**TAKE HOME CHALLENGE**

**Containerization, Deployment & Orchestrations of a Simple Ruby application**

Submitted By,

**DELWIN NORONHA**

delwin.noronha1007@gmail.com

Date: 19 Dec 2023

Contents

[**CHALLENGE** 2](#_Toc153967704)

[**APPROACH AND DESIGN CONSIDERATIONS** 4](#_Toc153967705)

[**SETUP** 5](#_Toc153967706)

[**PRE-REQUISITES** 6](#_Toc153967707)

[**INSTALLATION** 6](#_Toc153967708)

[**VALIDATION** 7](#_Toc153967709)

[**DEPLOY NEW IMAGE – The GitOps Way** 10](#_Toc153967710)

[**CLEANUP** 13](#_Toc153967711)

[**RESULTS** 14](#_Toc153967712)

[**CONCLUSION** 15](#_Toc153967713)

# **CHALLENGE**

Below, you will find the challenge(s) that we would like you to work on. We’re looking forward to seeing your skills put into action, and the task will also offer you more insights on the role and industry that we operate in.



<https://github.com/sawasy/http_server/blob/main/http_server.rb>

Overview

Given the provided Ruby app, design a pipeline and deployment strategy following the requirements illustrated below.

Modules

1. Pipeline design
2. Deploy
3. Improvements

**Module 1 - Pipeline design**

We have a homogenous mixture of environments based on cloud as well as on-premise Kubernetes clusters. Some of our applications run on multiple instances while others are standalone. The configuration of the auxiliary services (databases, caches etc.) might be different depending on the environment.

Design a pipeline that fits the needs of such an infrastructure for our app. (GitHub actions, helm, Argo CD are preferred).

**Module 2 - Deploy**

Run the app on local k8s cluster.

Part 1 - Containerize

Build a secure, scalable and robust container image.

Part 2 - Manifests

Write helm manifests to deploy the application to local Kubernetes cluster. Should satisfy following:

* Highly available and load balanced
* Ensuring the application is started before served with traffic
* Safeguards for ensuring healthy life cycle of applications
* Ensure zero downtime
* Endpoints of the web application are accessible outside the cluster

**Module 3 - Improvements**

This is an open-ended assignment, you are free to introduce changes, in the assigned time frame, to the application or in the instrumentation to meet your standards in terms of security, availability, reliability and observability.

# **APPROACH AND DESIGN CONSIDERATIONS**

The below considerations were taken to satisfy the mentioned requirements.

1. Containerize the application – This can be achieved using **Docker**. Below best practices are followed
   1. Use smallest possible base image / multistage build – to reduce image size + security
   2. Prevent root access in container - Security
   3. Order commands in Dockerfile to obtain highest level of caching – this will reduce build time and image size.
2. Manifests – **Helm** is used to manifest the deployments to Kubernetes cluster
3. Auxiliary services configuration – Can be achieved using Helm values. We can have different **helm values files** for different environment / applications. This will allow to configure different environment specific values like
   1. Database hosts
   2. Cache details
   3. Any other configurable endpoints, etc.

We can also make use of **secret management** tools like Vault, Secret Manager, GitHub secrets etc. to store and retrieve environment specific sensitive data like passwords.

1. Deploy to multiple Clusters – We can use **Argo CD** tool to manage multi-cluster deployments (both cloud and on-prem). This will follow GitOps and make it very easy to deploy applications to multiple clusters with least effort.
2. Local deployment – **K3D** Kubernetes is used. It has its own local container registry as well, where images can be imported. Docker in docker concept.
3. Load balancing – Using K3D provided **Traefik** LB.
4. CICD Pipeline – **Github actions** is used.
5. For high availability – **Autocscalers**, multi pods, affinity rules. (additionally – regional clusters, capacity planning, logging, monitoring, alerting etc)
6. **Readiness and Liveliness probes** – Used to make sure application is started before serving traffic, and ensure application health is regularly checked.
7. Zero downtime – Achieved using **rolling updates** and multiple pod replicas**.**
8. Application endpoints accessible outside cluster – Using **loadbalancer** service, ingress
9. TLS is not used, as this is a local setup. It is always recommended to used TLS encryption for production workloads.
10. This application is currently tested only in Linux / amd64 machine. Docker base image would require to be changed to support other architectures.
11. The application deployment is manifested with helm using minimal required values.
12. Applications will run on local host network. (localhost – 127.0.0.1)
13. ArgoCD UI will be exposed using port forwarding, as this is local setup.
14. Argo CD sync time has been set to 5s, it requires 1 time login.
15. The setup is limited with features required only for the challenge requirements.
16. Have fun implementing this setup!!

Additional Improvements / Considerations

1. **Security** – Enable RBACs in clusters, Use secret management tools, code scanning, proper auth processes, TLS, cert-manager.
2. **Reliability** – Logging, monitoring, alerting, tracing, auto scaling
3. **Observability** – We can use tools like Prometheus, Grafana, Jaeger for monitoring, alerting and tracing. This won’t be done for this challenge, considering the limited time given.
4. **Availability** – Multi region deployments, disaster recovery plans, capacity planning

