PropPulse CI/CD deployment (GitHub, Jenkins, DockerHub & Minikube)

Version	1.0.1
Prepared by	Rimah Houssameldine
Audience	ParkInnovation Team
Date	23/02/2024

Table of Contents

A. Introduction:

This guide walks through deploying a PHP application using GitHub, DockerHub, Jenkins, and Minikube. It covers setting up a GitHub repository, configuring DockerHub, implementing Jenkins for automation, and utilizing Minikube for local Kubernetes deployment. By following this documentation, users can establish a seamless CI/CD pipeline for efficient development, testing, and deployment of their PHP application.

1. Requirements:

- PC (Desktop, Laptop)
- Git, GitHub, Docker engine, DockerHub, Jenkins & Minikube
- VS Code
- Browser and Internet

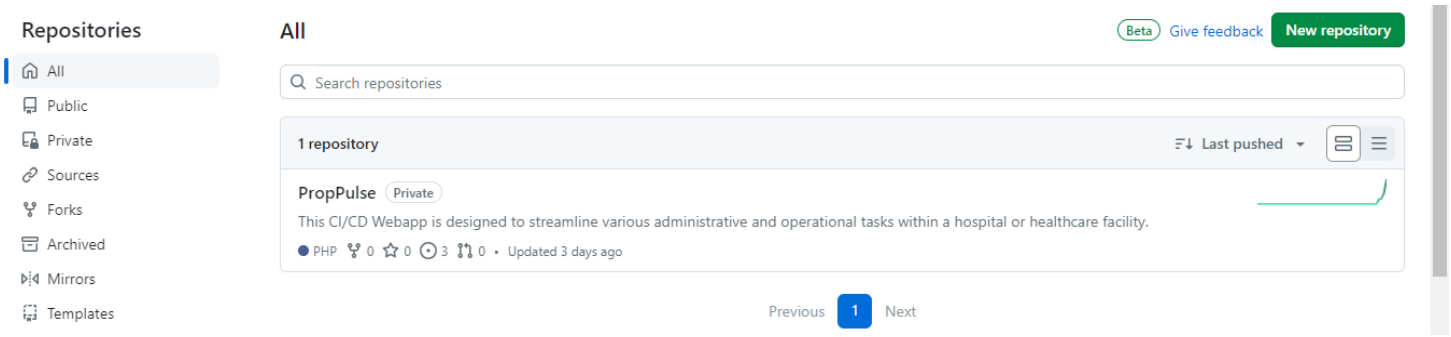





B. Steps for deploying PopPulse

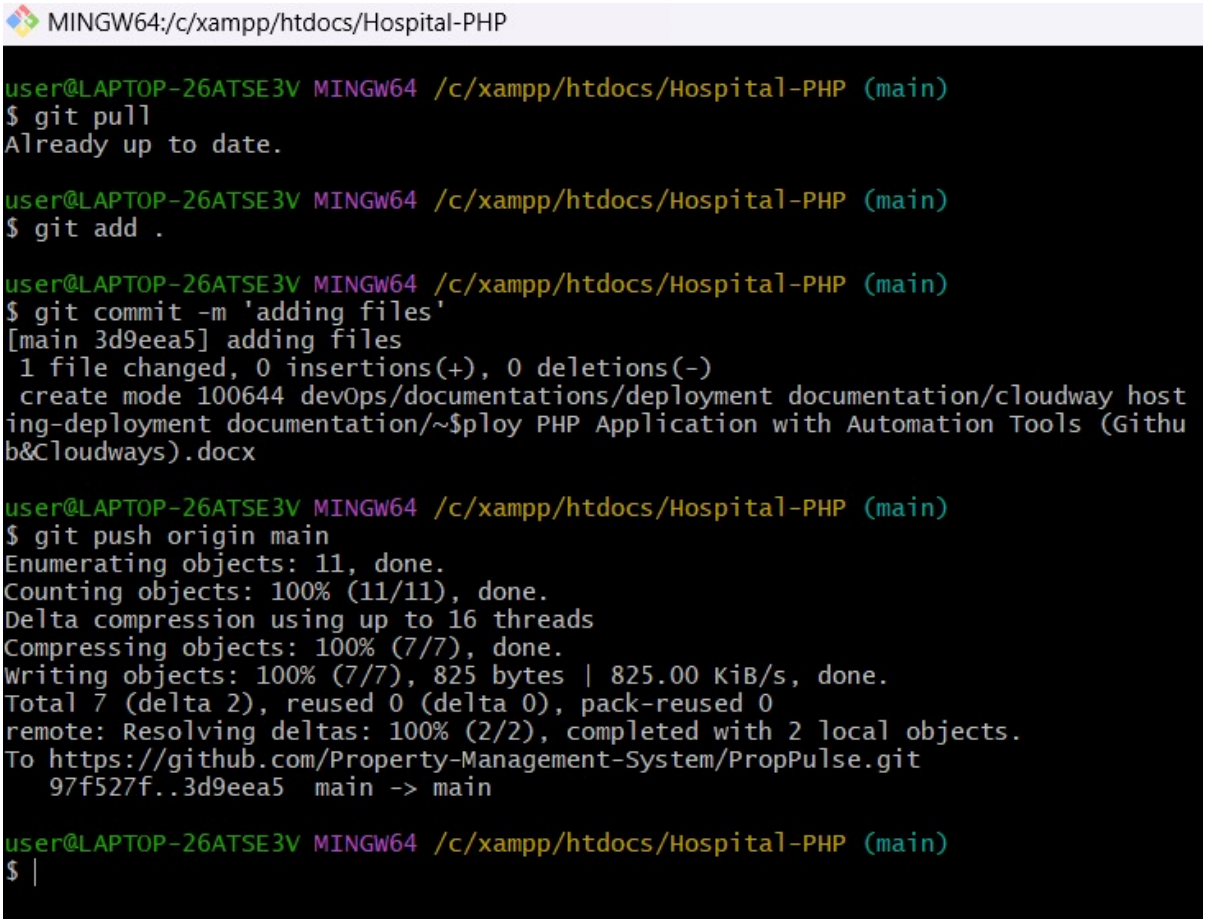
1. Create GitHub Repository

To create a new GitHub repository, navigate to your GitHub account, click on the "+" icon in the top-right corner, and select "New repository", then follow the prompts to configure repository settings and create it.



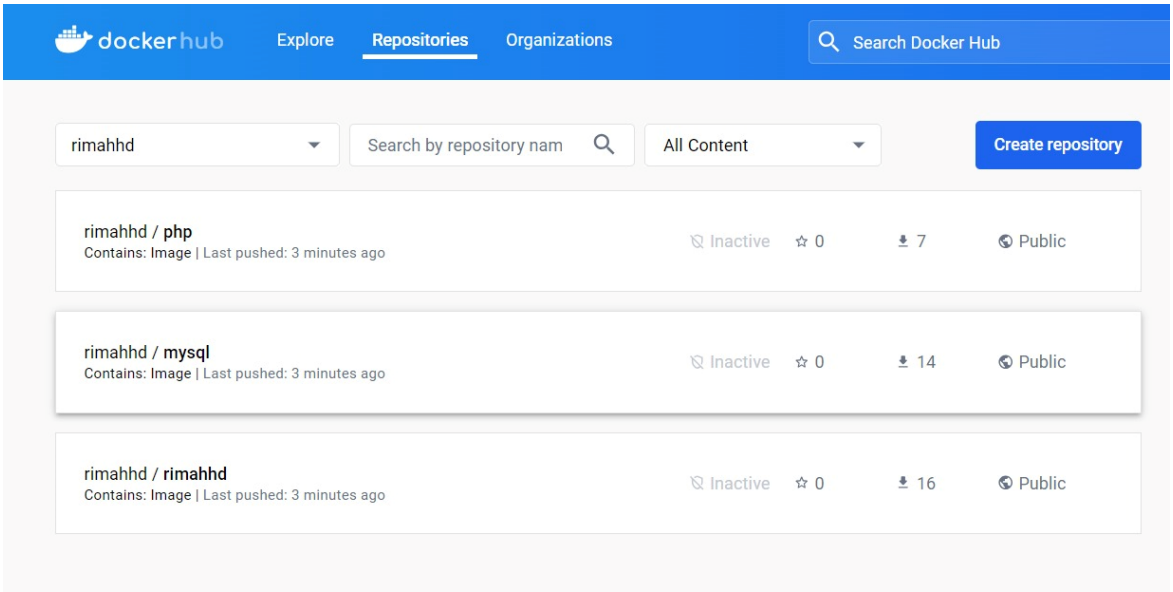
2. Push local files to the created repository

