(BSc) Applied Computing

Cloud & Networks

Heimdall - A Kubernetes Extension

Author:
Bryan Keane

Supervisor:
Lucy White

Contents

1 Plagiarism Declaration **4**

2 Acknowledgements **5**

3 Introduction **6**
3.1 Background . . . 6
3.2 Motivation . . . 6
3.3 Problem Statement . . . 7
3.4 Industry Example . . . 8
3.5 Aims and Objectives . . . 8
3.6 Risks . . . 9
3.7 Contributions . . . 9
3.8 Outline . . . 9

4 Methodology **9**
4.1 Agile . . . 10
4.2 Scrum Roles . . . 10
4.3 Scrum Artifacts . . . 10
4.4 Version Control . . . 10
4.5 Continuous Integration . . . 10
4.6 Continuous Delivery . . . 10
4.7 Testing Approach . . . 10
4.8 Open Source . . . 10

5 Technologies **10**
5.1 Kubernetes . . . 10

6 Tools **10**

7 Design **11**
7.1 System Architecture Overview . . . 11
7.2 Requirements . . . 11
7.3 Functional Requirements . . . 11
7.4 Non Functional Requirements . . . 11
7.5 Core Requirements and Stretch Goals . . . 11
7.6 User Stories . . . 11
7.7 Personal Stories . . . 11
7.8 User Definitions . . . 11
7.9 Models . . . 11

8 Prototypes **11**
8.1 Proof of Concept . . . 11

9 Reflection **11**

10 Summary **11**
10.1 Review . . . 11
10.2 Semester 2 Outline . . . 11

11 Appendices **11**

List of Figures

1 Model of Two Operators Watching a Single Resource's State 7
2 Model of Moodle, Tutors, and MySQL Industry Example 8
3 Model of the solution: Heimdall 9

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

Thank people... TODO

3 Introduction

Heimdall is an open-source Kubernetes tool which facilitates the stress-free development of containerized applications utilizing multiple Kubernetes Operators. Implementing this solution will allow developers to configure the custom controller to watch resources of their choice and connect the controller with their team's slack channel to receive interactive notifications when the resource is being updated by the incorrect operator. Users can go as far as setting an atomic owner of resources and blocking other operators from making changes.

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 (automatically managed and configured applications) to provide a comprehensive API solution to its customers. The API management features provided include

- Limiting the number of API requests based on the customer's purchased 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

(*Going to use rate-limit example or pod replica count example*) I encountered various problems during my internship. One of which was with RHOAM's rate-limiting service and is the motivation for Heimdall. Rate-limiting is done with the Marin3r Operator for OpenShift clusters (Red Hat's container orchestration platform). It works by injecting a rate-limiting container into the Pod being rate-limited. The container works by acting as a middleman for requests. It takes in requests and sends them to the destination container if the rate of requests has not reached the limit. The bug I found was that the port sends requests through the rate-limit container was being overwritten by a rogue Operator and set back by the owner Operator. This meant that customers who had RHOAM installed were not being consistently rate-limited. As a result, an influx of requests could have overloaded Pod CPU and Memory, causing needless stress to the cluster. It took multiple days of debugging to figure out what was going wrong and another week passed before my code changes were merged. This meant that this problem had been occurring on customer clusters for an unknown amount of time before the fix reached customers. After speaking to developers on various Red Hat teams, it became apparent that this problem was not unique to our team and is 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.

(*OLD PROBLEM STATEMENT - SAVING FOR REFERENCE*) Kubernetes natively supports multiple operators to watch the state of a single resource. This can cause problems like the one described in Figure 1. This can cause issues with object states and is the foundation of this project. During my Internship, I was working on a ticket which involved implementing pod count auto-scaling where the number of pods scales up based on the load each pod is experiencing. During this work, I observed some peculiar behaviour. When I forced the pod's load to increase the number of pods increased as expected, but shortly after the pod count returned to its original number. After observing for some time, it became apparent that something was scaling the pods up and down continuously. I discovered it was two OpenShift Operators fighting over the state of the pod. Each operator had the desired shape for the pod's deployment, where one operator wanted three replicas of the pod and the other operator wanted two replicas so they underwent a sort of tug-of-war, fighting over how many pods should be running. This is a common problem where two operators can watch the same resource and have conflicting resource definitions, so the operators' controllers fight to keep the pod in their preferred state. It wastes resources and can cause interruptions for the end user.

