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

IBM DevOps Pipeline Project

Project Plan

Software Engineering Project
23rd February 2021

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

Contents

1. Project Goals and Objectives

1.1. Background and Summary

This project is the core component of the Trinity College Software Engineering Module (CSU33013/CSU22013). For this project, we have been tasked with developing a continuous integration and continuous deployment pipeline. The client for this project is Criveti Mihai from IBM and we will be working to deliver a pipeline that suits his requirements.

The pipeline's primary function will be to build, bake, and deploy a web application that has been pushed to a git repository. The pipeline will also include static analysis, unit testing and checks for code coverage.

We have been given flexibility with regards to which technology we use for our pipeline but the application will need to be deployed to Red Hat OpenShift.

1.2. Goals

Our goal is to automate the repetitive and monotonous steps taken by developers during the development lifecycle. Steps that this process will automate include: building, testing, packaging and deploying the code. This will assist the client by speeding up their workflow and allow them to spend more time and attention on developing their code or application.

Our goal is to ensure that the product we deliver to our client meets the requirements we agreed upon at the beginning of the project.

1.3. Objectives

Create a pipeline using GitHub Actions 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.

Have our code available as an open-source project on GitHub.

2. Project Scope

2.1. Project Deliverables

Deliverables	Due Dates
Signed off Requirements Document	01/03/2021
Requirements Presentation	
IBM Progress Demonstration	16/03/2021
Project Plan	22/03/2021
Software Design Specification Document	
Final Presentation	06/04/2021
Project Development Report	23/04/2021
Project Management Report	
Software System (code bundle)	
Individual Reflective Essay	

2.2. Project Boundaries

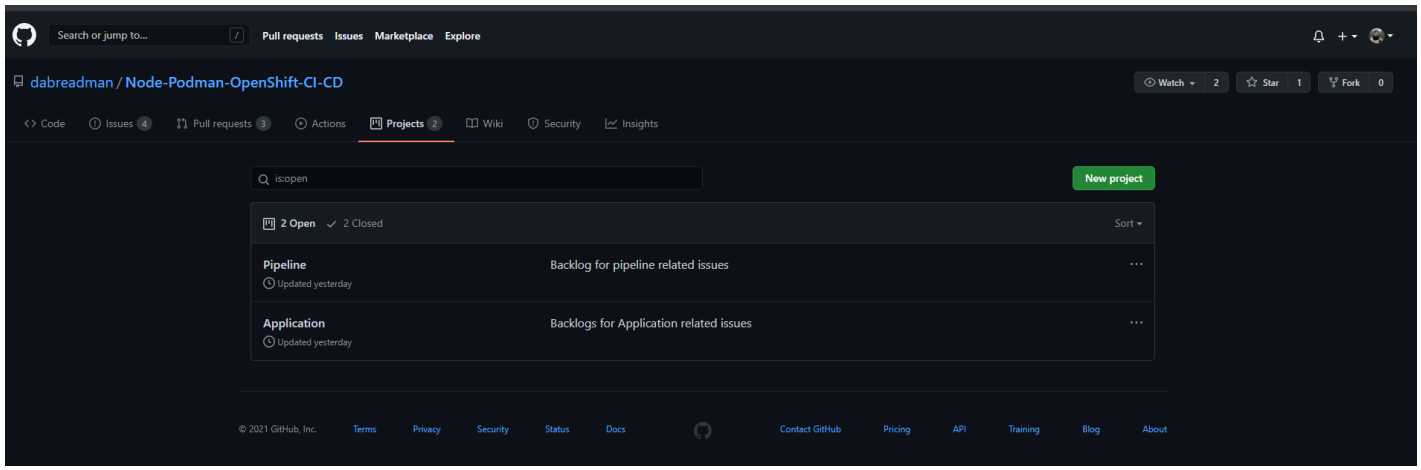
In Scope	• Automated building • Automated testing & static program analysis • Automated containerisation • Automated deployment to Red Hat OpenShift • Creating a web application to test the pipeline of the code.
Out of Scope	• Setting up a messaging system

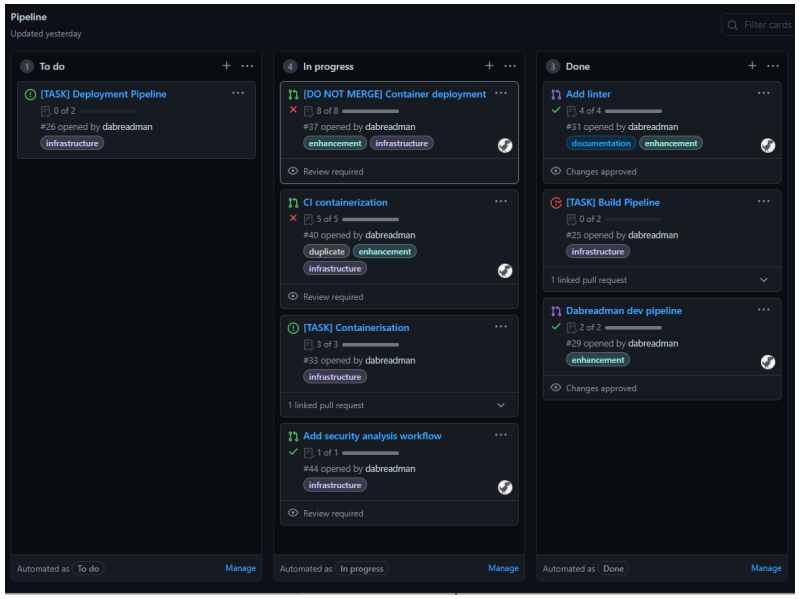
2.3. Product Backlog

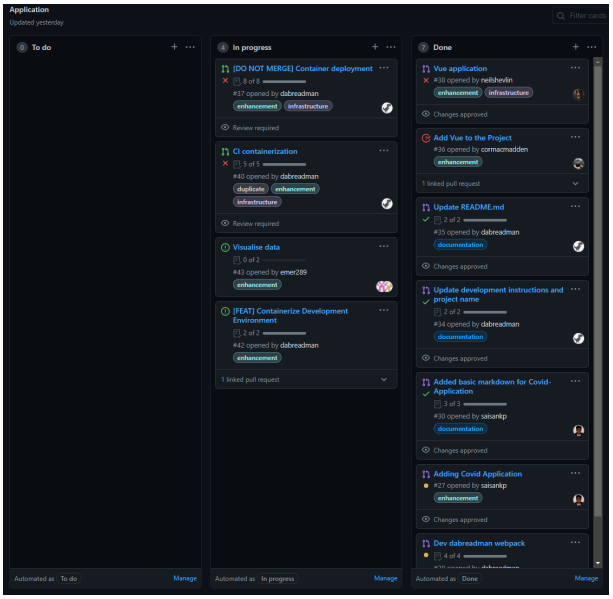
We are organising our product backlog using GitHub Project.

Below are screenshots of the backlog as of: 17/03/2021. of the repository.

We divided the backlog into two projects, one for the pipeline and the other for the application.







We decided to use GitHub Project to keep track of our tasks because of how well it is integrated with GitHub (which we had already decided to use) and the extra functionality.

