Software Build Automation Tools

Laboratory Exercises 2

Subject
Continuous Integration (CI) and its Role in Build Automation

Author:
PhD Hubert Zarzycki

The scope of laboratories

Introduction to Continuous Integration (CI)

Continuous Integration (CI) is a software development practice where developers frequently integrate code into a shared repository. The goal of CI is to detect and resolve integration errors early, ensuring that software remains in a deployable state. Each integration is verified by automated builds and tests, allowing teams to identify and fix issues quickly.

By implementing CI, development teams can:

- Reduce the risk of integration conflicts.
- Improve software quality through automated testing.
- Accelerate the development lifecycle by detecting defects early.

Popular CI/CD tools include GitLab CI/CD, Jenkins, and Travis CI, each offering automation capabilities for building, testing, and deploying applications.

Overview of CI Tools

1. GitLab CI/CD

GitLab CI/CD is an integrated continuous integration and continuous deployment (CI/CD) solution that comes as a part of GitLab. It enables developers to automate the process of building, testing, and deploying applications by defining pipeline workflows in a YAML configuration file named `.gitlab-ci.yml`. GitLab CI/CD runs jobs in a sequence of defined stages, ensuring a structured and automated development workflow.

Key Features:

- Built-in CI/CD – No need for third-party integrations; it works natively within GitLab.
- Pipeline as Code – Uses YAML configuration for defining CI/CD processes.
- Parallel Execution – Jobs can run simultaneously, reducing build times.
- Docker Integration – Supports Docker containers for consistent build environments.
- Auto DevOps – Predefined CI/CD templates to automate deployment.

- Security and Compliance – Features like secret management, scanning tools, and approval workflows.

GitLab Runners are lightweight agents responsible for executing CI/CD jobs. Runners can be shared, group-specific, or self-hosted for more control over execution environments.

2. Jenkins

Jenkins is an open-source CI/CD tool that automates building, testing, and deployment. It supports plugins that extend its functionality.

Key Features:

- Extensible with numerous plugins.
- Supports various build environments and scripting languages.
- Can be integrated with GitHub, GitLab, and other repositories.

3. Travis CI

Travis CI is a cloud-based CI/CD service primarily used for GitHub projects. It enables automated testing and deployment with minimal configuration.

Key Features:

- Easy integration with GitHub repositories.
- Supports multiple programming languages.
- Provides cloud-based build and test execution.

Setting Up a Simple CI Pipeline and Automating Builds

1. Create a GitLab Repository

- Log in to GitLab and create a new repository.
- Clone the repository to the local machine:
- `git clone https://gitlab.com/username/repository.git`
- `cd repository`

2. Create a .gitlab-ci.yml File

The `.gitlab-ci.yml` file defines the CI/CD pipeline. Below is an example for a Python project:

```
stages:
  - build
  - test
  - deploy

build:
  stage: build
  script:
    - echo "Building the project"
    - python setup.py install

test:
  stage: test
  script:
    - echo "Running tests"
    - pytest
deploy:
  stage: deploy
  script:
    - echo "Deploying application"
```

3. Push the Changes to GitLab

Commit and push the file to trigger the pipeline:

```
git add .gitlab-ci.yml
git commit -m "Add CI pipeline"
git push origin main
```

4. Monitor the CI Pipeline

- Navigate to CI/CD > Pipelines in GitLab.
- Check the status of the pipeline execution.
- Debug any errors if necessary.

Tasks

1. For the tasks create a report including:
   - repository link
   - `.gitlab-ci.yml` file content.
   - Screenshots of the pipeline execution.
   - Description of any issues faced and solutions implemented.
2. Configure and execute a simple GitLab CI pipeline. Instructions:
   - Create a GitLab repository and clone it to the local machine.
   - Create a `.gitlab-ci.yml` file with a basic build stage.
   - Push the file to the repository and check the pipeline execution in GitLab.

Expected Outcome**:** A successfully running CI pipeline in GitLab.

3. Adding Automated Tests to the Pipeline. Extend the pipeline to include automated testing. Instructions:
   - Modify the `.gitlab-ci.yml` file to include a test stage.
   - Use a testing framework such as PyTest (Python) or JUnit (Java).
   - Commit and push the changes, then verify pipeline execution.

Expected Outcome: Successful execution of the testing stage in the pipeline.

4. Debugging and Improving the Pipeline. Identify and resolve issues in a failing pipeline.
   - Introduce an intentional error in the codebase.
   - Observe the pipeline failure and analyze logs.

- Fix the error and push the corrected code.

Expected Outcome: Understanding of how to debug and improve a CI pipeline.

5. Please submit to the e-learning platform a short report summarizing the work, including challenges faced and solutions applied. Include a link to the GitHub repository.

Literature

Loeliger, J., & McCullough, M. (2012). *Version Control with Git: Powerful tools and techniques for collaborative software development.* O'Reilly Media.
Chacon, S., & Straub, B. (2014). *Pro Git.* Apress.
Rubalcaba, C. (2021). Git for Programmers*: Master Git for Effective Development and Deployment.* Apress.
GIT Documentation - *https://git-scm.com/doc*
GitHub Learning Lab - *https://lab.github.com/*
Atlassian GIT Tutorials - *https://www.atlassian.com/git/tutorials*
GitHub Actions Documentation - *https://docs.github.com/en/actions*
Jenkins CI/CD Documentation - *https://www.jenkins.io/doc/*