uc3m | Universidad Carlos III de Madrid

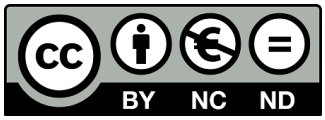
University Degree in Computer Science and Engineering
Academic Year 2020-2021

Bachelor Thesis

"GitOps continuous deployment and management tool for Kubernetes-based distributed systems"

Pablo Gómez-Caldito Gómez

Advisor: Francisco Javier García Blas
Leganés, September 2021



Agradecimientos

Primero de todo a mi familia, por darme siempre el apoyo y cariño que me ha permitido completar mis estudios. Especialmente a mi padre por ayudarme a entender las asignaturas que se me atascaban, a mi madre por todo lo que hace por mí.

También a mis amigos, por todos los buenos momentos que comparto con vosotros y estar siempre ahí cuando se os necesita. Sois muy valiosos para mí.

Por otro lado, a mis compañeros y amigos de la universidad tanto del campus de Colmenarejo como de Leganés. Especialmente a Adri, Ramón y Raúl, por ser mis compañeros de batalla en tantas prácticas, y los ratos en las cafeterías y discord.

También a mis compañeros de equipo en idealista por todo lo que me habéis enseñado y he aprendido trabajando a vuestro lado. Esos conocimientos me han ayudado mucho para tener una muy buena base para realizar este trabajo.

Finalmente a Javier, mi tutor. Por guiarme durante estos meses de trabajo semana a semana de forma intachable. Ojalá muchos más profesores tuvieran tu paciencia y buen trato hacia sus alumnos.

A todos, y a los que faltan por poner, tenéis mi más profundo agradecimiento.

GitOps continuous deployment and management tool for Kubernetes-based distributed systems

Bachelor Thesis

Pablo Gómez-Caldito Gómez

Abstract

The enormous and increasing amount of internet users has produced bigger and fluctuating workloads in computing infrastructures, and greater competition among software products over the last years. This has led to companies needing far more scalable services and faster delivery of new features. Responding to this demands appeared the DevOps philosophy and more modern infrastructures.

Kubernetes is a piece of software to manage container-based infrastructures and CI/CD is a software development practice to fasten the SDLC. Both are widely used by organizations implementing DevOps in their engineering teams. GitOps is way of performing Continuous Deployment which fits really well with Kubernetes. There are some projects that enable teams to adopt GitOps in Kubernetes clusters but are too complex for some use cases.

This Bachelor Thesis presents a GitOps tool for Kubernetes-based clusters focused on simplicity, following the Unix philosophy. This tool was designed from the ground up to run inside this clusters to ease its operations and has a modular design for better extensibility. It also uses a feature from Kubernetes called Server-Side Apply. The mentioned feature allows to have a very small deployment module compared to other alternatives by relying a lot of complex logic on the server side. Furthermore, the project is free and open-source so anyone benefit from it and the improvements made by the people who use it regularly.

Keywords: GitOps, Git, Kubernetes, Continuous Deployment, Containers, DevOps Philosophy, Free and Open-Source Software.

Herramienta de Despliegue Continuo GitOps y gestión para sistemas distribuidos basados en Kubernetes

Trabajo de Fin de Grado

Pablo Gómez-Caldito Gómez

Resumen

El enorme y creciente número de usuarios de internet produce cargas de trabajo más grandes y variables en infrastructuras de computación, y más competición entre productos basados en software en los últimos años. Esto ha llevado a que las compañias necesiten servicios mucho más escalables y una entrega mas rápida de nuevas funcionalidades. Respondiendo a estas necesidades ha aparecido la filosofía DevOps e infrastructuras más modernas.

Kubernetes es un software para gestionar infrastructuras basadas en contenedores y CI/CD es una practica para acelerar el Ciclo de Vida del Desarrollo de Software. Ambos son usados por empresas implementando DevOps en sus equipos de tecnología. GitOps es una manera de hacer Despliegue Continuo que encaja muy bien con Kubernetes. Hay algunos proyectos que permiten a los equipos adoptar GitOps enfocadas a clústeres de Kubernetes pero son demasiado complejas para algunos casos de uso.

Este Trabajo de Fin de Grado presenta una herramienta de GitOps para clústeres basados en Kubernetes enfocada en la simplicidad, siguiendo la filosofía Unix. Esta herramienta fue diseñada desde el inicio para ser ejecutada dentro de los clústers para facilitar sus operaciones y tiene un diseño modular para una mejor extensibilidad. También usa una funcionalidad de Kubernetes llamada Server-Side Apply. Dicha funcionalidad permite tener un módulo de despliegue mucho más sencillo que el resto de alternativas al delegar mucha lógica compleja en el lado del servidor. Además, el proyecto es Software Libre y de Código Abierto para que cualquiera se pueda beneficiar de él y las mejoras llevadas a cabo por los usuarios.

Palabras clave: GitOps, Git, Kubernetes, Despliegue Continuo, Contenedores, Filosofía DevOps, Software Libre y de Código Abierto.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

This first chapter gives a presentation of the developed Bachelor Thesis. It explains the motivation resulting in this project, the main objectives proposed to achieve, and lastly the document's structure.

1.1 Motivation

The enormous and increasing amount of internet users has produced bigger and fluctuating workloads in computing infrastructures, and greater competition among software products over the last years [1]. This has led to companies needing far more scalable services and faster delivery of new features. Responding to this demands appeared the DevOps philosophy [2] and more modern infrastructures.

Kubernetes [3] is a piece of software to manage container-based [4] infrastructures and CI/CD [5] is a software development practice to fasten the SDLC [6]. Both are widely used by organizations implementing DevOps in their engineering teams. GitOps [7] is way of performing Continuous Deployment [5] which fits really well with Kubernetes. There are some projects that enable teams to adopt GitOps in Kubernetes clusters but are too complex for some use cases.

This work presents three main contributions. The first one is a GitOps tool for Kubernetes-

based clusters focused on simplicity. Furthermore, presents an modular and extensible design to facilitate future updates. The second is a much simpler deployment module compared to other alternatives by relying a lot of complex logic on the server side. To achieve it uses a new feature from Kubernetes called Server-Side Apply. The third one is the community aspect of this tool. The project is free and open-source so anyone benefit from it and the improvements made by the people who use it regularly.

This project aims to build a tool that does one thing well and in a simple way, following the Unix philosophy [8]. In this project, we target at GitOps, in terms of continuous deployment based on information stored in Git repositories.

1.2 Objectives

The fundamental objectives sought with this project are associated to the build and enhancement of the said tool, and can be summarized in three,

Obtain a simple yet useful GitOps tool.

The resulting program is meant to be very simple but powerful nonetheless. It will only have a handful of key features which add plenty of value.

Tool working inside Kubernetes cluster.

Successfully have the tool running inside a Kubernetes-based cluster. This simplifies service operations and getting the credentials to communicate with the Kubernetes API.

Make the project free and open-source.

The project will be open-source so other people can benefit from it and contribute. It is a win-win situation for everyone and can also add revenue by selling support to other development teams who use it.

1.3 Document's Structure

This document is composed by 9 different chapters, each one providing insight on it from a different angle to give a complete understanding of the resulting work.

Chapter 1. Introduction.

Shows the reasons for developing the project, the objectives proposed to reach and how the document is structured.

Chapter 2. State of the Art.

Gives the reader an introduction on the developments of the field in which this project belongs. Also explains some technologies that are going to be used.

Chapter 3. System Overview

Provides a general overview of the tool. Explaining its background and a first high level description of it.

Chapter 4. Analysis

Serves to expose the found functional and non-functional requirements of the GitOps tool. Also explains the licenses of this tool and other projects used to build it.

Chapter 5. Design

Presents a closer look into the internals of the application and how the modules interact between them to get the desired behavior.

Chapter 6. Validation.

Depicts an explanation of the external tools, the testing methodology and the different tests performed.

Chapter 7. Use case.

Provides some insights on how a development team could use the resulting program .

Chapter 8. Project Plan.

Delivers an overview of the software development methodology used and planning done. Also a breakdown of the costs into different categories.

Chapter 9. Conclusions and future work.

Includes the culmination of this document and further improvements on the project.

CHAPTER 2

State of the Art

This chapter will provide an overview on the technologies and principles used to develop this project. In order to do this some websites, publications, and videos will be analyzed.

In the first two sections I cover a review of Kubernetes and the virtualization methods that uses. In the third one basic concepts of the Git VCS are explained because this project relies heavily on it. Lastly, the principles of continuous deployment, GitOps and the DevOps philosophy are exposed, which this project helps to implement.

2.1 OS-level virtualization and containers

OS-level virtualization enables to create isolated user-spaces that release on the host's kernel. From the point of view of a program running inside the guest OS, it can only see its contents and devices assigned to it, whereas the host can see everything on it.

These instances are invoked in different ways depending on their features and the Unix-like operating system they are used in. For instance, Linux users use Containers, FreeBSD uses Jails and Solaris ones Zones. The most basic way of OS-level virtualization is chroot, which is available in all Unix-like operating systems, only changing the apparent root directory for the current running process and its children. [9]

2.1.1 Containers

Containers are the one of most used solutions of OS-level virtualization in Linux. There are several implementations available like Docker, Podman, LXC and LXD. To accomplish the desired isolation, they rely on the Linux kernel, specially in they use cgroups and Linux namespaces.

LXC and LXD provide a containerized OS, while Docker and Podman are used for containerizing single applications. Also containers regardless its type can be, and normally are, deployed inside virtual machines.

- **Pros:** Less overhead compared to traditional VMs. By using the same kernel as the guest OS they are the same as a regular program in terms of performance but with more isolation. Also they have faster boot time, specially when containerizing single applications. [10]

- **Cons:** Storage persistence and snapshots. Traditional virtual machines have more mechanisms to keep the machine running and the data in it safe, like moving a VM from one host to another and taking snapshots. Containers are more ephemeral. This by itself is a con but if the containers are made stateless they can be really useful for deployments that scale horizontally.

2.2 Kubernetes

Kubernetes, also known as K8s, is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation [3]. With a system like this containers can be used in production environments.

Kubernetes does provide with [11]:

- **Service discovery and load balancing:** It provides an live inventory of the owned resources and manages the name resolution (DNS).

- **Storage orchestration:** It can mount multiple storage systems.

- **Automated rollouts and rollbacks:** The desired state of the system is specified in a declarative way and can be configured to change according to the load. It is able to do rollouts of new versions and rollbacks in a way to prevent any downtime.

- **Automatic bin packing:** Containers can be assigned RAM and CPU limits.

- **Self-healing:** Health checks can be defined to know when to replace containers. Also readiness checks are provided to not expose them to the client until they are ready to serve.

- **Secret and configuration management:** Sensitive information like container registry keys, SSH keys, and passwords can be stored and managed within k8s.

Kubernetes components can be seen in Figure 2-1.

