Waterford Institute *of* Technology

Project Feasibility Report

Bachelor of Science (Hons) Applied Computing

RubiX - Functions as a Container

Ciaran Roche - 20037160

Supervisor: Dr. Rosanne Birney
Second Reader: Joe Daly

April 16, 2019

Plagiarism Declaration

Unless otherwise stated all the work contained in this report is my own. I also state that I have not submitted this work for any other course of study leading to an academic award.

Acknowledgements

A very special gratitude goes to my supervisor Dr. Rosanne Birney for her advice and guidance through out this semester. Without it my project would not be where it is today. I also wish to extend my gratitude to the entire computing staff, who gave me the drive and motivation needed to push myself beyond my abilities.

My sincere thanks to Dr. Leigh Griffin, Aiden Keating, Laura Fitzgerald and the entire team at Red Hat Waterford. Their guidance and support throughout my internship and extending to my project has been incredible. Giving me not only the knowledge and tools needed for my final year of college but to progress my career in this field.

To all my family, without their encouragement and support I would not be writing this today. For that I will be forever grateful.

And finally, last but by no means least my fellow students. Their collaboration and friendship has made my time at WIT a memorable one. I wish you all the best in your FYP's and your future careers.

Contents

List of Figures

List of Tables

Glossary

API Application Programming Interfaces, a set of functions and procedures that allow the creation of applications which access the features or data of an operating system, application, or other service. vii, 13, 30

AWS Amazon Web Services, provides on-demand cloud computing platforms to individuals, companies and governments, on a paid subscription basis. 2

AWS Lambda Amazon Web Services Lambda, a service that lets you run code without provisioning or managing servers. 1

BaaS Backend as a Service, a cloud computing service model that serves as the middleware that provides developers with ways to connect their web and mobile applications to cloud services via API's and SDK's. 1

BSD-style license Berkeley Software Distribution style license, Is a family of permissive free software licenses, imposing minimal restrictions on the use and distribution of covered software. 12

Buildah A tool that facilitate building Open Container Initiative container images. 13

CLI Command Line Interface, is a means of interacting with a computer program where the user issues commands to the program in the form of successive lines of text. 4, 11–13, 30, 36

Cloud Foundry Foundation A nonprofit that oversees an open source platform and is a collaborative project of the Linux Foundation. 1

Container An isolated piece of software bundled with everything it needs to run on the host. 4, 13, 38

CRI Container Runtime Interface, at the lowest layers of a Kubernetes node is the software that starts and stops containers. vii

Cri-O Is an implementation of the Kubernetes CRI to enable using OCI compatible runtimes. It is a lightweight alternative to using Docker as the runtime for kubernetes. 13

Docker A piece of software which enables orchestration and management of containers on a host. 4, 12, 13, 24, 30

FaaC Functions as a Container, a term coined for this project as best describes the end result. Being a function that is a container allowing a user utilize Frameworks such as Knative. 3

FaaS Functions as a Service, a service is a category of cloud computing services that provides a platform allowing customers to develop, run, and manage application functionalities. Without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app. 1

GitHub Is a web-based hosting service for version control using Git. 14

GoLang Is an open source programming language designed by Google. Go is a statically typed, compiled language in the tradition of C, with the added benefits of memory safety, garbage collection, structural typing, and CSP-style concurrency. 11–15

JIRA Jira is a proprietary issue tracking product developed by Atlassian which allows bug tracking and agile project management.. 4

Kata Is a new open source project building extremely lightweight virtual machines that seamlessly plugs into the containers ecosystem. 13

Knative A set of middleware components that are essential to build modern, source-centric, and container-based applications that can run anywhere – on premises, in the cloud, or even in a third-party data center. vii, 1–3, 13

Kubernetes An open-source system for automating deployment, scaling, and management of containerised applications. 2, 4, 12, 24

OCI Open Container Initiative, Is a lightweight, open governance structure (project), formed under the auspices of the Linux Foundation, for the express purpose of creating open industry standards around container formats and runtime. vii, 13

Open-Source Is a type of computer software whose source code is released under a license in which the copyright holder grants users the rights to study, change, and distribute the software to anyone and for any purpose. 3, 11, 12

Quay.io is a cloud hosted service for building, storing and distributing Docker containers. 38

REST Representational State Transfer, a software architectural style that defines a set of constraints to be used for creating web services. Web services that conform to the REST architectural style, termed RESTful web services, provide interoperability between computer systems on the Internet. 2

S3 Amazon Simple Storage Service, provides object storage through a web service interface. 2

SDK Software Development Kit, Is a set of programs used by a computer programmer to write application programs.. vii, 4, 13, 14, 29

Serverless Applications that significantly depend on third-party services (knows as Backend as a Service or "BaaS") or on custom code that's run in ephemeral containers (Function as a Service or "FaaS"). 1–4, 14, 39

1 Introduction

1.1 Motivation

As seen in Figure 1, serverless computing is the number one growing cloud service (*RightScale 2018 State of the Cloud Report*, 2018). As with many trends in software, there is no one clear view of what serverless is. It was first used to describe applications that fully incorporate third-party, cloud-hosted applications and services. Another term often used to describe these type of architectures is Backend as a Service or BaaS. Today serverless is the term used to describe applications where the server-side logic is written by the applications developer but unlike traditional architectures it is event-triggered, meaning that a server is not constantly running but instead events trigger the compute containers (Fowler, 2018). This is often referred to as Functions as a Service or FaaS. A popular implementation of this is AWS Lambda. For the purpose of this report. I will be using the (Fowler, 2018) definition of serverless

Place	Service	Growth Rate	2017 Use	2018 Use
#1	Serverless	**75%**	12%	21%
#2	Container-as-a-service	**36%**	14%	19%
#3	DBaaS SQL	**26%**	35%	44%
#4	DBaaS NoSQL	**22%**	23%	28%
#5	DRaaS	**21%**	14%	17%

Figure 1: *Top Growing Cloud Services* – RightScale 2018 State of the Cloud Report *(2018)*

One of the main benefits and motivation for the project is that it reduces operational cost, as one only pays for the compute power they need as opposed to a constant running server instance. Another aspect of the operational costs stem from the development as developers can focus on business logic of their application over building the infrastructure around their application (Fowler, 2018).

The Cloud Foundry Foundation[1] conducted a global survey of their users and found that 22% are already using serverless technology (*Where PaaS, Containers and Serverless Stand in a Multi-Platform World*, 2018). And with almost half their users evaluating the technology. This kind of trend is supported by research conducted by Cloudability (*State of Cloud 2018 Report*, 2018). This suggests that the area of serverless computing is growing rapidly.

Further backing the growth of serverless, Google announced the launch of their Knative[2] project

[1] https://www.cloudfoundry.org – a multi-cloud application platform.
[2] https://cloud.google.com/knative/ – middleware components for container-based applications.

which is based on the Kubernetes[3] engine and developed by Google, Pivotal, IBM, Red Hat and SAP. Knative provides a suite of developer-focused middleware tools for building, deploying and managing serverless applications on the Kubernetes engine (Bryant, 2018).

The adoption rate of serverless technologies, mixed with a personal interest in the migration of conventional monolithic applications to microservice serverless applications, led me to explore problems around serverless that could be solved as a topic for my Final Year Project.

1.2 Problem Statement

Notwithstanding the many benefits outlined in Section 1.1, there are a number of identified problems with serverless technologies. A report by DigitalOcean highlighted that the biggest challenge faced by serverless was the ability to monitor and debug (*Currents: A quarterly report on developer trends in the cloud*, 2018). This was shown to be down to the layers of abstraction by particular vendors and their serverless services.
While Knative looks after the orchestration of a serverless application within Kubernetes seamlessly, thus reducing the number of layers of abstraction, it is up to the developer to build, configure and package their functions in a container. This pattern goes against one of the main benefits of serverless in that a developer can focus their time on the business logic over the configuration of the infrastructure thus reducing operational costs.
This project will look to solve both problems outlined above and will be discussed in detail in Section 1.4.

1.3 Industry Example

As discussed in Section 1 serverless adoption and usage is rising. It can be leveraged to fit almost any use case, but to put it in practical terms an example is provided here. A typical use case for a serverless function, which can be seen in Figure 2, is a function which is triggered when an image is uploaded to a web application. The web application would consist of a regular RESTful service. When an image is uploaded to the RESTful service, the function would back up the content to a persistent storage similar to S3 in AWS. This is an easy and very economical use case as a user would never have to worry about infrastructure provisioning or about scaling server instances to accommodate load. The function would not be consuming resources within the RESTful web service and would only be in involked when needed.

[3]https://kubernetes.io – container-orchestration system for automating deployment.

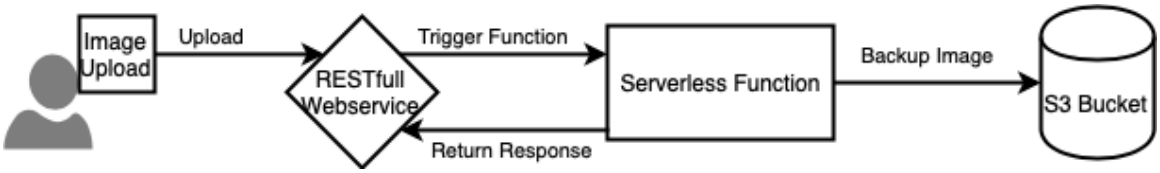

Figure 2: *Serverless Example*

1.4 Aims and Objectives

With the aim to solve the problems outlined in Section 1.2 the overall goal is to provide an open-source tool to allow software engineers to easily create FaaC to be utilised by Knative.

AO1 - Provide support for building FaaC in multiple languages. This will reduce the cost of entry, as developers can use a language that best fits their use case or expertise.

AO2 - The ability to easily debug functions. Solving the main identified problem with serverless, by providing the means for easy debugging will help increase adoption rate.

AO3 - Confidence in a Function as a Container being as secure as possible. By removing layers of abstraction the developers can have confidence in their containers being up to date and as secure as possible.

