Bachelor of Science (Hons) Applied Computing

Heimdall - A Kubernetes Extension

Author:
Bryan Keane

Supervisor:
Lucy White

Contents

List of Figures

List of Tables

1 Plagiarism Declaration

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and severe offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. I have read and understood the Assignment Regulations. I have identified and included the source of all other facts, ideas, opinions, and viewpoints in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source are acknowledged and the source cited are identified in the bibliography. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

2 Acknowledgements

I would like to express my deepest appreciation to my Mentors Ciaran Roche and Laura Fitzgerald who provided invaluable experience and contributions to this project. I would also like to extend my deepest gratitude to my Project Supervisor Lucy white for guiding and encouraging me throughout the process.

3 Introduction

This project, Heimdall, is an open-source Kubernetes Extension built for multi-operator Kubernetes environments. Implementing Heimdall will allow developers to configure it to watch resources of their choosing in a Kubernetes cluster. Heimdall will prevent any unwanted Operators from changing that resource, while simultaneously sending an interactive notification to the developer via a pre-configured Slack Channel regarding the issue.

3.1 Background

In 2022 I completed an 8-month internship at Red Hat for my 3rd-year work placement. I worked on the Red Hat OpenShift API Management (RHOAM) team. RHOAM utilises multiple OpenShift Operators, which are automatically managed and configured applications, to provide a comprehensive API solution to its customers. The API management features provided by RHOAM include: [Red Hat Developer, 2020]

- Limiting the number of API requests based on the customer's quota
- Creating security policies to manage API access
- Monitoring API health
- An API portal for sign-up and documentation

While working on this team, I had the opportunity to contribute to applications which utilize various technologies like Kubernetes, OpenShift, and Go Operators.

3.2 Motivation

Throughout my internship, I ran into a number of technical issues. One such issue centred around RHOAM's rate-limiting service which has become the motivation for my final year project, Heimdall. The issue in question was with the port through which API requests were being sent in order to be rate limited. That port was being overwritten by a conflicting Operator who had a different desired state for that resource. Rate limiting in this context refers to the maximum number of API calls allowed during a specified time period [IBM API-Connect, 2022]

RHOAM uses the Marin3r Operator to provide rate-limiting for RHOAM customers. It works by injecting a rate-limiting container into a Pod. The container then acts as a middleman for API requests by only redirecting requests to the destination container if the API request limit has not been reached. A more detailed explanation of this process will be discussed in Section 5.

It took several days of debugging to figure out what was going wrong and another week passed before my fix was merged. This meant that this problem had been occurring on customer clusters before my fix was rolled out to production. After speaking to developers on various Red Hat teams, it became apparent that this problem was not unique to our team and in fact, was an issue that all Kubernetes and OpenShift product teams face. Fixing this will not only benefit Red Hat teams but any team working on a Kubernetes-based application that utilizes Operators.

3.3 Problem Statement

In Kubernetes (K8s), resources (or objects) have a desired state which describes what the actual state of that resource should look like. The desired state can either lie in a YAML definition or in the code of a Controller. Operators and Controllers can watch that resource's actual state and continuously compare it with the desired state. If they do not match then it is the job of that Operator or Controller to make the necessary changes in order to synchronise them [Operator Pattern, 2022].

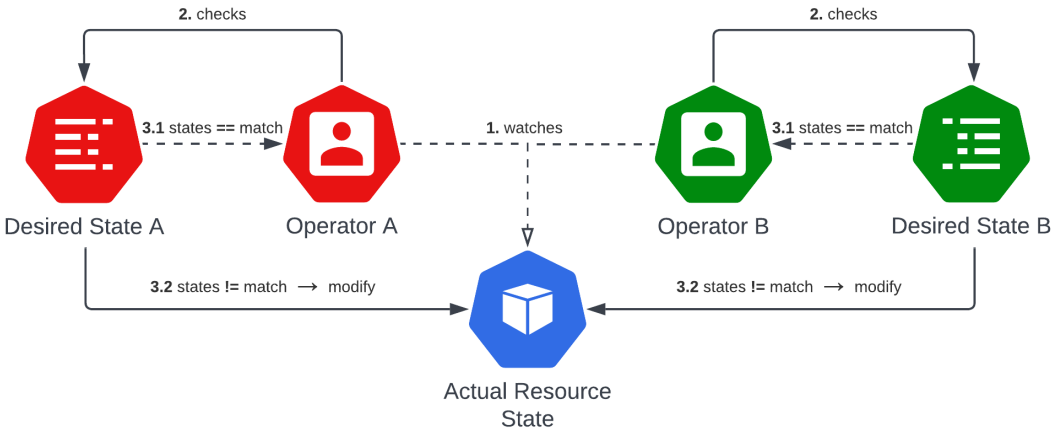

Figure 1: Model of The Problem - Two Operators Fighting Over a Resource's State

This pattern works well until two Operators with conflicting desired states are both set to reconcile a resource. This will cause the resource's state to continuously change as both Operators attempt to synchronise its actual state with their unique desired state. The model shown in Figure 1 shows this problem in action. Each Operator will do the following:

1. Watch a resource's actual state.
2. Check the desired state for that resource.
3. Compare the actual state with the desired state and
    3.1. *If they match* then return to Step 1. and repeat.
    3.2. *If they don't match* then continue to Step 4.
4. Modify the resource's actual state so that it matches the desired state.
5. Repeat this process continuously.