# **SETUP**

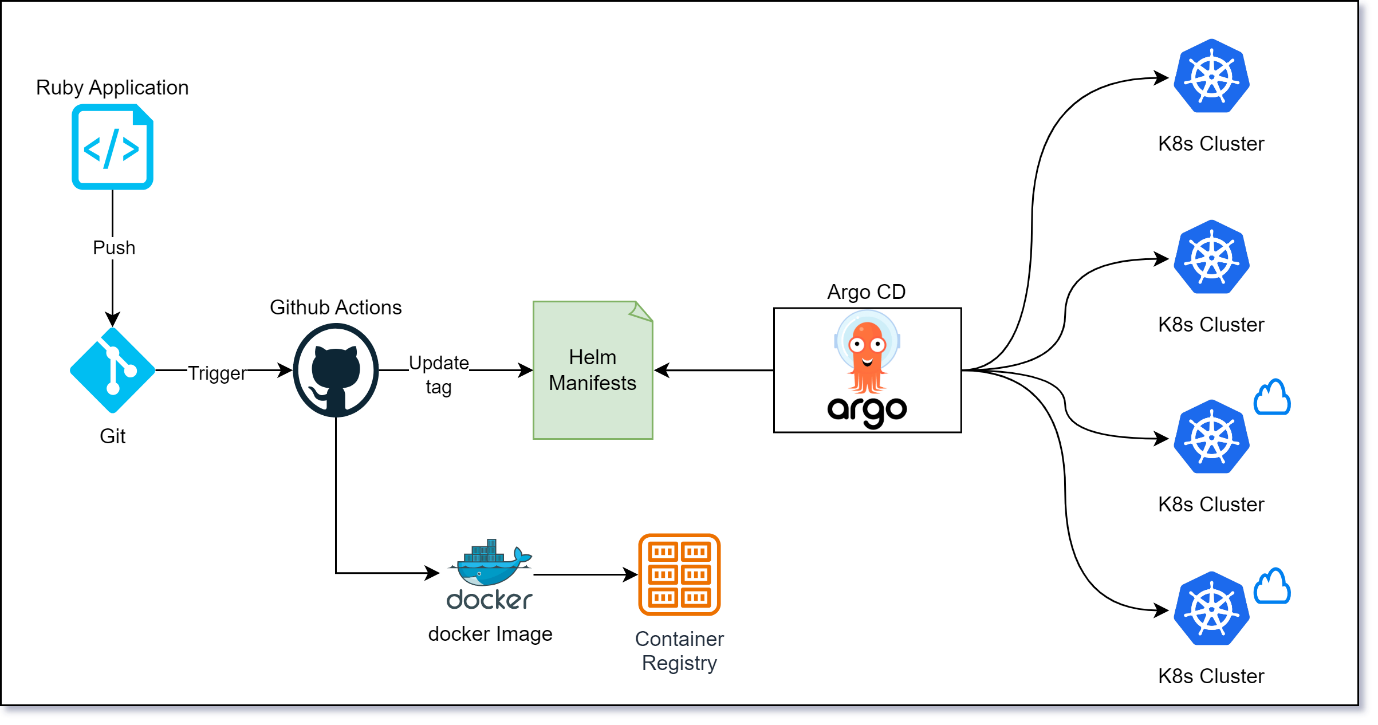
The below tools will be used

1. **Docker** – For containerization
2. **K3D** – To deploy K8s cluster locally and for local container registry.
3. **Helm** – To manifest Kubernetes deployments
4. **Argo** **CD** – For Kubernetes deployments

To install the setup, follow instructions mentioned in **README** attached with the code.

Detailed installation guide with screenshots provided in this document.

Below is a high-level diagram of this setup.



## **PRE-REQUISITES**

Make sure below tools are present in the system,

* Kubectl - to interact to Kubernetes cluster. <https://kubernetes.io/docs/tasks/tools/>
* Docker engine
* Bash
* Helm - https://helm.sh/docs/intro/install/
* Argo CD CLI (optional)
* K3D – To deploy K8s cluster locally and for local container registry - https://k3d.io/v5.6.0/#installation

Internet connection is required to download the necessary docker images and dependencies.

\*\* Note: If this setup is being run behind a proxy, please make sure the necessary proxy configuration is done in the system.

## **INSTALLATION**

1. Clone / Download the GitHub repo to your local - [**https://github.com/dnoronh/ruby-app-devops.git**](https://github.com/dnoronh/ruby-app-devops.git)
2. In bash, run the install\_setup.sh shell script using command