To push local files to a GitHub repository, first ensure you have initialized a Git repository locally using **git init**. Then, add your files using **git add .**, commit them using **git commit -m "Your commit message"**, and finally, push them to the GitHub repository using **git push origin master** (replace "master" with your branch name if applicable). Make sure you have the appropriate permissions to push to the repository and that you have configured the correct remote URL for your GitHub repository.



3. Create a DockerHub repository

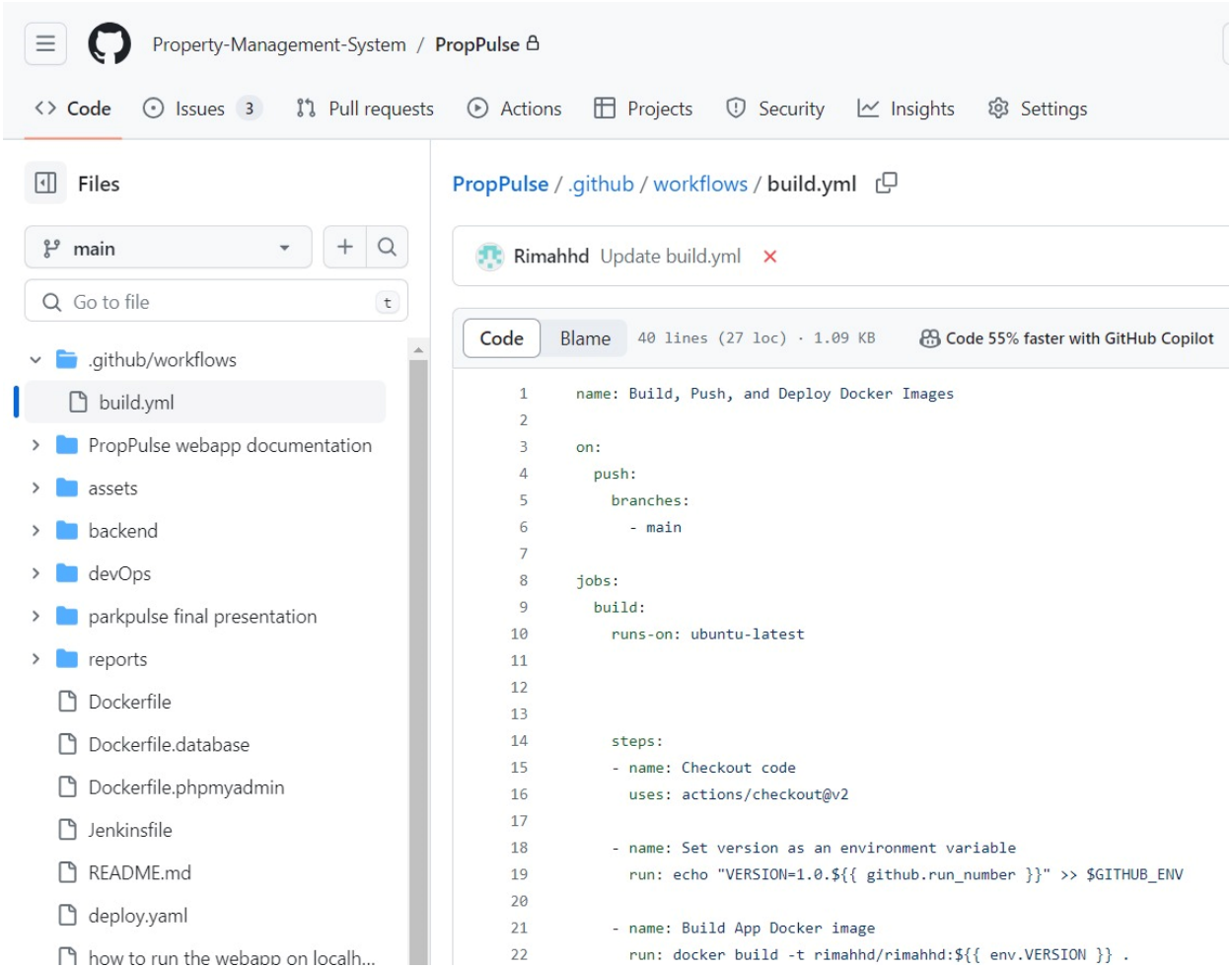
To create a DockerHub repository, log in to your DockerHub account, navigate to the dashboard, and click on the "Create Repository" button. Then, specify the repository name, description, visibility (public or private), and any other desired settings. Finally, click "Create" to finalize the creation of your DockerHub repository.