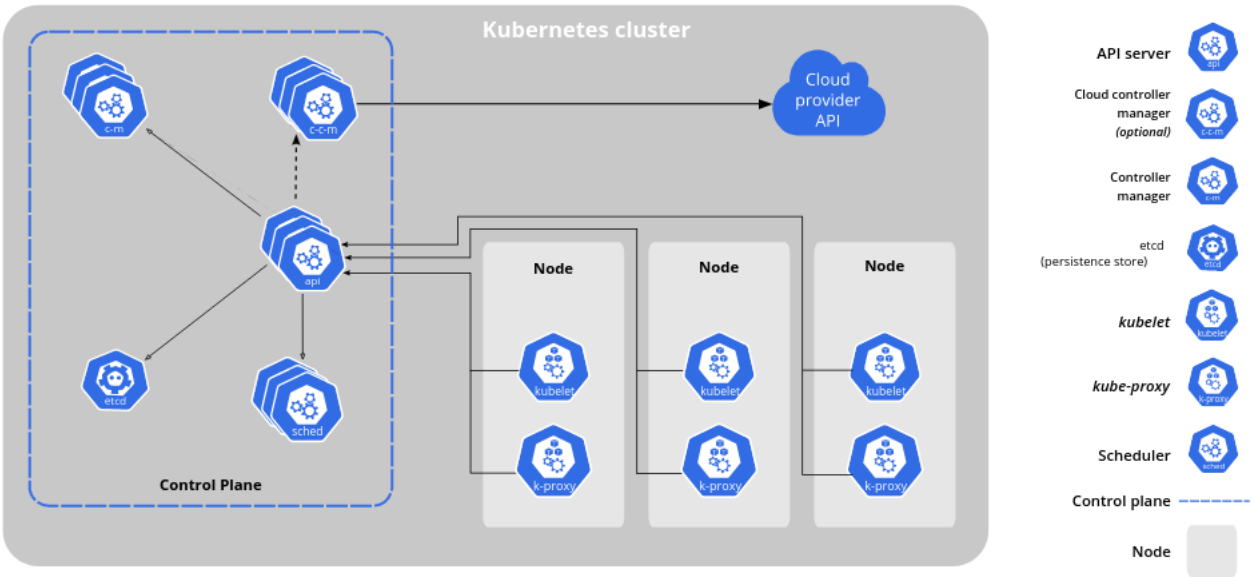


Figure 2-1: *K8s Components [12].*

2.3 Git

Git is a free and open-source distributed VCS (Version Control System) developed by Linus Torvalds in 2005 for development of the Linux kernel. It is normally used by programmers to develop source code in a collaborative way [13].

In order to save the current state of the code repository, a commit is made, in Git a commit is not stored as the difference with the previous version, but as an snapshot. Also to save space unaltered files are not stored again. Also it is distributed unlike other VCS like Apache Subversion, and developers perform operations against their local copies and not a central server. Those local copies are syncronized to and from a server called the origin when needed. It also supports branching, as a nearly every VCS, this means that lines of development can be split to not conflict with each other and merge them when it is appropriate. There are many

branching strategies to have an order when developing, Gitflow is a popular one and can be seen in Figure 2-2. When using Git servers like GitHub or Bitbucket developers who want to merge their branch into another can open something called a pull request to show the difference between them to other developers who will then approve it or not.

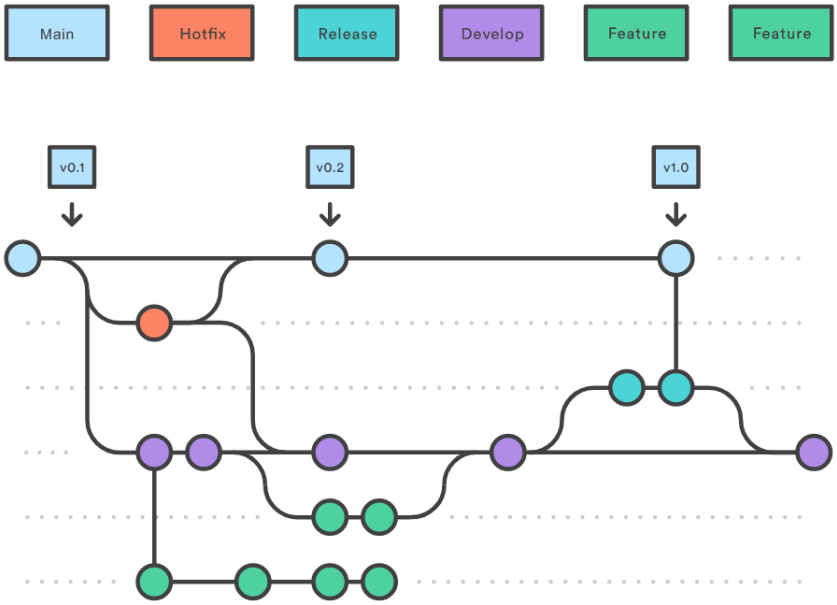


Figure 2-2: *Gitflow branching strategy [14].*

2.4 Continuous deployment, GitOps and the DevOps philosophy

2.4.1 Continuous deployment

Continuous deployment is the practice to deploy new versions of the code automatically. It aims to increase de frequency of deployments and remove manual error prone operations. This reduces the waste of resources in processes that involve many teams and eliminates the fear from releasing new features. [5]

When using continuous deployment, and any other operations model, rollback mechanisms should be present in case there are problems.

2.4.2 GitOps

GitOps is a model defined by WeaveWorks to manage continuous deployment taking advantage of declarative infrastructure defined as code stored in Git. What is reflected in the repository will be rolled to the servers automatically. Also each branch in the Git repository can be

matched with an environment. It is normally used in Kubernetes environments because it fits well with the declarative declaration of their state, but is not limited to them [15].

GitOps principles: [16]

- The entire system described declaratively,
- The canonical desired system state versioned in Git,
- Approved changes that can be automatically applied to the system, and
- Software agents to ensure correctness and alert on divergence.

Currently, GitOps does not need to use Git in order to work, it can use other VCSs but it was called in that manner because Git is the most used one.

2.4.3 DevOps philosophy

GitOps [2] helps to implement the DevOps philosophy in companies, which focuses on combining development and operations to reduce friction and fasten time to market. In this way developers can be the owners for their features across the whole SDLC (Software Development Life Cycle).

CHAPTER 3

System Overview

This chapter introduces the tool developed in this project, namely *Soup* and the background that has led to it. Later, Section 3.2 illustrates the functionality, how it works to accomplish it, a brief showcase of the modules which compose the program, and its operational environment.

3.1 Background

As descused in Chapter 2, GitOps is a model defined by WeaveWorks to manage continuous deployment, taking advantage of declarative infrastructure defined as code stored in Git. GitOps is usually employed in Kubernetes-based clusters and there are some options such as ArgoCD, Flux and Jenkins X. All these projects are quite complex and often cover too much. For instance, Jenkins X also does continuous integration and testing, which is a GitOps anti-pattern. [17]

This project aims to build a tool that does one thing well and in a simple way, following the Unix philosophy [8]. In this project, we target at GitOps, in terms of continuous deployment based on information stored in Git repositories.

So, for building this tool we will focus on the Git repository and the Kubernetes API to cope this problem.

3.2 Description

The solution developed for this project is called *Soup* and is available as free and open-source software in GitHub[1]. *Soup* is developed in Go programming language and contains everything needed to build it and install it in a cluster. In this way, the program is more useful to the community and can be improved by other contributors.

This solution enables the automatical deployment by using k8s manifests stored in Git repositories to a designated k8s-based cluster. Also, it can deploy each branch in an specified namespace. Kubernetes namespaces are used to separate resources in different isolated environments. Thus, names of resources need to be unique within a namespace, but not across namespaces.

The system is divided in three submodules, which will be detailed with much more detail in the Design Chapter:

- **Git module:** It is in charge of cloning the repository, retrieving the existing branches and checking out to them.
- **Branch configuration processing:** It retrieves the configuration file of the given branch, parses it and applies some logic to know what needs to be done. This configuration is placed in yaml files called `.soup.yml`.
- **Deployment module:** This module is in charge of making the actual changes over the Kubernetes cluster. It uses the Go Kubernetes API client in order to do it.

In order to simplify the deployment, a Kubernetes API functionality named "Server-Side Apply" is used. This enables kubernetes clients to send the manifests for it to being applied in the server side, most of the logic does not need to be in the client. Because of this the complexity of the deployment module can be low while providing everything needed.

Also the program is designed to run containerized in a k8s cluster. The diagram in Figure 3-1 provides an overview of how the system interacts both with Git and Kubernetes. It is compiled and the resulting binary is set to be executed with the container starting command. It is designed to run within the cluster because it takes the configuration needed to do the deployments from it. However, "Soup" could also run outside the cluster by adding some functionality to be able to choose whether the cluster configuration is retrieved from the cluster or contained in a file.

[1] `https://github.com/caldito/soup.git`

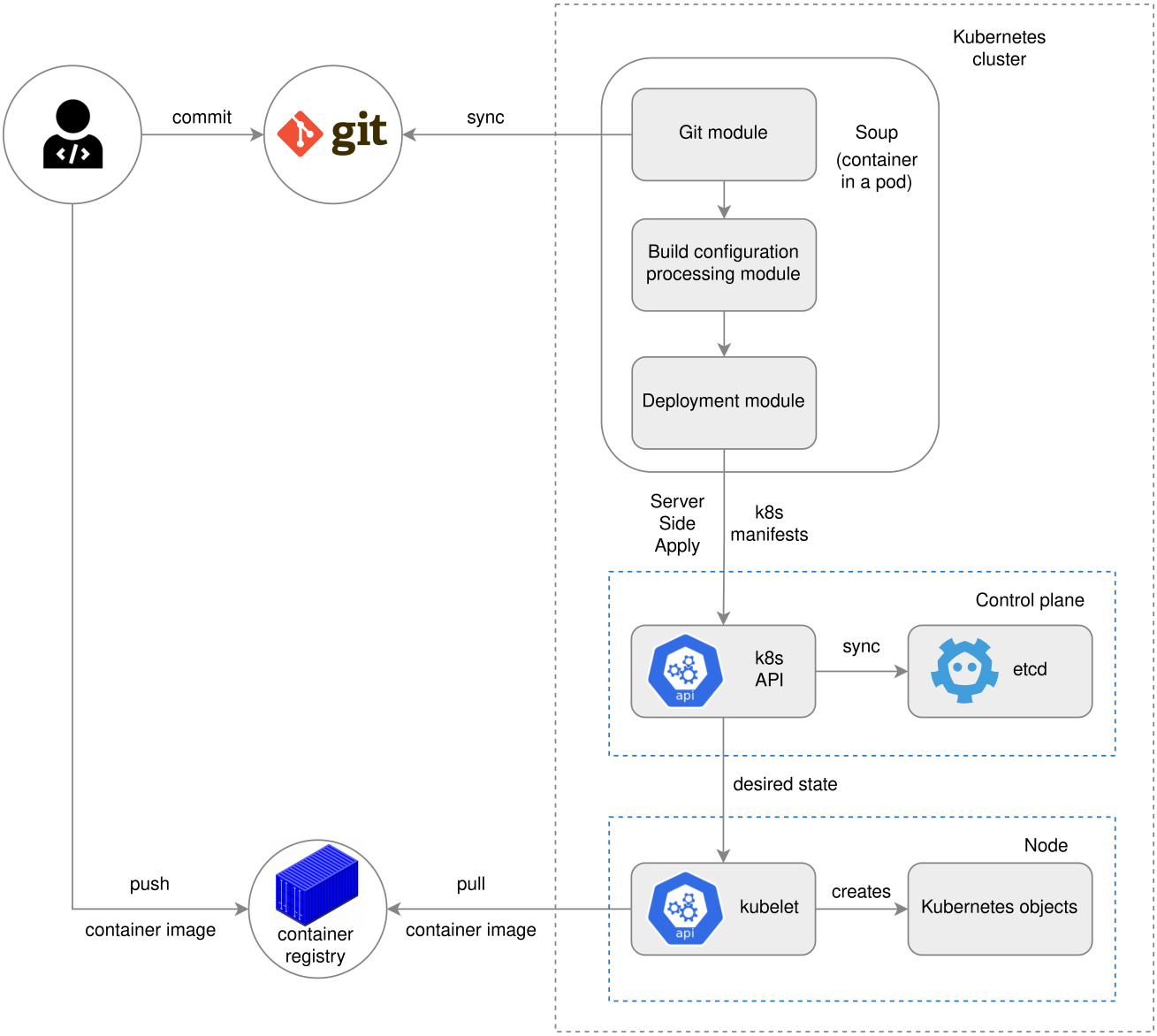


Figure 3-1: *System overview diagram.*

CHAPTER 4

Analysis

This chapter depicts the analysis followed prior to the construction of this project. Includes the requirement analysis and licenses regarding this work and other tools used to build it.