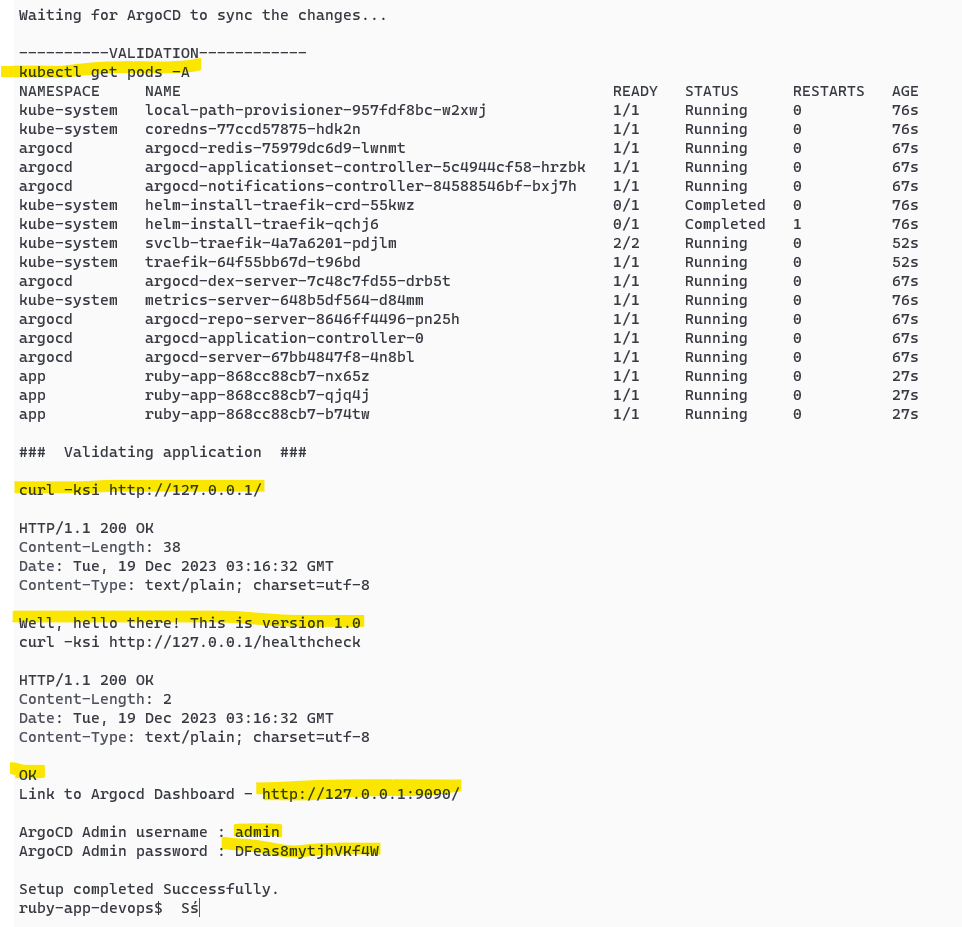
**./install\_setup.sh**

****

The script will perform the below actions

1. Create K3D cluster
2. Create docker image 1.0 for ruby app
3. Create docker image 2.0 for ruby app
4. Deploy Argo CD tool using helm charts
5. Create Argo CD application for ruby app
6. Port forwarding of Argo CD service.

This below can be seen post installation



Please use the mentioned Argo CD **username** and **password** for first time login to UI

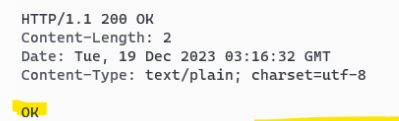
## **VALIDATION**

1. It will take around 1 min for the setup to complete. Once completed please validate if pods are up and running.

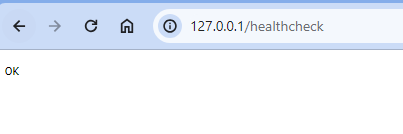
**kubectl get pods -A**

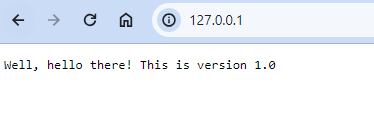
1. Validate if able to reach the ruby application

**curl -kis** [**http://127.0.0.1/healthcheck**](http://127.0.0.1/healthcheck)



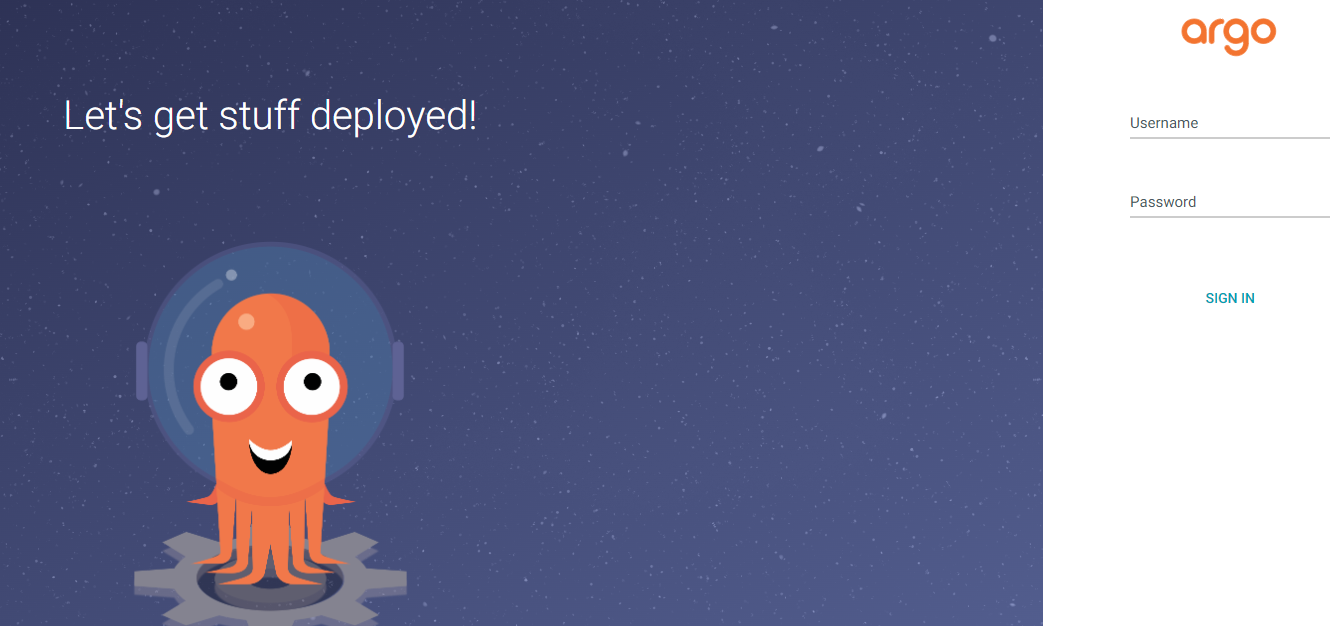
We can validate the same in bowser as well



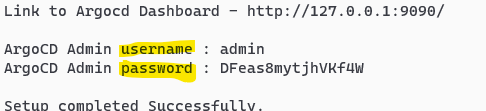


1. Validate if you are able to open Argo CD dashboard using below link.

[**http://127.0.0.1:9090/**](http://127.0.0.1:9090/)