AO4 - The means to test Functions as a Container locally before deployment. Alleviating a barrier to entry by providing a simple method to test functions locally without the need for dedicated development environments.

AO5 - The project to be transparent and developed in the open. This will allow for the community to mold and adapt the project to best fit their needs, overall increasing the quality and the uptake of the project.

AO6 - The means to control the life cycle of a Function as a Container. This will increase the speed of local development by providing the developer with the tools needed to fit their development needs, thus reducing the need to switch between tools.

The project will utilise common patterns that would be native to a software engineer already working in the space of Containers and Kubernetes. It will consist of the following:

- **CLI:** An easily extensible command line interface following the same work flow such as tools like Docker CLI and Kubernetes Kubectl.
- **SDK's:** A suite of SDK's to support the creation of Functions as a Container in multiple programming paradigms.
- **Showcase Application:** To demonstrate a modern serverless application running in the Kubernetes engine.

1.5 Contributions

The end result of this project will be a Framework which provides a suite of SDK's and a CLI that will reduce the barrier of entry to serverless development. The project will be developed in the open with the goal to build a community that will allow the development be guided by the needs of the serverless community as a whole. On the back of the project a number of applications to talk at conferences will be submitted, with a local tech talk already confirmed for Match 2019.

1.6 Outline

This report is broken into seven sections. This section, the first, is an overall introduction covering motives and the problem to be solved along with contributions that I intend to make. In section two the methodologies this project will follow are explored and discussed. The third section looks to examine the technological choices. Section four outlines the tools used to aid development of the project. The system architecture, requirements and user stories, make up the design of the project, which is captured in Section five. Section six discusses the prototype including a demonstration. Finally section seven reviews all completed work to date and looks toward work to be completed.

This project is sponsored by Red Hat, allowing access to tools such as JIRA along with acting as stakeholders in the project. As stakeholders Red Hat fulfill the role of product owner ensuring the development of this product meets the requirements set to solve the problem stated in Section 1.2. This is not a paid or monetary sponsorship.

This project is supervised by Dr. Rosanne Birney.

2 Methodology

An important decision that was made early on in the project was deciding on what methodology to follow, as this decision would have an impact on the project's timelines, deliverables and product quality. The two that were considered were Agile and the Waterfall approach. Both have notable strengths and weaknesses which were taken into account, and Agile was deemed the better fit for this project. Agile has several different flavours and I have opted to follow Scrum because all decisions made within the Scrum methodology are based on what is known, this will be discussed in Section 2.1. Scrum is also the most popular variant of Agile (*State of Agile Report*, 2018) and is well understood by the community.

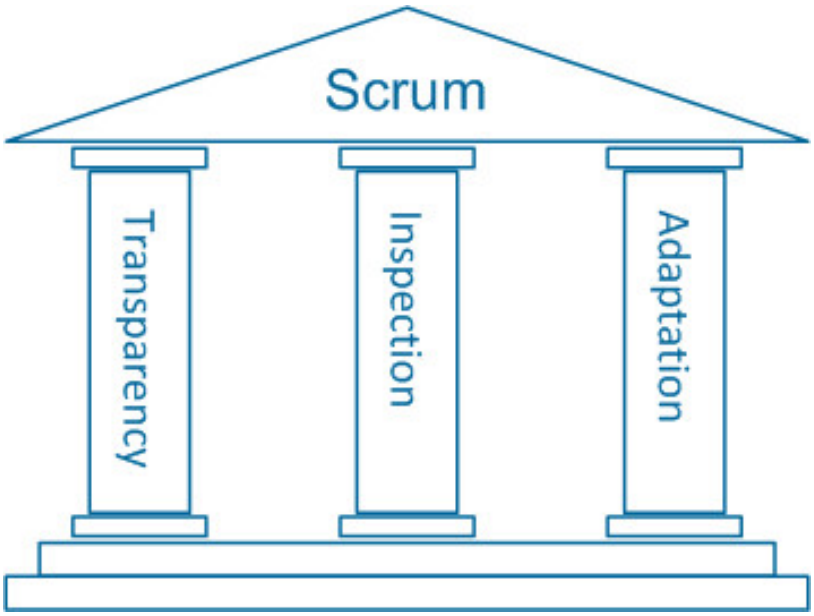

Figure 3: *3 Pillars of Scrum* The Scrum Guide™ *(2018)*

2.1 Agile

The Agile methodology comes in many flavors and frameworks (*Manifesto for Agile Software Development*, n.d.), and those who follow an Agile methodology do so by adapting it to best suit their needs. The same approach is taken for this project in that it follows closely to the Scrum framework. The Scrum Guide defines Scrum as a "framework for developing, delivering and sustaining complex products" (Schwaber and Sutherland, 2017). The Scrum framework suited this project due to it being based on empiricism, which asserts that decisions are based on what is known. It employs this decision flow through an incremental approach to optimise predictability and to control risk. The

Scrum framework can be broken into three pillars: transparency, inspection and adaptation.

2.1.1 Transparency

All aspects of a process must be visible to all who are involved and responsible for the outcome. Transparency requires these aspects to be defined so that any observers can share an understanding. I will achieve this by my Trello Cards and these reports.

2.1.2 Inspection

Scrum employs frequent inspections of Scrum artifacts, which are outlined in Section 2.3, and the overall progress toward a Sprint Goal to detect and identify and undesirable variances. These inspections should not be so frequent that they take away from the work but that they increase the standard and quality of the deliverables from each Sprint. The Scrum Artifacts, Section 2.3, will allow me achieve this.

2.1.3 Adaptation

During these inspections if it is determined that one or more aspects have deviated from the acceptable limits then the process must be adjusted. All adjustments must be made as soon as possible to minimize any further deviation. This will be achieved through the Scrum Artifacts.

2.1.4 Acceptance Criteria

For every story in the project, Acceptance Criteria will be set. It is this criteria that will ensure requirements are met per story and will be used by myself as a Developer and my ScrumMaster to ensure work is complete.

2.1.5 Definition of Done

A critical aspect to Scrum is the Definition of Done. This will be a list of activities that will be used to verify that the criteria is met and the features are truly complete.
As a lone developer in a time sensitive project, being able to definitively know when I am done with a task and can move on is essential to ensure the project meets its deadline and a product is delivered

2.2 Scrum Roles

Throughout this project a number of people will take on a number of different roles to fit with the Scrum Methodology. Some roles have been adapted to suit this project and are outlined below.

- **Scrum Master** - Dr. Rosanne Birney. Project supervisor Rosanne will also take the role as Scrum Master. As Scrum Master to this project Rosanne will facilitate weekly stand ups to ensure work is progressing with minimal impediments. This ensures that the project concludes with a high value product.
- **Product Owners** - Dr. Leigh Griffin (Red Hat) and Laura Fitzgerald (Red Hat). Taking an adapted approach to Product Owners it is their job to guide the overall scrum team through the scrum process. In comparison to the traditional Product Owner taking an authoritative role, this authority falls to myself as Developer to this project where I retain general say.
- **Developer** - Ciaran Roche. Following the traditional approach to a developer within a Scrum team. It is my job to do the work of delivering a potentially releasable product at the end of each Sprint. To that end all traditional team roles, such as front end, backend, UX will fall to me.
- **Stakeholder** - Aiden Keating (Red Hat). It is the job of the stakeholder to advise decisions based on their knowledge of the technologies used. As Aiden possesses several years working within the Kubernetes ecosystem along with many open source projects, his insight and advise will be critical during the early development days prior to community involvement.

2.3 Scrum Artifacts

Figure 4 shows artifacts as outlined in the Scrum Guide (Schwaber and Sutherland, 2017) which allow for inspection and adaptation of aspects of this project and how they are incorporated around a Sprint.

- **Sprint Planning** - There will be planning sessions to evaluate the project scope and refine the backlog to best suit the needs of the project. These meetings will involve the Product Owners. This meeting will define a goal for the next Sprint and will look at taking large development tasks and break them into smaller tasks and fit them to meet the overall goal for the coming Sprint.
- **Sprint** - The sprint is a time boxed unit where development work is carried out on the tasks assigned during the Sprint Planning artifact. These sprints will be relatively short to suit with the college semester and to allow for highly achievable goals and giving time to allow for any change. The plan is to settle on two week sprints to allow enough time for a shippable increment.

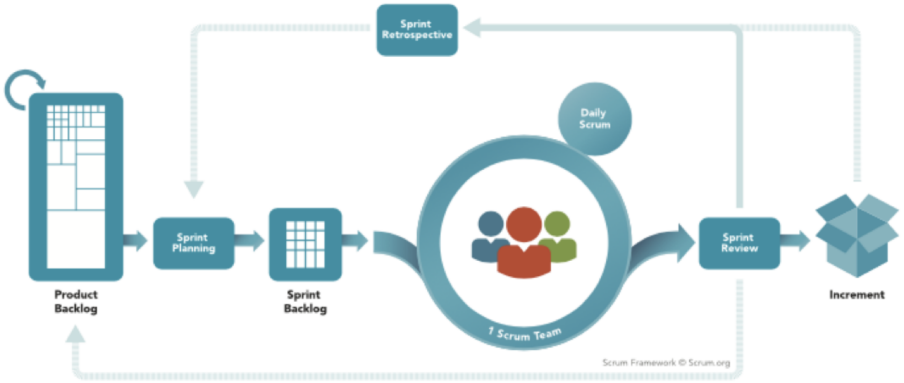

Figure 4: *Scrum Process Outline* The Scrum Guide™ *(2018)*

- **Sprint Review** - Once the sprint is over a review meeting will take place and the work toward the sprint goal will be evaluated. All work completed will be shown in a demo to the product owners. This will help gauge progress of the project and help with defining the next sprint during the following planning meeting. It will also allow us review the priority of the backlog.
- **Sprint Retrospective** - This meeting takes place following the review meeting, where it is looked at for areas that worked well and more so areas that did not work during the previous sprint. This allows scope for improvements in sprint performance, the outcome will be taken into consideration during the following planning meeting. This rapid feedback and course correction will ensure the success of the project.

