Team 81: Fun With Kubernetes

Jacob Balin Robert Detjens Nathan Hausman Paul Lim Mark Ser

15 March 2022

Contents

1 Executive Summary

Changes from Winter Term 2022: No changes have been made to this section.
Changes from Spring Term 2022: No changes have been made to this section.

1.1 Overall Goal

The goal of the project is to create a deployment pipeline for containerized applications on Raspberry Pi devices. With this pipeline, users should expect to be able to have their application containerized and deployed onto a Kubernetes Cluster. This cluster will be hosted on a Raspberry Pi to allow for affordable scaling of hardware resources with the addition of more Pis.

1.2 Term Plan

During the fall term, we focused on developing our iteration plan and design of the project, as well as performing initial setup of the needed hardware. For our iteration plan, we created a Trello board to manage and distribute the tasks needed to complete this project. We have also started Phase One of our development schedule: installing Kubernetes onto the Raspberry Pi. Additionally, each member also conducted research into the components used for their development phase. We will be completing the following during the winter term: building both stages of the deployment pipeline – the first, where the user-submitted application is built and containerized with Docker, and the second where the containerized app is deployed onto the Pi as a Kubernetes pod. Additionally, we will be completing the user-facing website for the deployment pipeline where a developer can submit jobs. During spring term, we plan to complete the following: integrating the developer site, deployment pipeline, and performing A/B testing on the completed project.

1.3 Identified Risks

We have identified three potential risks that could set back our schedule if not properly managed. We foresee that phases may be blocked on previous phases if they are not completed on time. To mitigate this, we plan to establish priorities for each feature so that development efforts can be focused on the minimum viable product first, before working on non-essential features. This will allow each development phase to be completed on time and not block the rest of the team. An additional risk is software incompatibility with ARM devices; Raspberry Pis are based on the ARM architecture, while some software is only built for x86. We have already encountered this problem in the setup process. All additional software will be reviewed to ensure compatibility and replaced with alternatives if needed. Finally, we may need additional hardware in order to test the product fully. If this is the case, we will contact the project partner or the instructional team to procure Pis as needed.

2 Background

Changes from Winter Term 2022: No changes have been made to this section.
Changes from Spring Term 2022: No changes have been made to this section.

2.1 Problem Statement

The objective of this capstone project is to build a workflow pipeline that can take a user-specified config file and launch a containerized Docker application using Kubernetes on a cluster of several Raspberry Pi computers. This allows the user to utilize a powerful system that can deploy web applications, machine learning, or any application to their choosing with a lower operating cost compared to industry solutions. By utilizing a Raspberry Pi cluster, the user can expect to be able to scale their workflow in either software or hardware. The user should also expect to be able to access their deployment through a website portal outside of the local network. This capstone project is targeted towards startups, researchers, and other groups of individuals that are looking to reduce their operating costs and reliance on cloud giants.

2.2 Existing Systems and Conditions

Currently, the primary solutions are cloud-based services such as AWS, Microsoft Azure, or Google Cloud. While useful in many contexts, the proposed solution would be cheaper in the long term for smaller services. While a Raspberry Pi cluster would have a one-time initial purchase for the hardware, cloud-based services would have a larger total cost, as there are recurring operational charges based on the computing power used.

This project will use the Docker containerization service alongside the Kubernetes container orchestration system to create a cluster on Raspberry Pis. Our project partner will be providing us with a Raspberry Pi with which to complete this project.

2.3 Motivation

The project partner is looking for alternative solutions to reduce their reliance on industry solutions and create a similar solution with a local Raspberry Pi cluster instead of servers.

2.4 List of Terms

Docker A containerization platform that packages software in a lightweight, isolated, and portable environment called a container. https://www.docker.com/

Kubernetes An open-source container-orchestration system for automating computer application development, scaling, and management. https://kubernetes.io/

Pipeline A set of automated processes that allow developers to reliably and efficiently compile, build, and deploy their code to their production compute platforms.

Raspberry Pi A low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. https://www.raspberrypi.org/

2.5 People Involved

Our team would like to acknowledge the project partners and stakeholders, including:

- Kyle Prouty (Project Partner)
- HP (sponsor / stakeholder)

3 Vision Statement

Changes from Winter Term 2022: No changes have been made to this section.
Changes from Spring Term 2022: No changes have been made to this section.

When our project is complete, a user will be able to quickly customize our pipeline to suit their workflow with a single configuration file. The setup will be just as simple as if they were to use another service. The user experience will be intuitive and easy to use and the system will be easily scalable.

The adoption of the project should decrease the operating costs to host simple microservices. It should also decrease reliance on current industry solutions and allow for more options in hosting.

3.1 Central hypotheses

3.1.1 Growth hypothesis

- We reduce the complexity of hosting a service
- The users can scale up and down their services as needed
- As the application is running on our own network, the users will be able to serve sites on custom domains; e.g. `www.go-beavs.com`

3.1.2 Value hypothesis

- Its learning curve will be less steep than industry solutions such as AWS, Microsoft Azure, or Google Cloud
- The users have the ability to scale their application as the backend is built on Docker and Kubernetes
- Users can deploy it on their own Raspberry Pi, or add additional Pis to the cluster

3.2 High-level requirements

- The deployment pipeline must provide connectivity between container system and internet

3.2.1 Functional requirements

- The deployment pipeline must run on a cluster of Raspberry Pis
- The deployment pipeline interface must be accessible by the local network
- The deployment pipeline must be able to deploy a user's service to the Kubernetes cluster from a user-provided configuration file
- The deployment pipeline must provide a way to connect the Kubernetes pods and their services to the Internet

3.2.2 Non-functional requirements

- The user must be able to view a dashboard of their service after the deployment through an user interface
- Non-technical users must be able to deploy their application with assistance from the documentation
- The deployment pipeline should use Kubernetes operators to automate certain tasks
- The deployment pipeline should provide scalability through adding or removing the use of more Raspberry Pis through the user interface

4 Success Measures & Stakeholders

Changes from Winter Term 2022: No changes have been made to this section.
Changes from Spring Term 2022: No changes have been made to this section.

4.1 Success Measures

4.1.1 Initial project research

- Research about Kubernetes and Kubernetes deployment
- Research about Raspberry Pi
- Configure remote access to the Raspberry Pi

By the end of this success measure, team members are more familiar with the technology that we will be using in this project.

4.1.2 Set up Kubernetes

- Install and configure Docker and Kubernetes
- Create pods within Kubernetes

By the end of this success measure, we should be able to deploy Kubernetes pods manually.

4.1.3 Configure Kubernetes UI/Dashboard

- Install custom solutions for better management of clusters and viewing cluster status

By the end of this success measure, team members will be able to manage the cluster from the management portal instead of through a CLI.

4.1.4 Develop a pipeline (part 1)

- User is able to go into the dashboard and submit a project with a config file
- Pipeline downloads the project and deploys project to Kubernetes pod

By the end of this success measure, users should be able to submit a config file to the pipeline.

4.1.5 Develop a pipeline (part 2)

- User is able to view uploaded project
- User should be able to see the pod status after successful deployment

By the end of this success measure, a user should be able to successfully launch their application.

4.1.6 Public facing

- User's project is able to receive traffic outside of the Raspberry Pi network

By the end of this success measure, public users should be able to access and make use of the deployed application.

4.2 Project Stakeholders

4.2.1 Project Partner

- Primary investor in this project.
- Provider of direction and basic structure of the project.
- Provider of the resources necessary to complete the project.

