DESIGN DOCUMENT FOR CI/CD PIPELINE

Issued
05/12/2023

Ghada Yousuf

TABLE OF

CONTENTS

THE INTRODUCTION

This document delineates the architecture and workflow of the Jenkins-based Continuous Integration/Continuous Delivery (CI/CD) pipeline, integral to the DevOps ecosystem. Designed for agility and reliability, the pipeline orchestrates the flow of code from development through to production, embodying the principles of CI/CD with a multi-branch strategy.

At the core of the process is an ASP.NET Core application, demonstrating Docker's containerization within a .NET environment. Each section of this guide will walk you through the pipeline stages, from automated builds to deployment, and provide a practical overview of running and managing the application, both locally and via Docker.

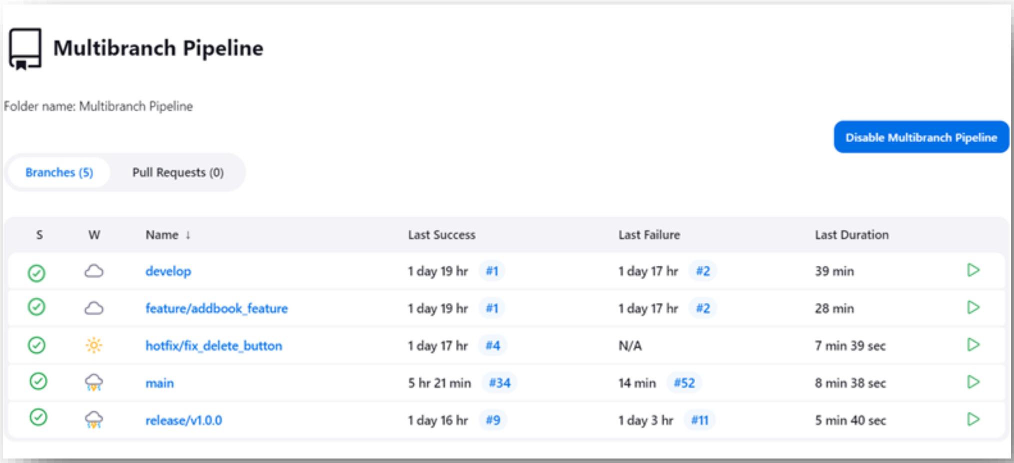
This guide aims to impart a clear understanding of the CI/CD operations, highlighting the seamless integration of development practices that ensure the codebase is production-ready at any given moment.

Through this document, you will gain insights into the structure and function of the pipeline stages, the strategic use of branches within the source code management, and the specifics of building and testing in an automated fashion. Furthermore, I will introduce you to the Book Tracker application that serves as the focal point of the CI/CD pipeline, detailing how to run it both locally and within a Docker contain

OVERVIEW OF THE SOURCE CODE MANAGEMENT AND BUILD AUTOMATION

Source Code Management

The CI/CD pipeline is designed to work in conjunction with a robust source code management strategy, utilizing a multi-branch setup as depicted in the provided Jenkins interface screenshot.



Folder name: Multibranch Pipeline

S	W	Name ↓	Last Success	Last Failure	Last Duration
✓	☁	develop	1 day 19 hr #1	1 day 17 hr #2	39 min
✓	☁	feature/addbook_feature	1 day 19 hr #1	1 day 17 hr #2	28 min
✓	☀	hotfix/fix_delete_button	1 day 17 hr #4	N/A	7 min 39 sec
✓	⛈	main	5 hr 21 min #34	14 min #52	8 min 38 sec
✓	⛈	release/v1.0.0	1 day 16 hr #9	1 day 3 hr #11	5 min 40 sec

Branching Strategy:

develop: Serves as the primary branch where the latest developments are merged and tested.

feature/*: Used for developing new features. Each feature has its own branch (e.g., feature/addbook_feature) which allows for isolated development without affecting the main or development codebases.

hotfix/*: Utilized for urgent fixes that need to be applied to the production code. These branches are used to quickly address critical issues (e.g., hotfix/fix_delete_button).

main: Represents the stable version of the application that is ready for production.

release/*: Supports preparing releases. When a release is ready, it's merged into main and tagged with a version number (e.g., release/v1.0.0).