2.4 Continuous Integration

Continuous Integration is the process where work is integrated frequently, usually on a daily basis by multiple people which leads to multiple integrations per day (Fowler, 2006). While this project will be developed by a single person it will have a continuous build cycle, this ensures a high quality overall, thus reducing the number of bugs. To achieve this, integration pipelines will be built for each repository associated with this project. The tools to be used will be outlined in more detail in the following sections.

Figure 5 shows the outline of these pipelines. When work is committed to GitHub through tool integrations with GitHub a number of checks will be triggered. Circle CI, which is discussed in Section 4.2.1 will be responsible for building the repository and carry out all unit tests in an isolated environment. Coveralls, which is discussed in Section 4.2.3, will check code base for test coverage and Sonarqube, which is discussed in Section 4.2.2 will analyse and advise of improvements to the code based on best-practices around technical debt

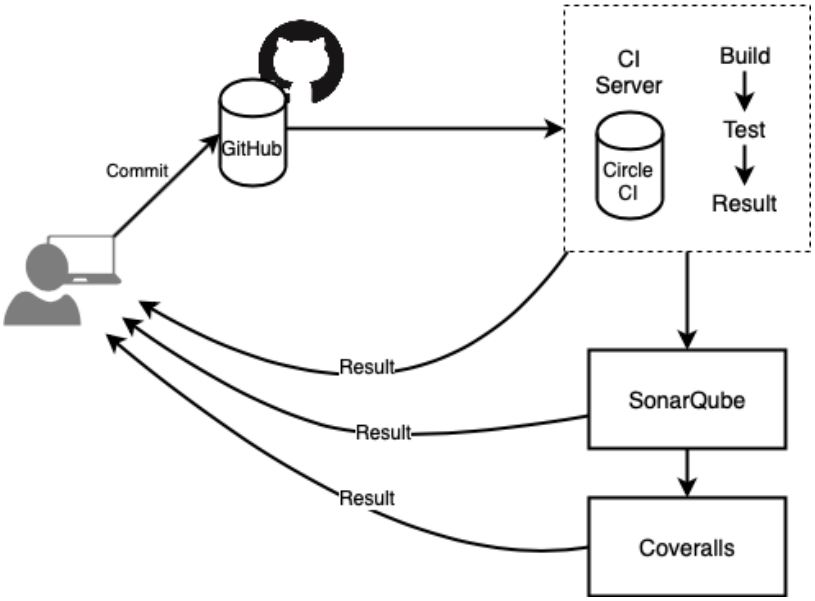

Figure 5: *Continuous Integration Workflow*

2.5 Continuous Integration/Continuous Delivery

Due to short development sprints, producing a shippable increment every two weeks. Continuous Delivery will be built into the Continuous Integration allowing for the automation of releases and bug fixes.

2.6 Testing Approach

Time was allocated to choose the testing approach for this project. The choice of testing would have significant effects on the project, from the cost of development to the overall quality of the project. Consideration was given to two methodologies, Test Driven Development and Behavior Driven Development (BDD). The latter was chosen for the approach to be followed for this project as Behavior Driven Development is the combination of Test Driven Development, domain-driven design and object-oriented analysis.
BDD is a set of practices that aim to reduce common wasteful activities in software development, such as rework needed due to misunderstood or vague requirements, or technical debt due to reluctance to refactor code. It achieves this by using a natural language to express desired behaviors and expected outcomes, based on these constructs test scripts are written and implemented into a CI/CD workflow (Solis and Wang, 2011). The result is a closer relationship to the acceptance criteria set at the user story level, thus BDD aligns closely with the chosen Agile Methodology which, was discussed in

Section 2.1.
Upon completion of the POC discussed in Section 6, constructs will be written outlining all expected and desired behaviors. These constructs will be used throughout the project to drive the development and tests to be written.

2.7 Open Source

A decision was made to ensure this project follows the Open Standards Requirement for Software set by the Open Source Initiative. In order to comply with the requirement and ensure that the project is legally Open Source, a number of criteria need to be met (*Open Standards Requirement for Software*, 2018).

- **No Intentional Secrets** - The project must not withhold any detail that is necessary for interoperable implementation. No versions of the project can be released if in violation to the OSR
- **Availability** - The project must be freely and publicly available under royalty-free terms.
- **Patents** - All patents associated with the project must be licensed under royalty-free terms for unrestricted use or be covered by a promise of non-assertion when practiced by open source software.
- **No Agreements** - There must not be any requirement for execution of a license agreement.
- **No OSR-Incompatible Dependencies** - Implementation of the standard must not require any other technology that fails to meet the criteria of this requirement.

A number of User Stories were defined around complying with the standard and creating the foundation for an Open Source community. These can be seen in Section 5.14.

3 Technologies

In order to solve the problems outlined in Section 1.2, the correct technologies needed to be chosen. To ensure a correct fit a number of investigations were undertaken. This fits in sync with the Agile methodology, Section 2.1, in that through early and often prototyping and investigation, the correct technology is chosen. The choices made resulted in a prototype application which is discussed in detail in Section 6. This section will be broken into the following sub sections, each outlining different aspects of the project and the technologies chosen which are to be implemented in the prototype:

- Command Line Interface
- Container Build Tools
- Software Development Kit

3.1 Command Line Interface

3.1.1 Investigation

Acceptance criteria, Section 2.1.4, was set in order to ensure the correct technology was chosen for the CLI.

- Must be easily extensible.
- Must be modular in design.
- Must appeal to developers.
- Must follow patterns users are familiar with.
- Must have minimal footprint.
- Must meet open-source criteria.

Based on the criteria, it was decided to go with GoLang and the reasoning will be discussed in detail in Section 3.1.2. Other technologies looked at include, NodeJs and Python but were dismissed because, while satisfying some of the acceptance criteria I felt GoLang would increase overall uptake of the project within the Open Source Community.

3.1.2 GoLang

GoLang[4] is an open source project developed by Google and distributed under a BSD-style license. GoLang was designed to be a modern computer programming paradigms to enable higher productivity while taking aspects from a number of different languages. GoLang is a specifically engineered language unlike others which have evolved in that the designers based characteristics of the language on what they considered to be the best elements of other modern languages. For example it is statically typed and efficient in the style of C++ and Java. It is productive and intuitive similar to that of Python or JavaScript.

Some of the main benefits include garbage collection and native concurrency while its novel type system allows for flexible and modular program construction (*GoLang*, 2018). The Stack Overflow Survey 2018 (*Stack Overflow Developer Survey 2018*, 2018) continues to show signs in GoLang's rise in popularity between developers which can be seen in Figure 6. This rise could be attributed to, tools like Docker and Kubernetes being built in GoLang. As these tools are open-source, willing contributions to them are made in GoLang. Utilizing the same language and libraries used to build these tools in the development of the prototype satisfies a number of the criteria set out in Section 3.1.1. GoLang appeals to developers, follows patterns users are familiar with and has a minimal footprint while meeting the open-source criteria.

Figure 6: *Stack Overflow Most Loved Languages* – Stack Overflow Developer Survey 2018 *(2018)*

GoLang features a plugin system. This allows developers build loosely coupled modular programs using packages that are compiled as shared object libraries. These libraries are then loaded and bound together dynamically at runtime (Vivien, 2017). By using GoLang's plugin system it will allow for greater extensibility in that each supported language by the tool will have its own library within the CLI maintaining a more modular structure to the overall design.

[4]https://golang.org – Google engineered programming language

3.2 Container Build Tools

3.2.1 Investigation

As of June 2015 due to the rise in popularity in container technologies, with emphasis on the rise of Docker, the Open Container Initiative (OCI) was born. It is the goal of the initiative to form an open standard in the Runtime Specification and the Image Specification of containers. Meaning containers could be built and run by a combination of tools and technologies (*OCI*, 2018). Just three years from the launch of OCI a number of interoperable tools are available. To find the best fit OCI tool for this project acceptance criteria was set for this investigation. The criteria is:

- Must be interoperable with technology chosen for CLI
- Must be OCI compliant
- Must support Runtime Specifications for local testing
- Must support Image Specifications for interoperablity with Knative

Based on the criteria outlined, Docker was chosen for the prototype and will be discussed in detail in Section 3.2.2. Other technologies looked at during this investigation have been Kata[5], Buildah[6] and Cri-O[7]. However, they were not deemed suitable because of the unavailability and maturity of their APIs.

3.2.2 Docker

Docker is a Container platform that controls the entire life cycle of a Container. A Container is a unit of software that packages up an applications code and all its dependencies and enables it to run on any computing environment with a suitable runtime. A Docker image is an executable package which includes everything needed to run. It is not until runtime that an image becomes a Container. This runtime can be any OCI compliant engine (*Docker*, 2018).

For the purpose of the prototype the focus will be on both the creation of an image and the execution of the image on the Docker engine. This can be satisfied through the use of the Docker GoLang SDK. An example can be seen in Figure 7, which shows the GoLang equivalent to using the Docker client to list all running containers within the Docker engine.

This shows that Docker satisfies the criteria set out in Section 3.2.1, as it is interoperable with the chosen technology for the CLI in Section 3.1. Docker is also OCI compliant and supporting both Runtime Specification and Imagine Specifications.

[5]https://katacontainers.io – Open Source Container building tool
[6]https://github.com/containers/buildah – Open Source Container building tool
[7]http://cri-o.io – Open Source Container runtime

```go
package main

import (
  "fmt"

  "github.com/docker/docker/api/types"
  "github.com/docker/docker/client"
  "golang.org/x/net/context"
)

// new environment
func main() {
  cli, err := client.NewEnvClient()
  if err != nil {
    panic(err)
  }
    // list all container options
  containers, err := cli.ContainerList(context.Background(), types.
      ContainerListOptions{})
  if err != nil {
    panic(err)
  }
    // foll all containers, print
  for _, container := range containers {
    fmt.Println(container.ID)
  }
}
```

Figure 7: *Docker SDK Example*

3.3 Software Development Kit

3.3.1 Investigation

It is the SDK's job to abstract all of the configuration away from the user and allow them to focus on the logic behind their functions. The SDK will add the most value to the project by reducing the cost of admission to the development of serverless functions. Due to this fact, a great deal of consideration had to be given to choose the initial supported paradigm. As this would be the flagship SDK to this project and will be used for the Community Launch of the project as well as demo's at a local tech talk.