4. Build docker images

To build three Docker images for frontend, backend, and database using GitHub workflows, you'll need to set up a workflow file (e.g., **.github/workflows/docker-build.yml**) in your repository.

Within this file, define jobs for building each image using Docker build commands.



yamlCopy

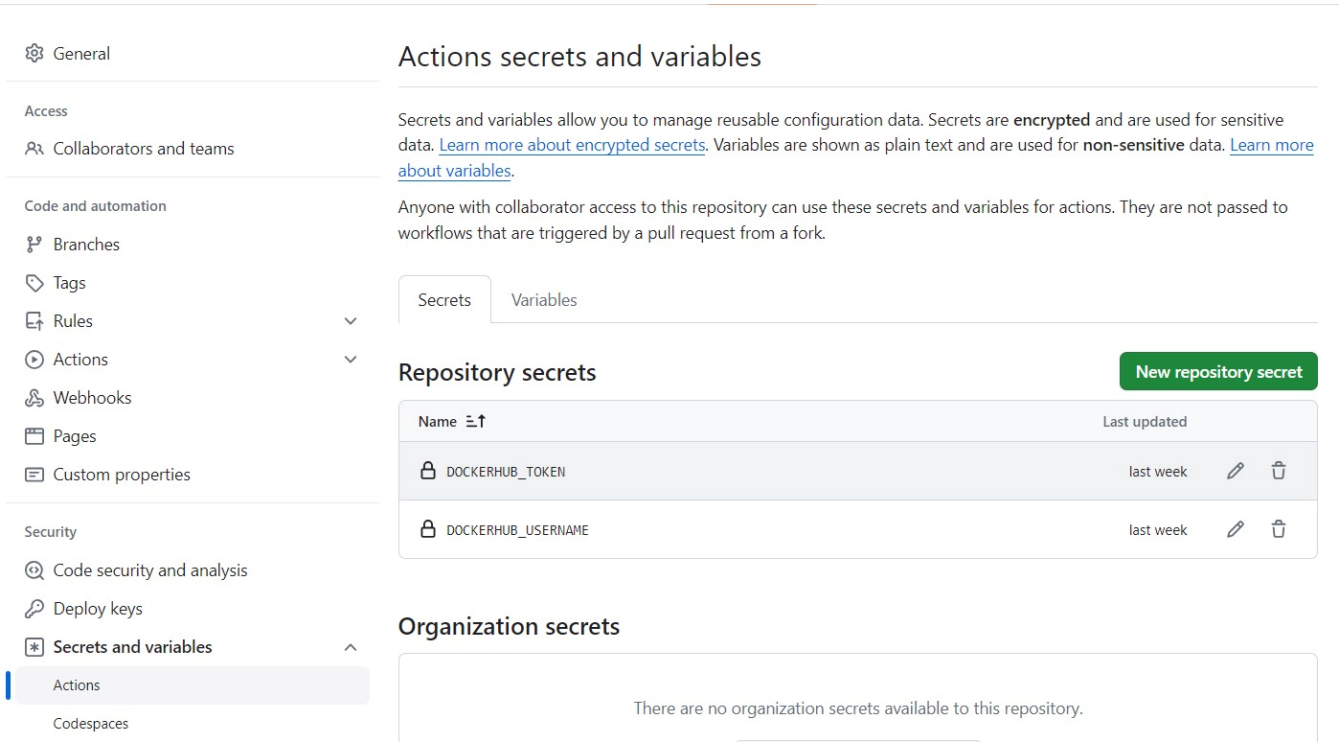
build.yml

```yaml
name: Build, Push, and Deploy Docker Images

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest



    steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: Set version as an environment variable
      run: echo "VERSION=1.0.${{ github.run_number }}" >> $GITHUB_ENV

    - name: Build App Docker image
      run: docker build -t rimahhd/rimahhd:${{ env.VERSION }} .

    - name: Build Database Docker image
      run: docker build -t rimahhd/mysql:${{ env.VERSION }} -f Dockerfile.database .

    - name: Build phpMyAdmin Docker image
      run: docker build -t rimahhd/php:${{ env.VERSION }} -f Dockerfile.phpmyadmin .

    - name: Log in to Docker
      run: echo "${{ secrets.DOCKERHUB_TOKEN }}" | docker login -u "${{ secrets.DOCKERHUB_USERNAME }}" --password-stdin

    - name: Push App Docker image
      run: docker push rimahhd/rimahhd:${{ env.VERSION }}

    - name: Push Database Docker image
      run: docker push rimahhd/mysql:${{ env.VERSION }}

    - name: Push phpMyAdmin Docker image
      run: docker push rimahhd/php:${{ env.VERSION }}
```