4.2.2 The Team

- Develop the deployment pipeline software
- Responsible for keeping the cluster operational and working

4.2.3 The Clients

- Use the cluster and management pipeline to host their application.

4.2.4 The Users

- Use client-deployed software on the cluster

5 Project Constraints & Risk

Changes from Winter Term 2022: No changes have been made to this section.
Changes from Spring Term 2022: No changes have been made to this section.

5.1 Prioritized Project Constraints

5.1.1 Constraint 1: Time

We have a maximum of 9 months from the start of this project to complete it. In this time we need to gain a complete understanding of all of the requirements for this project and how to successfully complete an implementation that meets those requirements. While developing this implementation, we also need to take into account any changes requirements. After the initial implementation is complete, we need time to test the project for any bugs and correct them. By the end of the 9 month period, we will have a completed and working implementation of this project (which will be as close to bug free as possible) that will be made available to users. The application we provide will have met all requirements that are set forth by our project partner.

5.1.2 Constraint 2: Resources

The original goal of this project was to have Kubernetes deployed across several Raspberry Pis. However, we currently only have one Pi on which to deploy our software. This may not be a problem for deliverables, but we will not be able to make sure our software is able to scale across multiple machines in a cluster.

5.1.3 Constraint 3: Scope

We may not have enough time to build a system where an user can submit any application to the system. Meaning, we have the system in mind where a user is submitting only a website application instead of having a system where the user can also submit a machine learning application.

5.2 Risk Management

Risk	Likelihood	Impact	Mitigation Strategy	Early Detection	Consequence
Inadequate hardware supplies	Likely	Medium	Check in with TA/Course Staff/Project partner on getting additional hardware supplies with a formal request form on our hardware request.	Do not have multiple raspberry pis in hand.	Not finishing a secondary element of the project.
Unable to complete project by the deadline	Unlikely	High	Review progress weekly and update the iteration plan to accommodate changing circumstances. An additional midpoint review should be conducted to see if any constraints may need to be adjusted.	The iteration plan ends up exceeding the deadline.	The project will be unfinished and possibly undeployable.
Bugs introduced to system that prevent project from functioning within constraints	Likely	Medium	Frequent unit testing performed during development of individual components of the system and integration testing performed when merging back into the main branch so any bug can be dealt with quickly.	Failed tests.	Bugs that go undiscovered can prevent users from successfully deploying their services.
Unauthorized access to development hardware	Unlikely	Medium	Key authorization will be required to log in to node preventing brute force password attacks. Code will also be backed up in Git & Github allowing for easy wipe & restore if compromised.	Keeping an eye on login attempts for unauthorised access.	Minor setback in development time while hardware is reset.

6 Iteration Plan & Estimate

Changes from Winter Term 2022: No changes have been made to this section.
Changes from Spring Term 2022: No changes have been made to this section.

6.1 Kubernetes Setup (Sprint 2 - 3)

a. Install Kubernetes on the Raspberry Pi **(1-2 days)**
b. Configure Kubernetes on the Raspberry Pi **(1-2 days)**
c. Connect to the Raspberry Pi Admin tools with K8S Lens **(3 Days)**
d. Test Kubernetes on the Raspberry **(3 days)**
e. Code Reviews and release **(4-5 Days)**

6.2 Kubernetes CI/CD Config File Integration (Sprint 3)

a. Research what settings is needed **(2-3 days)**
b. Work with Deployment UI Website team to define template format **(4-6 days)**
c. Create end-user documentation for config template **(1 days)**
d. Code Reviews and release **(4-5 Days)**

6.3 Github Container Build Pipeline (Sprint 4-5)

a. Create build script to pull repo from user-provided github **(1 day)**
b. Modify build script to build container image **(1-3 days)**
c. Coordinate with K8S Deployment team to define container handoff API **(2-3 days)**
d. Modify build script to send built image to K8S onboarding **(1-2 days)**
e. Modify script to ensure clean building environment **(1-2 days)**
f. Modify script to use user-selected generic Dockerfile if none provided **(3-4 days)** [**stretch**]
g. Integrate build pipeline with Github Actions **(5-7 days)** [**stretch**]
h. Code Reviews and release **(4-5 Days)**

6.4 K8S Deployment (Sprint 4 - 5)

a. Watch the images in the deployment area **(1-2 days)**
b. Deploy a new image in the deployment area **(2-3 days)**
c. Coordinate with Github Container Build team to define container handoff API **(2-3 days)**
d. Testing **(2-3 Days)**
e. Code Reviews and release **(4-5 Days)**

6.5 Deployment UI Website (User Facing) (Sprint 4-5)

a. Create a clean and usable web UI **(3-6 days)**
b. Containerize the web UI and self-host on the cluster **(1 day)**
c. Create an interface to pass applications as well as necessary configuration settings to the pipeline for deployment **(3-6 days)**
d. Set up an automated system for containerizing an application to be passed to the pipeline that is not yet containerized **(2-4 days)** [**stretch**]
e. Expose admin tools, such as Lens IDE **(2-7 days)**
f. Set up authentication **(3-5 days)**
g. Code reviews and release **(4-5 days)**

6.6 Integration (Sprint 6)

a. Integrate the backend service **(7 days)**
b. Work with the Kubernetes CI/CD Config File Integration, Github Container, and K8S Deployment team to integrate the backend service.

ii. Internally test (Alpha Testing) the full pipeline, so a developer should be able to deploy a web application with internal tools **(7-10 Days).**

b. Integrate the connection with the backend with the Deployment UI Website **(7-14 days)**
c. Work with Kubernetes CI/CD Config File Integration team to define template format **(4-6 days)**

ii. Internally test (Alpha Testing) the full pipeline, so a developer should be able to deploy a web application with the Deployment UI portal **(7-10 Days).**

iii. Code Review and Release **(7 days)**

6.7 Beta Testing (Sprint 7)

a. Utilize multiple test web application for beta testing **(7-14 days)**
b. Provide an user with a Github link that contains a web application project, instruct the new users to onboard the web application project to our service.