The approach taking in this investigation involved looking at the most common and popular programming languages. Thus giving a greater probability in the tool being of value to users and reducing the knowledge needed to use the tool. As can be seen in Figure 8, from the Stack Overflow Survey (*Stack Overflow Developer Survey 2018*, 2018), JavaScript is top of the poll. This is further backed from the GitHub Insights Statistics 2018 *The State of the Octoverse* (2018), viewable in Figure 9, which shows a total of 2.5 million Pull Requests. Pull Requests are a method for contributing to projects on GitHub, thus being a good indicator for usage of GoLang through

Figure 8: *Stack Overflow Most Popular Technologies* – Stack Overflow Developer Survey 2018 *(2018)*

contributions.

While based on the results of supporting JavaScript and supplying an SDK for JavaScript a decision was made to develop the SDK in TypeScript. The reasoning behind this decision will be discussed in Section 3.3.2. A number of other supported languages where also chosen, these will be put into the product backlog and discussed in Section 7.2. Among the other languages chosen where Python, GoLang and Java.

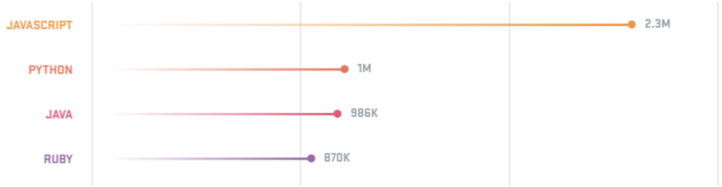

Figure 9: *GitHub Number of Pull Requests* – The State of the Octoverse *(2018)*

3.3.2 Typescript

TypeScript could be considered as modern JavaScript. This consideration is made as in comparison to JavaScript, TypeScript is strongly typed in that you must declare types and variables. Types can be summed up as the simplest units of data such as numbers, strings, structures and boolean values. Where as JavaScript, being dynamically typed, does not know what type a variable is until it is instantiated at run-time. This can lead to problems by false assumptions on variables, whereas with TypeScript these false assumptions can be caught during development. With added features like strict null checks, interfaces and schemas, TypeScript will overall improve the quality and usability of this project (*TypeScript*, 2018).

Further backing the decision to use TypeScript is its interoperablity to JavaScript this means that JavaScript modules can be used and understood by the TypeScript compiler. It also means that TypeScript transcompiles down to JavaScript meaning that any TypeScript code can be used in a JavaScript ecosystem. This will allow for the leveraging of the community and ecosystem behind JavaScript and being able to use available modules will overall decrease development time and allow for more complex features with less overhead.

4 Tools

4.1 Project Management Tools

4.1.1 JIRA

JIRA is a proprietary issue tracking product developed by Atlassian. It allows for the planning, tracking and management of Agile projects with stories and tasks broken down into a series of JIRA tickets. These tickets are stored in the backlog within JIRA allowing for the planning of sprints and managing the overall goal of each sprint.
As discussed in Section 5.6, to meet the void between JIRA, Agile and Scrum in what defines an epic, story and task a number of rules were set. Each story that was defined had a number of tasks associated with. These tasks were small achievable increments that would lead to satisfying the acceptance criteria set for each story. The story would be seen in JIRA as an Epic and the corresponding tasks would be outlined as tickets within JIRA.

4.1.2 Trello

Mentioned in Section 4.1.1, there is a disconnect between between JIRA, Agile and Scrum definitions. Due to this it was decided to incorporate Trello within the planning phases of the project. Trello is an online tool for tracking and project management. Due to its ease of use and a high standard of visualisation it made it easier to plan the User Stories for this project. The board used for planning can be found at Appendix C. This also allows for a higher level of abstraction, with Trello being more suited to story level discussions and JIRA more granular, technical level.

4.1.3 Trira

As Trello has a rich API a number of tools and plugins have been made to increase the usability of Trello. One tool which is used throughout this project is the Open-Source tool Trira. It can be found at Appendix D. It allows for the synchronisation between JIRA and Trello through the use of a command line tool. A separate Trello Board was set up for porting User Story Tasks to JIRA tickets. This can be found at Appendix E

4.1.4 GitHub

The architecture of this project as described in Section 5 will incorporate a number of different code bases. To allow for the ease of version control on multiple code bases GitHub is used. All code will

be housed within multiple repositories under an organisation on GitHub. The project willl adhere to semver standards for versioning.

4.2 Technical Tools

4.2.1 CircleCI

CircleCI is the tool chosen to handle all continuous integration of the platform. It will be incorporated into all GitHub repositories and will allow for the automation of deployments throughout the projects lifetime.

4.2.2 SonarQube

SonarQube is a Continuous Inspection tool which will be used in parallel to the continuous integration pipelines of the platform to ensure high code quality throughout.

4.2.3 Coveralls

In order to eliminate technical debt Coveralls is used to highlight untested areas of the codebase. It will work with the continuous integration to expose any gaps.

5 Design

5.1 System Architecture Overview

This project is essentially broke into multiple elements, each element consisting of its own architecture. The elements are as follows:

- Command Line Interface
- Software Development Kits
- Showcase Application

5.1.1 Command Line Interface

The CLI will be of modular design, consisting of a main application which will have common logic shared between all plugin modules. These plugin modules will consist of language specific logic associated with each of the supported languages of the product. The main application will utilise the Docker API to connect to the Docker Daemon. This can be seen illustrated in Figure 10

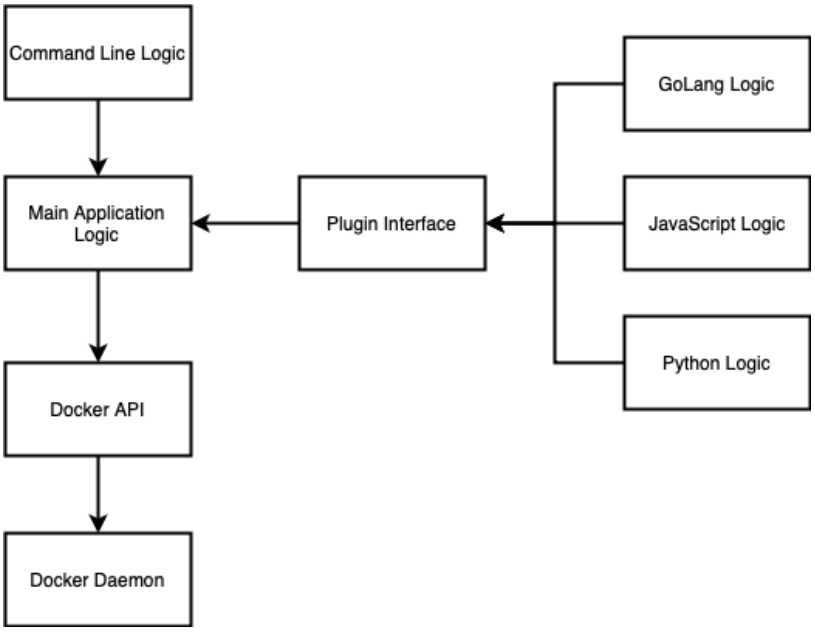

Figure 10: *Command Line Interface Architecture Diagram*

5.1.2 Software Development Kits

There will be a number of SDK's developed as part of this project, each SDK will correspond to a single supported language to the project. Each individual SDK will follow the same architecture. This architecture can be seen in the form of a Class Diagram in Section 5.15

5.1.3 Showcase Application

The full specification to the Showcase Application is to be decided during product backlog grooming. As of writing this report the high level overview to the showcase application consists of either a mobile application or a web application front end. These communicate to a K-Native backend running in a Kubernetes engine which will house a Function as a Container that communicates to an S3 Bucket. This can be seen illustrated in Figure 13.

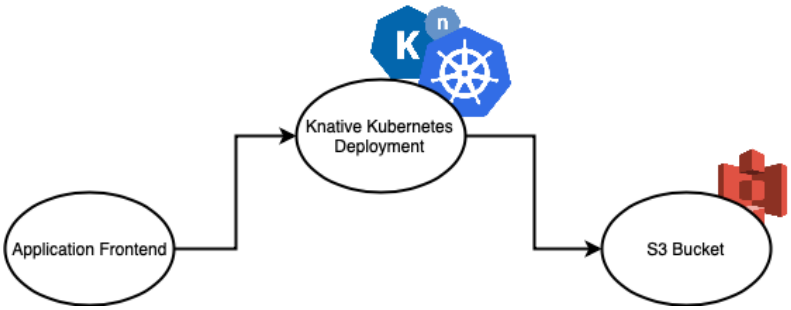

Figure 11: *Highlevel Showcase Architecture Diagram*

5.2 Requirements

Based on meetings with engineers currently working within the Knative project along with engineers working within RedHat, the problem which was discussed in Section 1.2 was identified. While identifying the problem a number of requirements were highlighted. These requirements will be split into two categories, Functional and Non-Functional.

It should be noted that these requirements were identified during the early stages of project planning and could be liable to change depending on the outcome of the Proof of Concept, discussed in Section 6.

5.3 Functional Requirements

The functional requirements listed below will be illustrated in both Section 5.16 and 5.17 as Sequence and Use Case Diagrams. Each requirement includes a rationale to help with understanding

the purpose of it.

R1 - Bootstrapping of a Function as a Container
To allow a user easily create Functions as a Container, without abstracting anything from the user, giving freedom to test, debug and customise as needed.

R2 - Create a Function as a Container Image
To improve the work flow by saving a developer from switching between tools throughout the creation of a Function as a Container lifecycle.

