DevOps Document

**Name:** Tony Jiang

**Semester:** 6

**Class:** RB04

|  |  |  |
| --- | --- | --- |
| Version | Date | Description |
| 0.1 | 28 Oct 24 | Initial document |
| 1 | 8 Dec 24 | Add version 2 |
| 1.1 | 19 Jan 25 | Add version 3 |

Contents

[Introduction 3](#_Toc188227083)

[CI/CD overview 3](#_Toc188227084)

[V3 3](#_Toc188227085)

[V2 3](#_Toc188227086)

[V1 4](#_Toc188227087)

[Actual CI/CD pipeline 5](#_Toc188227088)

# Introduction

This document contains the CI/CD pipeline, how it is set up, and the reasons for its configuration.

# CI/CD overview

This document provides an overview of the CI/CD pipeline for the project. It explains how the pipeline is set up, implemented, and versioned, leading to the final product version.

## V3

A diagram of a computer

Description automatically generated

In this version, I added integration tests to verify interactions between services, end-to-end tests to validate service endpoints by testing both happy flows and bad flows, and finally, deployment. After publishing the project on Docker Hub, it is automatically deployed to the cloud.

## V2

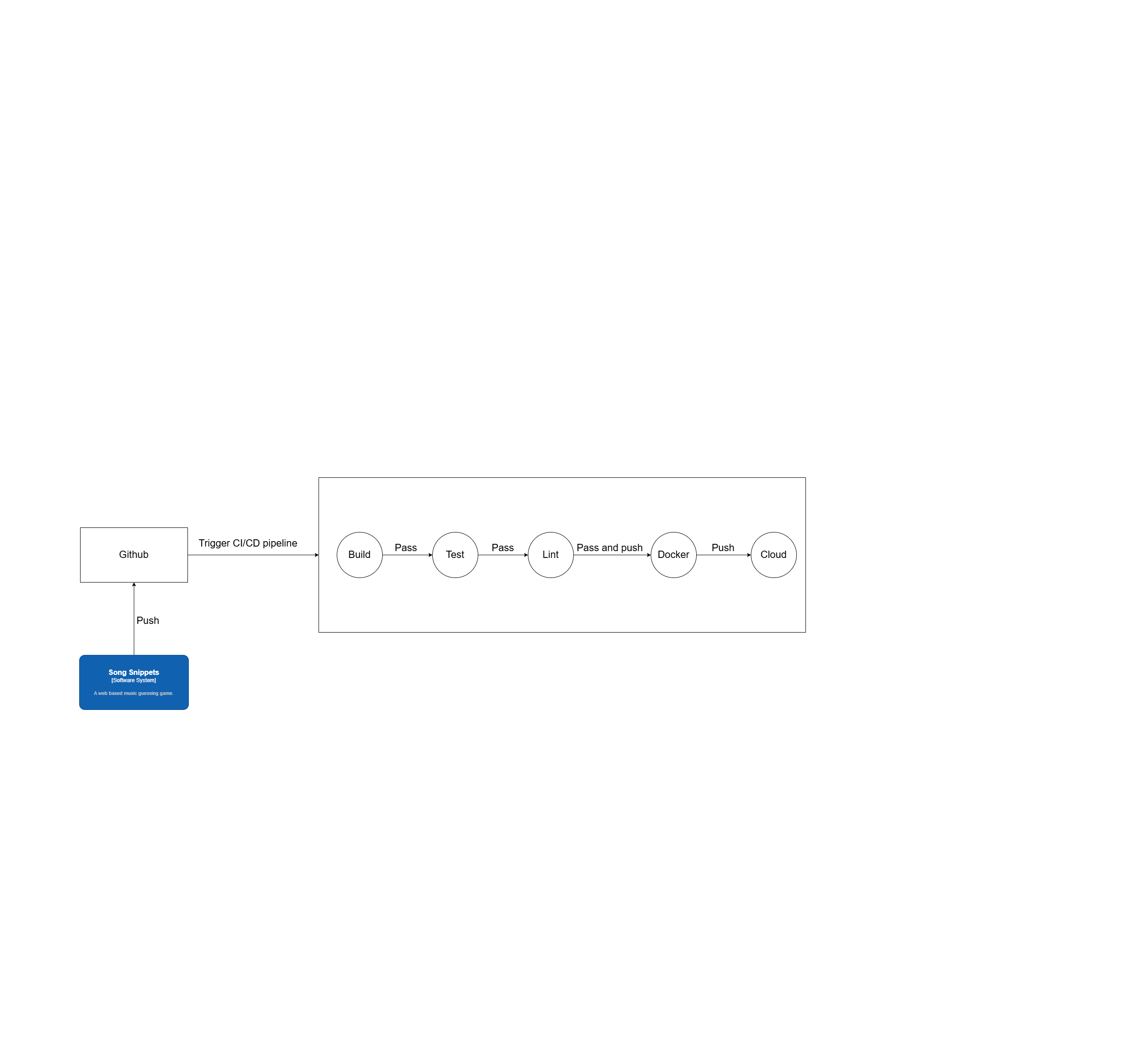
A diagram of a computer

Description automatically generated

In this version, the design structure is significantly different from the previous one. Build, Test, Lint, and Snyk now run in parallel, making the CI/CD pipeline faster compared to executing them sequentially. If any of these steps fail, the process will not proceed to Docker. If all four steps pass, Docker is executed, and the build is pushed to the cloud.

The Build task compiles the project to ensure there are no build issues. The Test task runs the unit tests to verify that no tests fail. Lint checks the code for any code smells and ensures the project follows best coding practices. Snyk scans for security vulnerabilities and outdated dependencies. Docker builds the Docker image, publishes it to Docker Hub and finally Cloud, It for deploying to project to cloud.