This strategy ensures that development efforts are organized and manageable, with clear paths for features to be developed, tested, and deployed.

Build Automation

The Jenkins multi-branch pipeline automates the building of each branch using the following process:

Automated Triggers: Each push to the repository triggers an automated build for the corresponding branch, ensuring that every change is promptly integrated and tested.

Build Process:

The **develop** and **feature** branches undergo a complete build cycle using MSBuild, which compiles the .NET Core application and prepares it for testing or deployment.

Upon successful build and test execution, **feature** branches can be merged into **develop**, which is then used to update **main** after thorough validation.

Hotfix branches are built and tested with the same rigor, allowing for quick production updates when necessary.

Release branches are built, and upon successful testing, they are deployed to the production environment.

Pipeline Overview

The pipeline, implemented using Jenkins, automates various stages including code checkout, build, testing, containerization, security scanning, image pushing, deployment to Google Kubernetes Engine (GKE), and monitoring and logging.

TOOLS AND TECHNOLOGIES

The CI/CD pipeline employs a variety of tools and technologies, each selected for its effectiveness in facilitating efficient and secure DevOps practices:

Jenkins: An automation server used to implement the CI/CD pipeline, enabling the automation of various development processes.

Google Kubernetes Engine (GKE): A managed, production-ready environment for deploying containerized applications, ensuring scalable and reliable deployment.

Google Cloud Storage: Used for storing build artifacts, offering high durability and accessibility.

Docker: Facilitates containerization of the application, ensuring consistent environments for development, testing, and production.

Trivy: A security scanner for Docker images, helping identify vulnerabilities and ensuring compliance with security standards.

Git: Source code management system, enabling efficient version control and collaboration.

IAM (Identity and Access Management) on Google Cloud: Manages access control by defining who (identity) has what access (role) to which resource, enhancing the security posture of cloud resources.

Google Cloud Monitoring: Provides visibility into the performance, uptime, and overall health of applications and infrastructure. This tool is crucial for observing the behavior of the application and infrastructure, allowing for timely detection and resolution of issues.

Google Cloud Logging: Enables the collection, analysis, and storage of logs from various GKE components. This is essential for troubleshooting, monitoring application behavior, and maintaining records for audit and compliance purposes.

The combination of these tools and technologies forms the backbone of the CI/CD pipeline, ensuring not only the automation of the build and deployment processes but also adherence to security and compliance standards, along with effective monitoring and management of

THE PIPELINE STAGES

A. Tools Used in the pipeline:

is 'MSBuild',Microsoft's build system for .NET applications.

B. Environment Variables

DOCKER_IMAGE: Specifies the name of the Docker image to be used for containerizing the application.

GKE_CREDENTIALS_ID: Identifier for Google Cloud credentials stored in Jenkins, used for authentication.

DOCKER_CREDENTIALS_ID: Identifier for Docker Hub credentials stored in Jenkins, used for pushing images.

DOCKER_TAG: A unique tag for the Docker image, combining the Jenkins build number and the Git commit hash.

C. Pipeline Stages:

1. Checkout

Purpose: Retrieves the source code from the specified Git repository.

Tool: Git.

Details: Checks out the code from the branch specified in the BRANCH_NAME environment variable.

Importance: This is the first step in any CI/CD pipeline, ensuring that the most current version of the source code is available for building and testing.

2. Build

Purpose: Compiles the .NET application and prepares it for deployment.

Tools: MSBuild, .NET Core.

Details: Restores dependencies and builds the application using MSBuild**.**

Importance: The build process transforms code into a tangible artifact that can be tested, deployed, and used. It's essential for identifying any compilation errors early.

3. Upload Build Artifacts to Cloud Storage (Bonus GCP Feature)

Purpose: This stage, serving as the bonus GCP feature, stores the compiled application artifacts in Google Cloud Storage, ensuring they are securely and reliably stored in the cloud.

