Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

IBM DevOps Pipeline Project

Requirements Document

Software Engineering Project
23rd February 2021

Cormac Madden
Emer Murphy
Tom Roberts
Prathamesh Sai Sankar
Neil Shevlin
Yi Xiang Tan

1. Introduction

1.1. Overview - Purpose of system

The purpose of building a CI/CD pipeline is to facilitate DevOps[1] — the amalgamation of cultural philosophies, practices, and tools that speeds up an organization's ability to deliver applications and services at high velocity. This allows a team to evolve and improve products quicker than organizations using traditional software development and infrastructure management processes.

Working together with IBM, we plan to build a GitOps pipeline to build and deploy an application to OpenShift. This automation will include performing static analysis, unit testing, code coverage, packaging in the form of a container using Podman[2] and deploying onto Red Hat OpenShift[3].

1.2. Scope

In order to build this GitOps CI/CD pipeline project, there are a few things that must be developed,

1. A web application using Node.js to test the deployment of the code.
2. Automated testing and static program analysis.
3. Automated deployment to Red Hat OpenShift.

1.3. Objectives and Success Criteria

Our objectives are:

Familiarise ourselves with the technologies available to build CI/CD Pipeline.

Decide on the suitable infrastructure to build the pipeline.

Develop a rudimentary webapp using Javascript & Node.js

Create a pipeline using GitActions that will:

- Build and image of the application using podman
- Perform static analysis, unit testing and code coverage on the image using Webdriver.
- Package our application in the form of a container.
- Deploy the application to Openshift.

Further develop the application to use a public Covid API.

Create more graph and visualisation options for the webApp

Create clear documentation of the plan and the process.

Deliver the code bundle to the client before April 23rd.

Our Success Criteria are:

Completing the objectives outlined.

Satisfying our client with the code we deliver.

Ultimately having a pipeline that will build and deploy an application on to OpenShift.

1.4. Definitions, abbreviations

- **CI/CD:** Continuous Integration/Continuous Delivery.
- **CI/CD pipeline:** This helps us to automate the steps in the software design process. The pipeline builds code, runs tests (CI), and safely deploys a new version of the application(CD). Automated pipelines remove manual errors, provide standardized feedback loops to developers, and enable fast product integration.
- **Standardisation:** This is a common format of a document or file that is used by one or more software developers while working on the same computer program. Software standards allow for the interoperability between different programs created by different developers.

2. Current system

Pipelines for building node.js applications, performing tests and deployment are widely available.

We will be using this project below, as a reference for some of our work.
https://github.com/christophernixon/DevOps-Pipeline-sweng?fbclid=IwAR0yLz6i-m9xEMPvn06DXHnyWXYT-p4yzV9VmGzWeRELuFqoo2L7NsHSL5g
This project was created in 2020 by another group of students doing a similar project to this.

Their pipeline had the following structure: **Git - Travis [React - Jest - Docker - IBM]**

We will be using GithubActions instead of Travis because one of our group members has some experience with it and setting it up is relatively straightforward.

We plan on using Javascript and Webdriver instead of React and Jest for developing and testing our application. We decided on this based on the experience of our team-members.

We will be using Podman instead of Docker. Podman is quite similar to Docker. They use the same commands and there is compatibility between podman and docker images.

We will also be deploying the pipeline to OpenShift instead of IBM cloud. This is one of the requirements from the client

Updated Pipeline Structure: **Git - GithubActions[Node.js - Webdriver - Podman - Openshift]**

Available CI solutions:

GitHub Actions	Built-in support by Github with documentations covering most of our use cases.
Travis CI	Ideal for open source projects that require testing in multiple environments.
Jenkins	Suited for larger projects that require a high degree of customization.

Other options include Gitlab, Circle CI and Codeship.

Version-control platforms:

GitHub	Distributed version control system.
SVN	A centralized version control system.
Apache Subversion	Distributed version control system.

Containerisation:

Podman	We are mandated to use this.
Docker	Uses OS-level virtualization to deliver software in packages called containers.