3.3.1 Industry Example

As an example, the open-source e-learning platform Moodle can be used. In an educational environment, one might have a Moodle Operator and a MySQL Operator to serve as Moodle's database. The MySQL Operator manages the storing, of course, student, and module information. This application may then install a Tutors Operator, which is another e-learning platform which houses course notes and lab work. Since there is already a MySQL database via the MySQL Operator, Tutors can use the same database to store course notes and lab work.

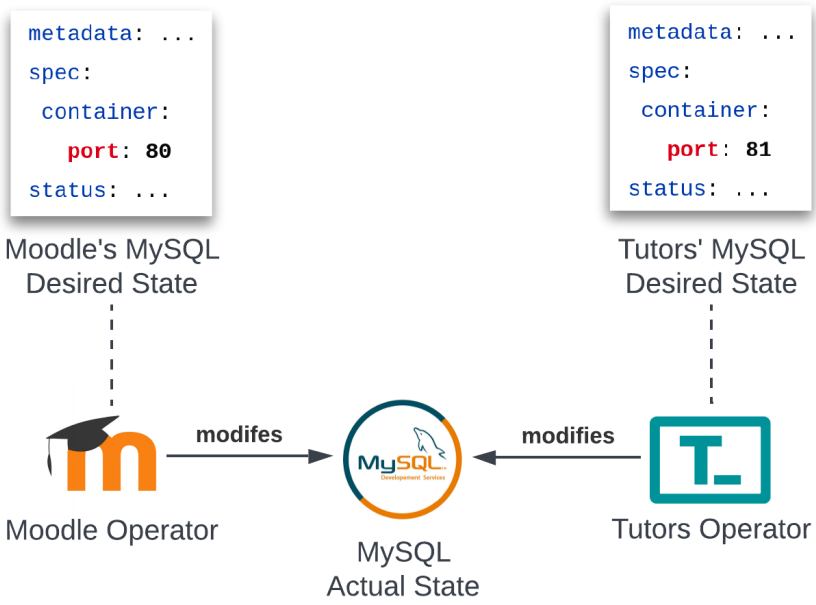

Figure 2: Simplified Model of Moodle, Tutors, and MySQL Industry Example

In this example, it would be very easy to misconfigure one of the Operators to have a dissimilar desired state for the MySQL resource. This would cause both Operators to constantly change that resource. Imagine Moodle wanted the port in which database access occurred to be port 80 and Tutors wanted the database access to occur through port 81. This would cause the MySQL resource's access port to be changed back and forth. If a lecturer or student attempts to access information through Moodle, but at that point in time the MySQL resource's port was configured by the Tutors Operator, the user would not be able to retrieve the data.

3.4 Aims and Objectives

The solution is to create a custom Kubernetes controller which will monitor resource states. A Kubernetes Operator can create a resource, become its owner, and will set the controller to watch that resource for changes. If another operator changes the resource the controller will trigger an alert, notify the developer via a slack integration and allow the developer to fix the problem without the need for time-consuming debugging.

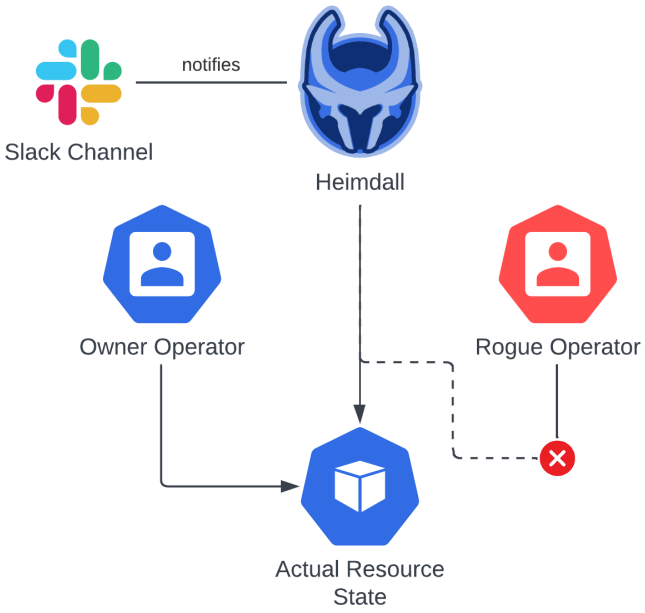

Figure 3: Model of the solution: Heimdall

Figure 3 models the proposed solution where the Owner Operator and Rogue Operator are attempting to change a resource's actual state. Once the Owner Operator is installed, it creates the resource with the addition of a label that Heimdall is looking for. Heimdall sees a new resource has been created with this label and begins monitoring its state. The Rogue Operator is installed and begins changing the resource.

