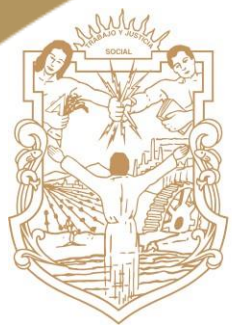

UTT

UNIVERSIDAD TECNOLÓGICA DE TIJUANA

GOBIERNO DE BAJA CALIFORNIA

TEMA:

Environment for Development and Continuous Integration

BY:

Derian Omar Tarango Mendez

GROUP:

10 B

COURSE:

Software development process management

PROFESSOR:

Ray Brunett Parra Galaviz

Tijuana, Baja California, 10 of january 2025

Environment for Development and Continuous Integration

1. Introduction

Preparing the development environment is a foundational step in software engineering. It ensures consistency, efficiency, and collaboration among team members. This document details the process for setting up an environment optimized for development and implementing Continuous Integration (CI) practices.

2. Importance of Environment Preparation

1. **Consistency: Ensures all developers work in a uniform setup, reducing errors.**
2. **Efficiency: Streamlines workflows and reduces setup time.**
3. **Collaboration: Facilitates seamless integration between team members.**
4. **Quality Assurance: Integrates CI tools to detect and fix issues early.**

3. Steps for Preparing the Development Environment

3.1 Define Requirements

- **Identify the tools, libraries, and frameworks required for the project.**
- **Ensure compatibility between versions of software and hardware.**

3.2 Version Control System (VCS)

- **Use Git for tracking changes and collaboration.**
- **Set up a repository on platforms like GitHub, GitLab, or Bitbucket.**
- **Define branching strategies (e.g., Git Flow, trunk-based development).**

3.3 Setup Development Tools

- **Install Integrated Development Environments (IDEs) like VS Code, IntelliJ IDEA, or Eclipse.**
- **Configure language-specific tools (e.g., compilers, linters).**

3.4 Dependency Management

- **Use tools like npm, pip, or Maven to handle libraries and packages.**
- **Create and share configuration files (e.g., package.json, requirements.txt).**

3.5 Containerization

- **Use Docker to create isolated and reproducible environments.**
- **Define Dockerfiles for the application and services.**

3.6 Database Setup

- **Install and configure database systems (e.g., MySQL, PostgreSQL, MongoDB).**
- **Populate with seed data for testing.**

3.7 Testing Frameworks

- **Install and configure testing tools (e.g., Jest, Mocha, JUnit).**
- **Automate tests for consistent validation.**

4. Steps for Implementing Continuous Integration

4.1 CI Tool Selection

- **Choose a CI tool (e.g., Jenkins, GitHub Actions, CircleCI, Travis CI).**
- **Ensure compatibility with the project's tech stack.**

4.2 Automate Builds

- **Set up build pipelines to compile and validate code automatically.**
- **Define build scripts (e.g., Gradle, Makefile).**

4.3 Automate Testing

- **Integrate unit, integration, and end-to-end tests into the CI pipeline.**
- **Configure tools for code coverage analysis.**

4.4 Deployment Setup

- **Implement deployment pipelines for staging and production environments.**
- **Use tools like Kubernetes or AWS CodePipeline for deployment automation.**

4.5 Notifications

- **Configure notifications for build and test results (e.g., Slack, email).**

5. Example Environment Setup

Scenario: Web Application Development

1. **Tech Stack: React (frontend), Node.js (backend), MongoDB (database).**
2. **Tools: VS Code, Docker, GitHub Actions.**
3. **CI Pipeline:**
    - **Step 1: Run linting and unit tests on every pull request.**
    - **Step 2: Build and test the application in a Docker container.**
    - **Step 3: Deploy to a staging server automatically.**

6. Best Practices

1. **Document Everything: Create onboarding documentation for new developers.**
2. **Automate Early: Introduce CI/CD practices from the start of the project.**
3. **Secure Environments: Protect credentials and sensitive data with tools like Vault or environment variables.**
4. **Regular Updates: Keep tools, libraries, and frameworks updated to avoid technical debt.**