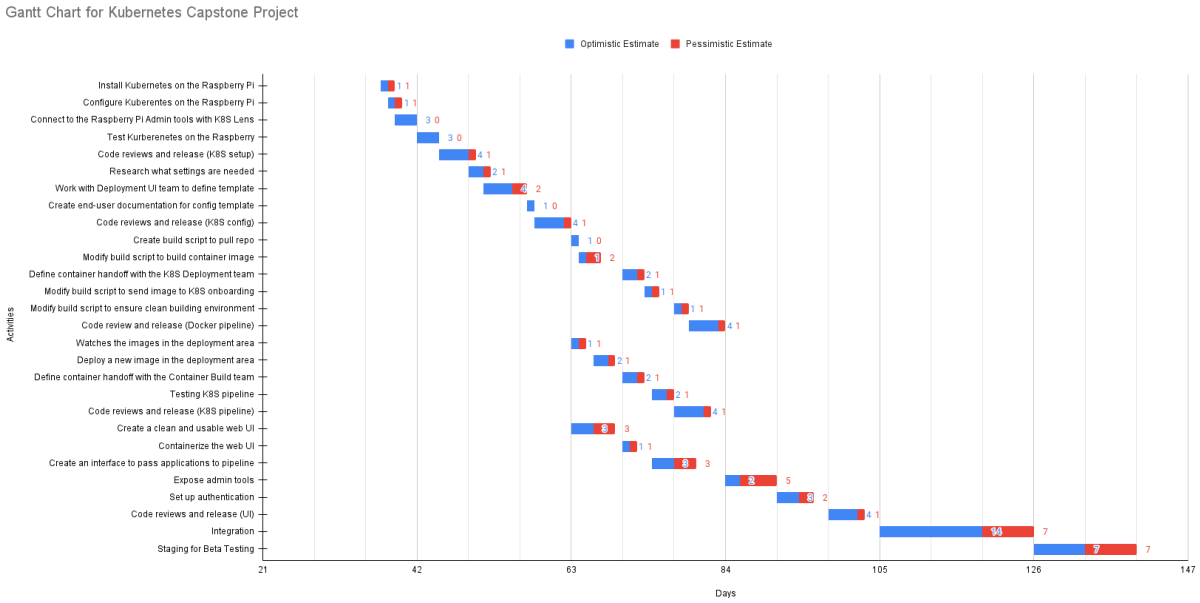


Figure 1: Gantt Chart

7 Individual Contributions

7.1 Mark Ser

Fun with Kubernetes: Distributed Web APIs

***Abstract*— This is a quick introduction to the overall view of the Fun with Kubernetes project and highlights the individual components of the projects and how the setup Raspberry Pi component will be tackled.**

Keywords—Kurbernetes, Docker, DevOps

INTRODUCTION

The goal of this project will be to build a Kubernetes cluster hosted from a raspberry pi. In the process we will leverage various operators for managing different services within the cluster management. After the cluster is set up, we will then build a deployment pipeline for deploying our containerized images as pods to the cluster. For this step we will build a deployment pipeline from scratch from the Kubernetes cluster we deployed. Lastly, we will leverage our new pipelines and deploy a web api or web app. [1]

PROJECT PROCESS DIAGRAM

Overall Project Flow Diagram

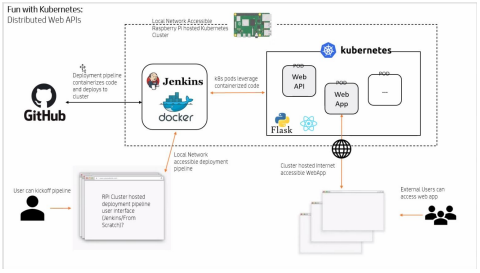


Figure 1: Overall Project Flow Diagram [1]

First, the user will access the system through an UI website dashboard that allows them to configure a deployment through the deployment pipeline. The user should be able to configure the application name, ports, and more. Then the deployment pipeline will run a custom deployment script that will pull the codebase from a github repo, build the codebase with a container image with version tags, and cleans up the build area. The deployment pipeline will have a second component where it will watch the staging area, where the container image is built, and deploy the new image in Kubernetes. Then a system admin should have the ability to view all of the active cluster statuses and manage any clusters: re-imaging, rebooting, etc.

Responsible component within Project

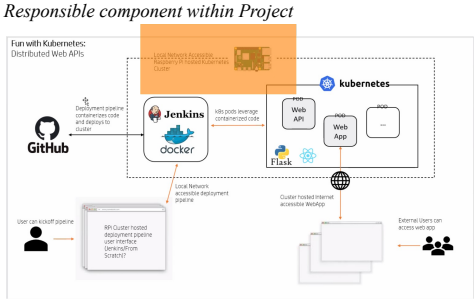


Figure 2: Responsible Component within Project [1]

Within the orange rectangle is highlighting the component that I am responsible / have ownership over. This component consists of configuring the raspberry pi for the project.

Expanded view of component

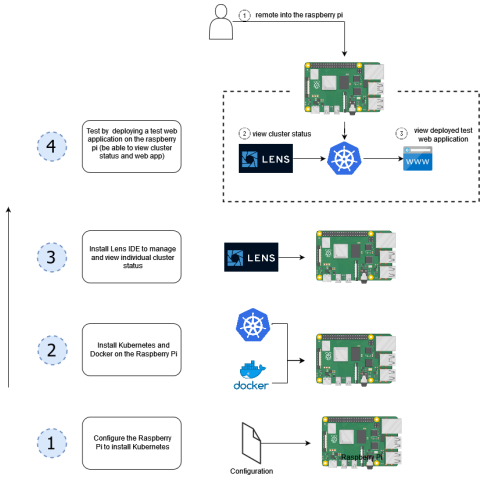


Figure 3: Expanded view of component

Here is an expanded view of the flow diagram of the component that I am taking ownership of for the Fun with Kubernetes project. I have split up the component into four smaller sections. We want to first configure the raspberry pi so that we are able to install Kurbernetes. In step 2, we want to instal kubernetes and docker onto the raspberry pi. In step 3, we want to install Lens, which is a system admin tool for kubernetes. In step 4, we want to verify that the raspberry pi has been successfully configured by deploying a test web application to the raspberry pi and seeing if a system admin is

able to utilize Lens on the deployed test web application and view the test web application through the configured test port.

User Stories for Setup Raspberry Pi

Here we will define user stories that will encapsulate the users and the project partner.

- As an individual contributor, I want to have the raspberry pi configured with Kurbernetes so that I can test my deployment pipeline.
- As a system admin, I want to be able to view the status of a deployed application so I can consider making adjustments to the cluster
- As an individual contributor, I want to have the raspberry pi configured with Kurbernetes so that I can configure Kurbenetes Configuration file depending on how it's installed/configured on the raspberry pi.
- As a user, I want to be able to view the entire kubernetes cluster status from a single UI dashboard so that I can view
- As a project partner, I want the individual contributors to work with Kubernetes so that they can learn more about Kuberentes and DevOps as a whole.

Personal Iteration Plan and Estimations

Timeline for Setup Raspberry Pi Component

This component is a major blocker for the other individual contributors as they require the raspberry pi to be setup before they can start working on their components in the project. So it's imperative that this component is finished as quickly and efficiently as possible. My current estimation is roughly 3 weeks, or all of Sprint 2 and a little of Sprint 3.

Week 1:

Research into various technology deployment systems for K8S that can run on a raspberry pi: KIND, K3D, microK8S, K8S, etc. Then configure and install the selected deployment system after discussing with the team.

Week 2:

Install K8SLens management system and validate the entire install with a test deployment. The raspberry pi should be able to deploy a test web application and a system admin user should be able to view the cluster status on the raspberry pi.

Week 3:

Conduct a code review and make any revisions as necessary.

As this is estimated to take roughly three weeks, I will be able to float around and help the other individual contributors on working on their components. Mostly, I believe that I will be helping the deployment pipeline owner as I will be the most familiar with the way that the raspberry pi is configured and can help them if any blockers arise from the configuration of the raspberry pi.

Solution Architecture

The technology stack, Kubernetes and K8SLens, were selected by our project partner. However, it is up to the owner on how to install Kubernetes in the raspberry pi. As mentioned in the Personal Iteration Plan and Estimations, there are many installation softwares for the raspberry pi. I believe that there are two main considerations: KIND and K3D that I will highlight here. I will also highlight microK8s for why it wasn't a fit for this project.

