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

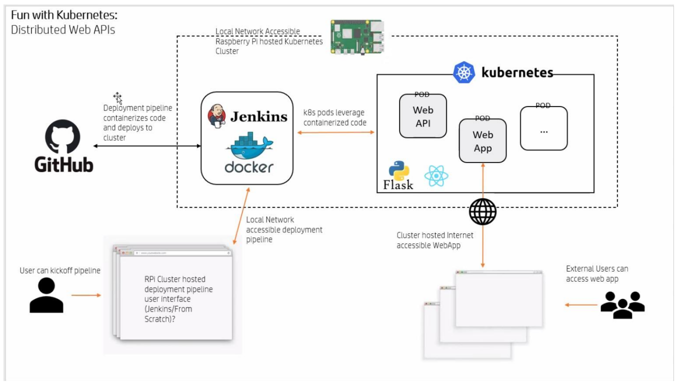
Fun with Kubernetes:
Distributed Web APIs
Local Network Accessible Raspberry Pi hosted Kubernetes Cluster
kubernetes
Deployment pipeline containerizes code and deploys to cluster
GitHub
Jenkins
docker
k8s pods leverage containerized code
POD
Web API
POD
Web App
POD
...
Flask
Local Network accessible deployment pipeline
Cluster hosted Internet accessible WebApp
User can kickoff pipeline
RPi Cluster hosted deployment pipeline user interface (Jenkins/From Scratch)?
External Users can access web app

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

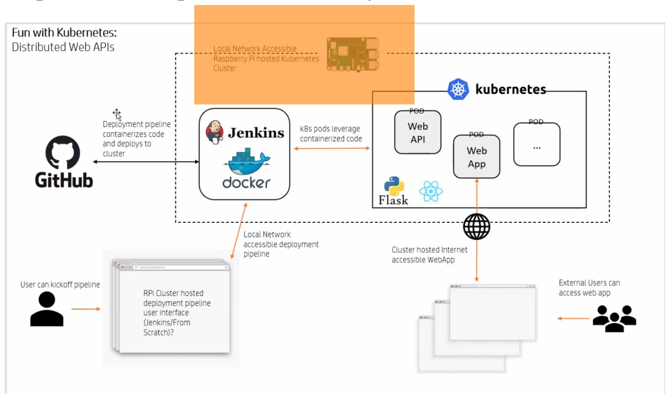
Fun with Kubernetes:
Distributed Web APIs
Local Network Accessible Raspberry Pi hosted Kubernetes Cluster
kubernetes
Deployment pipeline containerizes code and deploys to cluster
GitHub
Jenkins
docker
k8s pods leverage containerized code
POD
Web API
POD
Web App
POD
...
Flask
Local Network accessible deployment pipeline
Cluster hosted Internet accessible WebApp
User can kickoff pipeline
RPi Cluster hosted deployment pipeline user interface (Jenkins/From Scratch)?
External Users can access web app

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

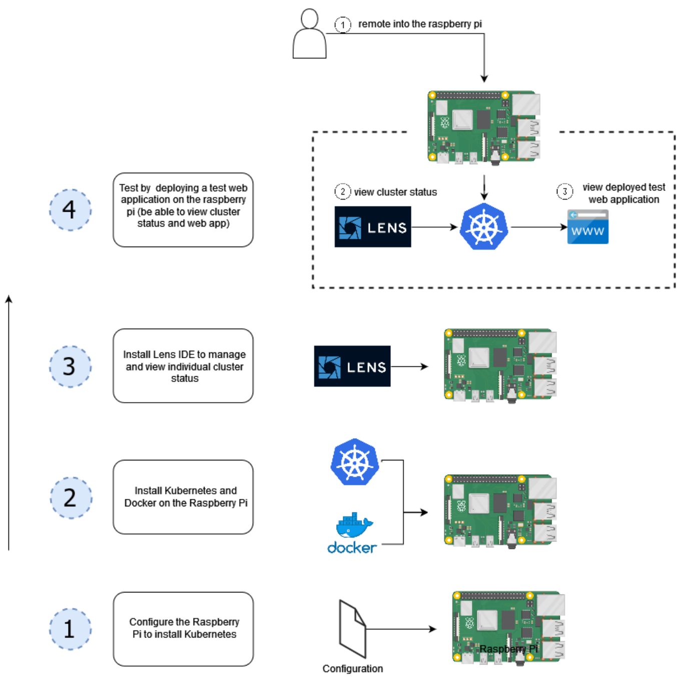
1 remote into the raspberry pi
2 view cluster status
LENS
3 view deployed test web application
WWW
4
Test by deploying a test web application on the raspberry pi (be able to view cluster status and web app)
3
Install Lens IDE to manage and view individual cluster status
LENS
2
Install Kubernetes and Docker on the Raspberry Pi
docker
1
Configure the Raspberry Pi to install Kubernetes
Configuration
Raspberry Pi

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].