3. Project Approach.

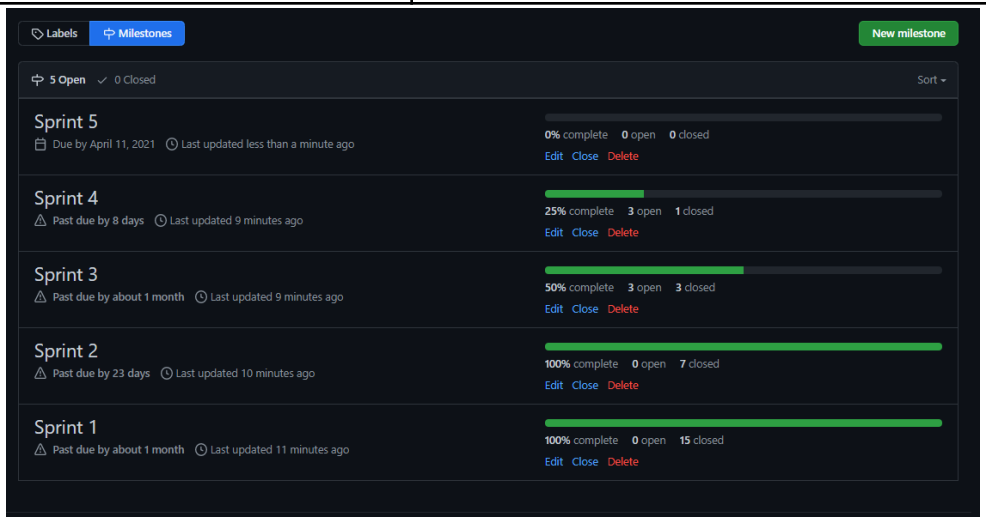
3.1. Scrum Sprints

The average duration for each of our sprints is two weeks. We decided we should have at least two scrum meetings for each sprint, however, when possible we aim to meet more.

At the start of each sprint, we clearly indicate the overall objectives of the sprint, the start and end dates of the sprint, and the organised dates for the following meetings.

We used GitHub Milestones to track the sprints and linked the relating issues to each milestone.

Sprint 1 (01/02/2021 - 14/02/2021)	• Communication setup • Project choices • Repository setup • Meeting schedule • Git/GitHub 101
Sprint 2 (15/02/2021 - 28/02/2021)	• Trivial Application • Technology Experimentations • Architectural Decisions • Requirements Document • Requirement Presentation
Sprint 3 (01/03/2021 - 14/03/2021)	• Vue Migration • Data Listing • Continuous Integration Pipeline • Project Documentation • Application Containerisation • Tests Containerisation
Sprint 4 (15/03/2021 - 28/03/2021)	• Development Containerisation • Continuous Delivery Pipeline • Data Visualisation • Coverage and Unit Tests • Project Plan • Software Design Specification



4. Project Organisation

4.1. The Team

Our team is made up of two third-year and four second-year students. Their prior experience with similar projects and prior technical skills are organised in the table below.

Name	Prior experience in projects	Prior Technical Skills
Yi Xiang Tan (dabreadman)	• Summer internship in localisation	• Web development • Scripting
Cormac Madden	• Developing Hololens Application	• Photoshop automation script writing,
Emer Murphy	• Programming Project CS1013	• Java • C
Tom Roberts	• Programming Project CS1013	• Java • C • Python
Prathamesh Sai Sankar	• Stock Market Visualizer using the Processing Java Library	• Java • C
Neil Shevlin	• Stock trading Application	• Web development

4.2. Staff Chart

The following are a series sprint staff chats outlining the roles of each member during the sprints and what each member has been and will be working on.

The scrum master role is rotated through the team for every sprint

Product owner	Mihai Criveti @ IBM
Team Leader	Yi Xiang Tan

Week 1	Completed	Ongoing
Yi Xiang Tan **[Scrum master]**	• Team organization • Project choices	• GitHub repository creation
Cormac Madden	• Project choices	• Technology research • Contact client
Emer Murphy	• Project choices	• Technology research
Tom Roberts	• Project choices	• Technology research
Prathamesh Sai Sankar	• Project choices	• Technology research
Neil Shevlin	• Project choices	• Technology research • System proposal documentation

Week 2	Completed	Ongoing
Yi Xiang Tan	• GitHub repository creation • GitHub messaging to Discord with webhook	• Requirements document
Cormac Madden **[Scrum master]**	• Technology research • Contact client	• Contact client • Technology research
Emer Murphy	• Technology research	• Solution technology research
Tom Roberts	• Technology research	• Requirements document
Prathamesh Sai Sankar	• Technology research	• Requirements document
Neil Shevlin	• Technology research • System proposal documentation	• Architectural decisions documentation

Week 3	Completed	Ongoing
Yi Xiang Tan	• Requirements document	• Creating backlogs
Cormac Madden	• Contact client	• Requirements Presentation
Emer Murphy **[Scrum master]**	• Solution technology research	• Requirements document
Tom Roberts	• Requirements document	• Requirements presentation
Prathamesh Sai Sankar	• Requirements document	• Requirements document
Neil Shevlin	• Architectural decisions documentation	• Creating backlogs

Week 4	Completed	Ongoing
Yi Xiang Tan	• Creating backlogs	• Build pipeline
Cormac Madden	• Requirements document	• Requirements presentation
Emer Murphy	• Requirements document	• Project plan
Tom Roberts **[Scrum master]**	• Requirements document	• Requirements presentation
Prathamesh Sai Sankar	• Requirements document	• Project initialization • Self-introduction research
Neil Shevlin		

Week 5	Completed	Ongoing
Yi Xiang Tan	• Creating CI pipeline	• Project containerization and CD pipeline
Cormac Madden	• Requirements document & presentation	• Vue migration
Emer Murphy	• Project timeline	• Data visualisation
Tom Roberts	• Requirements presentation	• Data visualisation

Prathamesh Sai Sankar **[Scrum master]**	• Project initialization	• Vue migration
Neil Shevlin		

Week 6	Completed	Ongoing
Yi Xiang Tan **[Scrum master]**	• Altering CI pipeline	• Building CD pipeline
Cormac Madden	• Vue setup	• Project plan
Emer Murphy	• Learning technologies	• Data visualisation
Tom Roberts	• Learning technologies	• Data visualisation
Prathamesh Sai Sankar	• Project initialization	• Vue migration
Neil Shevlin		• Project follow-up

Week 7	Completed	Ongoing
Yi Xiang Tan	• Application and development containerization	• Project plan
Cormac Madden	• Project plan	• Project plan
Emer Murphy	• Data visualisation	• Data visualisation
Tom Roberts	• Data visualisation	• Data visualisation
Prathamesh Sai Sankar	• Vue migration	• Test suite design
Neil Shevlin **[Scrum master]**	• Vue migration	• Application development

5. Risk Analysis

5.1. Risk Analysis

Identify the risks to the successful conclusion of the project and calculate their Risk Factor