A. Kurbernetes in a Docker (Kind)

Kurbernetes in a Docker, or Kind, is a deployment system that installs kubernetes in a raspberry pi. More specially, it can install on an arm64 platform (common operating software on a raspberry pi). What's notable about this solution is that it allows us to run the Kubernetes clusters locally and in CI pipelines (something that we can use for our deployment pipeline) with Docker[1] containers [3]. Whereas some of the other solutions do not use the Kurbernetes source code, and utilize a wrapper of Kurbernetes [3]. This solution however is slow for deployments, and has a large package size compared to the other solutions [3]. Something that we want to consider as we are running this on a single raspberry pi for the foreseeable future.

B. k3d

k3d is a lightweight wrapper running k3s (a lightweight wrapper of Kubernetes) in docker [4]. This solution is great because k3s is designed to be production ready, uses half the memory and in a small package size [5]. However since the solution uses a Kuberenetes wrapper, it doesn't have the full capabilities that Kubernetes offers which can limit what our system can do in the future. Also, since the solution is relatent on a user managed solution, it may lose support over the year and make our system obsolete.

C. microK8s

microK8s is a system that is created by Canonical, a UK-based computer software company that is focused on Ubuntu related projects [6]. It's optimized for quick and easy installation for single and multi node clusters [2]. It's also maintained by a company instead of a single user or a community, so there is some guarantee that this solution wont get obsolete in a few years time. However, since it's not containziered with Docker, this is a solution that may not be used for our use case.

[1] Docker is a set of platform as a service products that use OS-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries and configuration files; they can communicate with each other through well-defined channels [2]

References

[1] K. Prouty, "EECS project submission form," *Single Project*. [Online]. Available :https://eecs.oregonstate.edu/capstone/submission/pages/viewSingleProject.php?id=zTiP5qRHJYVi8iYw. [Accessed: 08-Nov-2021].

[2] The Chief I/O, "K3D vs K3s vs kind vs Microk8s vs Minikube," *thechief.io*. [Online]. Available: https://thechief.io/c/editorial/k3d-vs-k3s-vs-kind-vs-microk8s-vs-minikube/. [Accessed: 08-Nov-2021].

[3] "Kind," *kind*. [Online]. Available: https://kind.sigs.k8s.io/. [Accessed: 08-Nov-2021].

[4] "Overview¶," *k3d*. [Online]. Available: https://k3d.io/v5.0.3/#what-is-k3d. [Accessed: 08-Nov-2021].

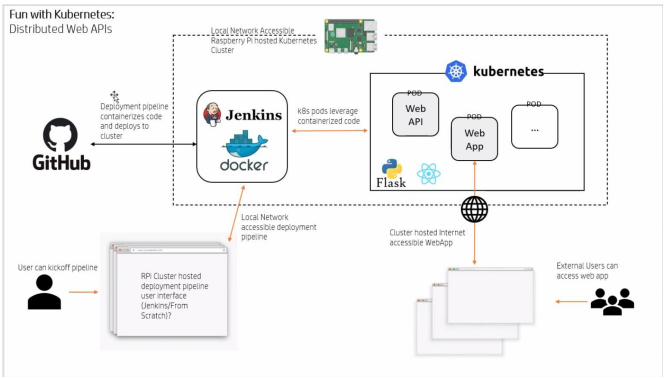
[5] k3s-io, "K3s-IO/K3s: Lightweight kubernetes," *GitHub*. [Online]. Available: https://github.com/k3s-io/k3s. [Accessed: 08-Nov-2021].

[6] "Canonical (company)," *Wikipedia*, 01-Nov-2021. [Online]. Available:https://en.wikipedia.org/wiki/Canonical_(company). [Accessed: 08-Nov-2021].

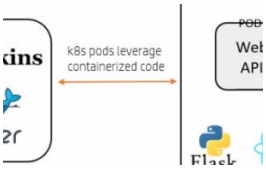
7.2 Jacob Balin

Section 1: Process Flows:

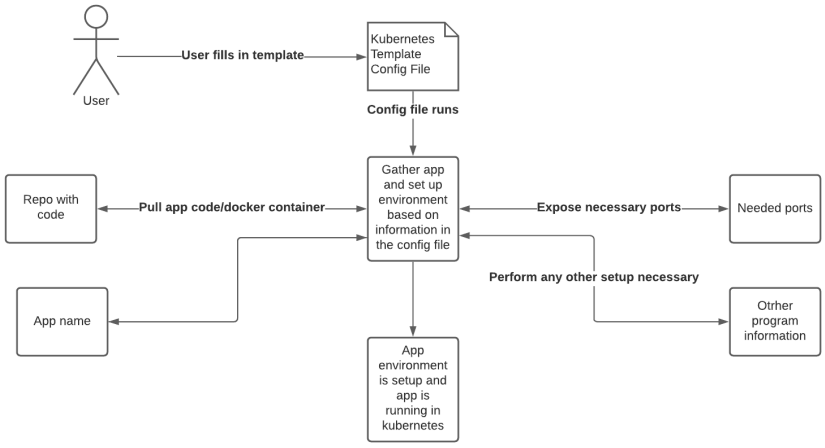
A. Major Project Components as provide by our project partner:



B. Individual Project Component:



I will be focused on this segment of the project:



The portion of this project that I am responsible for is the setup of the Kubernetes CI/CD configuration file which will be used to set up the Kubernetes environment for a user's application. This is phase 2 of 5 which we have defined as a group with our project partner. This will be done as follows. The user will be provided a template file which they will open and fill out with the information necessary to launch their application in Kubernetes. That completed configuration file will be used to set up the Kubernetes environment for the application. This will be done by completing setup tasks such as pulling any code or docker containers from any repos specified in the configuration file, exposing any ports provided in the file, naming the application appropriately, and completing any other necessary processes specified in the file. I will be responsible for designing the configuration file template which will be given to the user when they wish to launch the application, and then executing the instructions in the filled out configuration file.

Section 2: User Stories:

User story 1:

"As a user of this service, I need a template file that is clearly laid out, so that I can easily and accurately provide the information about my application that is needed to configure Kubernetes."

User Story 2:

"As the project partner, I need my team to create an easy to use configuration file, so that users of this project can successfully launch their application in a way that is as simple and user friendly as possible, and so that they do not face any frustrations with setup or other negative experiences using this service."

User Story 3:

"As a developer on this project, I need to develop an easy to use configuration file template that works correctly with the information filled in by the user, so that our service can successfully set up and launch the user's application without any errors or other problems."

User Story 4:

"As a user of the product produced by the client, I need to be able to use the system as it was intended to be used so that I know that the Kubernetes configuration was successful, which will provide me with a safer and more pleasant experience."

User Story 5:

"As a user of this service, I need the template configuration file to work properly with the information I provide it so that my application works correctly without any problems or bugs from the Kubernetes configuration process, some of which may not be as obvious as others, which could lead to problems for my users latter on."

Section 3: Personal Iteration Plan and Estimation:

I believe that work on this portion of the project will begin most likely at the start of Winter term, or possibly as early as the end of Fall term. This is because my portion of the project is in an early phase of the development. The only part of the project that is blocking my progress on this is phase 1, which entails installing Kubernetes and setting it up. This, however, should be a short and simple phase which will not block my phase, phase 2, for very long. The only other requirement that needs to be completed before beginning work on this phase is the general research our entire team will be doing in order to fully understand what we need to do to accomplish our phases and what the file product should look like. This research so far includes how to use the tools we will be working with which include Docker, Kubernetes, Flask, React, etc. Once we are all familiar with these tools, we can begin working on the phases of our project. I believe that we can complete this research by the end of the term. I also believe that it is possible for phase 1 to be completed by the end of the term or in early Winter term, which would allow work on phase 2 to begin.