## V1



This pipeline is a work in progress and will be enhanced as further research is conducted. The project repository is hosted on GitHub, and GitHub Actions is used to manage the CI/CD pipeline.

The pipeline is triggered whenever changes are pushed to the GitHub repository, regardless of the branch. It begins by building the project and checking for any issues. Next, it runs the existing unit tests to evaluate functionality coverage. Once the tests pass, the pipeline lints the project to identify code issues, code smells, and ensure adherence to best practices.

If all checks are successful, the pipeline pushes the build to Docker and subsequently deploys it to the cloud.

# Actual CI/CD pipeline

A screenshot of a computer

Description automatically generated

This is the actual CI/CD pipeline for the song service. It follows the CI/CD pipeline design that I planned. All tasks in the CI/CD pipeline were successfully completed. Below is the script for the CI/CD pipeline.

A close-up of a person

Description automatically generated

The first task is to build the project to ensure it compiles successfully and to identify any issues during the build process.



The second task is to run the unit tests to check if there are any changes in the functionalities. This is helpful because it eliminates the need for manual testing and alerts you when you need to update the tests or when changes to the functionalities introduce new issues.

A screenshot of a computer program

Description automatically generated

The integration test is used to test communication with other services or databases. Communication with other services can occur via REST API or message queue. In my case, I test the integration with other services using a message queue, specifically RabbitMQ. I need to mock both the database and RabbitMQ, and I use test containers for that. For the integration test, I focus on testing the data that is published. Additionally, the application has to run in the CI/CD pipeline for it to execute the test. It’s good to have the integration test so you don’t have to manually tested every time, it can be tested automatically.

A screenshot of a computer program

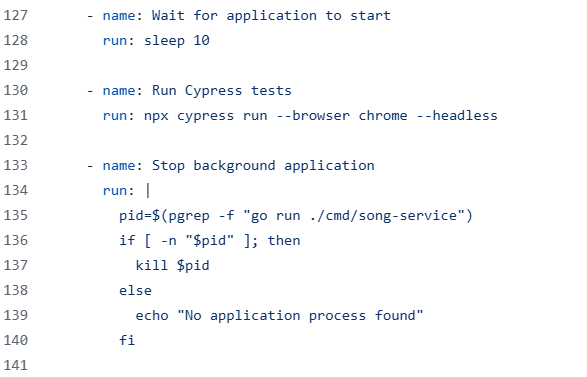
Description automatically generated



In the end-to-end test, I test the endpoint of my create song API. This test automatically checks the endpoint to ensure it returns the expected result. Also, I have to run the application to test the end to end and this can be done in the pipeline.

A screenshot of a computer program

Description automatically generated

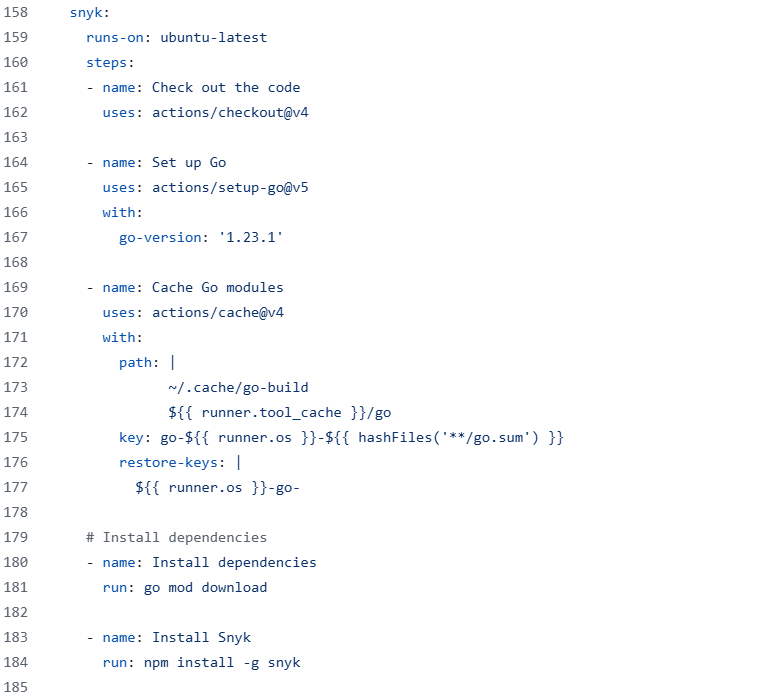


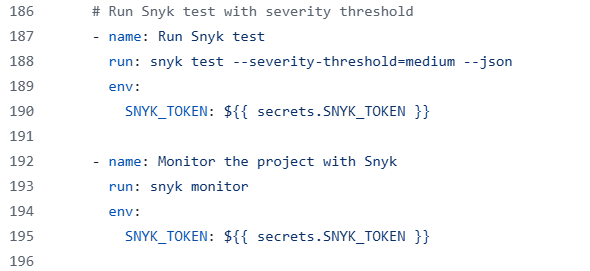
Linting is used to enforce coding best practices. It checks your entire codebase to ensure it follows best practices and identifies any code smells.

A close-up of a computer screen

Description automatically generated

Snyk is used for checking any outdated dependencies in the application.





The Docker task builds the Docker image and pushes it to Docker Hub once all tasks are completed, from the build to the Snyk task.

A screenshot of a computer program

Description automatically generated

After the Docker task is completed, the application is deployed. In my case, I’m using AWS Lambda functions. I deploy the “create song” function to AWS Lambda, ensuring that it is updated every time this pipeline is executed.

A screenshot of a computer program

Description automatically generated