3.3 Problem Statement

In Kubernetes (K8s), custom resources (or objects) have a state in which they should remain which are stored in YAML files called custom resource definitions. When an Operator is watching a resource's state, it is continuously comparing the state defined in that resource's YAML definition with its actual state. If these don't match, it is the Operator's job to make the necessary changes in order for the desired state and actual state to match [Operator Pattern, 2022].

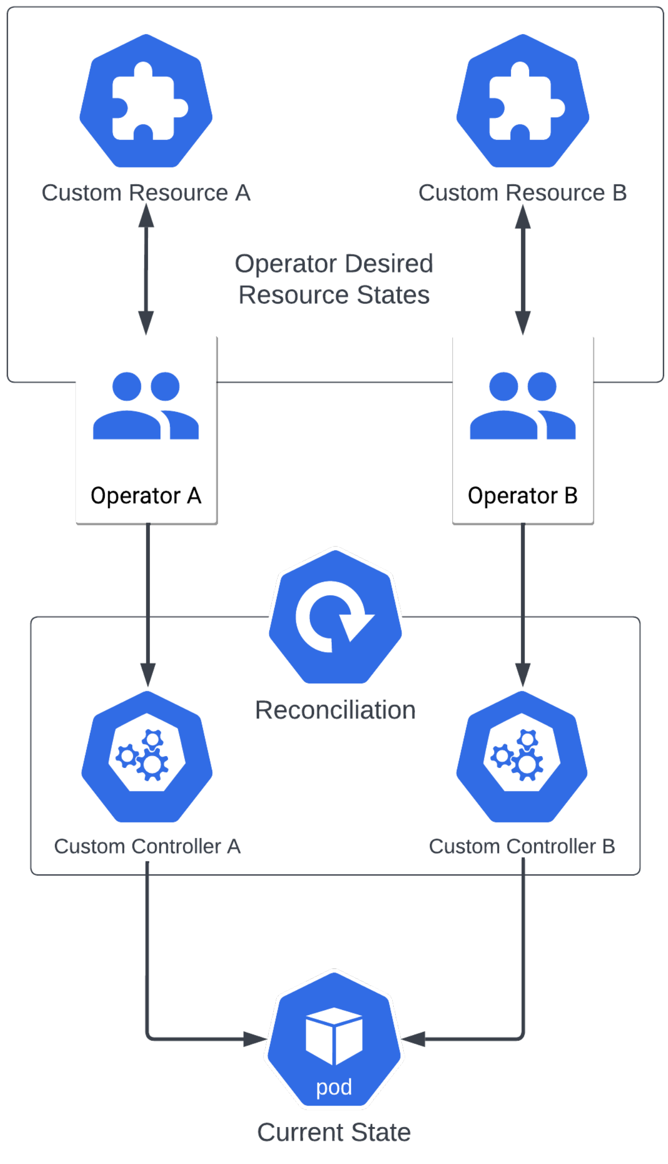

Figure 1: Model of Two Operators Watching a Single Resource's State

This pattern works well until two Operators with conflicting YAML definitions are both set to watch 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 watches their Custom Resource desired state and instructs their controller to make the necessary changes to ensure the actual resource state matches.

3.4 Industry Example

As an example, the open-source e-learning platform Moodle will be used. In an education environment, one might have a Moodle Operator and a MySQL Operator as that is the back-end database used by the platform. 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.