The minimum viable product for Heimdall involves the controller watching for non-owners changing the state of a resource. It will then generate an interactive notification for slack with details on the issue to allow the developers to find and fix the problem with relative ease. The stretch goals for Heimdall will involve allowing for an atomic owner of a resource to be set and blocking of changes from non-owners. This will not only allow the developer to fix the problem with ease but also stop the issue from occurring and prevent any downtime for the end user.

3.5 Risks

TODO(): this is a risky project as it has never been done before

3.6 Contributions

3.7 Outline

4 Methodology

The chosen methodology for software development teams is paramount for the successful planning and development of a product. Historically, the Waterfall Methodology was commonplace - but now Agile is the gold standard. Waterfall involved a distinct sequence of actions for engineers to follow. In short, it involved extensive design and planning before ever writing a line of code. Long documents detailing product design and strategy were written to fit the stakeholder's requirements. This proved to be ineffective as in most cases, holes in the design are discovered after beginning the implementation. In recent years, Agile has begun replacing this framework as it has proven much more efficient for software development teams [M. McCormick, 2012].

4.1 Agile

"Agile is an iterative approach to project management and software development that helps teams deliver value to their customers faster and with fewer headaches" [What is Agile?, 2022]. TODO() DISCUSS AGILE + SCRUM

4.2 Scrum Roles

Scrum has three main roles: Scrum Master, the Product Owner, and the Development Team. These are used to help describe the responsibilities of each stakeholder for a product.

4.3 Scrum Artifacts

Teams who practice Agile and Scrum methodologies often collect Scrum Artifacts. These are pieces of information that a product's stakeholders and the team developing it use to describe its development. The main Scrum Artifacts used for this project include Product Backlog Refinement, Sprint Planning, and Sprint Reviews. There are also various Extended Artifacts that are not included in the official Scrum Artifacts definition. These include reporting mechanisms like Burn down Charts.

4.4 Version Control

What is VCS, Git, GitHub,

4.5 Continuous Integration

Version control lies at the heart of Continuous Integration. CI is an Agile practice of integrating code changes to a product automatically from various contributors (product teams and open-source community contributions). It is a method used to consistently merge code changes into one central repository which runs automated tests and builds to ensure code functionality and integrity.

4.6 Continuous Delivery

Continuous Delivery is an approach

4.7 Testing Approach

4.8 Open Source

5 Technologies

5.1 Kubernetes

6 Tools

Minikube, Kubebuilder, Docker, Podman, Git, GitHub, Jira

7 Design

7.1 System Architecture Overview

7.2 Requirements

7.3 Functional Requirements

7.4 Non Functional Requirements

7.5 Core Requirements and Stretch Goals

7.6 User Stories

As a Developer, I want to be aware of changes to resources that I own. In order to be aware of changes to resources that I own, as a k8s developer, I want to be notified by some channel, such that I don't have to manually watch resources in a cluster. As a Developer, I want to control the cadence of alerts so that I can control noise created from alerts. As a Developer, I want to claim ownership of the resources that I control As a Developer, I want to control the changes to resources that I own.

7.7 Personal Stories

As a student, I want to have achieved First Class Honors, so that I can reach my academic goals As an aspiring Software Engineer, I want to have contributed a solution to a common problem faced by Software Engineers working with Kubernetes, so that I can make a valuable contribution..?? - something like that

7.8 User Definitions

7.9 Models

8 Prototypes

8.1 Proof of Concept

9 Reflection

10 Summary

10.1 Review

10.2 Semester 2 Outline

11 Appendices

References

[What is Agile?, 2022] *Atlassian*,
URL: https://www.atlassian.com/agile

[Operator Pattern, 2022] *Kubernetes*,
URL: https://kubernetes.io/docs/concepts/extend-kubernetes/operator/

[IBM API-Connect, 2022] *Understanding rate limits for APIs and Plans*,
URL: https://www.ibm.com/docs/en/api-connect/10.0.1.x?topic=connect-understanding-rate-limits-apis-plans

[M. McCormick, 2012] *Waterfall vs Agile Methodology*, M. McCormick,
2nd ed. MPCS, Inc., 2012.

[I. Sommerville, 2021] *Engineering Software Products: An Introduction to Modern Software Engineering*, 2021.

[Kubernetes Overview, 2022] *Kubernetes*,
URL: https://kubernetes.io/docs/concepts/overview/

[Red Hat Developer, 2020] *Red Hat OpenShift API Management*, Red Hat, Dec. 16, 2020.
URL: https://developers.redhat.com/products/red-hat-openshift-api-management/overview