4.1 Requirement Analysis

This section is dedicated to gather all the requirements needed for the desired functioning of the tool. They will define its behavior by determining its features and restrictions. The requirement analysis will be performed using the template table below.

ID		**Name**			
Importance		**Preference**		**Flexibility**	
Testability		**Origin**		**Dependencies**	
Description					
Achievement criteria					

Table 4.1: *Template for requirement analysis.*

- **ID:** Identifier corresponding to the requirement. Its format is <TYPE>R-XX, where <TYPE>

can be **F** for functional or **NF** for non-functional and XX for a two digit number.

- **Name:** Name given to the requirement.
- **Importance:** Degree of importance, it can take the values **Fundamental**, **Highly Useful** or **Useful**.
- **Preference:** Determines the order in which this requirements should be fulfilled. It can be **High**, **Medium** or **Low**.
- **Flexibility:** Determines if a requirement can be changed along the project's development (**Yes**) or not (**No**).
- **Testability:** Determines how easy can be tested, its possible values are **High**, **Medium** or **Low**.
- **Origin:** Stands for the origin of the requirement, its values can be **Client** or **Engineer**.
- **Dependencies:** Dependencies on other requirements.
- **Description:** Short explanation of the requirement.
- **Achievement criteria:** Test case that must be passed to determine the requirement is achieved.

Requirements can be functional or non-functional. Functional ones define what the system should do and the non-functional give details about how it will do those functions.

4.1.1 Functional Requirements

ID	FR-01	**Name**	Repository input parameter		
Importance	Fundamental	**Preference**	High	**Flexibility**	No
Testability	High	**Origin**	Client	**Dependencies**	
Description	The application must receive an input parameter called "repo" to choose the repository to perform the deployment from.				
Achievement criteria	Performs the operations using the indicated repository in the "repo" parameter.				

Table 4.2: *FR-01*

ID	FR-02	**Name**	Interval input parameter		
Importance	Useful	**Preference**	Medium	**Flexibility**	No
Testability	High	**Origin**	Client	**Dependencies**	
Description	The application must accept an input parameter called “interval” the amount of seconds that sleeps between executions. This should be an optional parameter.				
Achievement criteria	The program sleeps the received value in the the “interval” parameter in seconds.				

Table 4.3: *FR-02*

ID	FR-03	**Name**	Default interval value		
Importance	Fundamental	**Preference**	High	**Flexibility**	No
Testability	High	**Origin**	Client	**Dependencies**	FR-02
Description	The default interval between executions will be 120 seconds.				
Achievement criteria	The program will sleep 120 seconds when the "interval" parameter is not provided to the program .				

Table 4.4: *FR-03*

ID	FR-04	**Name**	Configuration file in repository		
Importance	Fundamental	**Preference**	High	**Flexibility**	No
Testability	High	**Origin**	Client	**Dependencies**	
Description	The program will check all the branches in the Git repository and read a configuration file called ".soup.yml" to know where and what to deploy. This file stores data in yaml format.				
Achievement criteria	The program uses the ".soup.yml" to know what to do when iterating over the different branches.				

Table 4.5: *FR-04*

ID	FR-05	**Name**	Nonexistent configuration file in repository		
Importance	Fundamental	**Preference**	High	**Flexibility**	No
Testability	High	**Origin**	Client	**Dependencies**	FR-02
Description	If in a branch of the repository does not exist the ".soup.yml" file the program will not perform any operations in that branch.				
Achievement criteria	The program runs correctly when using a repo which has a branch without ".soup.yml", it just prints a warning.				

Table 4.6: *FR-05*

ID	FR-06	**Name**	Namespaces in configuration file		
Importance	Fundamental	**Preference**	High	**Flexibility**	No
Testability	High	**Origin**	Client	**Dependencies**	FR-04
Description	The ".soup.yml has an array called "namespaces" which has objects with the fields "namespace" and "branch". This is to associate Git branches and k8s namespaces in which to deploy.				
Achievement criteria	The program deploys to the namespace that matches each branch				

Table 4.7: *FR-06*

ID	FR-07	**Name**	k8s manifests in configuration file		
Importance	Fundamental	**Preference**	High	**Flexibility**	No
Testability	High	**Origin**	Client	**Dependencies**	FR-04
Description	The ".soup.yml" has an array called "manifests" which indicates with manifests to deploy to the k8s cluster.				
Achievement criteria	The program applies the specified k8s manifests.				

Table 4.8: *FR-07*

ID	FR-08	**Name**	as-branch namespace in configuration file		
Importance	Highly Useful	**Preference**	High	**Flexibility**	No
Testability	High	**Origin**	Client	**Dependencies**	FR-04
Description	If the name of the "namespace" field is "as-branch" it deploys to a namespace called the same name as the "branch" field, changing namespace invalid characters to dashes.				
Achievement criteria	The program creates a new namespace called as the branch and deploys to it when "as-branch" is the namespace assigned to that branch.				

Table 4.9: *FR-08*

ID	FR-09	**Name**	branch selected with regular expression		
Importance	Highly Useful	**Preference**	High	**Flexibility**	No
Testability	High	**Origin**	Client	**Dependencies**	FR-04
Description	The ".soup.yml" can contain regular expressions to match branch names.				
Achievement criteria	The program works as expected using regular expressions with the branch names.				

Table 4.10: *FR-09*

ID	FR-10	**Name**	Execution in kubernetes cluster		
Importance	Fundamental	**Preference**	High	**Flexibility**	No
Testability	High	**Origin**	Client	**Dependencies**	
Description	The program is designed to be run as containerized and in a Kubernetes cluster.				
Achievement criteria	The program runs in a kubernetes cluster and it takes the credentials needed to deploy from it.				

Table 4.11: *FR-10*

4.1.2 Non-functional Requirements

ID	NFR-01	**Name**	Linux OS		
Importance	Fundamental	**Preference**	High	**Flexibility**	No
Testability	High	**Origin**	Client	**Dependencies**	
Description	The system must run on Linux.				
Achievement criteria	The result and behaviour of the execution is as expected under Linux.				

Table 4.12: *NFR-01*

ID	NFR-02	**Name**	install.yml		
Importance	Fundamental	**Preference**	High	**Flexibility**	No
Testability	High	**Origin**	Client	**Dependencies**	
Description	A k8s manifest should be provided that eploys the containerized program to the cluster.				
Achievement criteria	The install.yml file deploys the containerized program to the cluster as expected.				

Table 4.13: *NFR-02*

ID	NFR-03	**Name**	Make		
Importance	Fundamental	**Preference**	High	**Flexibility**	No
Testability	High	**Origin**	Client	**Dependencies**	
Description	The application must have a the Makefile used for multiple purposes.				
Achievement criteria	There is a file named Makefile with multiple targets.				

Table 4.14: *NFR-03*

ID	NFR-04	**Name**	Make compilation		
Importance	Fundamental	**Preference**	High	**Flexibility**	No
Testability	High	**Origin**	Client	**Dependencies**	NFR-03
Description	The Makefile must have a target for building the program.				
Achievement criteria	The Makefile has a target that builds the application as expected.				

Table 4.15: *NFR-04*

ID	NFR-05	**Name**	Make docker		
Importance	Fundamental	**Preference**	High	**Flexibility**	No
Testability	High	**Origin**	Client	**Dependencies**	NFR-03
Description	The Makefile must have a target for building the container with docker.				
Achievement criteria	The Makefile has a target that builds the container with docker as expected.				

Table 4.16: *NFR-05*

ID	NFR-06	**Name**	Make podman		
Importance	Fundamental	**Preference**	High	**Flexibility**	No
Testability	High	**Origin**	Client	**Dependencies**	NFR-03
Description	The Makefile must have a target for building the container with podman.				
Achievement criteria	The Makefile has a target that builds the container with docker as expected.				

Table 4.17: *NFR-06*

ID	NFR-07	**Name**	Make dependencies		
Importance	Useful	**Preference**	High	**Flexibility**	No
Testability	High	**Origin**	Client	**Dependencies**	NFR-03
Description	The Makefile must have a target for managing the external dependencies.				
Achievement criteria	The Makefile has a target that reflects the needed dependencies in the go.mod and go.sum files.				

Table 4.18: *NFR-07*

ID	NFR-08	**Name**	Make format		
Importance	Useful	**Preference**	High	**Flexibility**	No
Testability	High	**Origin**	Client	**Dependencies**	NFR-03
Description	The Makefile must have a target for formatting the source code according to Go standards.				
Achievement criteria	The Makefile has a target that formats the source code.				

Table 4.19: *NFR-08*

ID	NFR-09	**Name**	Logging		
Importance	Fundamental	**Preference**	Medium	**Flexibility**	No
Testability	High	**Origin**	Client	**Dependencies**	
Description	The application must log information about what happens in the program.				
Achievement criteria	The log gives an understanding on what happens in the program.				

Table 4.20: *NFR-09*

ID	NFR-10	**Name**	Error Information		
Importance	Fundamental	**Preference**	Medium	**Flexibility**	No
Testability	High	**Origin**	Client	**Dependencies**	
Description	The application must log errors and fatal errors.				
Achievement criteria	A failed execution shows the error.				

Table 4.21: *NFR-10*

ID	NFR-11	**Name**	Go version		
Importance	Highly Useful	**Preference**	High	**Flexibility**	No
Testability	High	**Origin**	Client	**Dependencies**	
Description	The version of Go that must be used in the application is 1.16.				
Achievement criteria	The result of the execution is the expected when using using Go 1.16 for compilation.				

Table 4.22: *NFR-11*

ID	NFR-12	**Name**	Kubernetes version		
Importance	Fundamental	**Preference**	High	**Flexibility**	No
Testability	High	**Origin**	Client	**Dependencies**	
Description	The application must work as expected when deployed in kubernetes v1.20 and higher.				
Achievement criteria	The execution of the application works as expected when deployed in kubernetes v1.20 and higher.				

Table 4.23: *NFR-12*

ID	NFR-13	**Name**	K8s Go API for deploy function		
Importance	Fundamental	**Preference**	Medium	**Flexibility**	No
Testability	High	**Origin**	Client	**Dependencies**	
Description	The system should use the k8s Go API to make the deployments.				
Achievement criteria	The system performs the deployments by using the k8s Go API.				