Tools: Google Cloud SDK (gcloud), Google Cloud Storage (gsutil).

Details: The pipeline authenticates with Google Cloud using a service account, a best practice for secure access management. It then uploads the build artifacts to a specified Cloud Storage bucket.
Importance: Provides a secure, scalable, and accessible storage solution for build artifacts, allowing for version control and facilitating rollbacks if needed.

4. Unit Tests

Purpose: This stage is crucial for ensuring code quality and reliability. It involves running unit tests, which are small and focused tests that validate the functionality of individual components or units of the application.

Tool: .NET Core CLI.

Details: The pipeline executes unit tests defined in the SampleApp.UnitTests project. These tests are designed to run quickly and isolate issues at the smallest level of the application.

Importance: Ensures the reliability and correctness of each part of the code, catching bugs early in the development cycle. Also encourages developers to write more modular, efficient, and maintainable code, which directly improves the overall quality of the application.

5. Dockersize

Purpose: Packages the application into a Docker container.

Tool: Docker.

Details: Builds a Docker image from the application, tagging it with a unique identifier.

Importance: Containerization encapsulates the application in a consistent environment, ensuring it runs the same way in any infrastructure.

6. Docker Security Scan

Purpose: Scans the Docker image for vulnerabilities.

Tool: Trivy.

Details: Runs a security scan on the Docker image using Trivy to identify potential vulnerabilities.

Importance: Identifies security issues within the container, enhancing the overall security posture of the application.

7. Push Docker Image

Purpose: Pushes the Docker image to Docker Hub.

Tools: Docker, Docker Hub.

Details: Authenticates with Docker Hub using credentials stored in Jenkins and pushes the tagged image.

Importance: Makes the image available for deployment and ensures versioning of container images.

8. Deploy to GKE (Bonus Point)

Purpose: Deploys the application to Google Kubernetes Engine.

Tools: Google Cloud SDK, Kubernetes CLI (kubectl).

Details: Authenticates with Google Cloud and applies Kubernetes deployment configurations to deploy the application.
Importance: Facilitates scalable and reliable deployment of the containerized application in a cloud environment.

9. Configure Monitoring and Logging

Purpose: Configures and verifies Google Cloud Monitoring and Logging for the application.

Tools: Google Cloud SDK.

Details: Checks the GKE cluster's logging and monitoring configurations.

Importance: Essential for ongoing oversight, performance tracking, and troubleshooting of the application in production.

10. Check Application Performance

Purpose: Monitors application performance metrics.

Tool: curl Command

Details: This stage is designed to measure the response time of a web application, ensuring that it meets performance standards.

Importance: Monitoring the response time of a web application is critical for maintaining a good user experience. Slow response times can lead to user dissatisfaction and can be indicative of underlying performance problems. This stage helps in proactively identifying such issues.