3. The Proposed System

3.1. Overview

Our design is for a complete CI/CD pipeline and a simple application to demonstrate the use of this pipeline. The application will visualise covid statistics on a dashboard. Version-control will be handled with Git to enable collaboration of work and track modifications to the files, and we'll use GitHub Actions to perform Continuous Integration. To test code quality we'll be using Webdriver. There won't be a database, it will be replaced with an external API for demonstration. The simple application to test the effectiveness of pipelines will be a Webapp made with JavaScript and Node js.

3.2. Functional Requirements

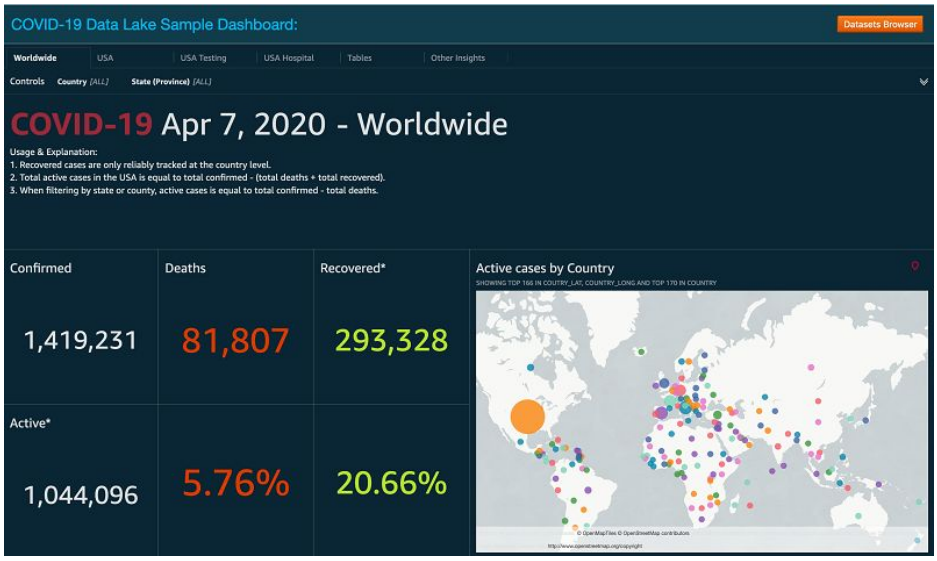
- The pipeline should take code submitted by developers on GitHub, build it. Then perform static analysis such as unit testing.
- The DevOps pipeline must package the application using podman, and then deploy the containers onto a Red Hat OpenShift Container Platform (OCP).
- The containers should then run individually on Linux systems using `podman`.

3.3. Non-functional requirements

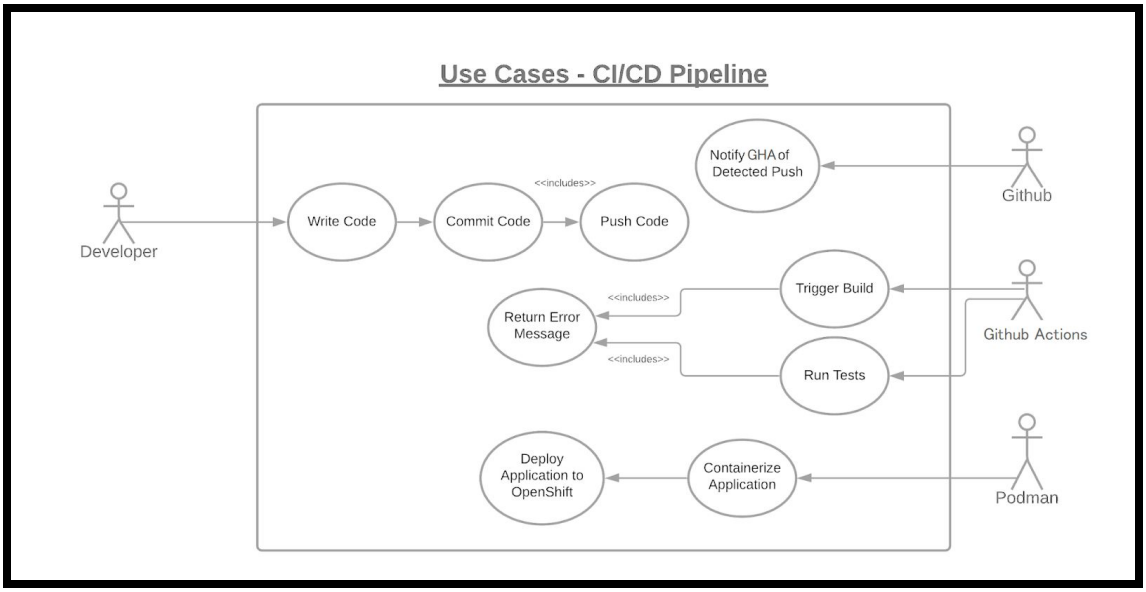
- **Agile**
    - Client meetup to demonstrate progress on a biweekly basis
- **Reliability**
    - Performs checks and code analysis for builds
- **Serviceability**
    - Clearly documented code
    - Documentation of key architectural decisions