Table 4.24: *NFR-13*

ID	NFR-14	**Name**	Deploy function as a library		
Importance	Useful	**Preference**	Medium	**Flexibility**	No
Testability	High	**Origin**	Client	**Dependencies**	
Description	The system should use the deploy function as a library and it could be used in other projects.				
Achievement criteria	The deploy function in in the `pkg/` directory so that can be used in other projects when this one is open-sourced.				

Table 4.25: *NFR-14*

4.2 Licenses

This section shows the legal aspects concerning this project. Starting with the license used for the tool and then the licenses from the principal dependencies and tools used during the development of this project.

4.2.1 Tool license

The tool is publicly available[1] and licensed under the Apache License 2.0[2]. It is a permissive license whose main conditions require preservation of copyright and license notices. Contributors provide an express grant of patent rights. Licensed works, modifications, and larger works may be distributed under different terms and without source code. [18]

4.2.2 Licenses of used projects

The projects used during the development of this work are licensed in the following way:

- **Go programming language** uses a custom permissive license which can be found in its repository [3].
- **Kubernetes Go client** uses the Apache License 2.0 which can be seen in its repository[4].
- **Go yaml** is under the Apache License 2.0, its details can be seen in its repository[5].
- **go-git** uses the Apache License 2.0, its details can be seen in its repository[6].

[1] `https://github.com/caldito/soup.git`
[2] `https://github.com/caldito/soup/blob/main/LICENSE`
[3] `https://github.com/golang/go/blob/master/LICENSE`
[4] `https://github.com/kubernetes/client-go/blob/master/LICENSE`
[5] `https://github.com/go-yaml/yaml/blob/v2/LICENSE`
[6] `https://github.com/go-git/go-git`

- **GNU Make** uses the GNU General Public License v3, its details can be consulted on its webpage[7].
- **Docker Community Edition** uses the Apache License 2.0, its details can be seen in its repository[8].
- **Podman** uses the Apache License 2.0, its details can be seen in its repository[9].

[7] `https://www.gnu.org/licenses/gpl-3.0.html`
[8] `https://github.com/docker/docker-ce/blob/master/LICENSE`
[9] `https://github.com/containers/podman/blob/main/LICENSE`

CHAPTER 5

Design

This chapter shows a more detailed explanation on the internals of the tool built. First it explains how this tool is designed to run as a daemon process, then a explicit description of each module and finally how they integrate all together to obtain the desired system.

5.1 Daemon process

This application is designed to run as a daemon process. This means that executed one time and is designed to run in the background forever or until it is stopped.

It reads the parameters at launch and then, it executes the rest of the program in a loop. It does sleep a certain amount of time after finishing an iteration.

```
init() // reads parameters
while(1) { // this loop is performed by the main() function, which is run after init()
    run()
    sleep(interval)
}
```

Listing 5.1: *Program daemon structure pseudocode.*

5.2 Module Description

The system is composed by modules to facilitate extensibility and maintainability in the future. This section provides an explanation on the four different modules found in this tool.

5.2.1 Parameter reading module

First of all the tool needs to receive some parameters in order to work. This module is in charge of receiving them. There are two of them:

- **Repository parameter**. This parameter expects a link to the Git repository which to perform GitOps. The name of this parameter is called `repo`. It is mandatory because there is no way the program can accomplish its goal without it.
- **Interval parameter**. This parameter expects an integer which is the number of seconds between executions. The name of this parameter is called `interval`. It is not required because if not provided a default value of 120 seconds is used.

This is performed in the `init()` function, as can be seen in Section 5.1. It is executed before anything and therefore is outside the main loop, this means it is executed just a single time. It stores the received values into a struct of type `ProgramConf` with global visibility so that it contains the program configuration and can be accessed from anywhere in the program.

```
type ProgramConf struct {
    Repo string
    Interval int
}
```

Listing 5.2: *ProgramConf struct.*

Also, parameters are received with the Go `flag` library and input should be in the following form so that they can be recognized correctly: `-parameter-name=parameter-value`.

5.2.2 Git module

This module permits to perform the necessary operations against the given Git repository. To cope with this, the `go-git` library is used, providing native Git implementation in Go. [1]

[1] `https://github.com/go-git/go-git`

When an iteration of the main loop starts, it clones the repository in a directory inside `/tmp/soup`. This means that a copy of the repository is stored in there, with all its branches and commits. Then, it retrieves all the branch names available in the remote with the function `getBranchNames`, which receives the cloned repository as parameter. It returns a string array with the names of the branches. Using that string array then it loops across the branch names and switches to them, also called checking out. After doing so to a to a given branch calls other modules to do the pertinent processing on it in order to accomplish GitOps.

Lastly, when the iteration finishes it removes the cloned repository to not occupy unnecessary space on the filesystem.

5.2.3 Build configuration processing module

In order to know what needs to be deployed and in which namespace of the cluster, in the repository is stored a file called `.soup.yml` with information related to that, there is an example of it in listing 5.3. It is written in YAML [19], a human-readable data-serialization language. It is very simple to read and write and uses identation to indicate nesting.

```yaml
---
namespaces:
  - namespace: "production"
    branch: "main"
  - namespace: "staging"
    branch: "develop"
  - namespace: "as-branch"
    branch: "features/*"
manifests:
  - "deployment.yml"
  - "service.yml"
```

Listing 5.3: *.soup.yml example.*

This file contains an array called `namespaces` that has objects with the attributes `namespace` and `branch`, which are both strings. The `namespaces` array is used to know where to deploy each branch. The `.soup.yml` file also contains an array of strings named `manifests`, which should contain the paths of the files to deploy. This file can be different across branches but that is not a problem because it searches for the branch the program is currently iterating on the `namespaces` array. It contains an array with info of what to do on different branches because is very impractical for this file to be different in every branch because branches merge together

frequently, so this configuration file tends to be the same across the different branches.

This module is in charge of processing it and know exactly which manifests should be deployed and in which namespace. In order to know that two functions are used:

- **getBuildConf():** This function reads the `.soup.yml` file and unmarshalls it to a BuildConf struct which can be seen in listing 5.4 which returns.
- **getNamespace():** Takes as parameters a *BuildConf* struct and the name of the branch and applies some logic to it to know in which namespace the manifests should be deployed.

```
type BuildConf struct {
        Namespaces []Namespace
        Manifests []string
}
```

Listing 5.4: *BuildConf struct.*

This module also contains two additional features, which provided by the `getNamespace()` function:

- **Branch name regular expressions:** Regular expressions are accepted to match with branch names.
- **Namespace as-branch:** if the namespace assigned to the current branch is `as-branch` it will deploy to a namespace called in the same way as the current branch, but changing "/" by "-" because slashes can not be in namespace names.

In the example provided in listing 5.3 the following will happen depending the branch the program is iterating on:

- **main:** Will deploy the specified manifests, **service.yml** and **deployment.yml**, to the `production` namespace.
- **develop:** Will deploy the specified manifests to the `production` namespace.
- **features/3:** Will deploy the specified manifests to the `features-3` namespace because the namespace is `as-branch` and matches the regular expression `features/*`.
- **bugs/4:** Will not deploy anything because it does not match any branch name.

5.2.4 Deployment module

This module is in charge of interacting with the kubernetes cluster to do the deployment. It consists in 3 functions:

- **deploy():** This function ties together the deployment module. Is in charge of calling the two functions below. Calls `DeclareNamespaceSSA()` a single time to ensure that the namespace exists and then it calls `DoSSA()` one time per manifest to deploy. It receives as parameters the target namespace and an array of manifests.
- **DeclareNamespaceSSA():** This method creates a manifest file for declaring the namespace, uses the DoSSA() method to create it and then it deletes the manifest file. This is called before invoking the `DoSSA()` function with other manifests to ensure that the target namespace exists.
- **DoSSA():** This is actually the core of the deployment module. It decodes a file to an `Unstructured` [2] struct, provided by the Go k8s client. It allows objects that do not have Golang structs registered to be manipulated generically. Then this is marshaled to JSON and sent to k8s to perform SSA (Server-Side Apply)

Server-Side Apply [20] helps users and controllers manage their resources through declarative configurations. Clients can create and modify their objects declaratively by sending their fully specified intent. A fully specified intent is a partial object that only includes the fields and values for which the user has an opinion. That intent either creates a new object or is combined, by the server, with the existing object.

5.3 Integration

For this tool to be useful, all the different modules have to work together to pursue the same objective. The modules are integrated as seen in Figure 5-1 to achieve this.

[2] `https://pkg.go.dev/k8s.io/apimachinery/pkg/apis/meta/v1/unstructured`

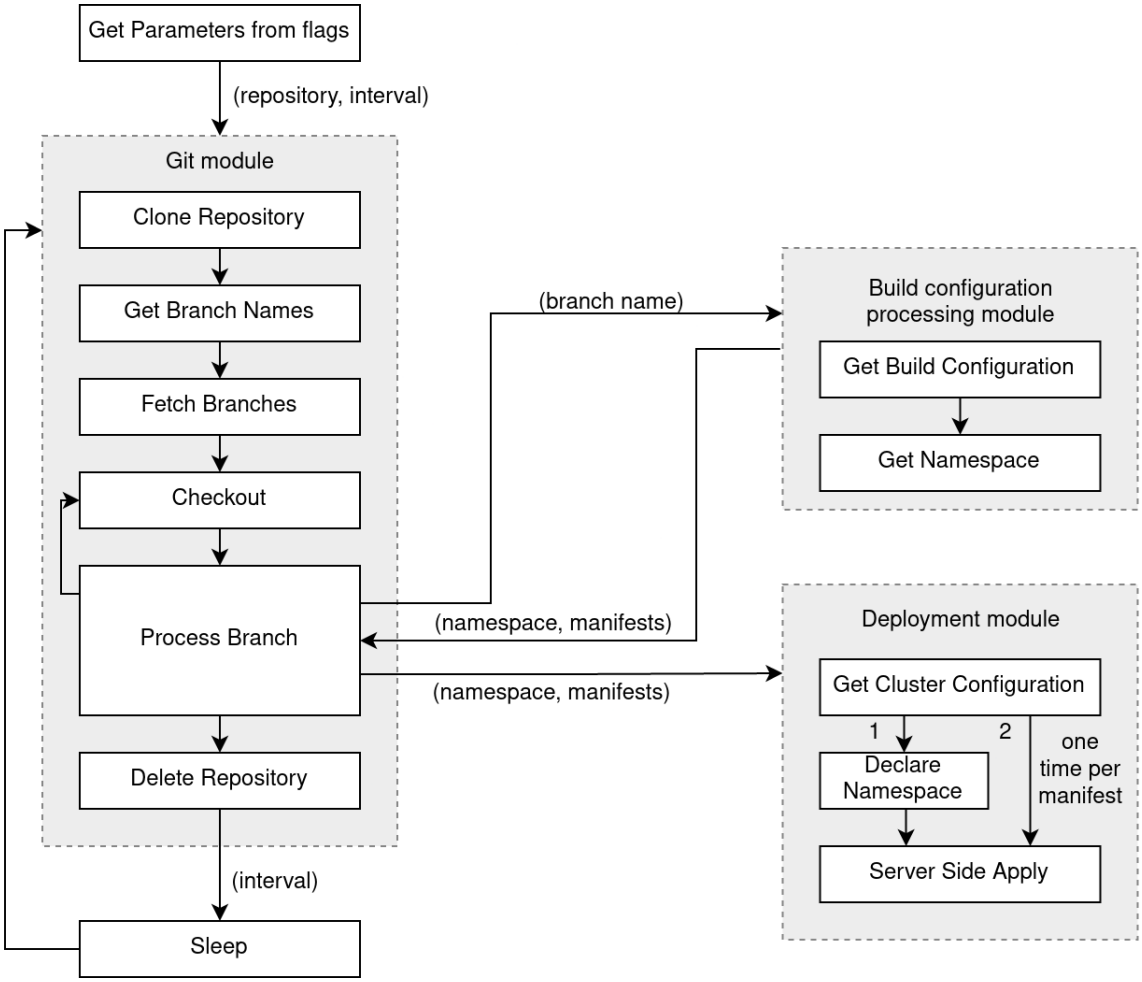


Figure 5-1: *Integration diagram.*

CHAPTER 6

Validation

This chapter shows the validation side of the project. First explains the external tools used for performing it, then the testing methology and lastly the set of tests used

6.1 External Tools

"Kubectl" is a command line tool that allows to control Kubernetes clusters, it is the only external tool used for testing the correct functioning of the program [21]. It communicates with the Kubernetes cluster's API via the command line, therefore allows to install resources, and inspect how they perform and the K8s cluster state. This tool is officially provided by Kubernetes and widely used for developers who interact with it.