****

Enter the username password provided in the installation output logs

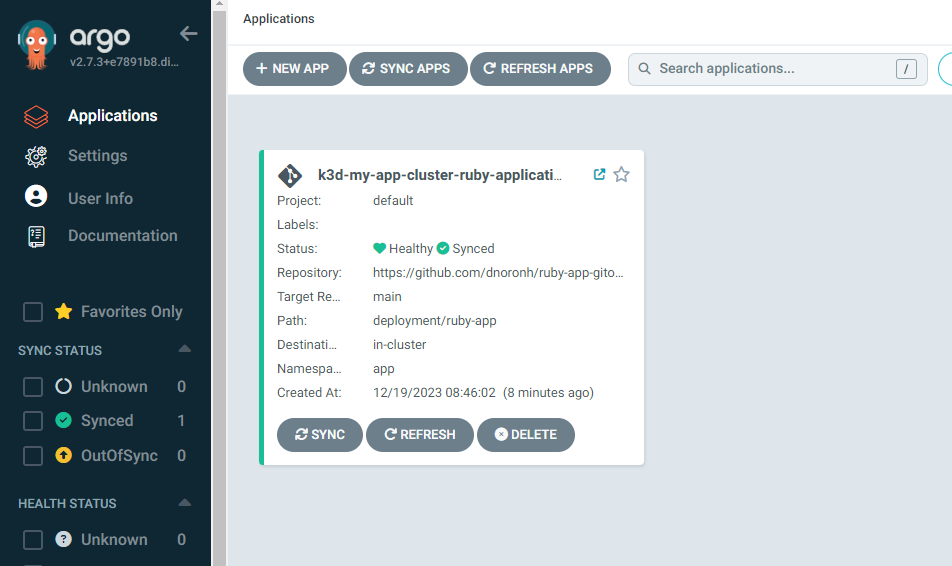
****

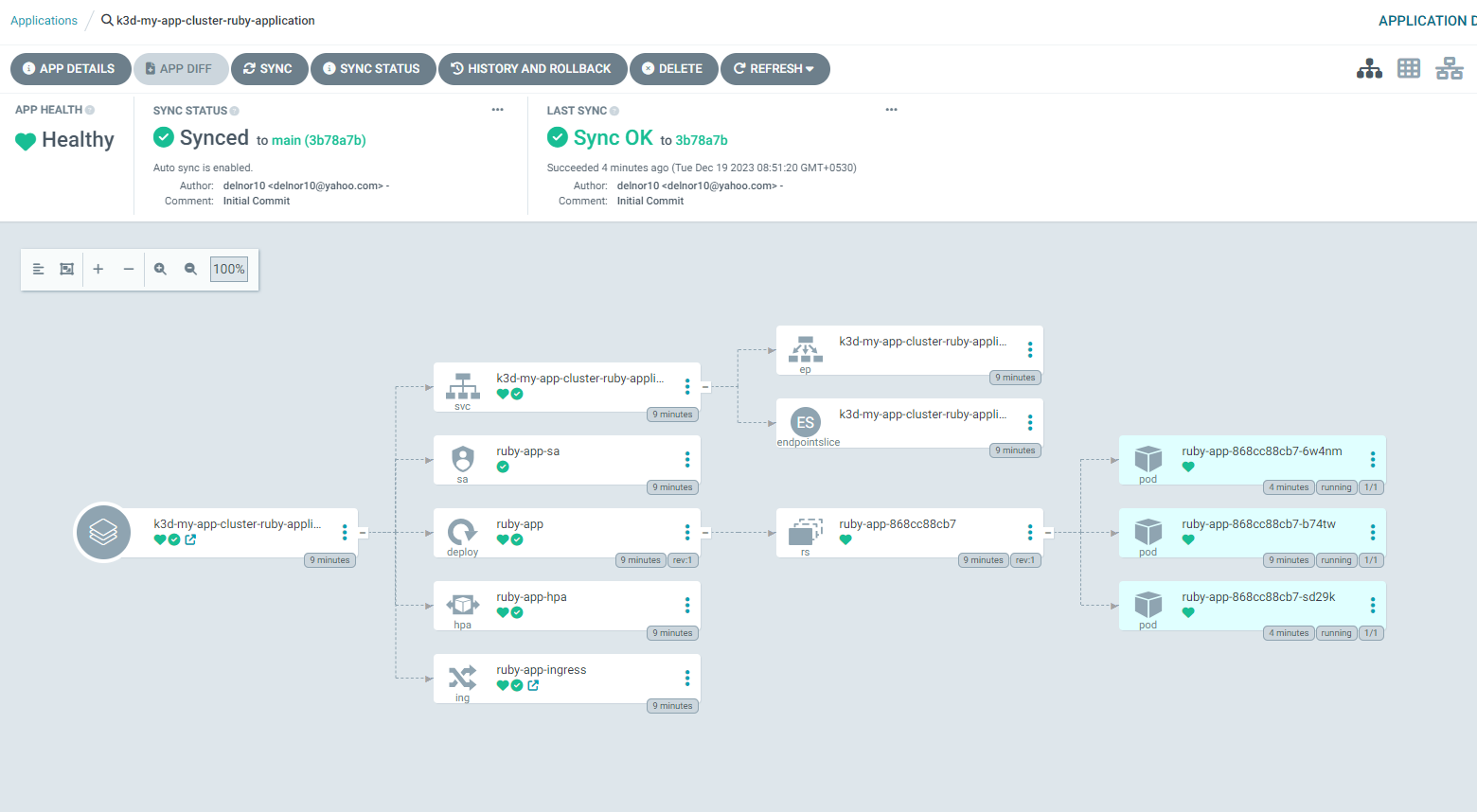
If you are not able to open the UI, please validate if port forward is enabled

Command **ps -f | grep port-forward**

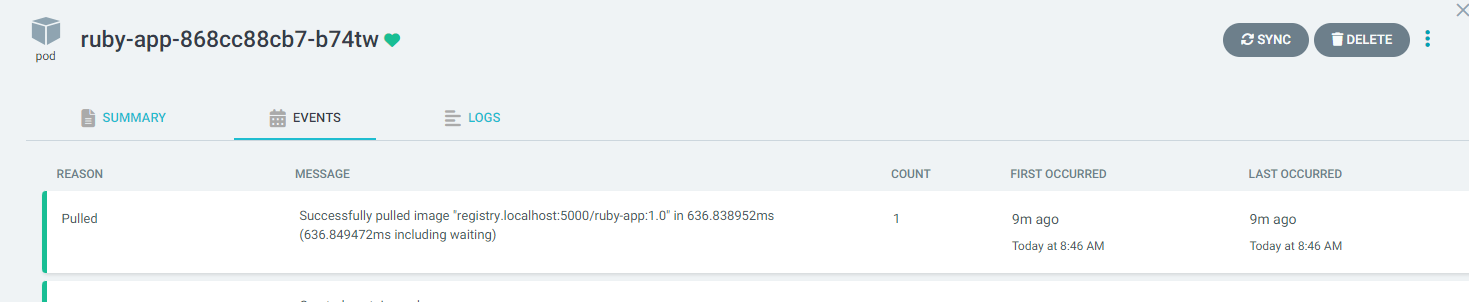
Run command below command to enable port forwarding if not already enabled