R3 - Build a Function as a Container
To allow for local deployment to an OCI compliant runtime independent of Knative and Kubernetes.

R4 - Push a Function as a Container to a Registry
To allow for a means of secure storage for a developers Functions as a Container without introducing adding complexity to a developers workflow by utilising systems already used by a developer.

R5 - List all Functions as a Container independent to regular containers
To improve visibility of a developers Functions as a Container isolating them from a regular container which may be on a users system.

R6 - Run a Function as a Container locally
To allow for local manual testing independent to Knative and Kubernetes.

R7 - Test a Function as a Container locally post deployment to Knative
To ensure trust by a developer in their Functions as a Container and to aid throughout the development cycle.

R8 - Stop a Function as a Container locally
To allow a developer stop a Function as a Container locally without the need to revert to another tool, thus improving the overall product experience.

R9 - Delete a Function as a Container locally
To improve workflow allowing a developer to clean their local host of any unwanted Functions as a Container without the need to revert to another tool.

5.4 Non-Functional Requirements

The following is a number of non-functional requirements that were identified in the early stages of planning. These requirements fall under a number of broad categories.

R10 - Maintainability: The product as a whole must remain maintainable by implementing procedures to allow for the software elements to be modified to correct faults, improve performance and overall re-factoring of the code structure.

R11 - Quality: Processes need to be in place to guarantee a high quality in the overall code base across all elements within the project. These processes should be built into CI/CD tools like those discussed in Section 2

R12 - Security: Systems to be in place to ensure unauthorised, accidental or unintended usage of a users Functions as a Container does not happen. Processes to ensure that the latest versions of dependencies are used where feasible [CVE].

R13 - Documentation: Any addition to the project must be accompanied by concise and practical documentation. This documentation must be easily accessible to users.

R14 - Performance: All software must provide a minimal footprint on a users host, along with minimal response times. Thus reducing the time needed on CPU usage along with space taken on disk.

5.5 Core Requirements and Stretch Goals

All requirements were broken into a number of user stories. These user stories can be seen in Section 5.6. On the creation of the user stories and the core requirements a minimum viable product (MVP) was defined and outlined. An MVP is a product with all core features given minimal functionality in order to provide feedback to further improve on in later sprits. Each story was tagged with an id which is reflected in the tables in Section 5.6 with tags beginning with *mvp*. Stretch goals can be identified by tags beginning with *s*.

5.6 User Stories

Following the Agile Methodology, which is discussed in Section 2.1, the planning of this project is broken into Epics and User Stories. As this project uses JIRA to track and plan the development

progress throughout sprints, which is discussed in Section 4.1.1, there is a disconnect between what is a JIRA epic and story in comparison to the definition of an Agile epic and story. In order to minimise this disconnect Trello was used during the planning, discussed in Section 4.1.2, and the following rules were applied.

User Stories would be broken into themes; each theme has an overview story. Each overview story will have a number of User Stories associated with it. These user stories would be treated as epics within JIRA and each epic will have a number of tasks associated with it.

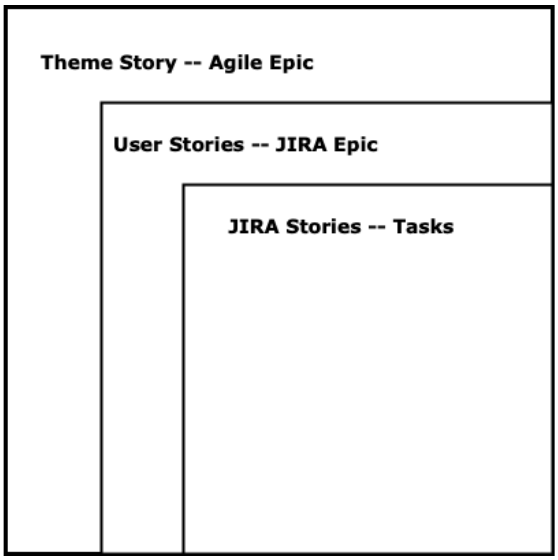

Figure 12: *Bridging the Agile – JIRA gap*

It must be noted that each User Story contains a *What, Why, How* along with *Acceptance Criteria* and *Tasks*. For brevity these will be left out of the document but can be found on the associated Trello Board found at Appendix C

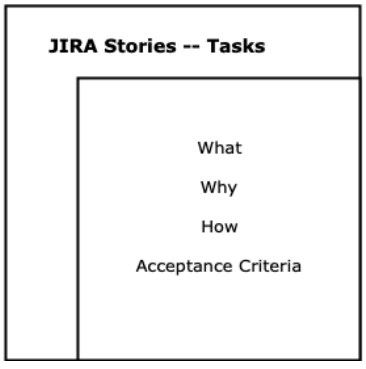

Figure 13: *JIRA Story – Task*

5.7 Themes

User Stories where thematically grouped under headings, to help with prioritisation and organisation. User story themes are as follows:

- Containers
- CLI
- Testing
- Security
- Developer
- Community
- ShowCase

5.8 User Definitions

A number users have been defined in order to add clarity to the user stories:
1 – General User Definition: A user of the product who would ideally be a developer who is used to working with tools like Docker and Kubernetes, minimal admin privileges are granted
2 – Community User Definition: A user of the product who is also a contributor.
3 – Admin User Definition: A developer to the product, with full admin and configuration privileges
NOTE: A Community User incorporates the user stories of a General user, while the Admin User incorporates all users.

Note : The following convention is used to number the User Stories, the *mvp* tag signifies the story to be included in the Minimum Viable Product. Whereas the *s* tag signifies the story to be considered a stretch goal.

5.9 Containers

Overview Story

As a General User, I want a container function so that I can have full control over my FaaS

#	As a	I want to be able to	such that
mvp1	General User	I want a way to handle HTTP requests in JavaScript	I can call my functions in a container
mvp2	General User	I want to be able to create a JavaScript container	I can leverage my own FaaS
s1	General User	I want to be able to create GoLang functions as a container	I am not limited to one programming language
s2	General User	I want to be able to create Python functions as a container	I am not limited to one programming language

Table 1: Container User Stories

5.10 Testing

Overview Story

As a General User I want to easily test my container functions so that I can highlight any errors quickly and efficiently.

#	As a	I want to be able to	such that
s9	General User	I want to easily be able to stress test my container functions	I can ensure reliability in the functions under load
mvp9	General User	I want to easily be able to test my functions outputs	I can easily test for errors without the need to write explicit tests
mvp10	Admin User	I want my production pushes to only occur when all tests are green	no unseen errors make it through my CI

Table 2: Testing User Stories

5.11 CLI

Overview Story

As a General User, I want to be able to control the lifecycle of my container functions so that I can increase productivity while maintaining full control and ownership of my container functions

#	**As a**	**I want to be able to**	**such that**
mvp3	General User	I want to bootstrap the creating of container functions	productivity of using container functions is increased abstracting the startup tasks
mvp4	General User	I want to easily create a container function	I maintain full control and ownership of my container function.
mvp5	General User	I want to be able to push my container function to a registry	I can easily access and store my functions
mvp6	General User	I want to be able to easily list what container functions I have created	they do not get lost between my other regular containers
mvp7	General User	I want to easily deploy a container function	I can easily test my container function.
s3	General User	I want to easily deploy container functions to Kubernetes	I do not need to switch between tools throughout the container function development process.
mvp8	General User	I want to be able to delete container functions	I can remove unused container functions without the need of using native tooling.
s4	General User	I want the ability to create multiple container functions at once	I can increase productivity by controlling multiple function container functions.
s5	General User	I want the ability to bootstrap multiple container functions at once	I can increase productivity by controlling multiple function container functions.
s6	General User	I want the ability to deploy multiple container functions locally at once	I can increase productivity by controlling multiple function container functions.
s7	General User	I want to be able to update container functions	I can easily reuse existing functions
s8	General User	I want to be able to roll back container function updates	I can roll back use cases in the event of a problem I can recover

Table 3: CLI User Stories

5.12 Security

Overview Story

As a General User I want to have confidence in my container functions such that the container functions are as secure as possible

#	As a	I want to be able to	such that
s10	General User	I want the ability to run my container functions in read only mode	I can ensure malicious code can not be injected or introduced to my containers during deployment
mvp11	Admin User	I want to have my systems protected from known CVEs	I can have assurance throughout my development
s11	General User	I want to grant read and write permissions to my function containers	I can control who owns the function

Table 4: Security User Stories

5.13 Developer

Overview Story

As an Admin User I want to ensure consistency throughout the project so that the project code base maintains a high standard

#	As a	I want to be able to	such that
mvp12	Admin User	I want nightly builds of the project	I can track the progress of the project and highlight any issues as early as possible into the development.
mvp13	Admin User	I want builds on every Pull Request	I can ensure no potential breaking changes are merged to the master branch
mvp14	Admin User	I want security vulnerability scans to be conducted as part of the CI	any potentially harmful vulnerabilities are highlighted early in the development process
mvp15	Admin User	I want the application to be available via a container registry	I can easily access, install and run the applications
mvp16	Admin User	I want to make my SDKs available for installation through package management tools	any user can easily access and use the SDK's independent to using the tool itself

Table 5: Developer User Stories

5.14 Community

Overview Story

As a General User I want this project to be developed in the open so that I can benefit from the transparency while being able to contribute to and give back to the project

#	**As a**	**I want to be able to**	**such that**
mvp17	Community User	I want a medium to be able to get involved in discussions around the project	I can participate and help shape the project
mvp18	Community User	I want a medium to be able to receive updates about the project	I can stay up to date with the project
mvp19	Community User	I want a medium to quickly talk to project developers	I can easily have any questions answered and share knowledge
mvp20	Community User	I want all repos open and under a single organisation	I can easily find and browse the available repos
mvp21	Community User	I want to have all documentation consolidated in one location	I can easily find and browse the documentation
mvp22	Community User	I want a clearly defined definition of done	I can easily contribute to the project and get involved