6.2 Testing Methodology

To perform the tests "Soup" was installed using the `install.yml` file provided in the repository with `kubectl apply`. When installing "Soup" the `install.yml` file provides the arguments to indicate the repository and execution interval. Then, the program is running in the cluster until is uninstalled using `kubectl delete` with the same installation manifest. When the program is running the state of the program and the resources that creates can be inspected to verify that

everything is working properly.

This end-to-end tests were performed since the installation manifest was ready. End-to-end tests, also called UI tests, verify that all the components of the system work together as expected. Also integration tests were performed in the "Git module" and "Build configuration processing modules". Integration tests verify that a module works as expected. This integration tests could run outside the cluster in the developer computer because did not need to interact with Kubernetes.

However, all this tests are manual which makes them time consuming. A further improvement is to have a automated tests [22] which are faster and more reliable, and also include unit tests. Unit tests validate functions in an isolated way and are highly recommended. Figure 6-1 shows the typical way of structuring an automated test plan among the different type of tests.

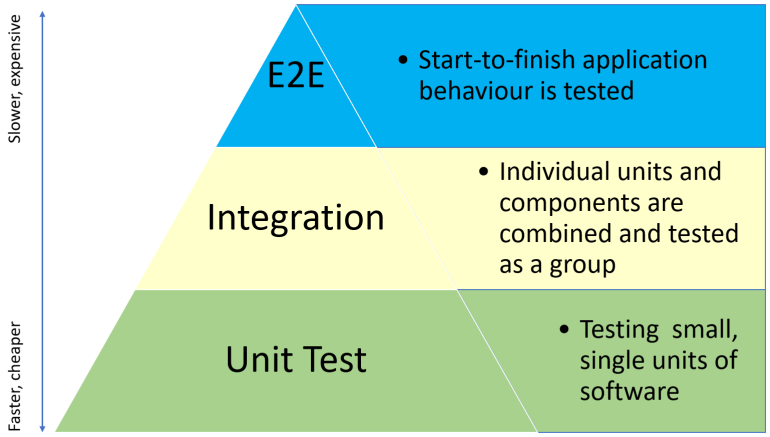


Figure 6-1: *[Testing pyramid]Testing pyramid [23]*

6.3 Test Performed

This sections shows all the test designed and performed to validate the system, they will be described using the following template:

Test ID		**Name**	
Related Requirements			
Description			

Table 6.1: *Validation test template.*

- **Test ID:** Unique identifier for the test.
- **Name:** Name given to the test.
- **Related requirements:** Requirements verified by the test.
- **Description:** Short description of the test.

Test ID	T-01	**Name**	Repo parameter existent
Related Requirements	FR-01		
Description	The program works as expected receiving the “repo" parameter.		

Table 6.2: *T-01*

Test ID	T-02	**Name**	Repo parameter nonexistent
Related Requirements	FR-01		
Description	The program fails as expected when the “repo" parameter is not received.		

Table 6.3: *T-02*

Test ID	T-03	**Name**	Interval parameter
Related Requirements	FR-02		
Description	The program sleeps the number of seconds received in the “interval” parameter between executions.		

Table 6.4: *T-03*

Test ID	T-04	**Name**	Default interval parameter
Related Requirements	FR-03		
Description	The program sleeps 120 seconds when the “interval" parameter is not provided to the program.		

Table 6.5: *T-04*

Test ID	T-05	**Name**	End-to-end testing
Related Requirements	FR-04, FR-05, FR-06, FR-07, FR-08, FR-09, FR-10		
Description	The program deploys the manifests to the namespaces as expected when using as input the testing repository[1].		

Table 6.6: *T-05*

Test ID	T-06	**Name**	Operating system
Related Requirements	NFR-01		
Description	The program works as expected in Kubernetes running on linux servers.		

Table 6.7: *T-06*

Test ID	T-07	**Name**	Installation manifest
Related Requirements	NFR-02		
Description	The `install.yml` manifest installs the program and grants to it the permissions it need in a Kubernetes cluster.		

Table 6.8: *T-07*

Test ID	T-08	**Name**	Makefile
Related Requirements	NFR-03, NFR-04, NFR-05, NFR-06, NFR-07, NFR-08		
Description	There is a Makefile which has targets that performs the following tasks as expected: building the program, creating the container Docker and Podman containers, getting the dependencies, formatting the code.		

Table 6.9: *T-08*

Test ID	T-09	**Name**	Log execution information shown
Related Requirements	NFR-09		
Description	The program prints information, warnings and errors about what happens during the execution as expected.		

Table 6.10: *T-09*

Test ID	T-10	**Name**	Fatal errors shown
Related Requirements	NFR-10		
Description	When the program crashes due to a fatal error prints information about what happened.		

Table 6.11: *T-10*

Test ID	T-11	**Name**	Go version
Related Requirements	NFR-11		
Description	The program works as expected when built with Go version 1.16.		

Table 6.12: *T-11*

Test ID	T-12	**Name**	Kubernetes version
Related Requirements	NFR-12, NFR-13		
Description	The program works as expected running in a cluster running Kubernetes version 1.20 and its API.		

Table 6.13: *T-12*

Test ID	T-13	**Name**	Deployment functions used as library
Related Requirements	NFR-14		
Description	The program provides the functions which interact with the K8s API to deploy as a library and can be used in other programs successfully.		

Table 6.14: *T-13*

Tables 6.15 and 6.16 show the traceability between this tests and the requirements defined in Chapter 4.

	FR-01	**FR-02**	**FR-03**	**FR-04**	**FR-05**	**FR-06**	**FR-07**	**FR-08**	**FR-09**	**FR-10**
T-01										
T-02										
T-03										
T-04										
T-05										
T-06										
T-07										
T-08										
T-09										
T-10										
T-11										
T-12										
T-13										

Table 6.15: *Traceability Matrix Tests - Functional Requirements*

	NFR-01	NFR-02	NFR-03	NFR-04	NFR-05	NFR-06	NFR-07	NFR-08	NFR-09	NFR-10	NFR-11	NFR-12	NFR-13	NFR-14
T-01														
T-02														
T-03														
T-04														
T-05														
T-06	■													
T-07		■												
T-08			■	■	■	■	■	■						
T-09									■					
T-10										■				
T-11											■			
T-12												■	■	
T-13														■

Table 6.16: *Traceability Matrix Tests - Non-Functional Requirements*

CHAPTER 7

Use case

Inside this chapter, we will be showing how users can install "Soup" in Kubernetes, the contents the GitOps repository should have as well as two repository organizations approaches, and also two different build configuration file use cases and when to use each of them.

7.1 Installation

Most users will interact first with "Soup" in its repository as is common in Open-source projects. There they can find the system overview diagram of Figure 3-1, how to install it and the features it provides. Then if a development team decides to use "Soup" will proceed to install it.

To install the program a user has to perform this tasks, which are detailed in the `README.md` file of the "Soup" repository:

- Step 1. Download the installation manifest with curl a seen in Listing 7.1
- Step 2. Edit the repo parameter in the manifest with the desired repository URL.
- Step 3. Apply the contents of the manifest to the cluster with `kubectl apply` as seen in Listing 7.2

Finally, the program will be running in the cluster performing GitOps with the given repository.

```
curl -O https://raw.githubusercontent.com/caldito/soup/main/manifests/install.yml
```

Listing 7.1: *Command to download installation manifest.*

```
kubectl apply -f install.yml
```

Listing 7.2: *Command to apply manifest to the cluster.*

7.2 Repository contents

The repository given to the program must have 2 elements:

- The `.soup.yml` file described in Subsection 5.2.3 to know in which namespace and which manifests to deploy.
- Kubernetes manifests that will be deployed.

Also, this repository can contain both the code and the GitOps configuration or just the GitOps configuration.

7.2.1 Build Configuration Files Convergence

As mentioned also in Subsection 5.2.3, the `.soup.yml` file tends to converge between all branches because they merge with each other. Due to this the build configuration file will not be changed often in most cases. Also, the best moment to set it up is in the first commits of the repository when there is only one branch. Some future work is to be able to gather the manifests to deploy from this file using regular expressions, as is done with the branch names. It will reduce even more the changes needed to perform in the build configuration file.

7.2.2 Containing just GitOps configuration

This is the recommended way of doing GitOps pipelines by Weaveworks and can be seen in Figure 7-1. The GitOps configuration is stored in a different repository as the code.

Its changes can be automated more with less effort. Makes easier to differentiate the continuous integration from the continuous deployment process. Good the cluster runs multiple applications with different codebases. However, this can produce too many overhead that exceeds the benefits in small projects.