Commit this workflow file to your repository, and each time you push changes to the specified branch, GitHub Actions will automatically trigger the workflow to build your Docker images. Make sure to authenticate with DockerHub in your workflow if your images are pushed to DockerHub.

Note on how to create secrets:

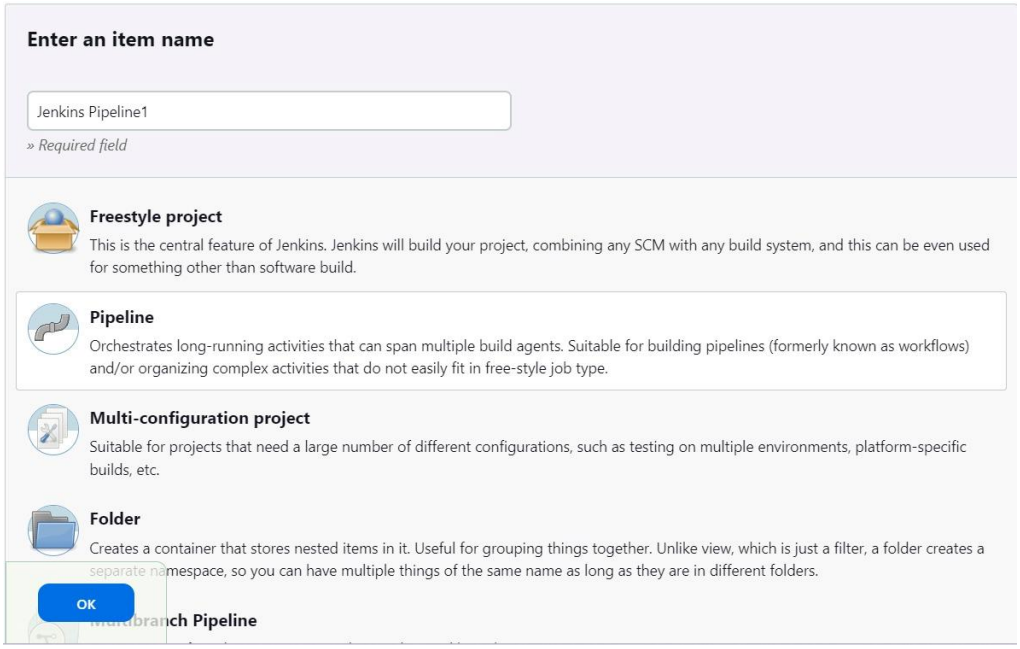
To create GitHub secrets, go to your repository's settings, navigate to the "Secrets" section, and click "New repository secret", then add the secret name and value, and save it.