3.4. System prototype (models)

User interface mockups



Use cases



Use Case Descriptions

Name: Push Code **Participating Actor:** Developer **Entry Condition:** The developer must have previously written their code and committed it properly. **Exit Condition:** The developer has successfully pushed their code. **Normal Scenario:** 1. The developer writes their code to the best of their ability. 2. The developer commits their code that they are satisfied with. 3. The developer pushes their code onto the code repository. **Error Scenario:** ● The developer tries to push without a commit beforehand.	**Name:** Notify Github Actions of Detected Push **Participating Actor:** Github **Entry Condition:** The developer must have pushed code to the code repository for Github to notify Github Actions. **Exit Condition:** Github Actions has been successfully notified, to kick off the CI build. **Normal Scenario:** 1. The developer pushes their code onto the code repository. 2. Github recognizes this push and notifies Github Actions. 3. Github Actions receives a notification from Github and starts the CI build. **Error Scenario:** ● Github notifying Github Actions when there was no push to the code repository.

Name: Trigger Build
Participating Actor: Github Actions

Entry Condition: Github has successfully notified Github Actions of a code push.
Exit Condition: The code will be successfully built.

Normal Scenario:

1. Github Actions receives a notification from Github.
2. Github Actions starts the CI build.

Error Scenario:

- There was an error when building the code.

Name: Run Tests
Participating Actor: Github Actions

Entry Condition: Github has successfully notified Github Actions of a code push.
Exit Condition: The code will be successfully tested.

Normal Scenario:

1. Github Actions receives a notification from Github.
2. Github Actions starts the QA testing with selenium.

Error Scenario:

- There was an error when testing the code.

Name: Containerize Application
Participating Actor: Podman

Entry Condition: The code has been successfully built and tested.
Exit Condition: The application will be containerized.

Normal Scenario:

1. The code has been successfully built and tested.
2. Podman containerizes the application.

Error Scenario:

- Trying to containerize an application that has not been built and tested.

Name: Deploy Application to OpenShift
Participating Actor: Podman

Entry Condition: The application has been containerized.
Exit Condition: The application will be deployed on OpenShift.

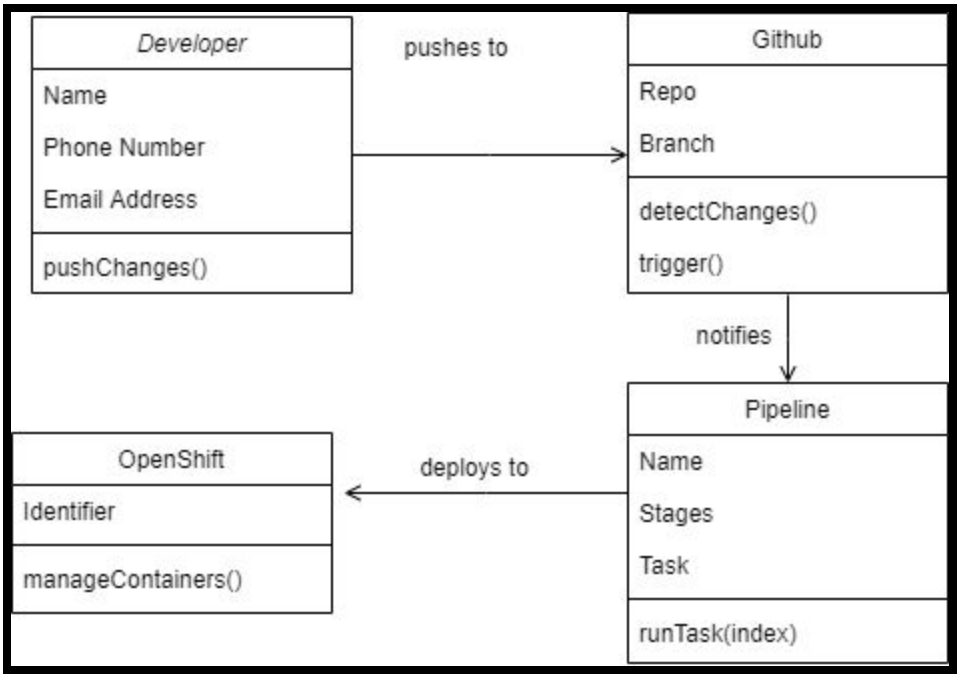
Normal Scenario:

1. The application has been containerized already.
2. OpenShift manages the containers that have been deployed.

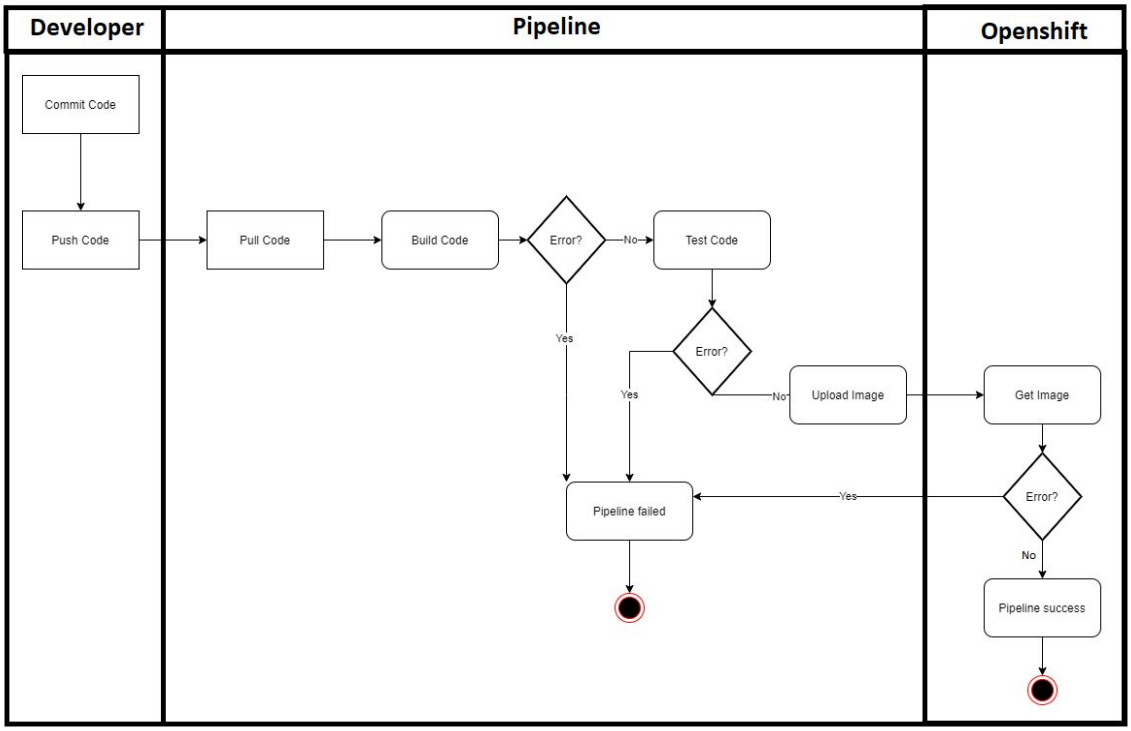
Error Scenario:

- Deploying an application that has not been containerized.

3.4.1. Object model



3.4.2.Dynamic model



References

What is DevOps? - Amazon AWS

Podman

Red Hat OpenShift

https://learn.openshift.com

Deploying to OpenShift using GitHub Actions

OpenShift Pipelines with Jenkins, Tekton and more...

Podman and Buildah for Docker users – Red Hat Developer Blog

https://developers.redhat.com/developer-sandbox