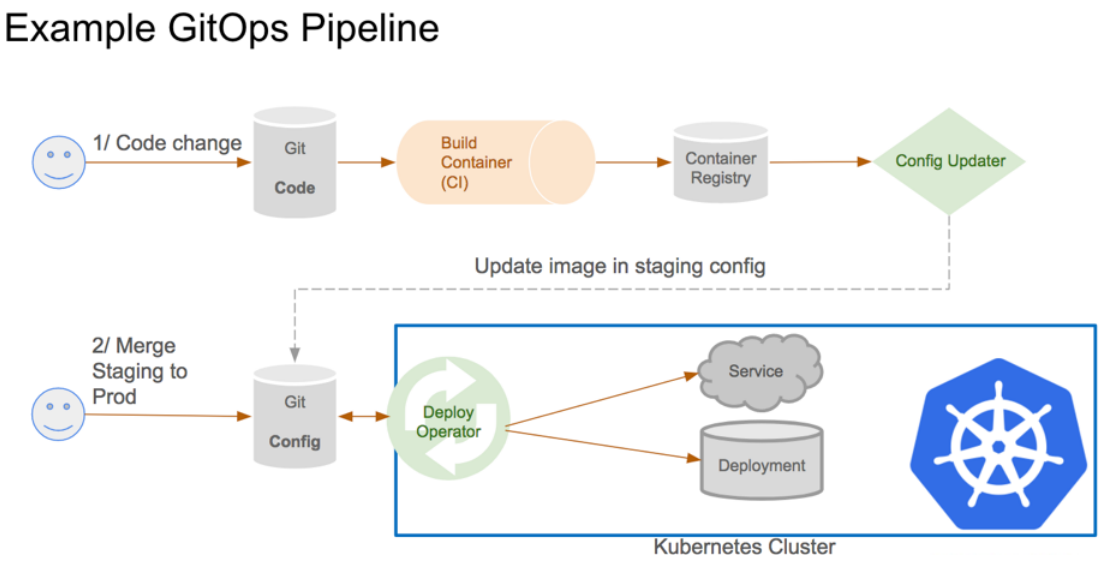


Figure 7-1: *Weaveworks GitOps pipeline [16].*

7.2.3 Containing Code and GitOps configuration

Following this organization the code and the GitOps configuration live together in the same repository. This approach can be better if a single application is deployed to Kubernetes or when the distributed system code is stored in a monorepo [24]. This is not very common, but can be done, for instance when the application is a monolith that would benefit from K8s scalability.

7.3 Deployment approaches

7.3.1 Namespace per branch

To have a very fast feedback loop the best way is to deploy each branch to a different namespace. In this way the infrastructure can be seen in action before merging it to develop and modifying the staging branch. New additions in the infrastructure or version upgrades can be seen in isolated environments. The staging environment gets new features tested in other environments and is meant to be as close as possible to production. If everything looks good in staging the changes can be transferred to production by merging the develop branch into main.

Listing 7.3 shows a build configuration file that allows to deploy the main, develop and feature branches. This are the most used branches in a Gitflow branching strategy[14]. The production namespace will be deployed after the contents in the main branch, the staging one

after the contents of develop and per feature branch a new namespace will be created.

```
namespaces:
  - namespace: "production"
    branch: "main"
  - namespace: "staging"
    branch: "develop"
  - namespace: "as-branch"
    branch: "features/*"
manifests:
  - "deployment.yml"
  - "service.yml"
```

Listing 7.3: *.soup.yml file for namespace per branch.*

Listing 7.4 shows a shorter `.soup.yml` which just deploys the given manifests in a namespace called in the same way as the branch. It is simpler but sometimes you want to have more control because the name of the branch differs from the namespace name.

```
namespaces:
  - namespace: "as-branch"
    branch: "*"
manifests:
  - "deployment.yml"
  - "service.yml"
```

Listing 7.4: *.soup.yml file for namespace per branch shorter.*

As a downside of this approach, is that "soup" does not delete the namespaces created of a branch when it is deleted. This feature is planned as future work because is quite an important feature for this use case. Also, it is planned to allow to choose which namespaces can and which can be deleted, because there are some environments which should be up always like production and staging, also called pre-production.

Another downside which not depends on "Soup" features is the cost to do this approach. If there are a lot of existing branches the RAM and CPU requirements for the cluster is much bigger. It also happens that developers usually forget to close their feature branches after they are merged into develop, increasing the impact of this issue. A way to mitigate it is to delete automatically after a certain amount of time merged feature branches.

7.3.2 Production and staging

A more conservative approach would be to deploy only deploy only the manifests in master and develop, each one to a different namespace. Right now "Soup" adapts really well to this approach because these namespaces are meant to be always up. When iterating over the main branch it will deploy to production namespace and in the develop branch to the staging namespace, however to do this would add more complexity to the system.

Another benefit against the use case approach described in Subsection 7.3.1 is the much cheaper cluster because only two namespaces are used.

```yaml
namespaces:
  - namespace: "production"
    branch: "main"
  - namespace: "staging"
    branch: "develop"
manifests:
  - "deployment.yml"
  - "service.yml"
```

Listing 7.5: *.soup.yml file only for production and staging.*

CHAPTER 8

Project Plan

In this chapter we focus on the planning side of the project. First it focuses on the management of the software development project and the methodology used. Then on a detailed breakdown of the expenses for the costumer.

8.1 Management

This section details the management of this software development project. Explains the methodologies considered and the one used for the it. Then illustrates the different iterations performed and the time consumed by each of them.

8.1.1 Methodology

To organize and plan the project an adequate software development methodology should be used. Four different ones were analyzed [6].:

- **Waterfall Model:** This one is the more traditional approach on engineering. A phase should be completed before continuing to the next one. It is less flexible than the next ones and. A working system can only be obtained at the end, after all phases are completed, so its feedback loop often is not sufficient.

- **Prototype Model:** Consists on building a prototype which will be evaluated and improved in following iterations after more information is gathered after seeing users interact with it. It aims to fasten the time to market and reduce cost.
- **Incremental Model:** This model divides the work into parts which can be built and tested separately. It is good for following the customer requirements but a working system can not be tested until the end.
- **Spiral Model:** This one is close to the incremental model but takes risk analysis very seriously. Each phase is divided in four parts: Planning, Risk Analysis, Engineering and Evaluation.

As this project focuses on building a simple yet useful tool the natural choice is the prototype model. After developing the prototype and using it we will perform some iterations to solve the most important problems and add valuable features. The project evolution following this model can be seen in Figure 8-1.

8.1.2 Planning

This subsection explains the activities performed in each iteration and their extent in time.

- **Iteration 1**: Git and parameter reading modules. The base of the prototype will be the Git module, the branches of the repository are checked out and iterated in a loop.
- **Iteration 2**: Build configuration processing module. Here, the build configuration of each branch is obtained from the `.soup.yml` file. With this module ready the program knows in which namespace to deploy and which Kubernetes manifests.
- **Iteration 3**: Build scripts and installation manifest. This is needed to fasten development and be able to to perform testing inside Kubernetes.
- **Iteration 4**: Deployment module. Finally, the project is able to deploy the manifests to a K8s cluster. With this iteration finished the prototype is done.
- **Iteration 5**: Improve installation manifest. The K8s manifests are improved to grant the necessary permissions to the "Soup" program inside the cluster.
- **Iteration 6**: Improve error handling. In this last iteration, the error handling is improved to not end the program when minor errors occur. Also some refactoring was performed in this iteration.

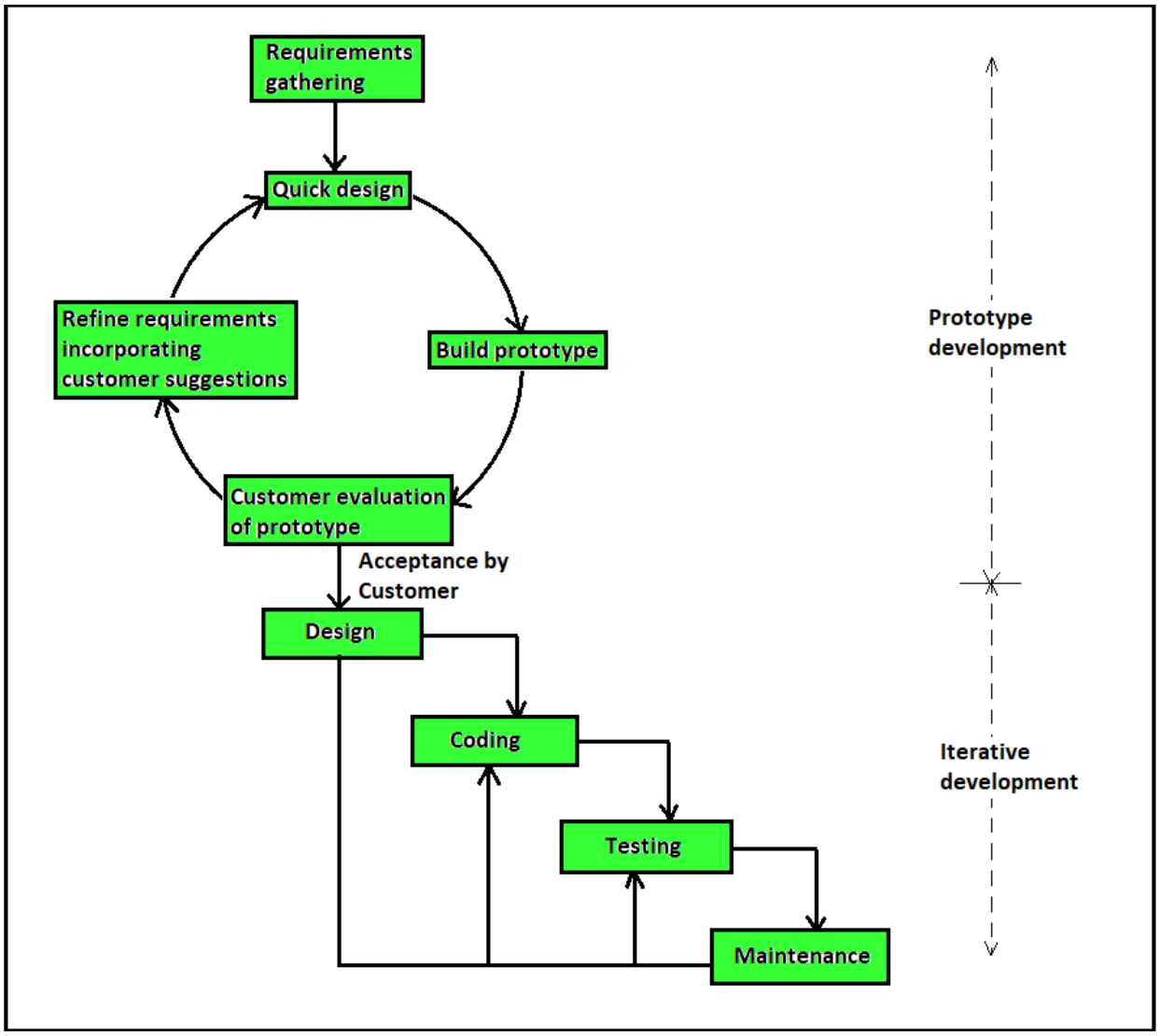


Figure 8-1: *Prototype model phases.[25]*

Figure 8-2 shows the different implementation phases. The red ones refer to building the prototype, the green ones to iterating over it and the blue one is about developer usability.

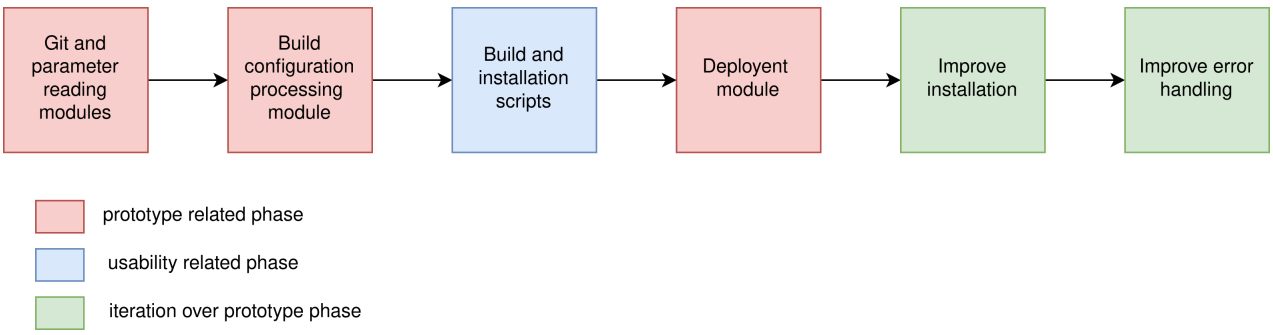


Figure 8-2: *Implementation phases.*

The initial time estimation for completing this work was 7 months, starting on 1 January 2021 and ending on 31 July 2020. However, the project was completed in 8 months, mostly due to some complications during the development of the deployment module. 10 hours of work were dedicated each week, 2 hours a day each workday. Table 8.1 shows the duration of each iteration with its start and end dates. Also, Figure 8-3 shows the initial estimation Gantt chart and Figure 8-4 shows the final Gantt chart detailing the duration and tasks performed during each phase.

