins UI at http://localhost:8080 3. Create the first admin user and install suggested plugins
User Story 2: Create a Basic Freestyle Job	Create a new **Freestyle Project** to run a Python script stored in a GitHub repository.	1. Job name: python-ci-job 2. Configure to: • Pull source code from Git • Run a simple Python script (hello.py) 3. Add build step: python hello.py
User Story 3: Connect Jenkins with Git Repository	Integrate Jenkins with a GitHub repository where the Python project is stored.	1. Use Git plugin (installed by default) 2. Configure job source code management: • Repository URL: https://github.com/your-username/python-ci-demo.git • Branch: main (or master) 3. Verify successful cloning via job console output
User Story 4: Execute a Simple CI Pipeline	Build a CI pipeline that: • Pulls the code from Git • Executes the Python script • Prints output and shows build status	1. Run the job and check console logs for script output 2. Observe the job status: ✓ Success or ✗ Failure
User Story 5: Enhance Job Using Plugins	Use at least **two plugins** to enhance the job functionality. **Suggested Plugins:** • **Email Extension Plugin** – Send email on failure • **JUnit Plugin** – Publish test results (optional: if unit tests exist) • **Build Timeout Plugin** – Auto-abort stuck jobs • **Git Parameter Plugin** – Allow branch selection before build	1. Install selected plugins from **Manage Jenkins → Plugin Manager** 2. Add plugin configuration to the existing job 3. Show how the enhancement helps in CI visibility or control

Performance Outcome:

Upon completing this case study, learners will achieve the following learning outcomes:

- The learner is able to install and configure Jenkins, create a basic freestyle job, integrate with a Git repository, run a simple Python-based CI pipeline, and enhance job functionality using Jenkins plugins.
- This confirms the learner’s basic competency in automating software builds using Jenkins.