COMP.SE.140
Continuous Development and Deployment -DevOps

Project Report

By

Saurabh Chauhan

(150917632)

Introduction

This report provides an overview of the Continuous Integration and Continuous Deployment (CI/CD) pipeline implemented for the DevOps project. The pipeline consists of three main stages: build, test, and deploy. Each stage performs specific tasks to ensure the project's code quality, reliability, and successful deployment.

CI/CD Pipeline Overview

1. Build Stage

In the build stage, the primary objective is to prepare the application for deployment by removing existing containers and images, performing a system prune, and building Docker images using the provided Docker Compose configuration. The build stage is triggered on each push to the project repository. Key actions in this stage include:

Execution of a PowerShell script (**remove_containers_and_images.ps1**) to remove existing containers and images.

Cleaning up the Docker environment using the **docker system prune** command.

Building Docker images based on the project's **Docker Compose configuration**.

2. Test Stage

The test stage focuses on ensuring the reliability and functionality of the application. It involves starting the application in Docker

containers, running tests, and validating the results. Key actions in this stage include:

Starting the application in Docker containers using **docker compose up -d**.

Running a series of tests using Postman collections (**DevopsProject.postman_collection.json**) via Newman.

Verifying the status of containers during and after testing using PowerShell scripts (**check_containers.ps1** and **check_containers_shutdown.ps1**).

3. Deploy Stage

The deploy stage is responsible for deploying the application in a production-like environment. In this case, the application is started in Docker containers. Key actions in this stage include:

Bringing up the application in Docker containers using **docker compose up -d**.

Pipeline Configuration

The pipeline is configured to run only when changes are pushed to the "project" branch. Additionally, each stage is tagged with "local," indicating that the pipeline is designed for local development and testing environments.

Testing Approach and Tools Used

Postman:

Used for creating and executing API tests.

Provides a visual environment for designing requests, defining test scripts, and managing collections.

JavaScript Test Scripts:

Leveraged JavaScript within Postman for scripting test scenarios.

Test scripts include assertions to validate response status codes, response content, and perform complex validation logic.

Asynchronous Operations Handling:

Utilized timeouts and asynchronous request handling in test scripts to accommodate delays in state changes and ensure accurate validation.

Dynamic Data Validation:

Employed dynamic data validation, such as comparing logs or states obtained in different stages of the test.

The Postman collection effectively captures the functional aspects of the DevOps project and ensures that the application behaves as expected under various scenarios. It provides a comprehensive suite of tests covering state changes, response validations, and log sequences, contributing to a robust and reliable testing strategy for the project.

Test Docker Containers Running:

Ensure that all necessary Docker containers are running as expected after the application deployment.

Created a shell script (**check_containers.ps1**) that uses Docker CLI commands to check the status of running containers.

Integrate this script into your CI/CD pipeline, possibly within the "test" stage, after the deployment of your application.

Test Docker Containers Stopped on SHUTDOWN:

Ensure that all Docker containers are stopped when the application state is set to SHUTDOWN.

Created another shell script (**check_containers_shutdown.ps1**) that uses Docker CLI commands to check if containers are stopped after the SHUTDOWN state is set.

Integrated this script into CI/CD pipeline with "test" stage, after the application state is set to SHUTDOWN.

Postman Collection Overview

The Postman collection consists of several API requests, each representing a specific scenario in the DevOps project. Here are some key aspects of the testing implementation:

Get Messages From Gateway:

Sends a GET request to retrieve messages from the gateway.

Contains tests for verifying that the status code is 200 and that the response contains logs.

Set State to INIT, PAUSED, RUNNING:

Sends PUT requests to set the application state to INIT, PAUSED, and RUNNING.

Each request has associated test scripts that validate the state change and perform additional checks, such as verifying logs after specific time intervals.

Return RUNNING/PAUSED State Correctly:

Sends GET requests to retrieve the current state and has associated test scripts to validate that the correct state is returned.

Return Correct Run Logs:

Sends a series of requests to change the state, fetch logs, and validate the logs against the expected sequence.

Utilizes timeout functions to account for delays introduced by asynchronous operations.