Risk Element	Impact (1 to 5)	Likelihood (1 to 5)	Risk Factor (I*L)
Unfamiliarity of technology	4	4	16
Excessive planning of architecture	3	5	15
Errors due to different development environment	3	5	15
The result is different from the requirements	5	2	10
Progress stall due to miscommunication	4	2	8
Inferior code quality	2	4	8

5.2. Risk Mitigation

The risks (ordered by their risk factor) and measures that will be taken to reduce them

Risk	Measures to Reduce Risk
Unfamiliarity of technology	• Early research • Congregated resource for learning the technologies • Project references • Explanation of workflow and technologies
Excessive planning of architecture	• Making architectural decision early on • Using popular technology stack • Writing transferable code for possible pivoting
Errors due to different development environment	• Containerised application • Containerised development environment • Module version lock files
The result is different from the requirements	• Bi-weekly client meetup with progress demonstration • Create backlogs from requirements documents • References similar products
Progress stall due to miscommunication	• Weekly scrums with progress updates and task allocation • Detailed feature request • Mandatory code reviews with required status checks
Inferior code quality	• Branch protection policies • Mandatory code review • Linting and testing suite

6. Project Controls

Outline the mechanisms and tools used to control all aspects of the project execution, progress, and quality, deliverables, deadlines and communication.

Tools	Functionality
Git	Source control
GitHub	Code hosting platform
GitHub Projects	Outline of tasks
Discord	Structured communication and video conference
Jist Meets	Recording of video conference
Google Meet	Video Conference with client
Google Drive	Collaboration of documentation and storage solution of said documentation

7. Communication

7.1. Client Communication

We communicate with our client Criveti Mihai over email and through our biweekly meetings on Google Meet. We have set up a calendar event so that he can reschedule it if needs be, and so that we are all sent a reminder shortly before each meeting.

Meeting	Date	Outcome
Introductory Meeting	15/02/2021 @ 15:00	• Learned the expected result from this project and Criveti's main priorities. • Received a useful list of learning resources from Criveti.
Requirements Sign Off	01/03/2021 @ 15:00	• Set expectation of minimal viable product
Progress Demonstration	16/03/2021 @ 15:30	• Get feedback on the demonstration • Assured progression and direction

7.2. Project Team Meetings

Outline the schedule of team meetings and any requested reports

Our main method of communication is through our Discord Server. This is an informal place that allows us to ask questions at any time and receive a quick response. It is beneficial when it comes to scheduling impromptu meetings or notifying others of changes regarding the project.

Finally, one of the most important methods of communication is our weekly scrum meetings. Here we can work together, ask questions, air out any issues, and plan for the next week. Communication is key to keep the project running smoothly and remaining on track.

Date	Result
03/02/2021	• Self-introduction • Project choice
10/02/2021	• Meeting schedule • Interest expression • Position allocation • Task allocation • Default communication channel
17/02/2021	• Requirements documents review • Architectural Decisions • Git 101
24/02/2021	• Workflow demonstration • Progress demonstration
03/03/2021	• Requirements presentation review • Requirements presentation allocation • Requirements presentation practice
10/03/2021	• Project rundown • Git revision • Performance review

8. Appendix

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 web application using Javascript & Node.js

Create a pipeline using GitHub Actions that will:

- Build and image of the application using Podman
- Perform static analysis, unit testing and code coverage on the image using WebDriver and Jest.
- 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 web application.

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 used an existing pipeline used by IBM as a reference for a system that is currently available. That pipeline had the following structure: **Git - Travis [React - Jest - Docker - IBM]**

We will be using GitHub Actions 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 Podman and docker images are compatible.

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 - GitHub Actions [Node.js - WebDriver - Podman - OpenShift]**

Available CI solutions:

GitHub Actions	Built-in support by GitHub with documentation 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. Can run rootless.
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 web application made with JavaScript and Node js.

3.2. Functional Requirements

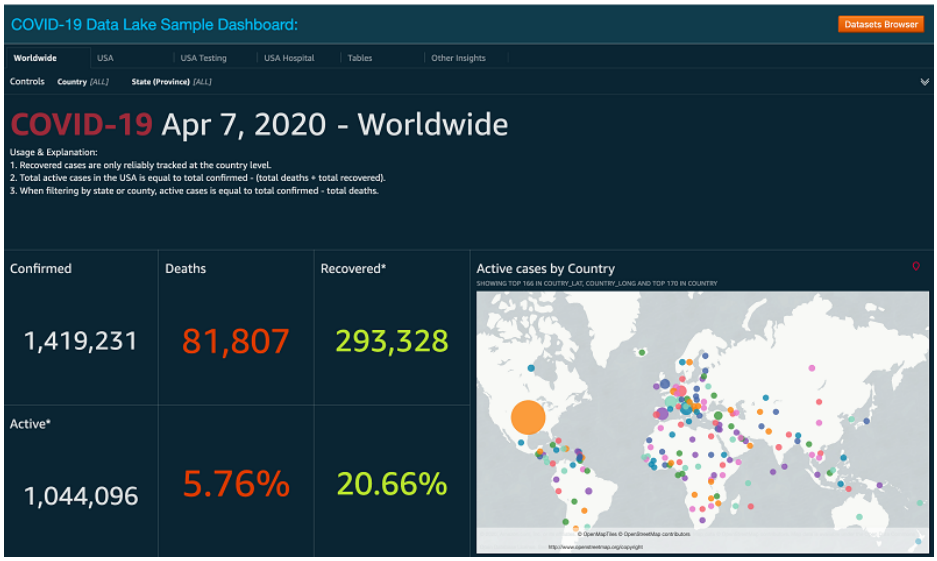
- The pipeline should take code submitted by developers on GitHub, build it. Then perform static analysis such as unit testing.
- The DevOps pipeline must package the application using Podman, and then deploy the containers onto a Red Hat OpenShift Container Platform (OCP).
- The containers should then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