In addition, since each phase is quite inclusive of elements of the other phases, we will each help each other on each phase. We have decided to work in a system where one person is the leader and main contributor of a phase. When someone is done with or can not work on their phase, they can help the others to complete their phases. This way we will all contribute and fully understand each phase of the development of this project.

Section 4: Solution Architecture:

For this project, we have chosen to use Kubernetes to deploy user applications because it is a free and easy to use option. According to the Kubernetes documentation [1], it allows a user to continuously serve and update their containerized software so that their users are able to connect to their service constantly. The article by Red Hat [2] identifies Kubernetes as a solution that "automates many of the manual processes

involved in deploying, managing, and scaling containerized applications." We hope to use this functionality to allow users to deploy an application on a Raspberry Pi which will be accessible to the public for a much lower cost than hosting their applications on other services. According to Red Hat [2] Kubernetes works by creating a cluster of machines running some form of Linux which are used to deploy applications which are made up of containers (such as Docker containers). In addition, Kubernetes allows a developer to schedule updates to go out to these pods in order to keep the running applications up to date and usable. With that in mind, I will be responsible for setting up the configuration template which users will complete to start up their Kubernetes environment. According to the Kubernetes documentation on configuration [3], the configuration format recommended for Kubernetes is YAML. This section of the Kubernetes documentation [3] also recommends the use of labels and kubectl. In this context, labels, "identify semantic attributes of your application or Deployment..." and kubectl is a command which searches a directory for any configuration files which are used in Kubernetes configuration. Because we plan to use the YAML format for the configuration file, I will likely develop a template, using labels, which will guide our user to filling out all of their configuration information.

There are a few ways I think the template could be laid out. The first is that the user will be provided with one template file of which we have created. The labels in this file will allow the user to easily fill in the correct information. However, because some users may have more information that needs to be included than others, this template will need to be very large and all inclusive to possible pieces of configuration information the user may want to specify. The other possible method I have come up with is creating multiple templates, each of which holds different information. In this case, there will be a general template file which will hold crucial information that most if not all users will need to set up Kubernetes, while the other templates will hold configuration information that is less crucial. These less crucial templates will each contain information in a certain category so that the users can easily select the appropriate set of templates in order to provide Kubernetes with all of the necessary configuration information. If this template option is chosen, then kubectl will be used to gather the completed templates and execute the configuration.

Kubernetes is a system which allows users to easily deploy and maintain their applications. In order to ensure that this functionality is provided to the users of our service, we need to develop a template that is easy to use and understand. My goal is to design a template that will accurately configure a user's Kubernetes environment without sacrificing any usability. Usability is important because we do not know what level of knowledge about Kubernetes our users will have, and thus it is important to make a template that is as easy to understand as possible. Accuracy is also very important, as if there are any faults in configuration, the user will not be able to launch

their application successfully. That is why it is important to me to create a template which accommodates these two principles. This will be my responsibility, and with the help of my team we will accomplish this task.

Sources:

[1] Kubernetes. “Learn Kubernetes Basics.” Kubernetes. https://kubernetes.io/docs/tutorials/kubernetes-basics/ (accessed November 7, 2021).

[2] Red Hat. “What is Kubernetes?” Red Hat. https://www.redhat.com/en/topics/containers/what-is-kubernetes (accessed November 7, 2021).

[3] Kubernetes. “Configuration Best Practices.” Kubernetes. https://kubernetes.io/docs/concepts/configuration/overview/ (accessed November 7, 2021).