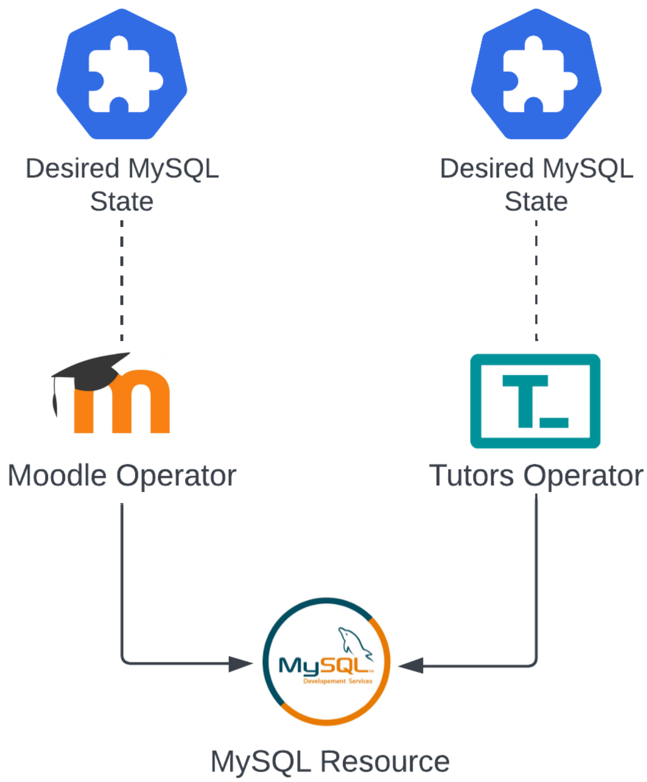

Figure 2: Model of Moodle, Tutors, and MySQL Industry Example

In this example, it would be very easy to misconfigure one of the Operators to have a diverging 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 8000 and Tutors wanted the database access to occur through port 8001. This would cause the MySQL resource's access port to be changed back and forth. If a lecturer or student was attempting 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.

3.5 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 the resource is changed by another operator the controller will fire 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 models the proposed solution where Operator A represents the owner operator of the Monitored Resource, and Operator B represents the rogue operator which is causing the resource to change from the owner's desired state. Once Operator A is installed, it creates the resource and instructs Heimdall to monitor the resource's state. Once Operator B is installed and changes the resource. Heimdall sees this occurring and will send the developers a notification through slack which will remove the need for debugging and allow the developer to get a fix pushed as soon as possible. The stretch goal for this product would be to completely block Operator B from changing the resource so the developers have an interim solution to the bug while they push a fix to the relevant operator, preventing any customer downtime.

3.6 Risks

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

3.7 Contributions

3.8 Outline

4 Methodology

The chosen methodology for software development teams is paramount for the successful planning and development of a product. Currently, the most common methodology used is Agile, but before this, the Waterfall Methodology was the gold standard. This method of development involves 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].

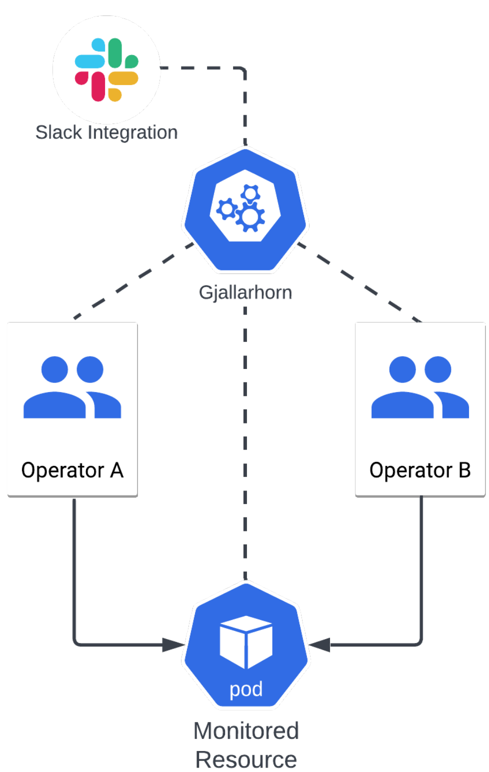

Figure 3: Model of the solution: Heimdall

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] (accessed Oct. 24, 2022). *Atlassian*,
URL: https://www.atlassian.com/agile

[Operator Pattern, 2022] (accessed Nov. 03, 2022). *Kubernetes*,
URL: https://kubernetes.io/docs/concepts/extend-kubernetes/operator/

[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/