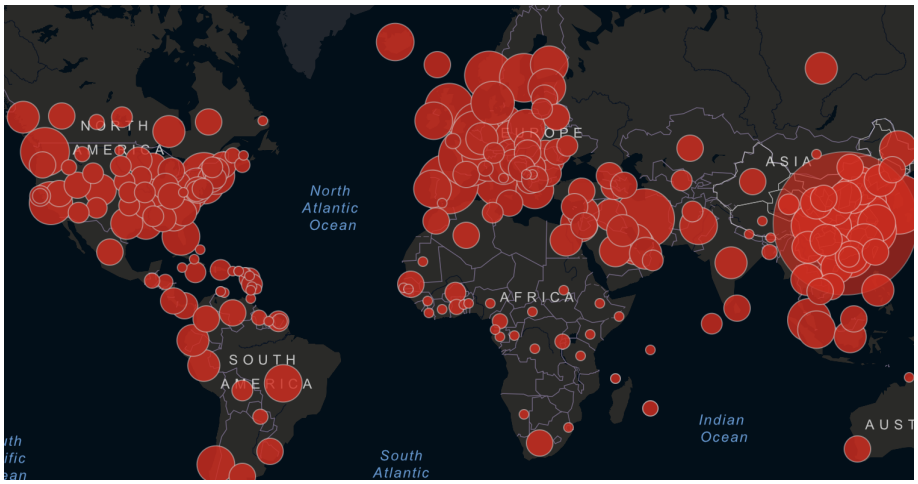
3.3. Non-functional requirements

- **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

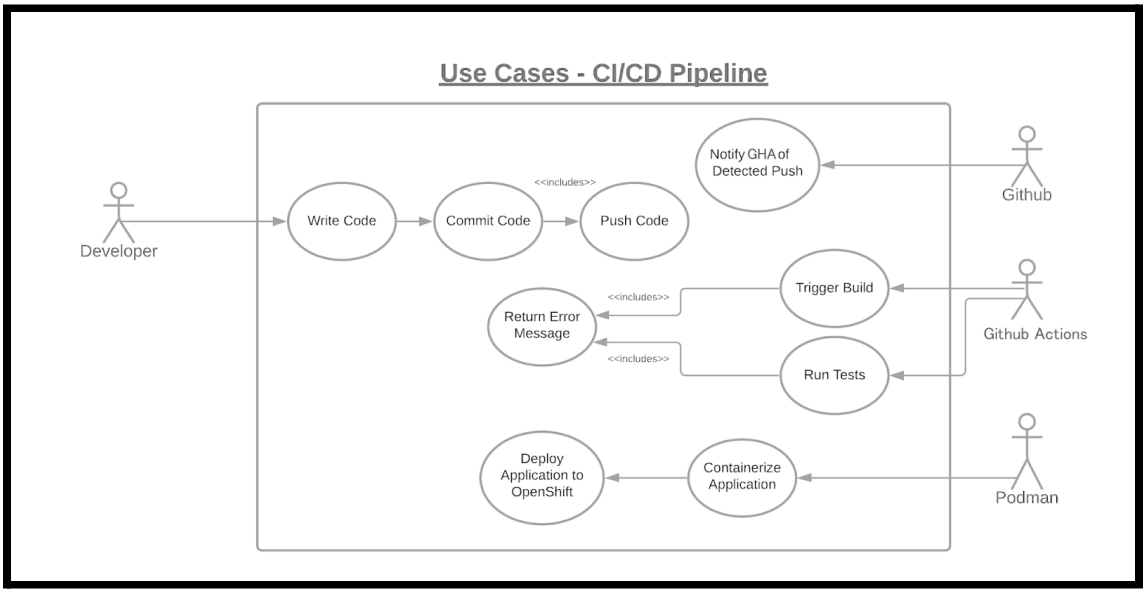
3.4. System prototype (models)

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.

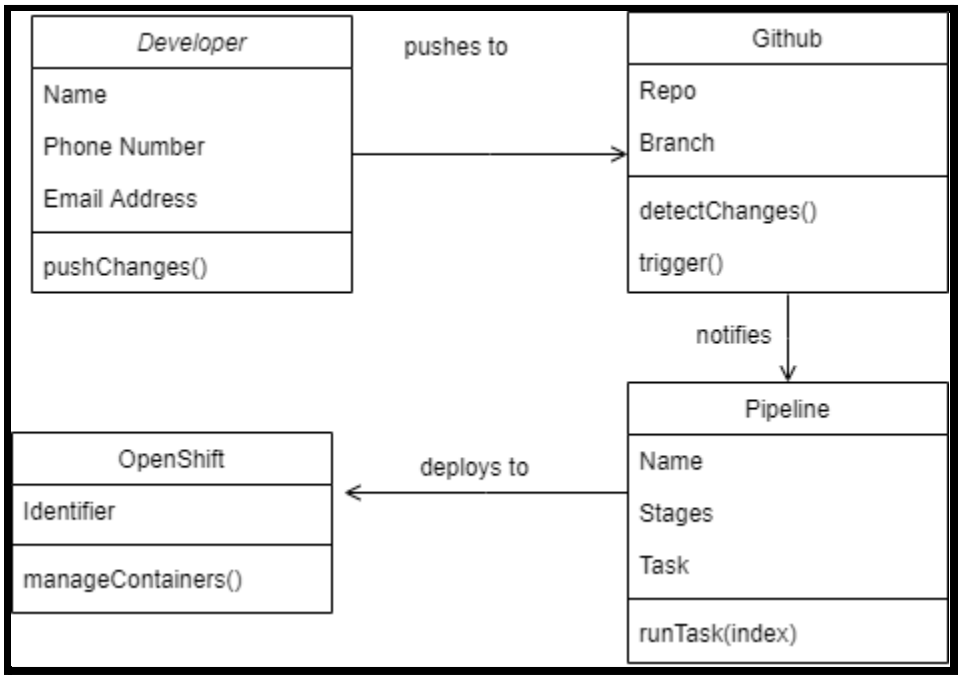
Normal Scenario:

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

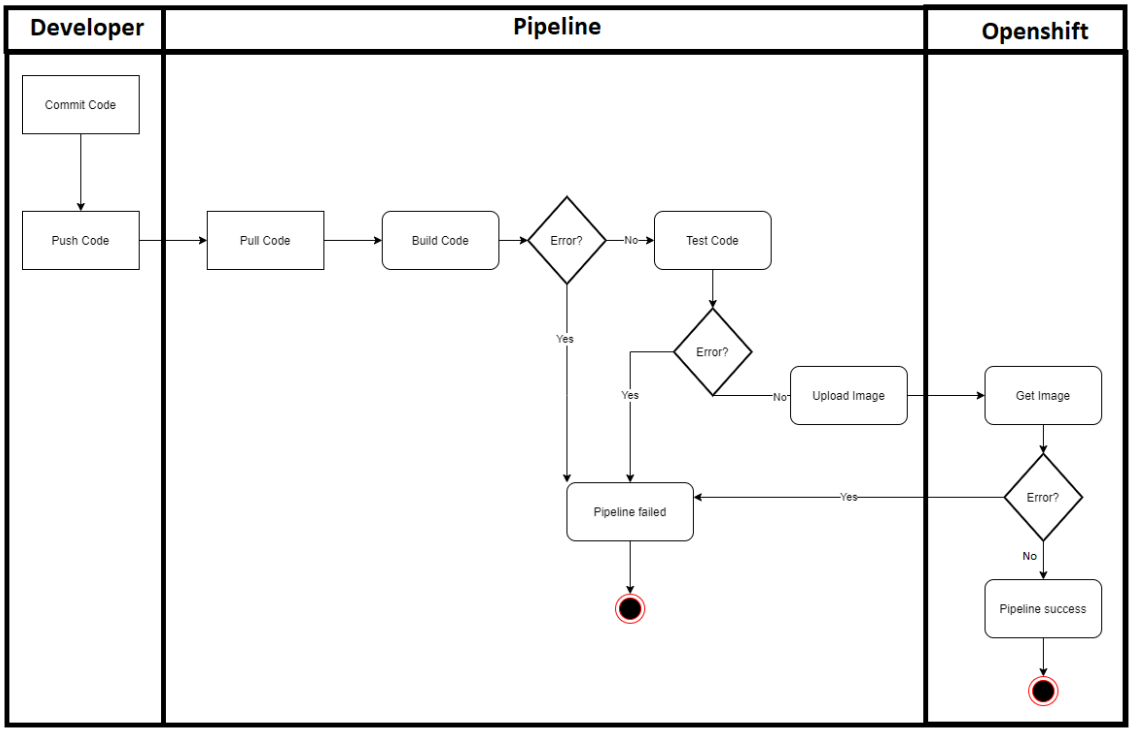
Error Scenario:

- 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

Software Design Specification

1. Introduction

1.1. Overview - Purpose of system

The purpose of building a CI/CD pipeline is to support **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 develop 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 Scope	• Automated building • Automated testing & static program analysis • Automated containerisation • Automated deployment to Red Hat OpenShift • Creating a web application to test the pipeline of the code.
Out of Scope	• Setting up a messaging system

1.3. 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.
- **Containerisation:** The process of packaging an application along with its required libraries, frameworks, and configuration files together so that it can be run in various computing environments efficiently.
- **Deployment:** This refers to the process of running an application on a server or device.
- **Image:** an immutable file that contains the source code, libraries, dependencies, tools, and other files needed for an application to run.