Task	Start
Requirements gathering and prototype design	**Fri 1/1/21**
Customer communication	Fri 1/1/21
Requirements and preliminary design	Fri 1/8/21
Git and parameter reading modules	**Fri 2/12/21**
Customer communication and design	Fri 2/12/21
Engineering	Wed 2/17/21
Validation	Fri 3/5/21
Build configuration processing module	**Thu 3/11/21**
Customer communication and design	Thu 3/11/21
Engineering	Mon 3/15/21
Validation	Tue 3/23/21
Build scripts and installation manifest	**Thu 3/25/21**
Customer communication and design	Thu 3/25/21
Engineering	Mon 3/29/21
Validation	Tue 4/6/21
Deployment module	**Thu 4/8/21**
Customer communication and design	Thu 4/8/21
Engineering	Thu 4/15/21
Validation	Thu 5/13/21
Improvement iterations	**Thu 5/27/21**
Engineering	Thu 5/27/21
Validation	Thu 6/10/21
Documentation	**Thu 6/24/21**
Drafting	Thu 6/24/21
Supervision and final report	Thu 7/22/21

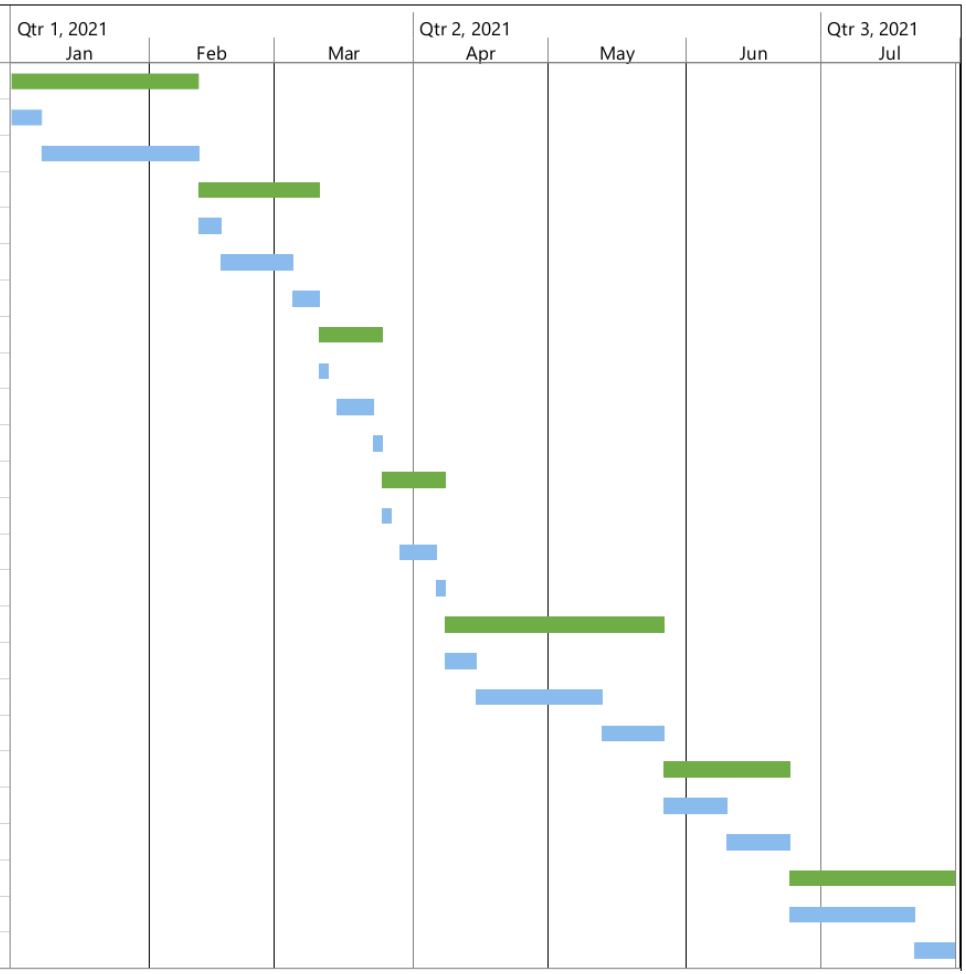


Figure 8-3: *Project's initial estimation Gantt diagram.*

Task	Start	Finish
Requirements gathering and prototype design	**Fri 1/1/21**	**Fri 2/19/21**
Customer communication	Fri 1/1/21	Fri 1/8/21
Requirements and preliminary design	Mon 1/11/21	Fri 2/19/21
Git and parameter reading modules	**Mon 2/22/21**	**Mon 3/15/21**
Customer communication and design	Mon 2/22/21	Wed 2/24/21
Engineering	Thu 2/25/21	Wed 3/10/21
Validation	Thu 3/11/21	Mon 3/15/21
Build configuration processing module	**Tue 3/16/21**	**Fri 3/26/21**
Customer communication and design	Tue 3/16/21	Wed 3/17/21
Engineering	Thu 3/18/21	Wed 3/24/21
Validation	Thu 3/25/21	Fri 3/26/21
Build scripts and installation manifest	**Mon 3/29/21**	**Fri 4/9/21**
Customer communication and design	Mon 3/29/21	Tue 3/30/21
Engineering	Wed 3/31/21	Wed 4/7/21
Validation	Thu 4/8/21	Fri 4/9/21
Deployment module	**Mon 4/12/21**	**Fri 6/18/21**
Customer communication and design	Mon 4/12/21	Fri 4/16/21
Engineering	Mon 4/19/21	Fri 6/4/21
Validation	Mon 6/7/21	Fri 6/18/21
Improve installation manifest	**Mon 6/21/21**	**Wed 6/30/21**
Engineering	Mon 6/21/21	Thu 6/24/21
Validation	Fri 6/25/21	Wed 6/30/21
Improve error handling	**Thu 7/1/21**	**Fri 7/23/21**
Engineering	Thu 7/1/21	Mon 7/19/21
Validation	Tue 7/20/21	Fri 7/23/21
Documentation	**Mon 7/26/21**	**Tue 8/31/21**
Drafting	Mon 7/26/21	Fri 8/20/21
Supervision and final report	Mon 8/23/21	Tue 8/31/21

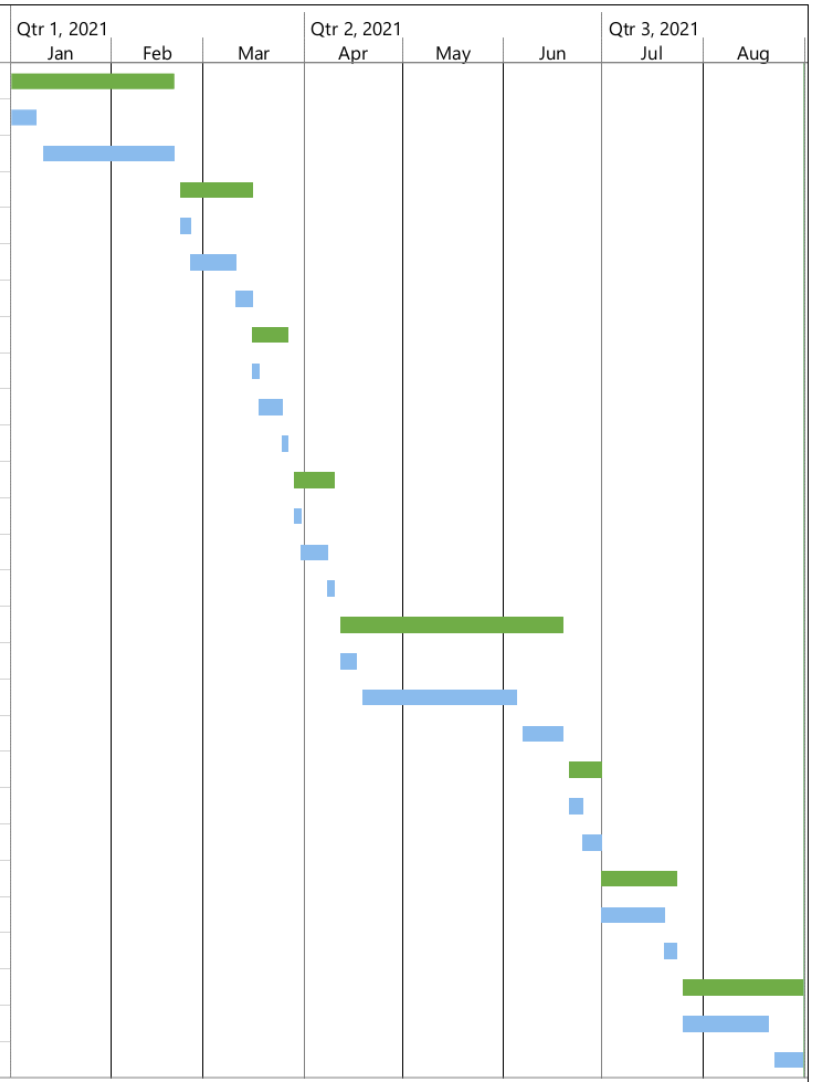


Figure 8-4: *Project's final Gantt diagram.*

Iteration	Description	Duration	Start Date	End date
-	Requirements gathering and prototype design	50 days	1/01/21	19/02/21
1	Git and parameter reading modules	24 days	20/02/21	15/03/21
2	Build configuration processing module	11 days	16/03/21	26/03/21
3	Build scripts and installation manifest	14 days	27/03/21	9/04/21
4	Deployment module	71 days	10/04/21	19/06/21
5	Improve installation manifest	11 days	20/06/21	30/06/21
6	Improve error handling	24 days	1/07/21	24/07/21
-	Documentation	38 days	25/07/21	31/08/21

Table 8.1: *Iteration Planning.*

8.2 Budget

This section contains a detailed breakdown of the costs of the project. The different categories are the Human Resources, Equipment, Software, Consumables and other costs. Then the total budget is obtained, which contains also the benefits, risks and taxes.

This project has taken a total of 8 months of work, starting in January 2021 and finishing in August of 2021. In each month there were 22 working days considered, with 2 hours each day. The research manager contributed also with 30 hours, which makes a total of 382 days of work. Table 8.2 shows the project's budget summary.