5. Build a pipeline on jenkins

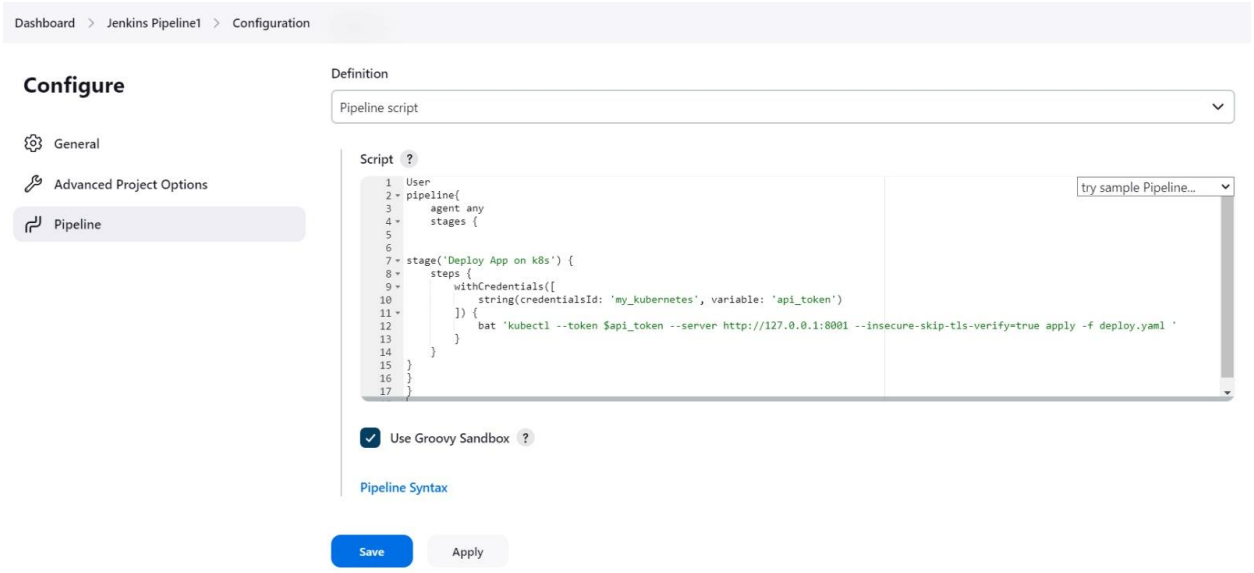
To build a Jenkins pipeline, follow these steps:

1. Install Jenkins
2. Install Pipeline plugins
3. Create a Jenkinsfile
4. Define stages and steps
5. Configure Jenkins job
6. Run the job
7. Monitor pipeline execution
8. Iterate and improve