Software Design Specification

2. System Design

2.1. Design Overview

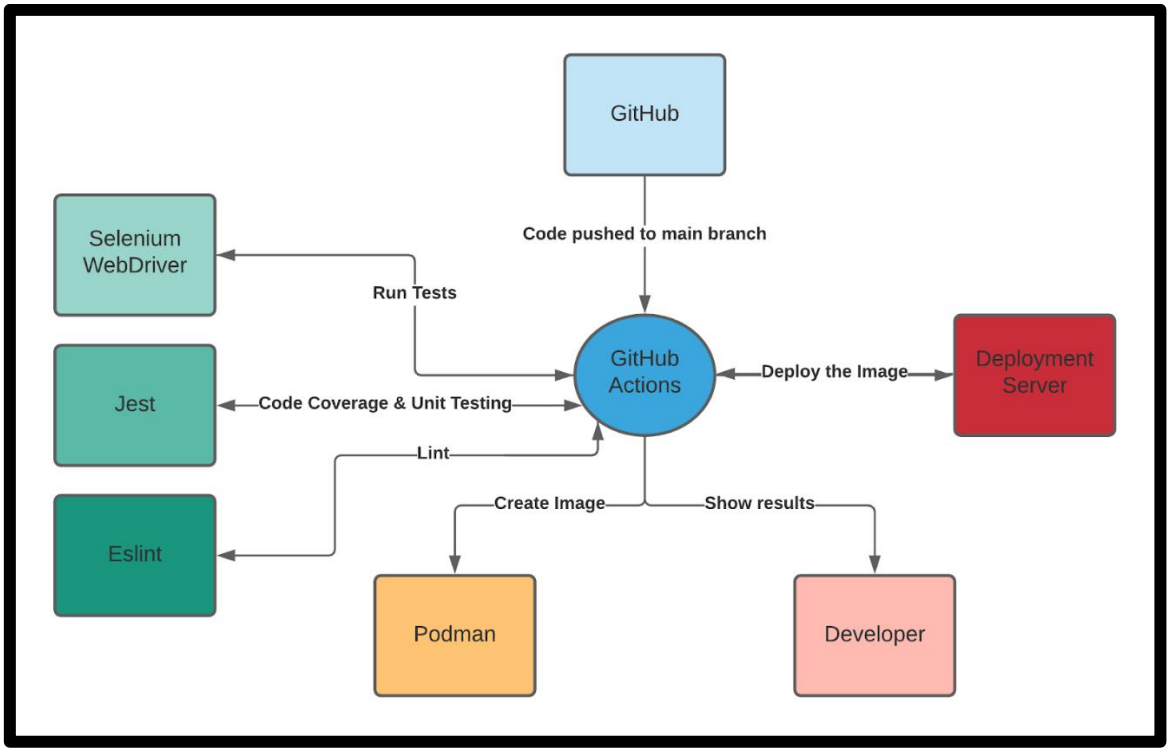
Our design is for a fully functional **CI/CD pipeline** and a **simple application** to demonstrate the use of this pipeline. This will allow developers to write, commit, and push code into a repository, and the code will continuously be built and tested.

2.1.1. High-level overview of how the system is implemented, what tools, frameworks and languages are used etc.

The application we will use to demonstrate out pipeline will be a simple application showing **Covid-19** [4] **statistics on a dashboard**. The version-control will be handled with **Git** [5] to enable collaboration of work and track modifications to the files and we will use **GitHub Actions** [6] to perform Continuous Integration. For testing, we will be using **Selenium Webdriver** [7]. For code coverage and static unit testing, we will be using **Jest** [8]. For linting, we will use **Eslint** [9]**.** For containerization, we will use **Podman.** There will not be a database, it will be replaced with an **external API - Rapid API** [10] for demonstration. Finally, the application will be deployed onto **Openshift.** This simple application will be a Webapp made with **Vue.js** [11] for the front-end, and **Node.js** [12] for the back-end.