7.3 Robert Detjens

WIC: Individual Contribution

Robert Detjens

Process Flows

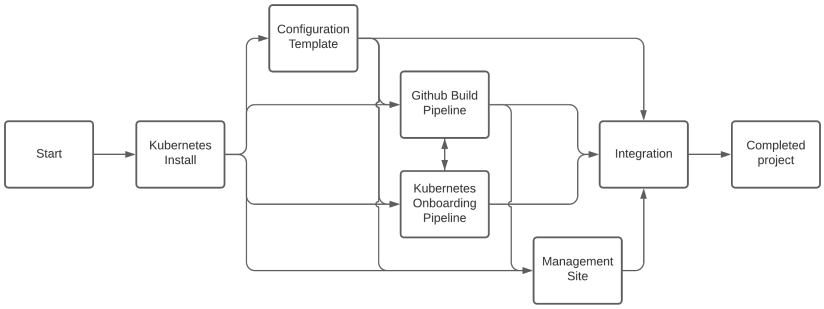


Figure 1: Task flow overview

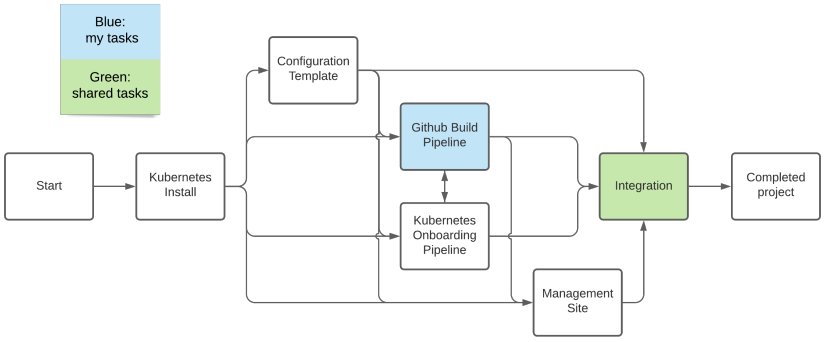


Figure 2: My task responsibilities

1

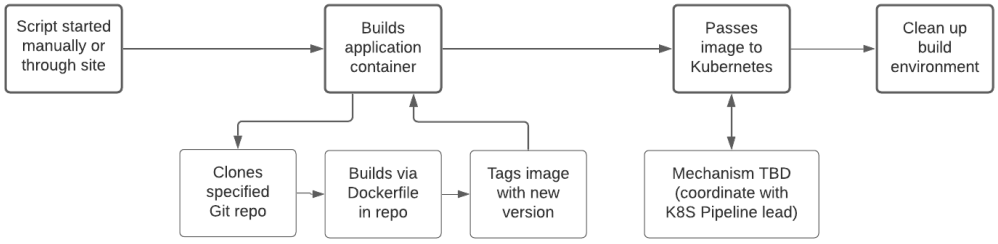


Figure 3: Github Image Pipeline details

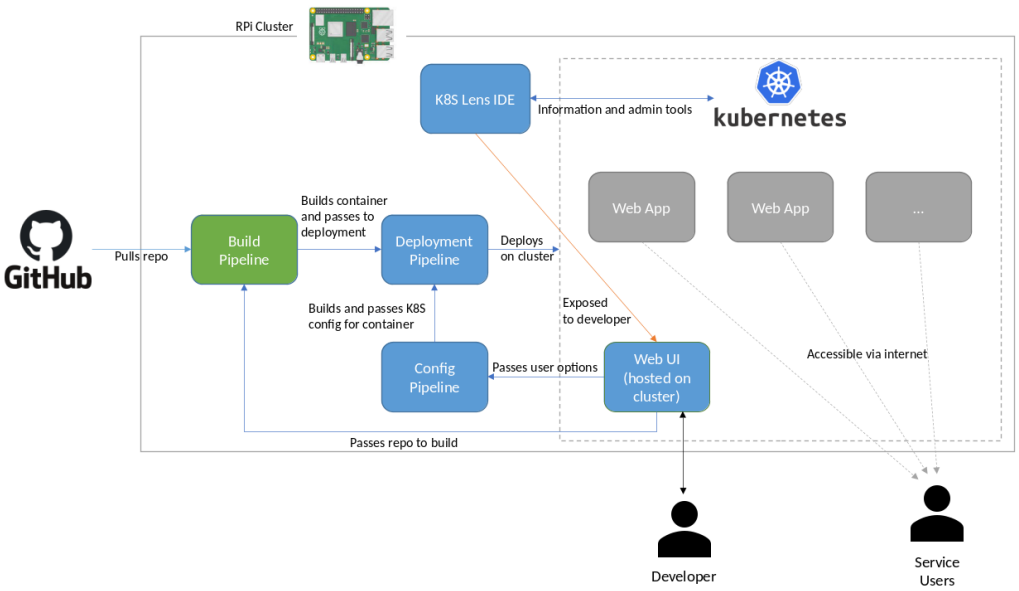


Figure 4: General project overview

User Stories

As a user of the deployment service, I want my application to be built quickly so that I can rapidly develop my application and not wait on the build service.

As a user of the deployment service, I want my applications to be built correctly according the the rules I set so that I know the application works as intended.

As a user of the deployment service, I want my applications to be built reproducibly so that the built image is the same as would be created in a local dev environment.

As a developer, I want the deployment pipeline to be simple so that it can be easily worked on and maintained with new features.

As the project partner, I want a working pipeline so that the full product can be delivered.

As the end user of a deployed project, I want the container to be built correctly so that the application is functional.

Personal Iteration Plan

The phase of the project I am responsible for is the Docker image build pipeline – turning a user's application repository into a deployable container image. This phase is blocked on the Kubernetes install (phase 1) and the user-provided config file design (phase 2). Once these have been completed in Sprint 3, that unblocks my phase to begin implementation in Sprint 4. The final steps of this build pipeline need to be coordinated with the Kubernetes image onboarding pipeline (phase 4), and as such will be blocking until we can coordinate that handoff procedure.

While my phase is blocking, I will be assisting my other teammates with the current phase being worked on, such as assisting in designing the template file for Phase 2 and integrating all components together.

Broken up by sprint, the plan is as follows:

- Sprint 2: Assist in setting up access to the Pi to facilitate the install
- Sprint 3: Assist team in researching needed config values for config file
- Sprint 4: Begin implementing build pipeline
- Sprint 5: Finish implementing build pipeline and coordinate image handoff procedure
- Sprint 6: Assist in final integration of components together
- Sprint 7: Help manage beta testing and feedback

Solution Architecture

Our team will be working on the project in 5 discrete phases in order to maximize available time and share the work of managing parts of the pipeline between each team member. Most of our components are easily modularized, however some depend on each other and will require each component team to coordinate solutions where their domains overlap.

The first phase must be completed before working on subsequent phases, as all following phases require interaction with Kubernetes. We will be using the K3S implementation of Kubernetes [1], as it is resource-light while being feature-complete. We do not need the more powerful features of full-fat K8S detailed in [2] as this cluster will be running only on one machine, not hundreds. The smaller footprint of k3s will also more of the compute power available for client applications.

We will also be using Lens [3] to provide an admin dashboard for the Kubernetes cluster, both for internal use and eventually for exposing management to end developers. This was recommended by our project partner and from our research we agree with their recommendation.

List of Terms

Docker A containerization platform that packages software in a lightweight, isolated, and portable environment called a container. https://www.docker.com/

Kubernetes An open-source container-orchestration system for automating computer application development, scaling, and management. https://kubernetes.io/

Pipeline A set of automated processes that allow developers to reliably and efficiently compile, build, and deploy their code to their production compute platforms.

Container A fully-contained application that bundles all of its dependencies in one package and can be deployed easily

Repository A hosted version of source code tracked via a version control program such as Git

References

[1] k3s-io, "K3s Lightweight Kubernetes," *GitHub repository*. Rancher, 2021.Available: https://github.com/k3s-io/k3s

[2] H. Roy, "K8s vs K3s: The comprehensive difference," *p3r*. p3r, Jul. 2021.Available: https://www.p3r.one/k8s-vs-k3s/

[3] lensapp, "Lens - The Kubernetes IDE," *Github repository*. Mirantis, Inc.Available: https://github.com/lensapp/lens

7.4 Paul Lim

Section 1: Process Flows using two or more visualizations/diagrams

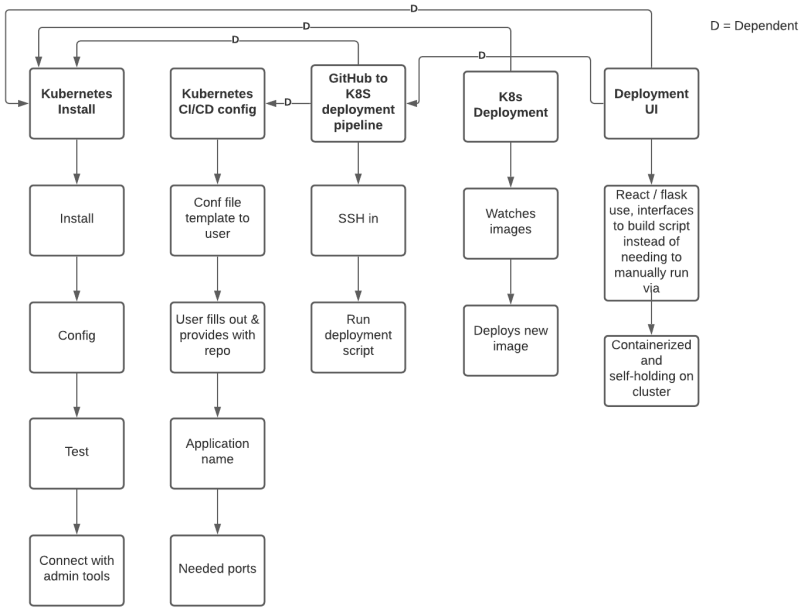


Figure 1: High level dependency chart of major tasks

The chart above is a dependency chart of our major tasks and their individual steps. The majority of the tasks are dependent upon the first major task, Kubernetes Installation, being complete. There are also several other dependencies, such as the GitHub to Kubernetes deployment pipeline being dependent on the Kubernetes CI/CD config, that need to be kept in mind as the team proceeds through each sprint.

