EE 6254: DEVOPS ENGINEERING

ASSIGNMENT 3

Group Number : 47

Group Members : EG/2020/4112 : Perera H.L.D.U.G.

EG/2020/4232 : THARSI S.

Date : 27/06/2024

Part 1: CICD Design Diagram

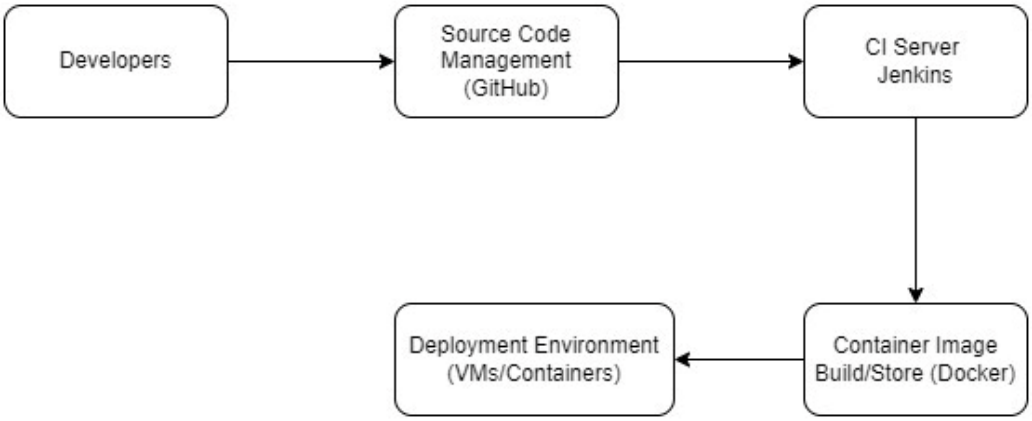

Figure 1: CICD Design Diagram

Explanation

Developers

Developers write and maintain the code for applications. They use different programming languages and tools to add new features, fix problems, and make the application run better. When they make changes, they save their code in a Git repository like GitLab/GitHub. This involves writing a commit message that explains the changes and pushing the code to the repository. Version control systems like Git help developers keep track of changes, collaborate with team members, and go back to earlier versions if necessary. To ensure high-quality code and follow coding standards, developers often review each other's code.

Git Repository

A Git repository is a central place to store code, providing version control and efficient code management. It keeps track of every change made to the code, allowing developers to revert to previous versions if needed. Git's branching and merging features let multiple developers work on different features or fixes at the same time without interfering with each other’s work. Pull requests allow for code reviews and discussions before changes are merged into the main branch. The repository also connects with CI/CD pipelines to automate building and deploying the code.

CI Server (Jenkins)

Jenkins is the main tool used for Continuous Integration (CI). It automates building, testing, and integrating code changes to ensure the code is always ready to deploy. When a commit push is detected in the Git repository, Jenkins pulls the latest code, builds it, and runs various tests to check its functionality. Jenkins then packages the application into a deployable format, like a Docker image or a JAR file. It also provides detailed logs and reports to help developers quickly find and fix issues.

Container Image Build/Store (Docker)

We use Docker to containerize our application, packaging it and its dependencies into a single, portable container. Docker images are created using Dockerfiles, which define the application's environment, dependencies, and startup commands. After they are built, these images are stored in a Docker registry, like Docker Hub or a private registry. The Docker registry manages versions and ensures the correct images are used during deployment. It also supports image scanning to detect vulnerabilities and enforce security policies. Containerization ensures consistency across different environments, from development to production.

Deployment Environment (VMs/Containers)

The deployment environment uses virtual machines (VMs) or containers to run the application VMs provide an isolated environment with a dedicated operating system, while containers share the host OS kernel but provide isolated user space. Kubernetes manages the deployment of these containers on VMs or container hosts. It ensures efficient use of resources and smooth updates. Kubernetes schedules containers based on their resource needs and availability, and it can automatically add or remove containers based on demand. This helps the application handle different levels of traffic without needing manual changes.

Connectivity in the CI/CD Pipeline

- Developers push code to the Git repository.
- Jenkins pulls the code from the Git repository, builds it, and pushes the build artifacts to the artifact storage.
- Jenkins triggers Docker image builds, which are stored in a Docker registry.

Part 2: Automation Approach

Application Tools and Dependencies

Node

- Version: 22.1.0.

React

- Version: 18.2.0
  - Purpose: Frontend javascript library for Web application Development

Express

- Purpose: Backend using Nodejs for Web application Development

MongoDB

- MongoDB Atlas(online)
- Purpose: Database management system for storing and managing application data.

Automated Application Deployment

DevOps Tools

GitHub

- Git – Version 2.4
- Purpose: GitHub is used for version control and as a code repository. It tracks code changes and allows developers to collaborate. GitHub supports branching, pull requests, and code reviews to ensure code quality and smooth integration of new features. It also integrates with CI/CD pipelines to trigger automated processes like builds and deployments when code is committed.

Jenkins

- Version: 2.375
- Purpose: Jenkins is used for Continuous Integration (CI) and Continuous Deployment (CD). It automates building, testing, and deploying applications. Jenkins pulls the latest code from GitHub, compiles it, runs tests to ensure it works, and packages it into deployable formats. It supports many plugins to extend its features and work with various DevOps tools.

Docker

- Version: 26.1.4
- Purpose: Docker is a tool that helps package applications and their dependencies into portable containers. This ensures that the application runs consistently across different environments, from development to production. Docker simplifies the process of setting up development environments, testing, and deployment. Containers are lightweight and isolated, providing a secure and efficient way to run applications.

Automated Deployment Process Using Jenkins CI/CD Pipeline

The deployment of our application is fully automated using a Continuous Integration/Continuous Deployment (CI/CD) pipeline managed by Jenkins. Here's a detailed breakdown of how it works:

Code Commit

1. Developer Actions:
    - Developers commit code changes to the GitHub.
    - Each commit has a commit message describing the code changes.
2. Triggering the Pipeline:
    - When new code changes are committed to the GitHub, a webhook configured in GitHub triggers the CI/CD pipeline in Jenkins.
    - Then Jenkins initiates the build process.

Build Stage

1. Code Retrieval:
    - Jenkins pulls the latest code from the GitHub repository, ensuring it has the most up-to-date version of the application.
2. Build Process:
    - Jenkins uses build tools like Maven to compile the code.
    - During this stage, all dependencies are resolved, and the application is compiled into executable binaries or packages.

Containerization

1. Building Docker Images:
    - Docker builds images from the application artifacts.
    - Each image includes the application and its runtime environment, ensuring consistency across different deployment environments.
2. Pushing to Docker Registry:
    - The built Docker images are tagged with version numbers and pushed to a Docker registry, such as Docker Hub or a private registry.
    - This registry acts as a central repository for all container images, making them available for deployment.

By integrating these tools and processes, our CI/CD pipeline ensures a robust, efficient, and automated deployment process that maintains high standards of code quality, reliability, and performance.