6. Add the required credentials

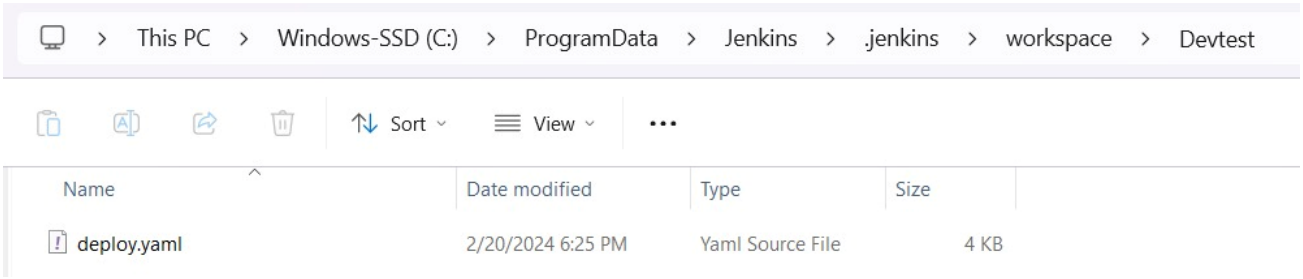
To add credentials in Jenkins:



1. Navigate to the Jenkins dashboard and click on "Manage Jenkins".
2. Select "Manage Credentials" from the options.
3. Click on the "Global credentials" domain (or any other domain where you want to store your credentials).

4. Click on "Add Credentials" on the left sidebar.
5. Choose the type of credential you want to add (e.g., Username with password, SSH username with private key, Secret text, etc.).
6. Fill in the required fields with your credential information.
7. Optionally, provide an ID and description for the credential.
8. Click "OK" to save the credential.

Finally, add a deploy.yaml file



The provided YAML file defines Kubernetes resources for deploying a web application, MySQL database, and PHPMyAdmin:

- **Web Application**:
  - Exposed through a NodePort service on port 80.
  - Deployed using a single replica Deployment.
- **MySQL Database**:
  - Exposed internally using a ClusterIP service on port 3306.
  - Managed by a StatefulSet for persistent storage.
- **PHPMyAdmin**:
  - Exposed through a NodePort service on port 8080.
  - Deployed using a single replica Deployment to manage the MySQL database.

These resources are configured with appropriate labels, ports, and environment variables for inter-component communication and functionality. Adjustments may be necessary based on specific deployment requirements and configurations.

7. Create DB, FE & BE docker files

Dockerfile.database > ...

```
# Use an official MySQL runtime as a parent image
FROM mysql:latest

# Set the MySQL root password
ENV MYSQL_ROOT_PASSWORD=root

# Create a database and user
ENV MYSQL_DATABASE=hmisphp
ENV MYSQL_USER=root
ENV MYSQL_PASSWORD=

# When container starts, execute the following SQL script
COPY ./init.sql /docker-entrypoint-initdb.d/
```

Dockerfile.phpmyadmin > ...

```
# Use an official phpMyAdmin image as base
FROM phpmyadmin/phpmyadmin:latest

# Set environment variables for MySQL connection
ENV PMA_HOST=localhost
ENV PMA_USER=root
ENV PMA_PASSWORD=

# The port phpMyAdmin will run on
EXPOSE 80

# Start phpMyAdmin
CMD ["apache2-foreground"]
```

Dockerfile > ...

```
# Use an existing Apache image with PHP support as a base
FROM php:apache

# Install the MySQLi extension
RUN docker-php-ext-install mysqli

# Set the working directory in the container
WORKDIR /var/www/html

# Copy HTML and PHP files from the host into the container
COPY . .

# Expose port 80 to the outside world
EXPOSE 80
```

Start Minikube

To start Minikube using Docker engine, follow these steps:

1. **Install Docker**: If you haven't already, download and install Docker from the official Docker website according to your operating system.
2. **Install Minikube**: Download and install Minikube by following the instructions provided in the Minikube documentation, ensuring that you have the latest version installed.
3. **Start Minikube**: Open a terminal or command prompt and run the following command to start Minikube with the Docker driver:

   sqlCopy code

   ```
   start
   ```

   This command initializes Minikube with Docker as the driver for managing the Kubernetes cluster.
4. **Verify Minikube**: Once Minikube has started successfully, verify its status by running:

   luaCopy code

   ```
   status
   ```

   You should see the status of Minikube as "Running".
5. **Manage your Kubernetes cluster**: You can now use Minikube to manage your Kubernetes cluster locally. You can interact with the cluster using kubectl, the Kubernetes command-line tool.