**kubectl port-forward svc/argocd-server -n argocd 9090:80 > /dev/null &**

****

****

We can see that image version ‘1.0’ is deployed to Kubernetes via Argo CD



As best practices for GitOps, we have the manifests in a different Gihub Repo. Argo CD will be monitoring this repository. - <https://github.com/dnoronh/ruby-app-gitops-repo>

## **DEPLOY NEW IMAGE – The GitOps Way**

To perform the build and deploy of version 2.0 of the app, we will use **GitHub Actions** workflow.

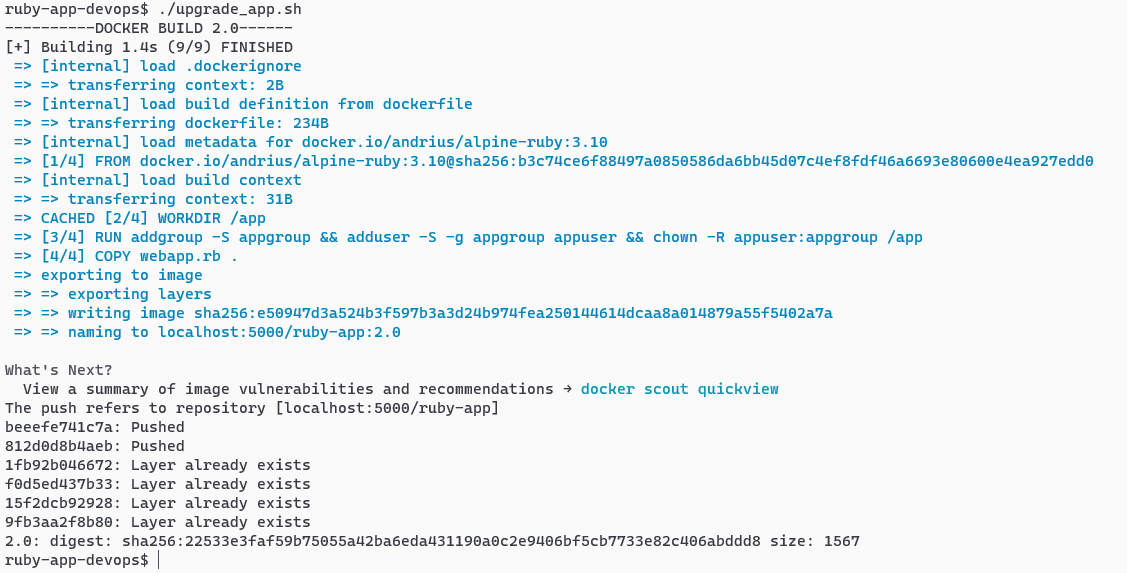
GitHub actions pipeline will perform the below steps

1. Build the docker image
2. Tag and push the image to registry (this step would be done manually in our case, as we are using a local registry)
3. Update the image tag in manifest (helm values) – Ideally it is better to use short **GIT\_SHA** for the tags, but in this demo, we will use the tag ‘2.0’ for the newer version.

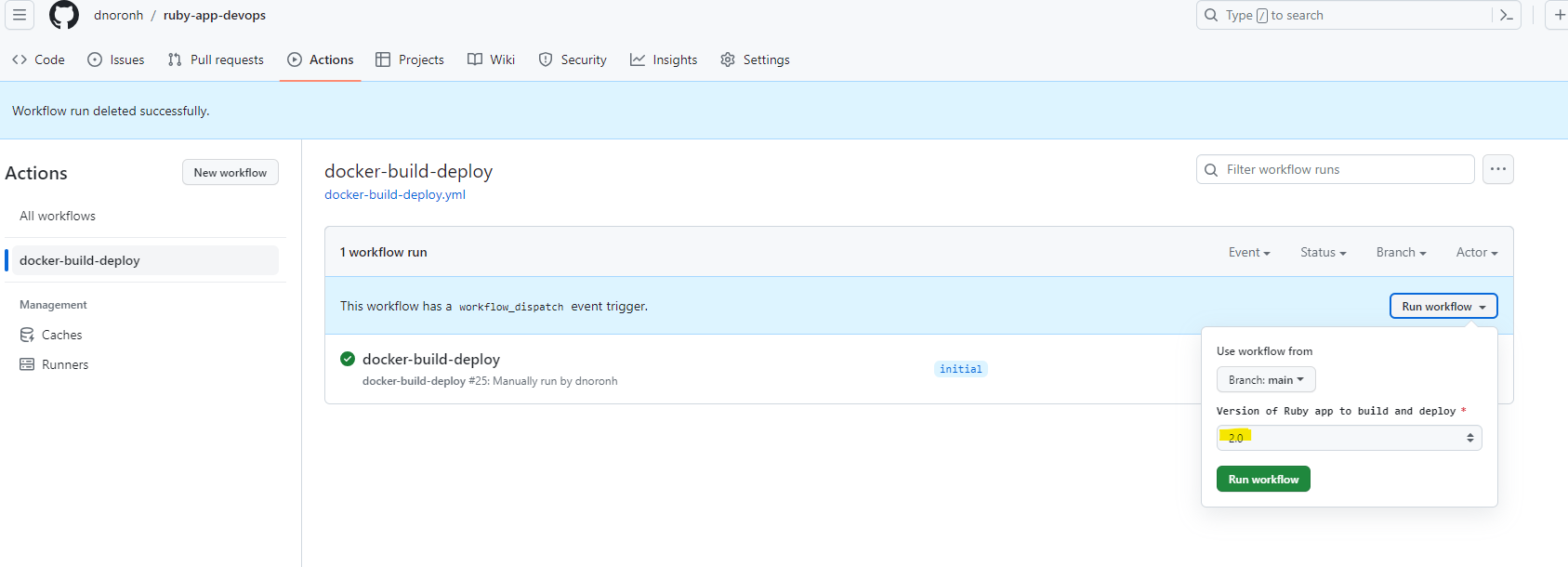
Argo CD will sync the changes to the cluster and deploy the new version 2.0