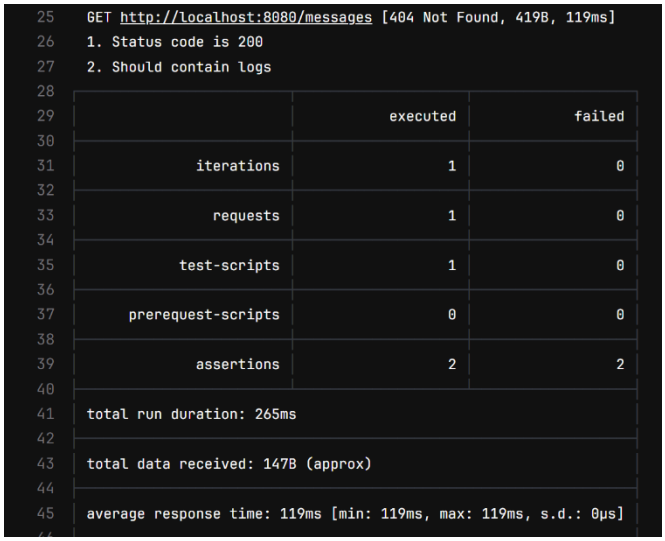
Example of Pipeline Runs

TDD has been followed it can be confirmed by looking at pipeline run history form gitlab instance provided.

But still few example are provided here

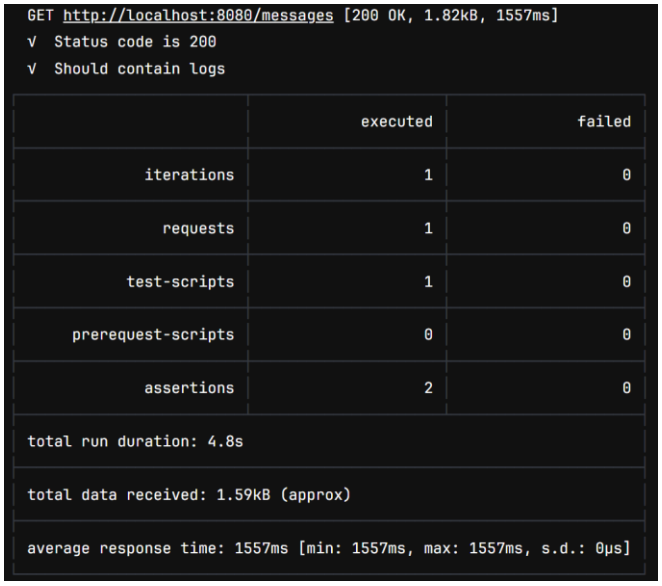
First test were added for GET /messages request and pipline was ran

https://compse140.devops-gitlab.rd.tuni.fi/saurabh.chauhan/saurabh.chauhan_private_project/-/jobs/3052



After code was added pipline ran clean

https://compse140.devops-gitlab.rd.tuni.fi/saurabh.chauhan/saurabh.chauhan_private_project/-/jobs/3056



Main Learnings

Containerization and Docker Compose:

Learning: Efficient use of Docker Compose to manage and orchestrate containerized services during the build, test, and deployment stages.

Insight: Understanding how to leverage Docker and Docker Compose for local development and CI/CD processes.

Use of PowerShell Scripts:

Learning: Incorporation of PowerShell scripts for tasks like container and image removal.

Insight: PowerShell's utility in automating Windows-centric tasks and integrating them seamlessly into the pipeline.

State Management and Synchronization:

Learning: Addressing asynchronous operations and managing states effectively during testing.

Insight: Recognizing the importance of synchronization in testing scenarios involving state changes.

Worst Difficulties:

Asynchronous Operations in Testing:

Challenge: Handling asynchronous behaviors during testing, particularly in scenarios where timing is crucial.

Insight: The complexity introduced by asynchronous operations may require fine-tuning and careful consideration in test script design.

Synchronization Challenges:

Challenge: Ensuring synchronization between different stages of the pipeline, especially when dealing with state changes and container lifecycle.

Insight: Synchronization issues can lead to false positives/negatives in test outcomes and may require additional attention.

Areas for Improvement:

Enhanced Logging and Debugging:

Recommendation: Consider enhancing logging and debugging mechanisms within test scripts to facilitate troubleshooting.

Insight: Detailed logs can provide valuable insights into the execution process, aiding in identifying issues.

Effort Estimation:

Docker Build and Cleanup Scripts: Approximately 10 hours

Postman Test Scripting: Approximately 15 hours

Integration of PowerShell Scripts: Approximately 5 hours

Addressing Asynchronous Operations: Approximately 8 hours

Synchronization Handling: Approximately 7 hours

Logging and Debugging Enhancements: Approximately 5 hours

Refinement of State Change Scenarios: Approximately 8 hours

Conclusion:

The project demonstrates a solid understanding and implementation of CI/CD practices, utilizing Docker, Postman, and PowerShell scripts. The main learnings include effective containerization, API testing, and automation with PowerShell. Challenges related to asynchronous operations and synchronization emphasize the importance of meticulous scripting. Recommendations for enhanced logging, refinement of scenarios, and continuous improvement will contribute to the project's robustness. The estimated effort reflects the comprehensive nature of the project, covering various aspects of the CI/CD pipeline and testing strategies.