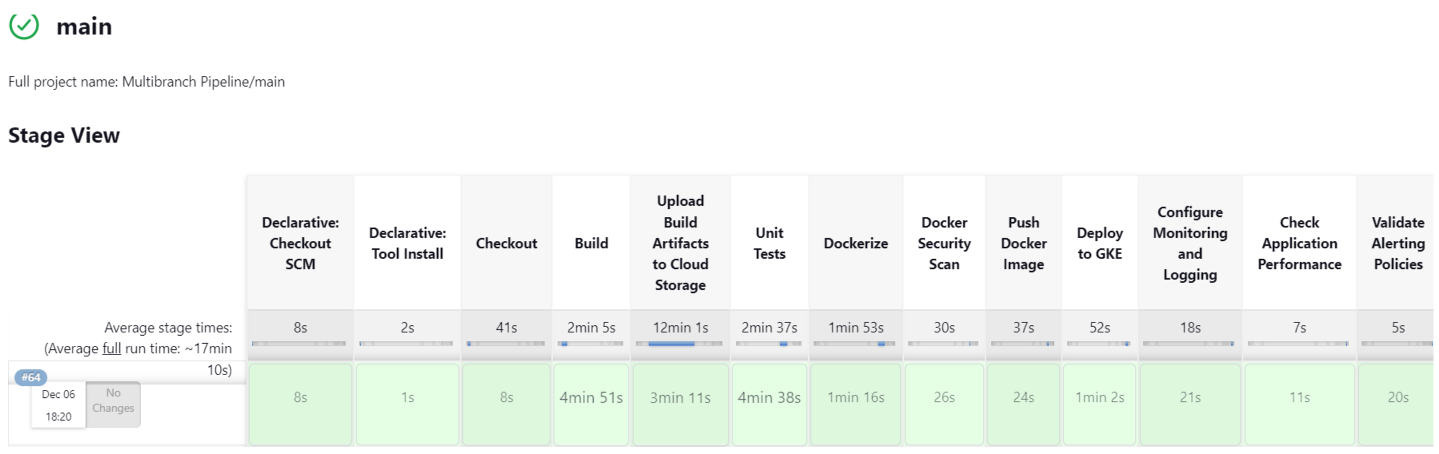
11. Validate Alerting Policies

Purpose: Ensures alerting policies are correctly set up in Google Cloud Monitoring.

Tool: Google Cloud SDK.

Details: Lists and validates the current alerting policies set up in Cloud Monitoring.

Importance: Confirms that critical alerts are in place for proactive issue resolution and system reliability.



main

Full project name: Multibranch Pipeline/main

Stage View

	Declarative: Checkout SCM	Declarative: Tool Install	Checkout	Build	Upload Build Artifacts to Cloud Storage	Unit Tests	Dockerize	Docker Security Scan	Push Docker Image	Deploy to GKE	Configure Monitoring and Logging	Check Application Performance	Validate Alerting Policies
Average stage times: (Average full run time: ~17min 10s)	8s	2s	41s	2min 5s	12min 1s	2min 37s	1min 53s	30s	37s	52s	18s	7s	5s
#64 Dec 06 18:20 No Changes	8s	1s	8s	4min 51s	3min 11s	4min 38s	1min 16s	26s	24s	1min 2s	21s	11s	20s

ACCESSING THE JENKINS PIPELINE SCRIPT

Locating the Script on GitHub

The Jenkins pipeline script, crucial for automating the CI/CD process, is hosted on GitHub. This section provides instructions on how to access and review the script.

Repository Details

GitHub Repository: ghadayi/devOps

This repository contains the scripts and configuration files used in the CI/CD pipeline.

Steps to Access the Pipeline Script

1. **Navigate to the Repository:**

Open your web browser and go to https://github.com/ghadayi/devOps.

This link directs you to the main page of the repository.

2. **Locate the Pipeline Script:**

Once in the repository, look for a file named pipeline.js.

This file contains the Jenkins pipeline script used for the project.

3. **Reviewing the Script:**

Click on the Jenkinsfile to view the contents.

GitHub provides a user-friendly interface to read the script directly in the browser.

Alternatively, you can clone or download the repository to access the file locally.

4. **Understanding the Pipeline language**

The pipeline script is written in Groovy, a powerful, versatile, and modern programming language tailored for the **Java** platform

OVERVIEW OF THE BOOK TRACKER APPLICATION

Introduction

The Book Tracker application is an ASP.NET Core application designed as part of a tutorial for a Medium post. It demonstrates the configuration of Docker for a .NET Core application, serving as a practical example of integrating modern web app development with containerization.

Application Description

Functionality: The application is a simple book tracker that allows users to maintain a list of books, add new entries, and delete books no longer in their collection.

In-Memory Database: For demonstration purposes, it uses an in-memory database that gets reset with mock data every time the application starts. This simplifies the setup and focuses on the core functionality without the need for external database configuration.

Technology Stack

.NET Core SDK: The application is built using the .NET Core SDK, highlighting its capabilities in building and running cross-platform web applications.

Docker: The application is containerized using Docker, showcasing how Docker can be used to build, ship, and run applications in isolated environments.

Running the Application

Local Execution: Instructions are provided to run the application locally using the .NET Core CLI commands.

Docker Execution: Steps to build a Docker image and run it in a container are outlined, demonstrating the ease of deploying applications with Docker.