To demonstrate this in our local cluster,

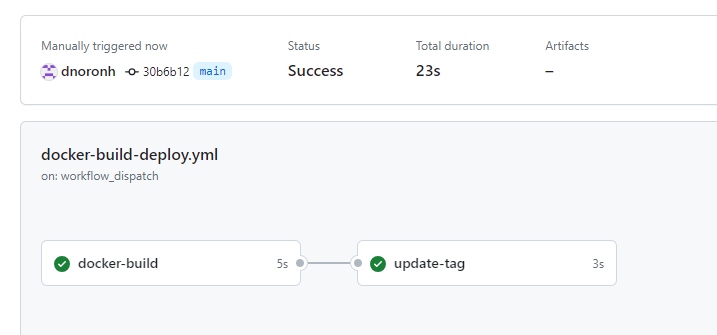
In our case, image 2.0 version is already built during installation. We just need to run the Github actions pipeline to update the tag in helm values.

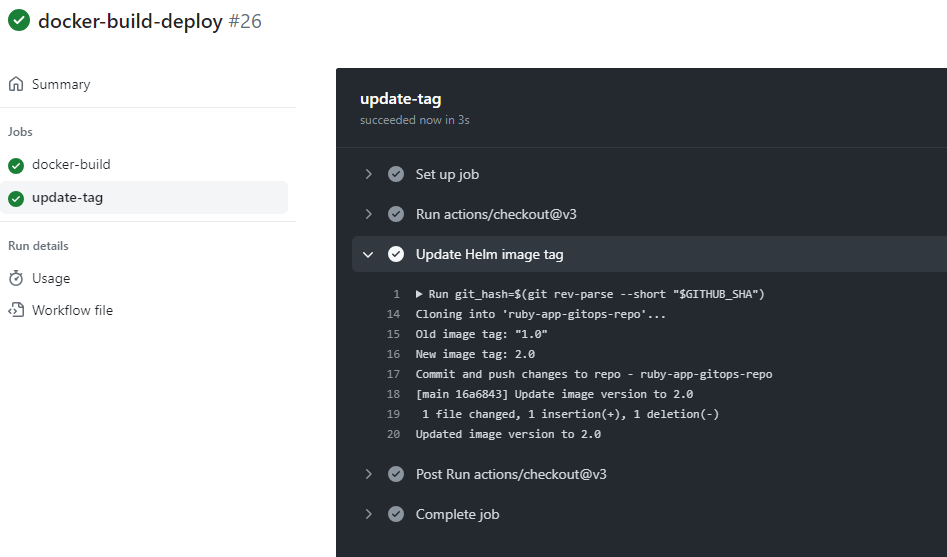


Run the GitHub actions workflow – <https://github.com/dnoronh/ruby-app-devops/actions/workflows/docker-build-deploy.yml>



This workflow will perform a dummy docker build, and update the image version in manifest repo to 2.0