Author	Pablo Gómez-Caldito Gómez
Advisor	Francisco Javier García Blas
Duration	8 months
Total Budget	23,884.24 €

Table 8.2: *Project's budget summary.*

When calculating the costs related to the equipment and software, a 5 year amortization period was considered. So the resulting cost for them is obtained with this formula:

$$Total\ cost = \frac{Total\ Price}{60\ months} \cdot 8\ months \tag{8.1}$$

8.2.1 Human Resources costs

This subsection details the cost in human resources. This costs are reflected in Table 8.3, showing the hourly salary, hours spend and the total cost. There are two roles in this project, the Technical Lead and the Computer Engineer, taken by the tutor and the student respectively. The salaries are based on the 2021 Salary Guide of Spain made by Hays, a professional recruitment agency. [1].

Human resources cost			
Role	**Salary/hour**	**Hours**	**Total**
Technical Lead	40.00 €	30	1,200.00 €
Computer Engineer	30.00 €	352	10,560.00 €
Total HHRR cost	11,760.00 €		

Table 8.3: *Human Resources costs*

8.2.2 Equipment costs

This subsection details the cost in equipment during the development of project, this costs are reflected in Table 8.4. As said earlier a 5-year amortization period was considered.

Equipment costs			
Equipment	**Price per unit**	**Cost per month**	**Cost (8 months)**
Personal computer	2,000.00 €	33.33 €	266.66 €
Kubernetes cluster cloud servers	N/A	30.00 €	240.00 €
Printer	300.00 €	5.00 €	40.00 €
Total equipment cost	546.66 €		

Table 8.4: *Equipment costs.*

8.2.3 Software costs

This subsection details the cost in software during the development of project, this costs are reflected in Table 8.5. As said earlier a 5-year amortization period was considered, the same as in equipment costs.

[1] `https://cloud.email.hays.com/Guia-Salarial-2021-profesionales`

Software costs			
Software	**License Price**	**Cost per moth**	**Cost (8 months)**
Ubuntu 20.04 Desktop	0.00 €	0.00 €	0.00 €
LaTeX	0.00 €	0.00 €	0.00 €
Go programming language	0.00 €	0.00 €	0.00 €
Kubernetes	0.00 €	0.00 €	0.00 €
Docker Community Edition	0.00 €	0.00 €	0.00 €
Podman	0.00 €	0.00 €	0.00 €
Total software cost	**0.00 €**		

Table 8.5: *Software costs.*

8.2.4 Consumables costs

This subsection details the cost in consumables during the development of project, this costs are reflected in Table 8.6. These include office material like pens, paper sheets, post-its and notebooks, and also the printer toner.

Consumables costs	
Consumable	**Cost**
Office material	30.00 €
Printer toner	30.00 €
Total consumables cost	**60.00 €**

Table 8.6: *Consumable costs*

8.2.5 Other costs

This subsection details the cost in consumables during the development of project, this costs are reflected in Table 8.7. Indirect costs are the ones originated by the use of an office such as electricity, internet or water, and represent a 20% of the human resources expenses.

Other costs	
Concept	**Cost**
Indirect costs	2,352.00 €
Total other costs	**2,352.00 €**

Table 8.7: *Other costs.*

8.2.6 Total budget

This final subsection of the budget adds together all the costs described earlier and also adds the risks and benefits, which are a 15% and 20% of the costs respectively. Finally, it adds the 21% of taxes on top of that to obtain the resulting total budget of this project, **23,884.24 €**. This is all illustrated in Table 8.8.

Total costs	
Concept	**Cost**
Human resources	11,760.00 €
Equipment resources	546.66 €
Software resources	0.00 €
Consumable resources	60.00 €
Other costs	2,352.00 €
Total costs	**14,718.66 €**
Risks (15%)	2,207.80 €
Benefits (20%)	2,943.74 €
Total after risks and benefits	**19,870.20 €**
Taxes (21%)	3,974.04 €
Total after taxes	**23,884.24 €**
Total Budget	**23,884.24 €**

Table 8.8: *Project Budget.*

8.3 Socioeconomic Impact

This program is FOSS (Free and Open Source Software), this means that the program source code is available. This results in a series of benefits:

- **Transparency:** People can read the source code so they know what they are actually running on their servers.
- **Contributions:** Any developer can contribute to improve the tool by submitting pull requests with new features or bug fixes.
- **Free of charge:** The tool can be used for free, so can benefit anyone despite its socioeconomic status.
- **Eases adoption:** The project can be continued by others by forking it in case the development ceases.

Also, being open source does not mean that the tool can not be monetized. After the project gets a significant user base we support can be sold to users who need it.

CHAPTER 9

Conclusions and Future Work

This chapter is the one which puts and end on the document. First analyzes the achievement of the proposed objectives and then shows the future work.

9.1 Objectives Achievement

The main goal of our project was to build a tool that just does GitOps well and in a simple way, following the Unix philosophy. The objectives established in Section 1.2 have been achieved during the development of the project.

O1 Obtain a simple yet useful GitOps tool.

Chapter 3 gives an overview on the resulting tool and Chapter 5 provides a lot more detail on how it works. The end result is a very simple program that enables development teams to adopt GitOps.

O2 Tool working inside Kubernetes cluster.

The tool was designed from the start to be deployed inside Kubernetes-based clusters, and it does it as expected. This was explained within the 3.2 in more detail.

O3 Make the project free and open-source.

In Subsection 4.2.1 we state the license of the tool. The tool is now available as a FOSS project called “Soup”. Anyone can add their contributions by submitting a pull request, use for free or make a copy of it and change it.

9.2 Future Work

This section proposes further work on this project that would be beneficial.

Outside cluster deployment option.

Give the option of deploying the “Soup” program outside a k8s cluster, in some cases it would be better. In this case the credentials to communicate with the k8s API should be passed to the tool, because it will not be able to get them from the cluster because runs outside of it.

Ephemeral namespaces.

To delete the created namespaces when branches related to them are deleted. This is specially useful when a namespace is created per branch as stated in Subsection 7.3.1

Repository authentication.

Right now it does perform any authentication to use the given repository, so only works with repositories that do not require it. So this will be a future feature because is a painful limitation.

Handling multiple repositories.

It would be a nice addition because some use cases would benefit from it and is not complicated to implement.“Soup” could handle multiple repositories without major changes in its structure, this change only would affect the parameter reading and git modules.

Signal handling for graceful shutdown.

Right now the application does not handle the shutdown in any particular way. This could lead to shutting down in the middle of a deployment process. Because of this, a next feature would be to able to handle signals to perform a graceful shutdown, in which it would wait a bit for the deployment to complete before ending.

Regex for manifests in `.soup.yml` files.

This is a next feature in the road map because it would make a lot more flexible the build configuration file. Users could say that any file in a folder or that any file ending in an specific way should be deployed.

Automated tests.

An automated tests are a must have in software projects. And specially in open-source free software, the program should be tested automatically and in a reproducible way in order for people to adopt “Soup” and contribute to it.

Bibliography

[1] Cisco. (Mar. 2020). "Cisco annual internet report (2018–2023) white paper," [Online]. Available: https://kubernetes.io (visited on 09/05/2021).

[2] P. Jha and R. Khan, "A review paper on devops: Beginning and more to know," *International Journal of Computer Applications*, vol. 180, pp. 16–20, Jun. 2018. DOI: 10.5120/ijca2018917253.

[3] K. P. Contributors. (Jul. 2021). "What is kubernetes?" [Online]. Available: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/ (visited on 09/05/2021).

[4] O. Laadan and J. Nieh, "Operating system virtualization: Practice and experience," Jan. 2010. DOI: 10.1145/1815695.1815717.

[5] M. Shahin, M. Ali Babar, and L. Zhu, "Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices," *IEEE Access*, vol. PP, Mar. 2017. DOI: 10.1109/ACCESS.2017.2685629.

[6] S. Kumar and P. Dubey, "Software development life cycle (sdlc) analytical comparison and survey on traditional and agile methodology," *ABHINAV NATIONAL MONTHLY REFEREED JOURNAL OF RESEARCH IN SCIENCE TECHNOLOGY*, vol. 2, pp. 22–30, Aug. 2013.

[7] A. A. Richardson. (Aug. 2017). "Gitops - operations by pull request," [Online]. Available: https://www.weave.works/blog/gitops-operations-by-pull-request (visited on 09/05/2021).

[8] E. S. Raymond, *The Art of UNIX Programming*. Pearson Education, 2003.

[9] Á. Kovács, "Comparison of different linux containers," in *2017 40th International Conference on Telecommunications and Signal Processing (TSP)*, Jul. 2017, pp. 47–51. DOI: 10.1109/TSP.2017.8075934.

[10] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and linux containers," *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 171–172, 2015.

[11] K. P. Contributors. (Jul. 2021). "Why you need kubernetes and what it can do," [Online]. Available: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/#why-you-need-kubernetes-and-what-can-it-do (visited on 09/05/2021).

[12] ——, (Aug. 2021). "Kubernetes components," [Online]. Available: https://kubernetes.io/docs/concepts/overview/components (visited on 09/05/2021).

[13] S. Chacon and B. Straub, *Pro Git*, 2nd. USA: Apress, 2014.

[14] V. Driessen. (Jan. 2010). "A successful git branching model," [Online]. Available: https://nvie.com/posts/a-successful-git-branching-model (visited on 09/05/2021).

[15] Weaveworks. (Aug. 2018). "What is gitops," [Online]. Available: https://www.weave.works/blog/what-is-gitops-really (visited on 09/05/2021).

[16] ——, (Jan. 2010). "Guide to gitops," [Online]. Available: https://www.weave.works/technologies/gitops (visited on 09/05/2021).

[17] A. I. Dmitrichenko. (Jul. 2018). "Kubernetes anti-patterns: Let's do gitops, not ciops!" [Online]. Available: https://www.weave.works/blog/kubernetes-anti-patterns-let-s-do-gitops-not-ciops (visited on 09/05/2021).

[18] A. S. Foundation. (2004). "Apache license, version 2.0," [Online]. Available: https://www.apache.org/licenses/LICENSE-2.0.html (visited on 09/06/2021).

[19] C. E. Oren Ben-Kiki and I. döt Net. (Oct. 2009). "Yaml ain't markup language (yaml™) version 1.2," [Online]. Available: https://yaml.org/spec/1.2/ (visited on 09/05/2021).

[20] K. P. Contributors. (Aug. 2021). "Server-side apply," [Online]. Available: https://kubernetes.io/docs/reference/using-api/server-side-apply (visited on 09/05/2021).

[21] ——, (Jun. 2021). "Overview of kubectl," [Online]. Available: https://kubernetes.io/docs/reference/kubectl/overview/ (visited on 09/05/2021).

[22] P. Mahajan, H. Shedge, and U. Patkar, "Automation testing in software organization," *International Journal of Computer Applications Technology and Research*, vol. 5, pp. 198–201, Apr. 2016. DOI: 10.7753/IJCATR0504.1004.

[23] M. D. Bono. (Apr. 2020). "Agile, unit testing and quality," [Online]. Available: https://marcellodelbono.it/agile-unit-testing/ (visited on 09/05/2021).

[24] T. Fernández. (Mar. 2021). "What is monorepo? (and should you use it?)" [Online]. Available: https://semaphoreci.com/blog/what-is-monorepo (visited on 09/05/2021).

[25] S. K. Pal. (Apr. 2018). "Software engineering | phases of prototyping model | set – 2," [Online]. Available: https://www.geeksforgeeks.org/software-engineering-phases-prototyping-model-set-2/ (visited on 09/05/2021).