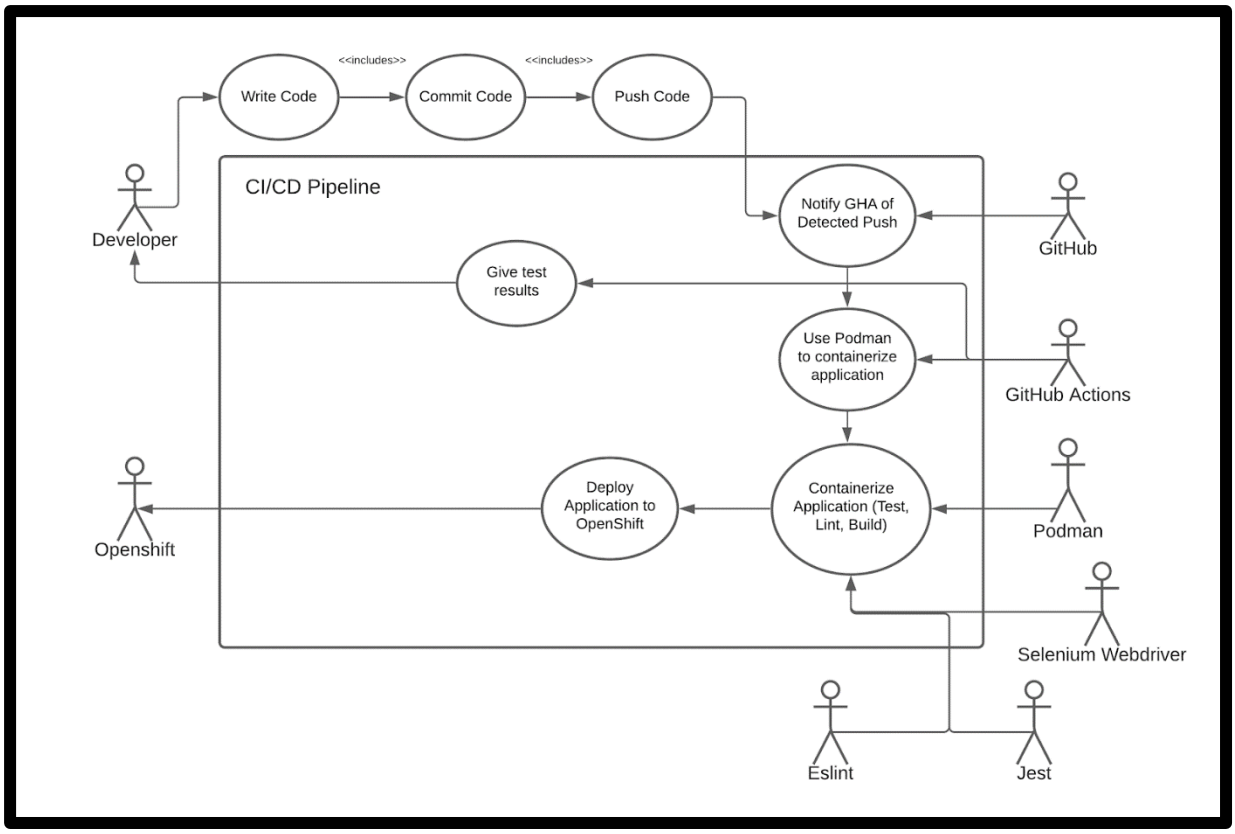
2.2. System Design Models

2.2.1. System Context



Software Design Specification

2.2.2. Use cases (from Requirements)



Name: Write, Commit, Push Code **Participating Actor:** Developer **Entry Condition:** The developer must change the code on their local machine. **Exit Condition:** The developer has successfully written, committed the code into a git repository, triggering the CI/CD pipeline. **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:**	**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:**

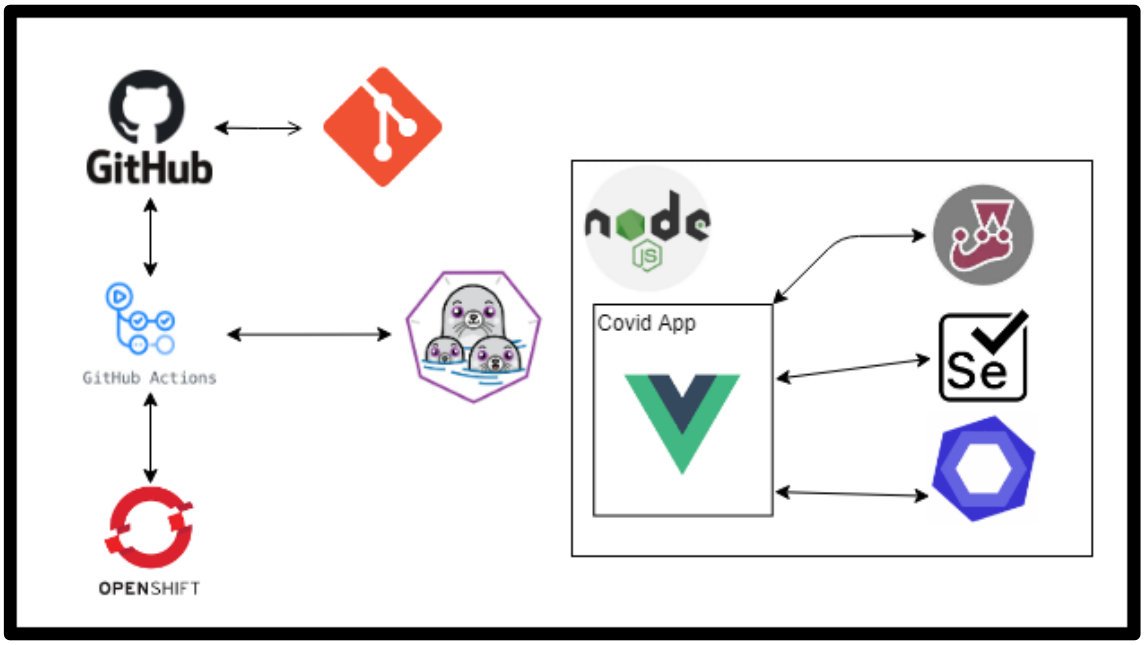
Software Design Specification

• The developer tries to push without a commit beforehand.	• GitHub notifying GitHub Actions when there was no push to the code repository.
Name: Use Podman to Containerize Application **Participating Actor:** GitHub Actions **Entry Condition:** GitHub has successfully notified GitHub Actions of a code push. **Exit Condition:** Podman will start to containerize the application. **Normal Scenario:** 1. Github Actions receives a notification from Github. 2. Github Actions notifies Podman to containerize the application. **Error Scenario:** • Trying to notify Podman without a notification of a code push.	**Name:** Containerize Application (Test, Lint, Build) **Participating Actor:** Podman, Selenium Webdriver, Jest, Eslint. **Entry Condition:** Podman was informed to containerize the application. **Exit Condition:** The application will be containerized, and tasks such as testing, linting, and building will be completed. **Normal Scenario:** 1. Podman containerises the application. 2. Testing is done with Selenium Webdriver and code coverage with Jest. 3. Linting is done with Eslint. 4. The application is built. **Error Scenario:** • Containerizing the application without code coverage from Jest.
Name: Deploy Application to Openshift **Participating Actor:** Openshift **Entry Condition:** The application has been containerized. **Exit Condition:** Openshift will manage the containers. **Normal Scenario:** 1. The application has been fully containerized and finished static unit testing. 2. The container will be handled by Openshift. **Error Scenario:**	**Name:** Give test results **Participating Actor:** Github Actions **Entry Condition:** The application has gone through testing, linting, and building. **Exit Condition:** The developer can see if the application has passed all tests and if the application has been built. **Normal Scenario:** 1. The CI/CD pipeline tests the application, and builds the application. 2. The developer gets the results from the tests and from building the application.

Software Design Specification

• Trying to deploy an application that is not containerized yet.	**Error Scenario:** • Giving test results when no tests and building has occured.

2.2.3. System Architecture



Our system architecture is displayed above and shows how the different components of our project work cohesively as part of a system. We use a git repository on GitHub to store our source code, which is mainly written using Javascript, using Vue.js and Node.js. GitHub Actions recognizes any push to this repository and uses Podman for containerization. After the application is containerized, carry out steps such as building the code, testing the code, and linting it. This is with the help of Selenium Webdriver, Jest, and Eslint.