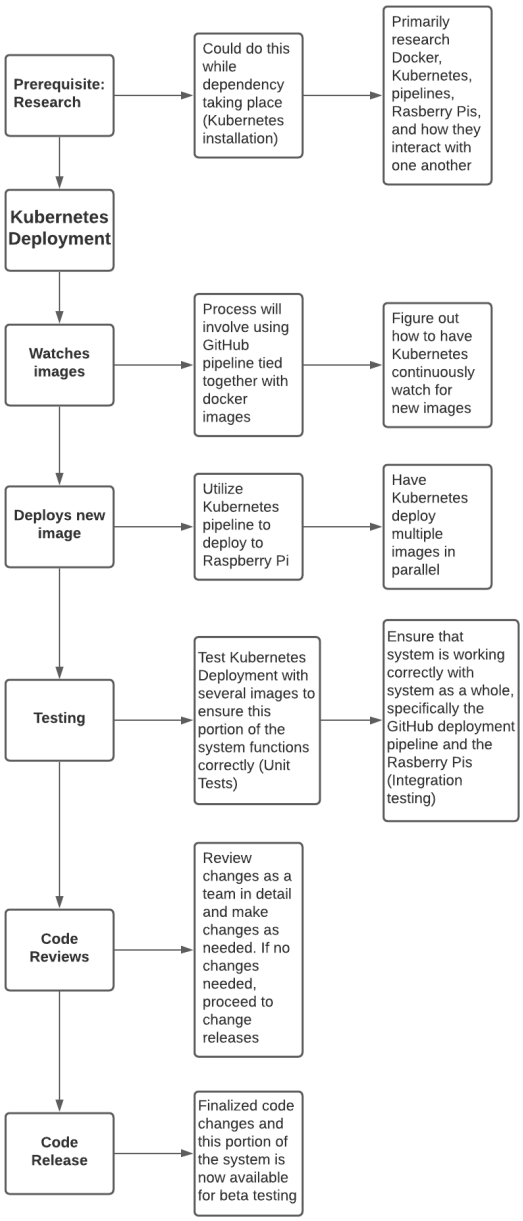


Figure 2: Individual project deliverables

The image above is a graphic outlining individual project deliverables. Arrows that point down describe the order in which the tasks are to be completed, whereas arrows to the right go into more detail as to what the individual task is.

Section 2: User Stories

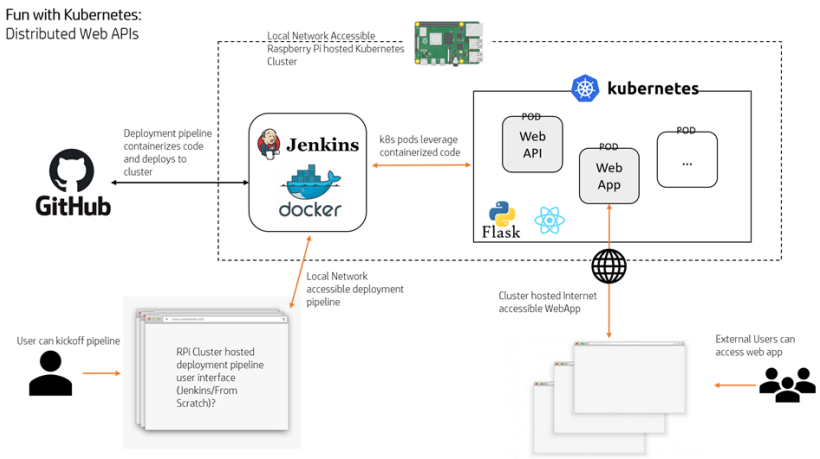
- User Story 1:
  As a user, one needs to be able to go into the project dashboard and submit a config file to the pipeline so one can customize how the application works to tailor it to my use case.
- User Story 2:
  As a user, one needs to be able to see the status of the project being uploaded to the pipeline so one can handle potential errors accordingly.
- User Story 3:
  As a user, one needs to be able to track metrics, such as API requests or daily hits, through a user-friendly dashboard in order to analyze the application's performance.
- User Story 4:
  As a user, one needs to be able to run my application locally through the Raspberry Pi network in order to debug and test the application before publicly deploying it.
- User Story 5:
  As a customer, one needs to be able to access the deployed application from my personal computer in order to utilize the application deployed from the Raspberry Pi network.

Section 3: Personal iteration plan and estimations

Based on team discussions, implementation of these user stories (Figure 1) will begin Winter 2022. Many of the major tasks are dependent on research of topics such as Kubernetes, Docker, and deployment pipelines, so the team will use the remainder of the Fall 2021 term to focus on becoming familiar with these technologies.Several of the user stories also have dependencies. For example, both User Stories 4 and 5 are

dependent on the entire application pipeline being set up. If a team member is blocked on a current User Story, one should plan on assisting their teammate on the blocker rather than attempting to switch to another User Story, unless it proves to be inefficient for more than one teammate to work on that task. For the individual project deliverables (Figure 2), implementation will also begin Winter 2022. This is because these tasks are dependent on the Kubernetes Installation (Figure 1), which is expected to be completed by the end of Fall 2021. While Kubernetes is being installed, the prerequisite research needed for the Kubernetes Deployment can be completed (Figure 2).

Section 4: Solution Architecture



The architecture was designed by the project partner, Kyle Prouty. The design is intended to be a framework built on Raspberry Pis that developers can use to test their application on before deploying it onto their primary platform. The framework is versatile because of the deployment pipeline, which allows users to customize how the framework reacts if their code is pushed to GitHub, if tests fail, or other unexpected behavior occurs. Although Kyle designed this project to make local testing for developers easier, he also kept in mind that our team is comprised of students and chose relevant technologies for us to research and get experience with.