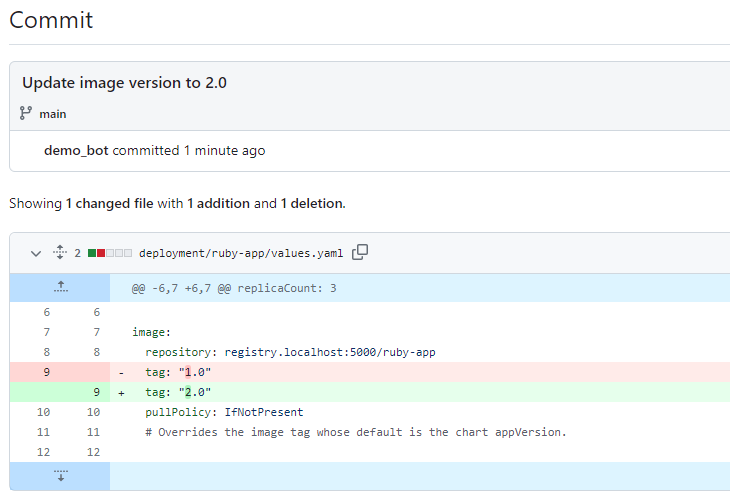




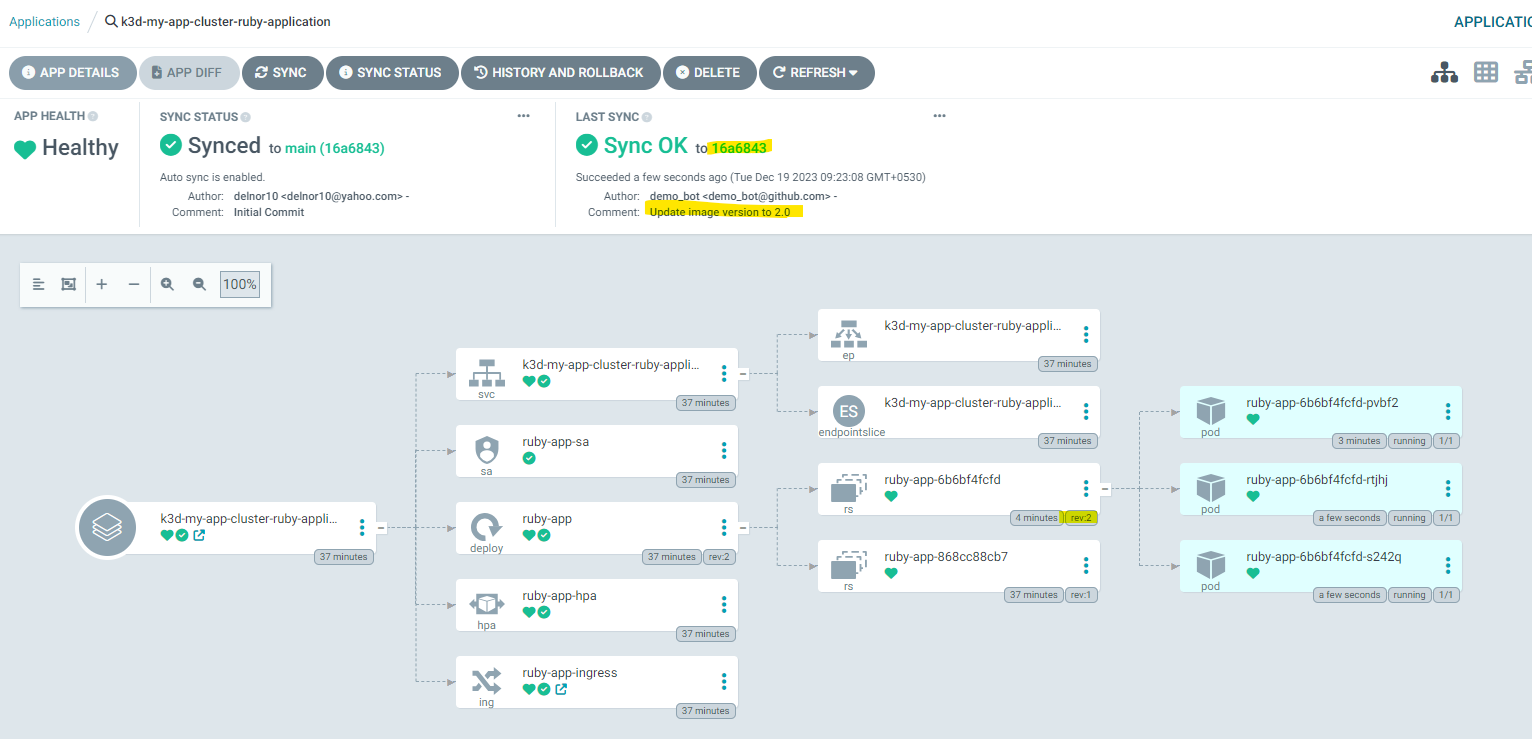
Let’s validate the changes done

Go to <https://github.com/dnoronh/ruby-app-gitops-repo>

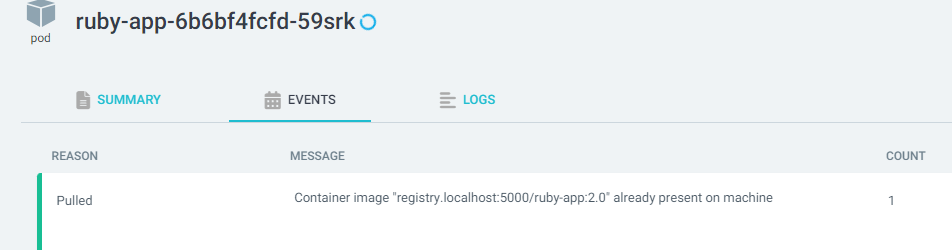
View the latest commit done



Now observe the changes in Argo CD or by sending a request to the app

****

You can observe now that version 2.0 of the app is deployed

****

****

Validate the application endpoints

****

We can observe that Argo CD has deployed the latest version of the image and it is running successfully.

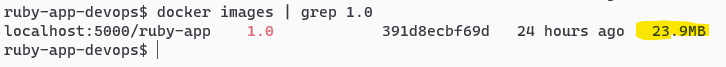
## **CLEANUP**

To cleanup the setup, run the below command

**./install\_setup.sh cleanup**

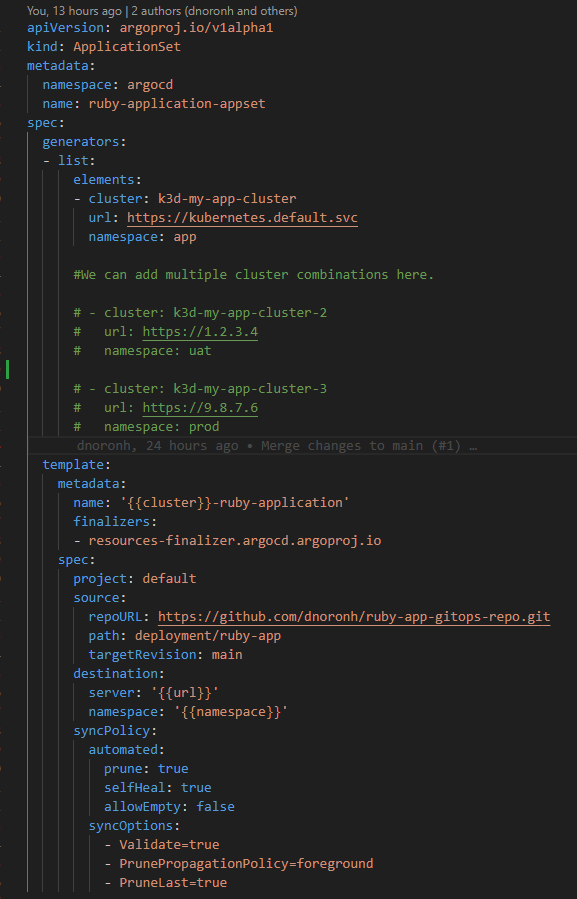
# **RESULTS**

1. **Containerization** – The docker image for the Ruby app is built following industry best practices to achieve lower image sizes and best security. Additionally, non-root access is provided to the container. We can see that the final image is only 24MB. This is a huge advantage when it comes to microservices architecture.