Table 6: Community User Stories

5.15 SDK Class Diagram

Outlined in Section 5.1.2, there will be a number of SDK's developed throughout this project. Each SDK will follow the class structure outlined in Figure 14

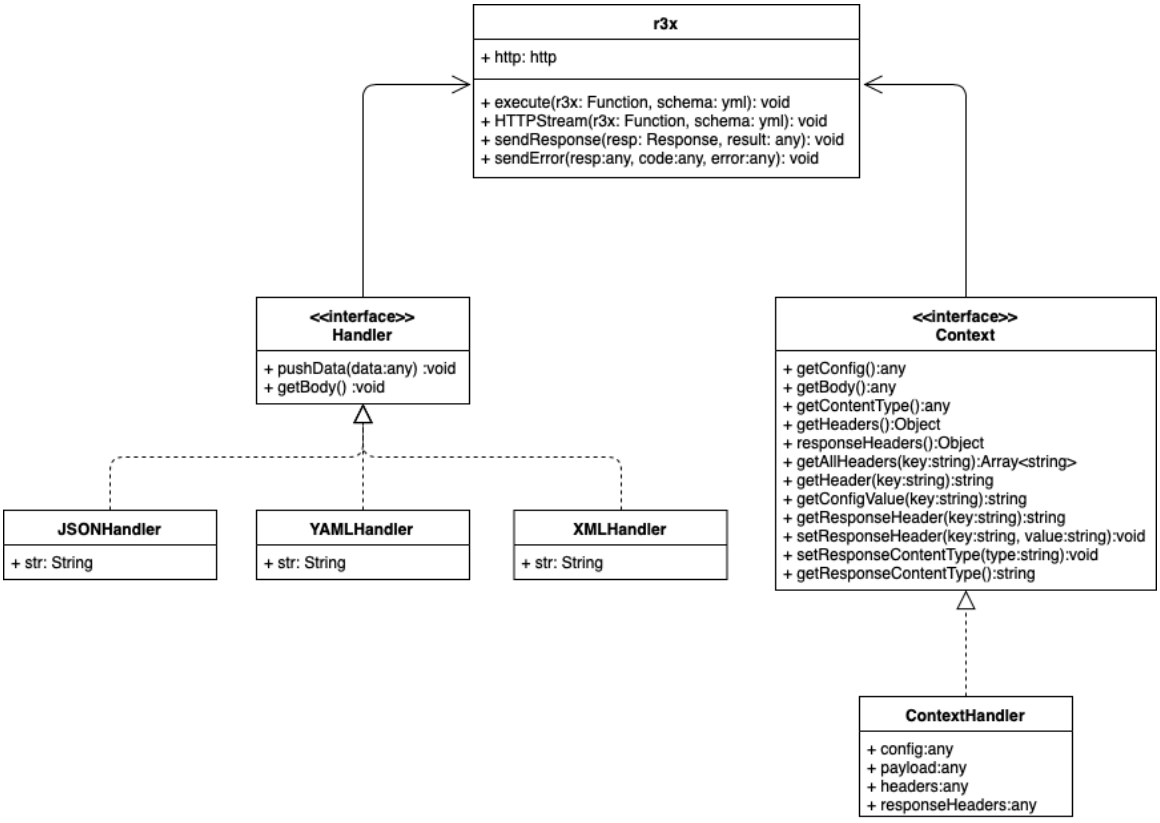

Figure 14: *SDK Class Diagram*

5.16 CLI Sequence Diagram

A number of sequence diagrams have been created to show object interactions between the CLI, Docker API and Docker Daemon.

5.16.1 Building an Image

Building an image via the command line, to improve the work flow by saving a developer from switching between tools throughout the creation of a Function as a Container lifecycle.

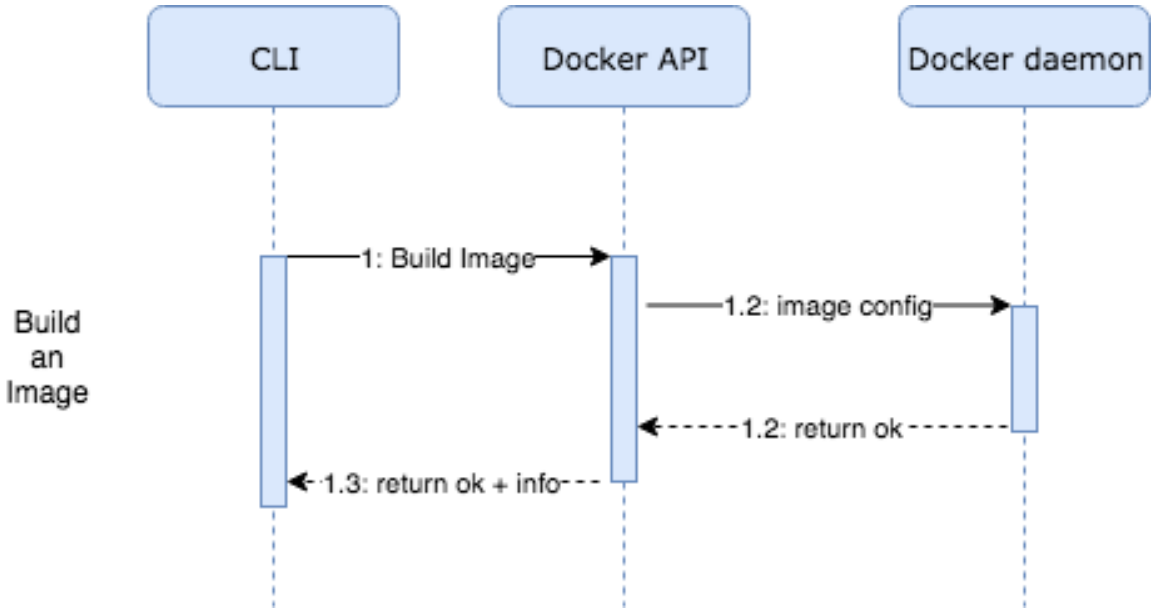

Figure 15: *CLI Sequence Diagram for Building Images*

Sequence Steps

1. CLI sends build image command to Docker API
2. Docker API receives command and triggers build image in Docker Daemon
3. Docker Daemon returns response 200 OK status and built image
4. Docker API returns response and image information to CLI

5.16.2 Building a Container

Building a Container locally, to allow for local deployment to an OCI compliant runtime independent of Knative and Kubernetes.

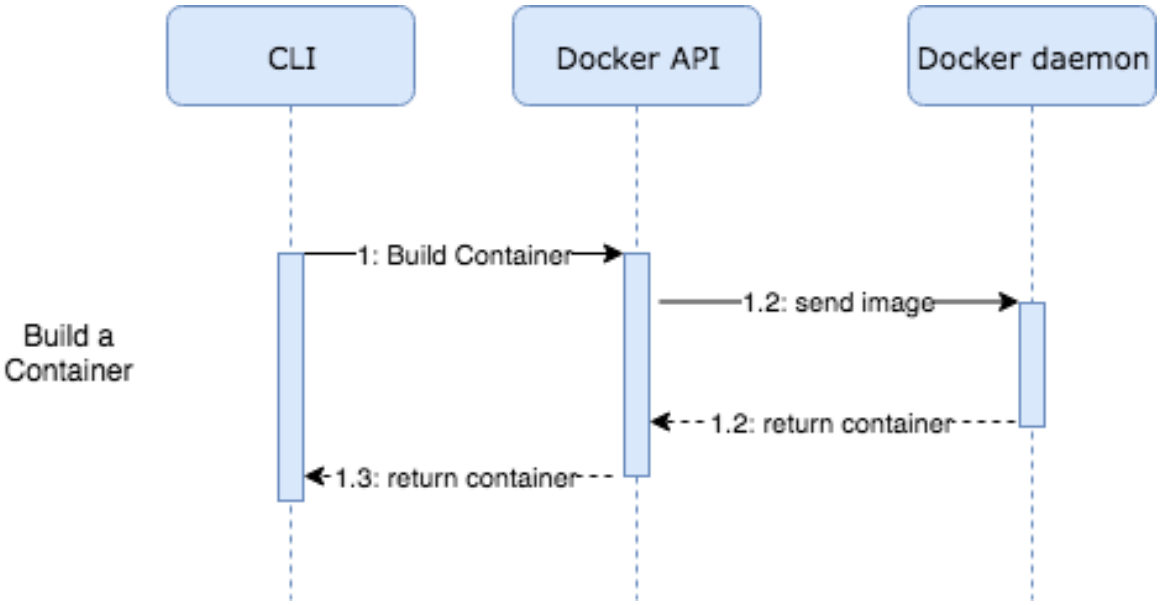

Figure 16: *CLI Sequence Diagram for Building Containers*

Sequence Steps

1. CLI sends build container command to Docker API
2. Docker API receives command and triggers build container in Docker Daemon
3. Docker Daemon returns response 200 OK status and built container
4. Docker API returns response 200 OK status and container information to CLI

5.16.3 Starting Containers

To allow for local manual testing independent to Knative and Kubernetes.