2.2.3.1 Architectural Decisions

Architectural Decision	Git
Problem Statement	Multiple version control solutions to be chosen from.
Motivations	To enable collaboration of work and track modifications to the files
Alternatives	Subversion
Justification	Distributed, no assets large enough to validate the use of subversion
References	https://git-scm.com/

Software Design Specification

Architectural Decision	Web application (JS, Node.js)
Problem Statement	Multiple applications to be chosen from.
Motivations	To develop a dummy application used in testing the effectiveness of pipelines.
Alternatives	Ruby on Rails, React, Typescript
Justification	Application is not within scope to validate the use of additional frameworks (React/Typescript), could also simulate the agile environment with unplanned changes.
References	https://nodejs.org/en/

Architectural Decision	No database
Problem Statement	Inclusion of database to be chosen from.
Motivations	To decide application structure
Alternatives	Postgres, MySql, MongoDB, GraphQL
Justification	Not within requirements and unnecessarily added complexity, could easily be replaced with Rapid API for demonstration.
References	https://rapidapi.com/marketplace

Software Design Specification

Architectural Decision	Selenium Webdriver
Problem Statement	Multiple testing frameworks to be chosen from.
Motivations	To test code quality
Alternatives	Jest, Puppeteer
Justification	It is extensive (and less limiting), open-sourced, and Neil has extensive experience with it.
References	https://www.selenium.dev/documentation/en/webdriver/

Architectural Decision	GitHub Actions
Problem Statement	Multiple pipelining solutions to be chosen from.
Motivations	To support CI/CD use cases of application development.
Alternatives	Travis, Jenkins, DevOps
Justification	Built-in support by GitHub with documentations covering most of our use cases.
References	https://github.com/features/actions

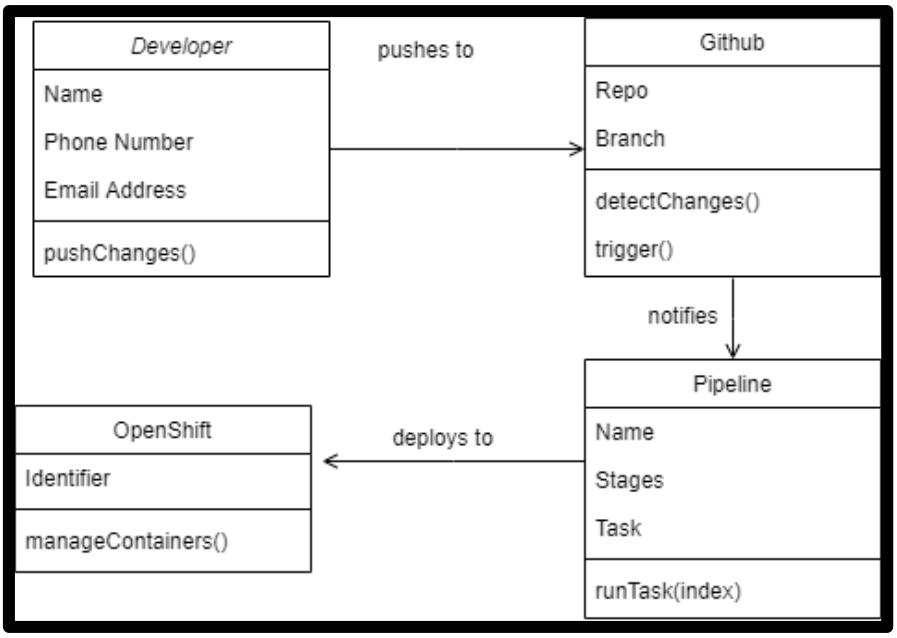
Software Design Specification

Architectural Decision	Vue.js
Problem Statement	Multiple frontend framework to be chosen from.
Motivations	To integrate the frontend framework for easier development.
Alternatives	React, Next.js
Justification	Vue is the middle group from Vanilla JS and React JS, while still providing many benefits and out-of-the-box tooling.
References	https://vuejs.org/

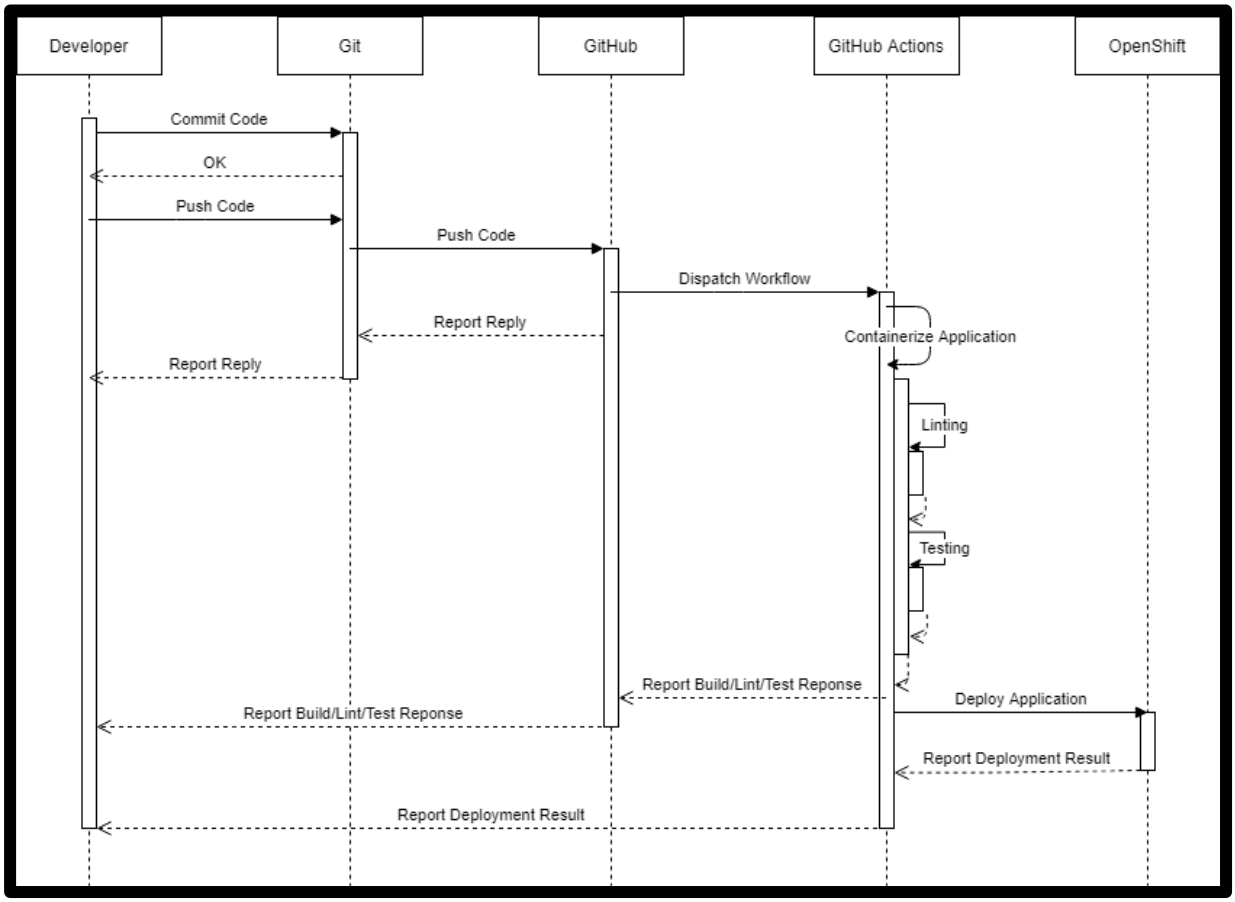
Architectural Decision	Jest
Problem Statement	Webdriver is not suitable for code coverage.
Motivations	Code coverage test.
Alternatives	Cypress, Mocha, Chai
Justification	Jest has the lowest technology overhead and highest capability
References	https://jestjs.io/