1. **GitHub actions** – With Github actions workflows, we were able to observe the automated docker build as well as updating of the latest image tag in the manifest. This allows continuous build and deployment for faster releases. Additionally, we can add code / container scanning as part of the build pipeline to make the application more secure.
2. **Helm** – Using helm we were able to easily manifest the Kubernetes deployments. Helm values files introduce a flexible approach to configuration, enabling developers to customize settings for each instance and environment effortlessly. This eliminates the need to hardcode configurations directly into the codebase, and the same code can be seamlessly deployed across diverse environments, promoting consistency and efficiency. The values can be any interface connection details, ports, values that differ over environments.
3. **Argo CD** – Using Argo CD and its application Set object we were able to easily deploy the application to the Kubernetes cluster, and we also observed how application Set can be used to deploy the application to multiple clusters and namespaces irrespective of where they are hosted. This flexibility ensures that our application deployment remains agile and adaptable to varying infrastructural demands.

Below is the Application Set configuration used for our deployment. This can be further enhanced to support deployments to multiple clusters. Clusters can be setup in Argo CD using Kubernetes Secrets.



1. **K3D** - is a highly effective tool for swiftly creating local Kubernetes clusters for testing purposes. Its easy configurability and effortless destruction make it advantageous, especially in the context of the Docker in Docker concept.

In this scenario, we could setup multiple K3D clusters and use Argo CD to demonstrate multi-cluster deployments, but due to short time available, I am restricting it to single cluster.

# **CONCLUSION**

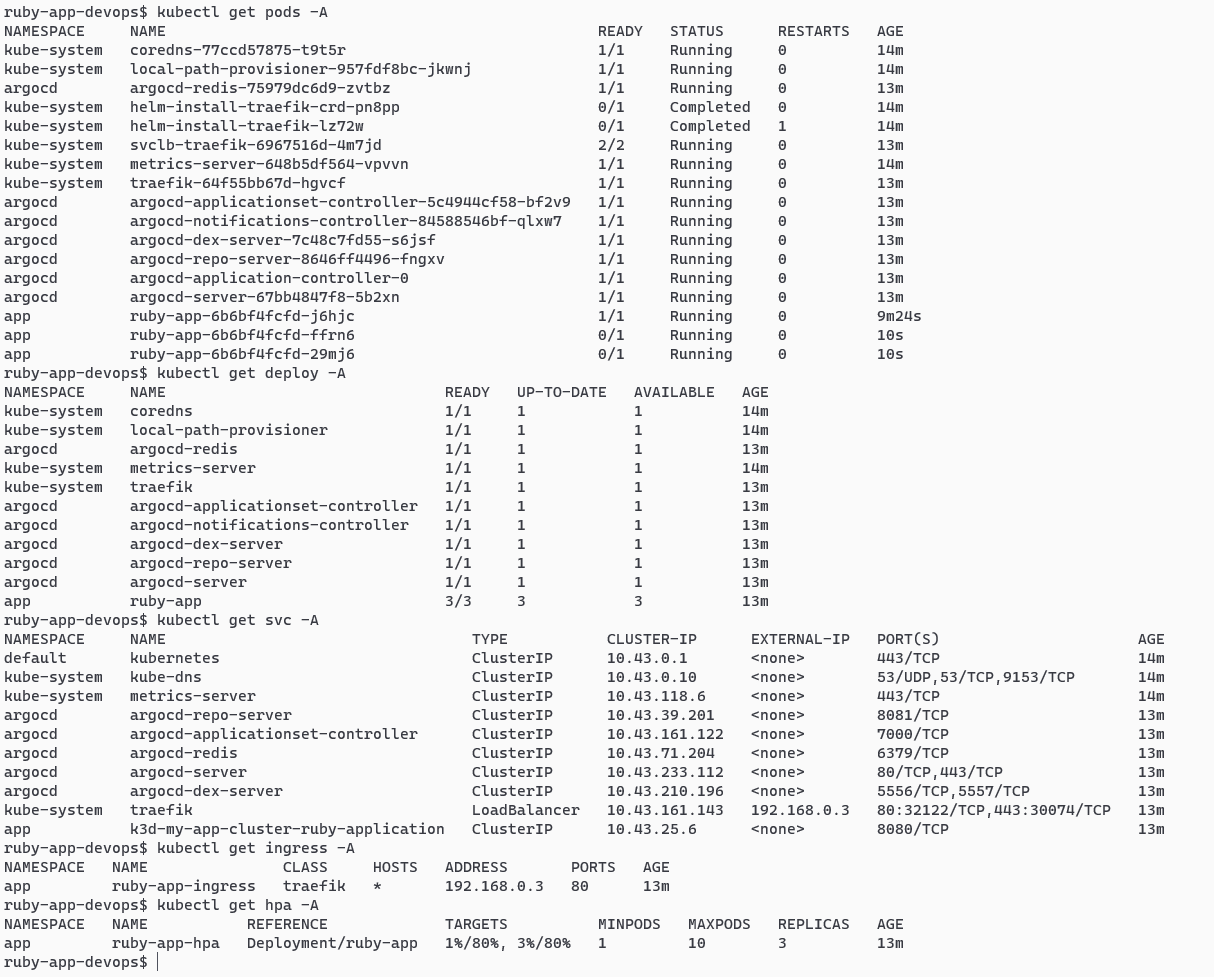
Through this challenge, a flexible solution was devised for the continuous build and deployment of a Ruby application in a hybrid, multi-cluster Kubernetes environment.

The pipeline, orchestrated using GitHub Actions, automated the containerization of the Ruby application based on industry best practices. Helm was instrumental in generating Kubernetes manifests, providing flexibility for deployments tailored to specific instances and environments. Argo CD streamlined the deployment process following the GitOps model, ensuring the deployment of the latest application version across relevant instances.

The integration of observability tools such as Prometheus, Grafana, Jaeger, and Datadog further enhances the reliability of applications hosted in the clusters. This deployment and improvement strategy positions the Ruby app for sustained success in dynamic and demanding operational landscapes.

Few Additional screenshots

Few relevant K8s objects



Screenshots from argoCD