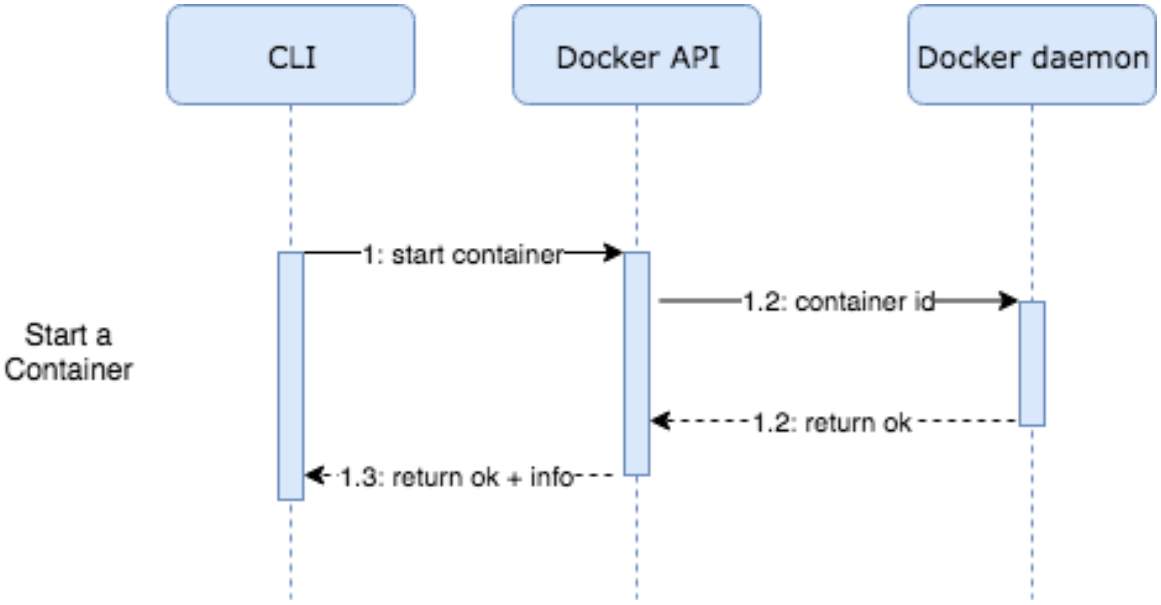

Figure 17: *CLI Sequence Diagram for Starting Containers*

Sequence Steps

1. CLI sends start container command to Docker API
2. Docker API receives command and triggers the daemon to start a container via the container ID
3. Docker Daemon returns response status 200 OK
4. Docker API returns response status 200 OK to CLI

5.16.4 Deleting Images

Deleting Images, to improve workflow allowing a developer to clean their local host of any unwanted Functions as a Container without the need to revert to another tool.

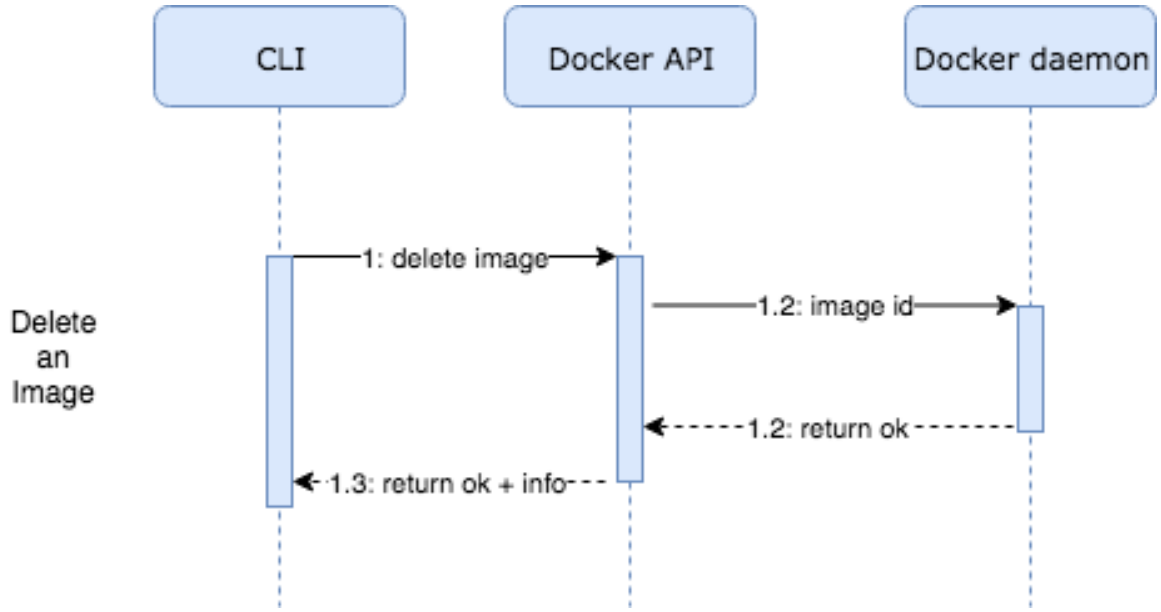

Figure 18: *CLI Sequence Diagram for Deleting Images*

Sequence Steps

1. CLI sends delete image command to Docker API
2. Docker API receives command and forwards image id to Docker Daemon
3. Docker Daemon returns response status 200 OK on deletion of image
4. Docker API returns response status 200 OK and information to CLI

5.16.5 Stopping Containers

To allow a developer stop a Function as a Container locally with out the need to revert to another tool, thus improving the over all product experience.

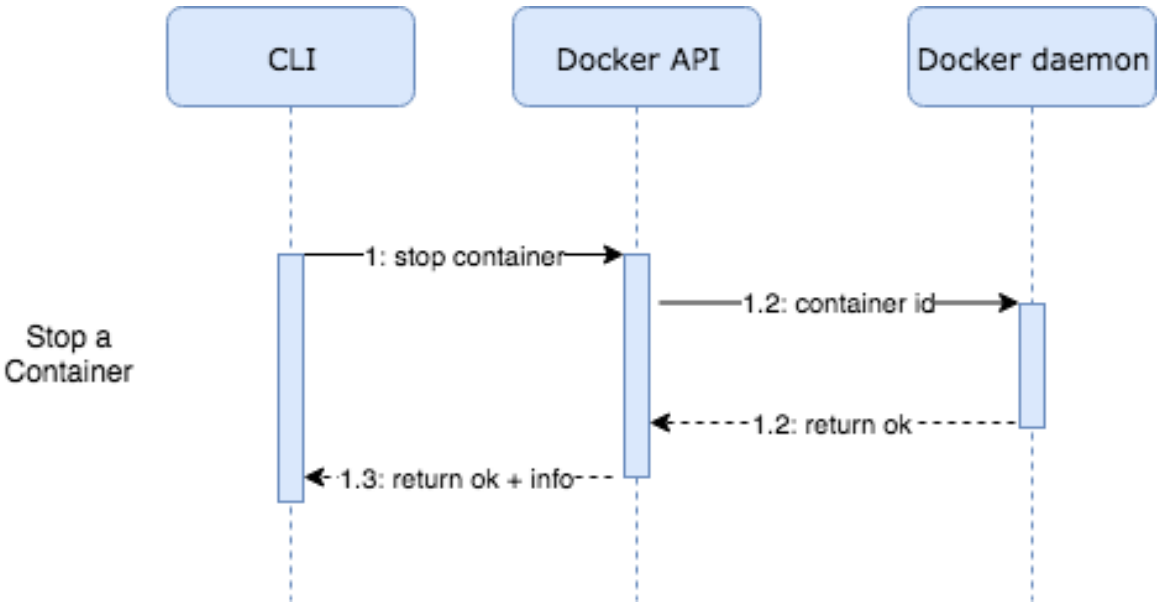

Figure 19: *CLI Sequence Diagram for Stopping Containers*

Sequence Steps

1. CLI sends stop container command to Docker API
2. Docker API receives command and forwards stop container to Docker Daemon with the container ID
3. Docker Daemon stops container and returns response 200 OK
4. Docker API returns response 200 OK and container information to CLI

5.16.6 Listing Containers

To improve visibility of a developers Functions as a Container isolating them from a regular container which may be on a users system.

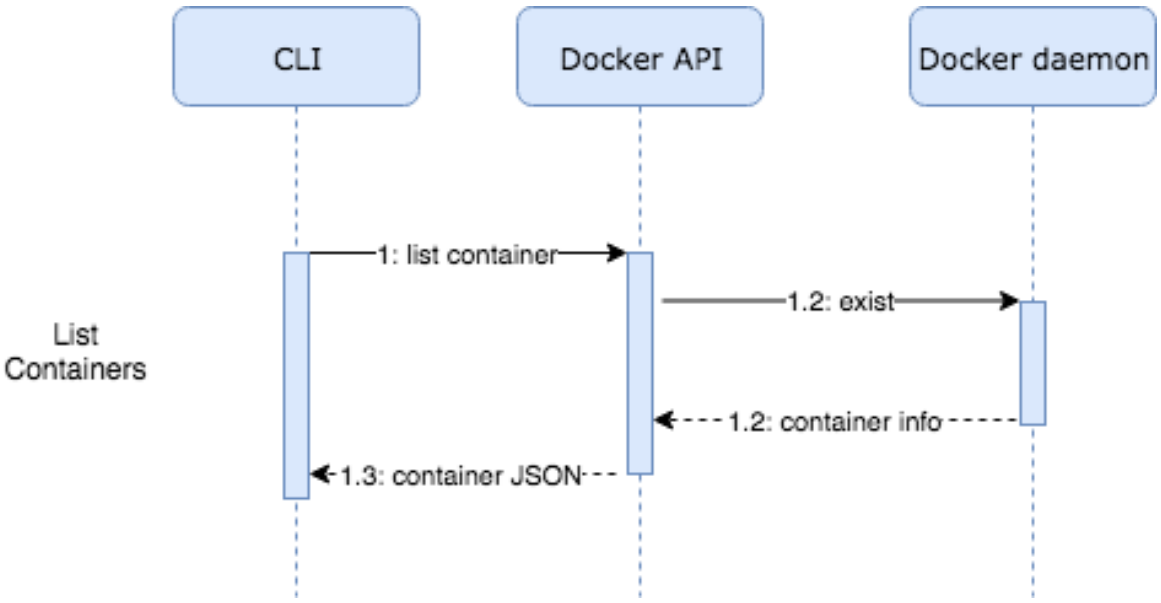

Figure 20: *CLI Sequence Diagram for Listing Images*

Sequence Steps

1. CLI sends list image command to Docker API
2. Docker API receives command and forwards command to Docker Daemon
3. Docker Daemon returns response 200 OK all containers
4. Docker API returns response 200 OK and container information to CLI, the CLI parses information and returns list of Function as a Container to user.

5.17 CLI Use Case Diagram

A representation of a typical users interactions with the Docker Daemon via the CLI durning development of a Function as a Container.