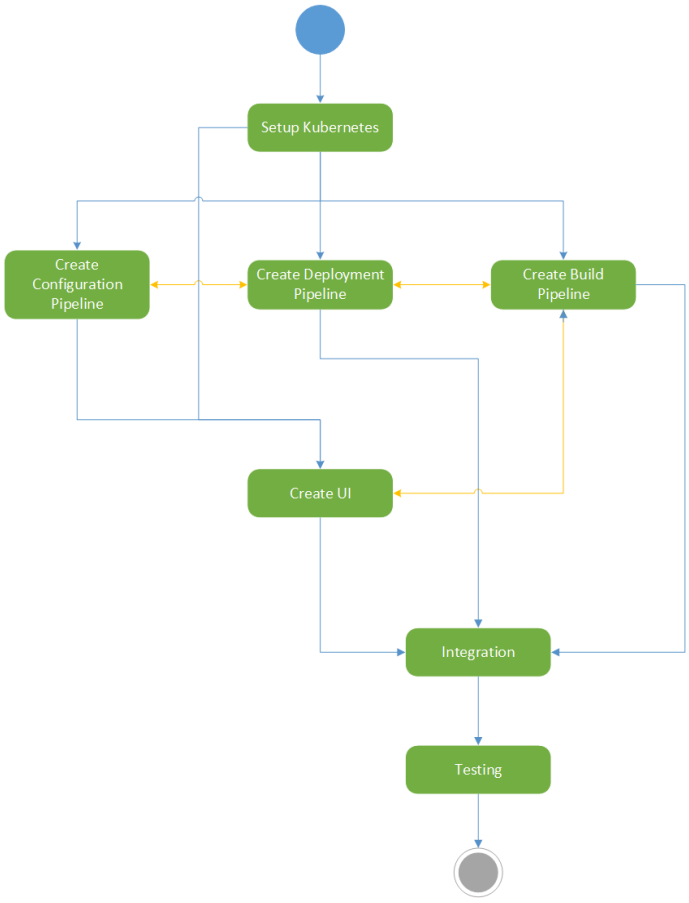
7.5 Nathan Hausman

Individual Contribution to Kubernetes Capstone Project

Nathan Hausman

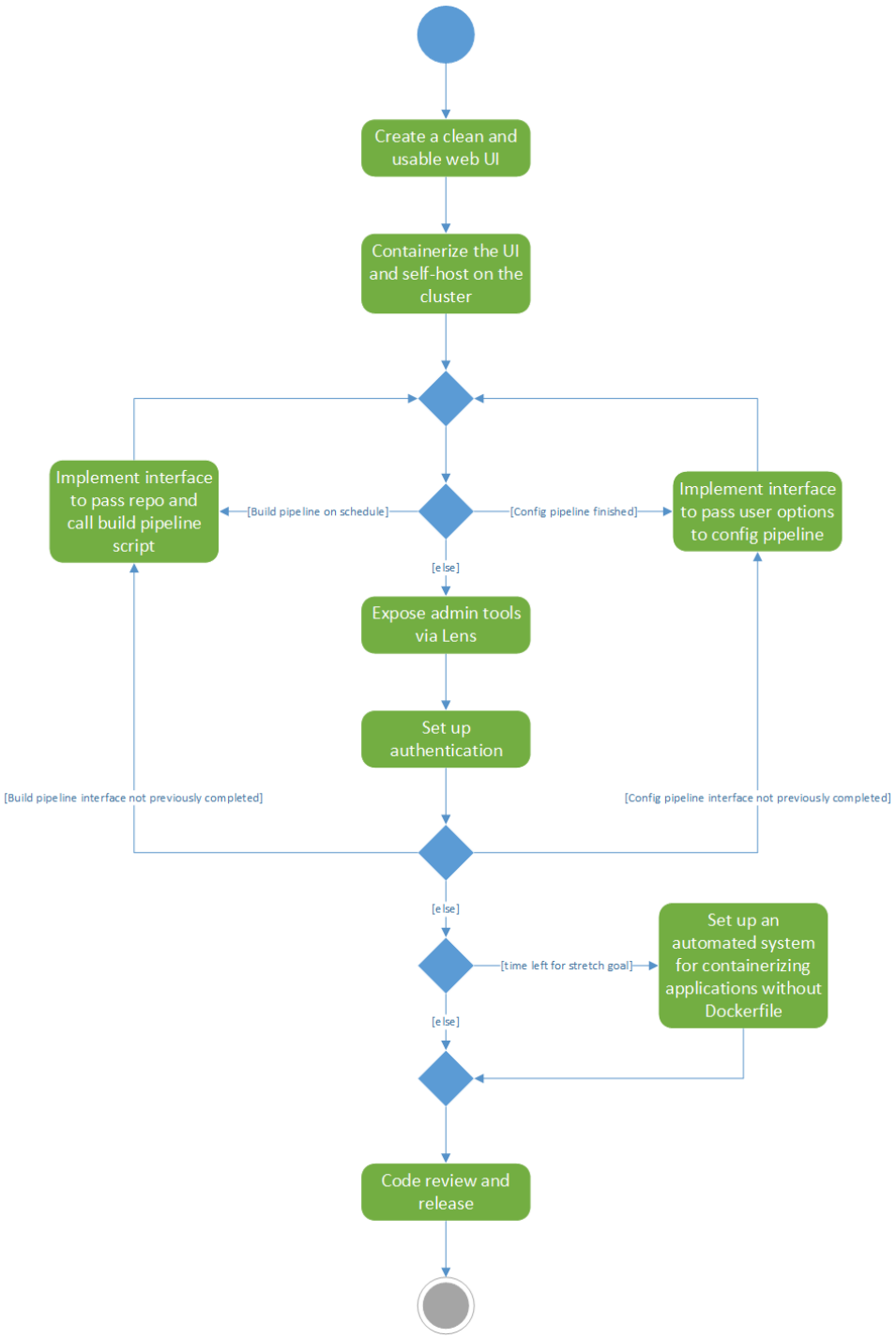
Section 1: Process Flows

Overall System: (Yellow lines indicate flexible integration rather than strict dependency)



Hausman 2

Pipeline UI:



Hausman 3

Section 2: User Stories

User Story #1:

As a user,
I need to pass an application to the pipeline through the UI
So that I don't have to spend time containerizing or setting up deployment to Kubernetes.

User Story #2:

As a user,
I need to access data and interact with the deployed applications in an intuitive way through the UI
So that I can more efficiently work with my applications without needing to interact directly with Kubernetes.

User Story #3:

As a user
I need security protecting the pipeline UI
So that only authorized individuals may modify or add services to the Kubernetes cluster.

User Story #4:

As a developer
I need certain settings from the user when passing an application
So that it can be deployed to Kubernetes properly.

User Story #5:

As a sponsor
I need a clean and usable UI for accessing the pipeline
So that it is easy to use and understand for users who may not have in-depth knowledge of Kubernetes or Docker.

Section 3: Personal Iteration Plan

I have decided to begin this section of the project at the start of winter term and I estimate it to require all of sprint 4 and 5 (1/3 to 2/13). My current estimates for the different sections of the project are as follows:

- Create a clean and usable web UI **(3-6 days)**
- Containerize the web UI and self-host on the cluster **(1 day)**
- Create an interface to pass applications as well as necessary configuration settings to the pipeline for deployment **(3-6 days)**
- Expose admin tools, such as Lens IDE **(2-7 days)**
- Set up authentication **(3-5 days)**

- Set up an automated system for containerizing an application to be passed to the pipeline that is not yet containerized **(2-4 days) [stretch]**
- Code reviews and release **(4-5 days)**

As with every section of this project, my team and I will break these tasks down into smaller chunks as the need arises to split up work.

A few parts of this section are interconnected with other parts of the project. While most can be integrated while being developed in parallel, there are three aspects of the project that need to be completed before I can begin development.

The first requirement is that Kubernetes and Docker need to be installed and set up on the devices, as I will be containerizing the UI and hosting it on the cluster. This should be completed well before I begin work on the UI as all other sections of the project depend on this being complete.

The second requirement is that the config pipeline section of the project must at least have the required user inputs defined, so that I can create the form to input the user's options. Ideally, the config pipeline should be completed so that I can quickly begin work on passing the user options to it, but this is more flexible and can be moved around if the config section of the project is delayed for some reason.

The third requirement is that Kubernetes Lens must be installed and connected to the cluster. Right now, the details of this are currently uncertain, as we are researching how exactly we intend to connect it and expose it via the UI. This research needs to be completed and Lens set up before sprint 5, when I begin working on that portion of the project.

Additionally, there is some research that I must do prior to beginning this section of the project. This research includes deciding on a web framework for the UI, how to connect and expose information about Kubernetes and its services as well as admin tools through Lens, and how security and authentication should be implemented. I plan on working on this research during sprint 3 and winter break. During this time, I will also help my team members with setting up Lens and working on the Kubernetes configuration pipeline, as both are involved with my section of the project. However, it is unlikely I will be able to help the other sections of the project much, as I have the longest section of the project, and will therefore be relying on the help of others who finish their parts first instead. After my section is done, we will all be working together on finishing the integration and testing the entire pipeline, as well as making any changes necessary to improve and refine the product.

Section 4: Solution Architecture

It was decided to deploy the UI as a Docker container like any of the other services that run on Kubernetes so it can take advantage of the pre-existing infrastructure to deliver the service.

Hausman 5

The Lens IDE was suggested as a way to display a dashboard for controlling the cluster by the project partner. However, if a better solution is found during research, it is possible that it may be replaced.

The method of authentication has not yet been decided, and will have to be researched, however simply opening up the pipeline UI to any connection is a security risk, and thus some sort of login is necessary.

8 Appendix: Winter Term Updates (3/15/2022)

After discussing the state of our project with our project partner and stakeholder, we have all agreed the current project is still inline with the vision and goals detailed in the previous version of this document constructed during Fall Term 2021.