Software Design Specification

2.2.4. Class Diagrams

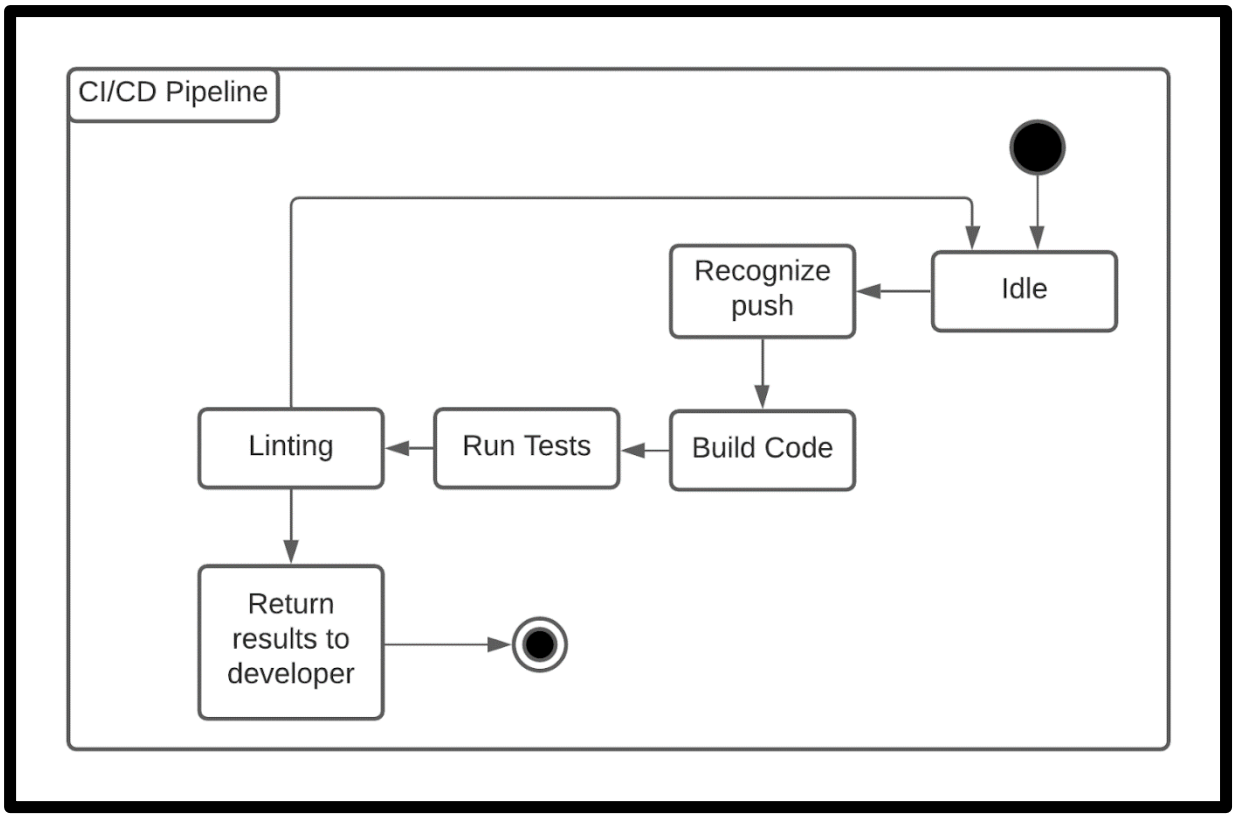


2.2.5. Sequence Diagrams



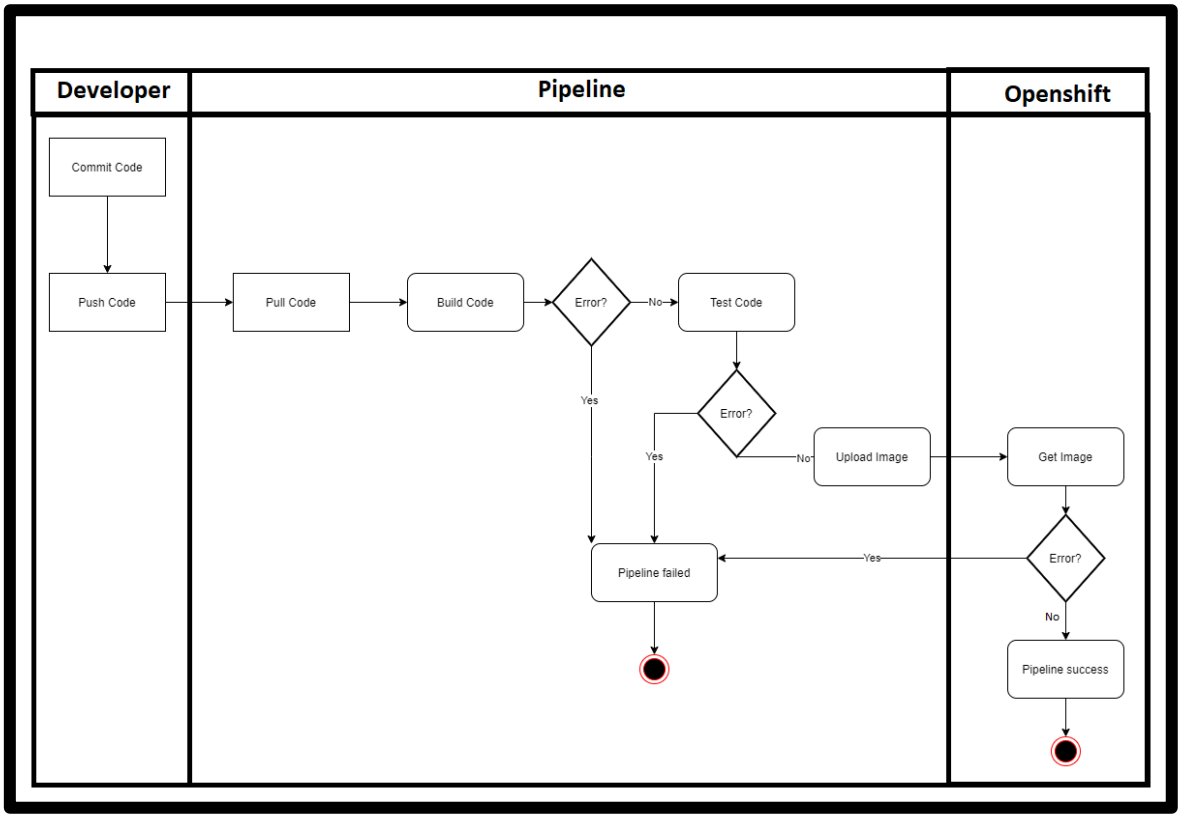
Software Design Specification

2.2.6. State Diagrams



2.2.7. Other relevant models

Dynamic Model



Software Design Specification

References

1. What is DevOps? - Amazon AWS
2. Podman
3. Red Hat OpenShift
4. COVID-19
5. Git
6. GitHub Actions
7. Selenium Webdriver
8. Jest
9. Eslint
10. Rapid API
11. Vue.js
12. Node.js

Other useful resources

- https://learn.openshift.com
- https://www.openshift.com/blog/deploying-to-openshift-using-github-actions
- https://redhatspain.com/openshift-pipelines/
- https://developers.redhat.com/blog/2019/02/21/podman-and-buildah-for-docker-users/
- https://developers.redhat.com/developer-sandbox