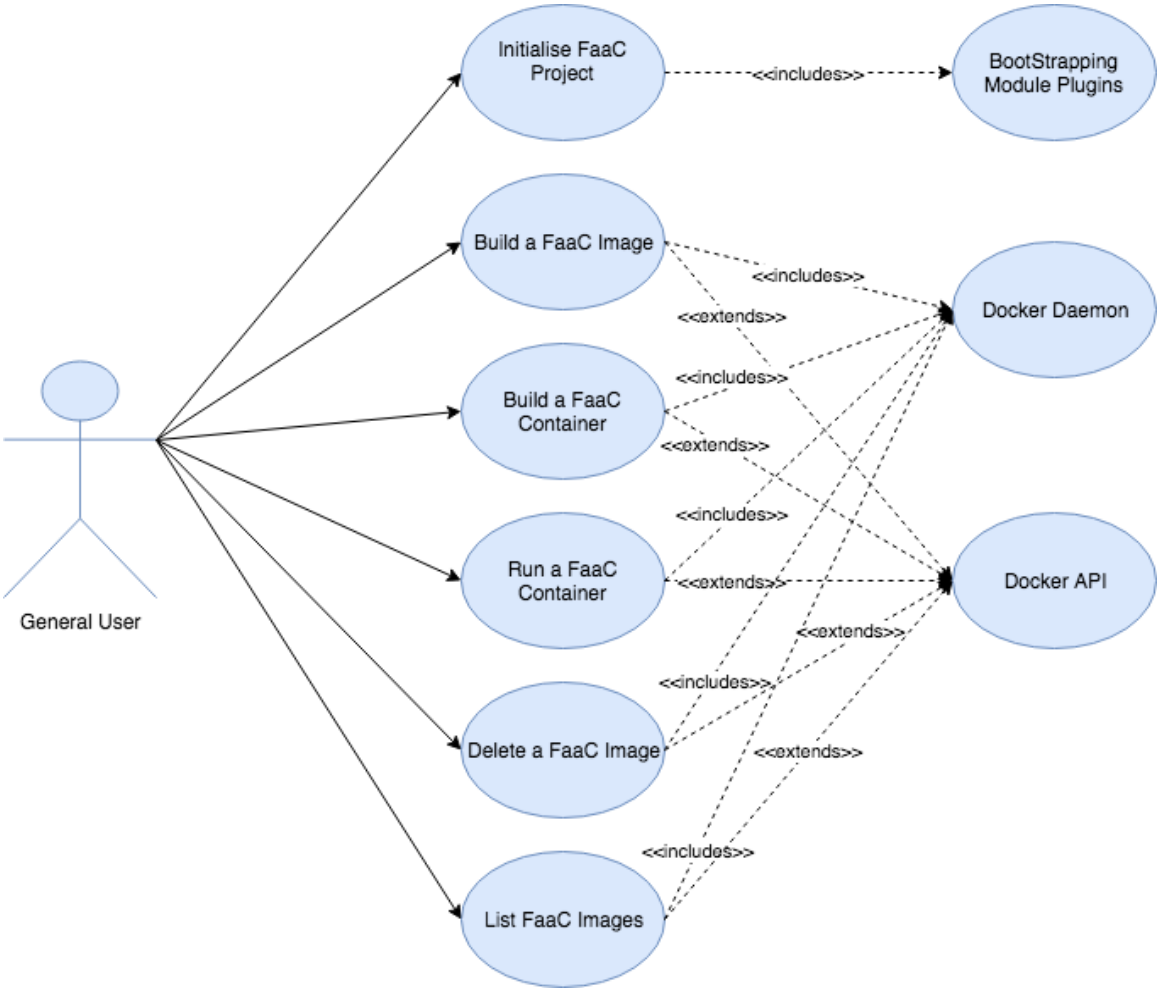

Figure 21: *CLI Use Case Diagram*

6 Implementation of Prototype

To ensure full technical feasibility, two Prototypes were developed. The first was used as a benchmark for further iterations, ensuring that work carried out within this project not only improves the user experience, but also improves on the existing solution technically. The second prototype was developed following the design laid out in Section 5.

6.1 Prototype 1

As outlined in Section 1.4, essentially a Function as a Container is a function wrapped in a server and packaged as an image. Through the development of this prototype a server was built in Typescript. Included in this server were two endpoints that satisfy a HTTP Get Request and a HTTP Post Request. This was built with the aid of a common Node Module *Express*[8] and followed recommended patterns of Middlewares / pipelines. This is a simple abstraction in that the output of one unit/function is the input for the next
This prototype serves as a benchmark for further implementations as the project looks to provide a better solution to the development of a Function as a Container. It provides a baseline of a typical Container for us to compare. The image for this prototype can be found at Appendix F.

6.2 Prototype 2

The resulting code from this Prototype of the SDK can be found at Appendix G, along with a showcase application built to demo it, which can be found at Appendix H. Completed in this Prototype is the ground work for the SDK, written in Typescript allowing a user to create a Function as a Container. This provided the feasibility for the project and the design outlined throughout this paper.

6.3 Prototype Summary

Prototype	Size	Vulnerabilities-	Latency
One	75.3MB	90	15ms
Two	23.4MB	0	7ms

Table 7: Prototype Comparison Table

[8]https://expressjs.com – Minimalist web framework for Node.js

A number of arbitrary statistics where taken from the two prototypes based on freely available statistics from Quay.io[9]. These results can be seen in Table 7. As we can see our prototype following the design structure outlined in this report has made improvements in both size and latency, while reducing the number of vulnerabilities within our Container. This in my opinion is due to stripping away third party modules and dependencies, as it allows us to focus on what we need with out accumulating a large dependency tree which we do not control. This prototype will now be focused on for the remainder of this project.

[9]https://quay.io – Container Repository

7 Summary

7.1 Review of Work Completed

Throughout this Semester I set out to identify a problem within the serverless space. Upon identifying the problem a proposed solution was sought. This led to the identification of technologies which could solve the problem. A number of tools and methodologies were then chosen to help in the development of this solution. A number of design patterns were explored and looked at, resulting in the architecture seen in this report. Finally a benchmark prototype was built, with a second prototype being developed and improving on the benchmark set.

7.2 Semester Two Work to Complete

With the prototype complete, and due to this project being Agile a backlog refinement meeting will be held with all stakeholders in this project to groom the backlog and make adjustments where needed that best suit the current state of the project. With the reduction of unknowns, work in Semester Two will be incremental releasable improvements on what is already complete.

There will be a local tech talk on Knative and building an optimal Function as a Container, held during Semester two. An Open Source Community release of the project will follow and will include a hackaton to gain valuable insights from the Kubernetes community on the product as a whole.

The intention is to complete the MVP stories and achieve all core functionality. Any stretch goals completed will be a bonus for this project and will serve as a launch point for community involvement.

Appendices

A GitHub Organization

https://github.com/rubixFunctions

B Circle CI Repository

https://circleci.com/gh/rubixFunctions

C Trello User Stories

https://trello.com/invite/b/1tnWUrMa/9be869b4b608c1a9c8fb40ff4723f18b/r3x-user-stories

D Trira GitHub

https://github.com/aerogear/trira

E Trello Trira Board

https://trello.com/invite/b/VgT5oxJ3/d6a4fa66d09c13e37f3b6c0a7969d3a7/rubix-task-planning-board

F Prototype One

https://quay.io/repository/ciaranroche/r3x-poc-1

G Prototype Two

https://github.com/rubixFunctions/r3x-js-sdk

H SDK Showcase

https://github.com/rubixFunctions/r3x-js-showcase

Bibliography

Bryant, D. (2018), 'Google releases knative: A kubernetes framework to build, deploy, and manage serverless workloads'.
URL: *https://www.infoq.com/news/2018/07/knative-kubernetes-serverless*

Currents: A quarterly report on developer trends in the cloud (2018).
URL: *https://www.digitalocean.com/currents/june-2018/*

Docker (2018).
URL: *https://www.docker.com/*

Fowler, M. (2006), 'Continuous integration'.
URL: *https://martinfowler.com/articles/continuousIntegration.html*

Fowler, M. (2018), 'Serverless Architectures'. [online] Accessed: 6/10/2018.
URL: *https://martinfowler.com/articles/serverless.html*

GoLang (2018).
URL: *https://golang.org/doc/*

Manifesto for Agile Software Development (n.d.).
URL: *https://agilemanifesto.org/*

OCI (2018).
URL: *https://www.opencontainers.org/about*

Open Standards Requirement for Software (2018).
URL: *https://opensource.org/osr*

RightScale 2018 State of the Cloud Report (2018).
URL: *https://www.rightscale.com/lp/state-of-the-cloud*

Schwaber, K. and Sutherland, J. (2017), 'The Scrum Guide'. [online] Accessed: 29/10/2018.
URL: *https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf*

Solis, C. and Wang, X. (2011).
URL: *https://ieeexplore.ieee.org/document/6068372?arnumber=6068372*

Stack Overflow Developer Survey 2018 (2018).
URL: *https://insights.stackoverflow.com/survey/2018/*

State of Agile Report (2018).
URL: *https://explore.versionone.com/state-of-agile/versionone-12th-annual-state-of-agile-report*

State of Cloud 2018 Report (2018).
URL: *http://get.cloudability.com/ebook-state-of-cloud-2018.html*

The Scrum Guide^{TM} (2018).
URL: *https://www.scrumguides.org/scrum-guide.html*

The State of the Octoverse (2018).
URL: *https://octoverse.github.com/*

TypeScript (2018).
URL: *https://www.typescriptlang.org/*

Vivien, V. (2017), 'Writing modular go programs with plugins – learning the go programming language – medium'.
URL: *https://medium.com/learning-the-go-programming-language/writing-modular-go-programs-with-plugins-ec46381ee1a9*

Where PaaS, Containers and Serverless Stand in a Multi-Platform World (2018).
URL: *https://www.cloudfoundry.org/multi-platform-trend-report-2018/*