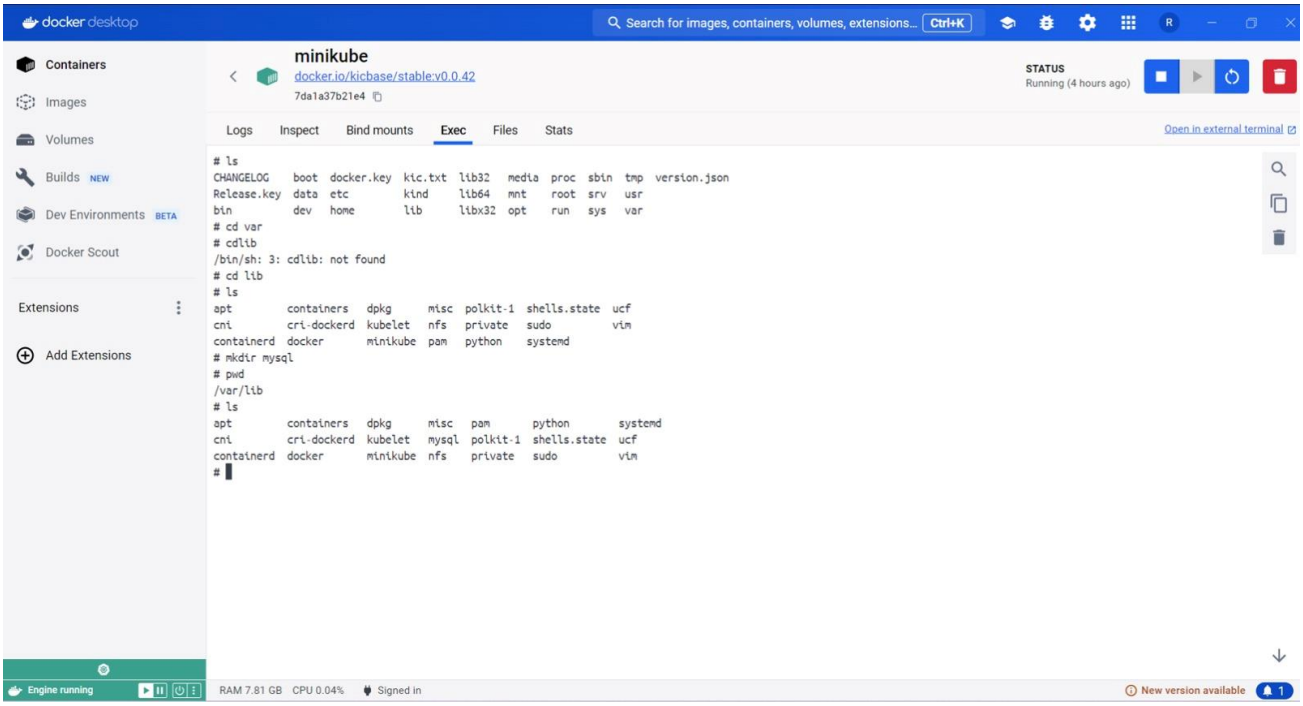
By following these steps, you can start Minikube using Docker as its driver, enabling you to run a Kubernetes cluster locally for development and testing purposes.

```
user@LAPTOP-26ATSE3V MINGW64 ~
$ minikube start
W0216 15:25:06.576217   24192 main.go:291] Unable to resolve the current Docker
CLI context "default": context "default": context not found: open C:\Users\user\
.docker\contexts\meta\37a8eec1ce19687d132fe29051dca629d164e2c4958ba141d5f4133a33
f0688f\meta.json: The system cannot find the path specified.
* minikube v1.32.0 on Microsoft Windows 11 Home Single Language 10.0.22631.3155
Build 22631.3155
* Using the docker driver based on existing profile
* Starting control plane node minikube in cluster minikube
* Pulling base image ...
* Restarting existing docker container for "minikube" ...
* Preparing Kubernetes v1.28.3 on Docker 24.0.7 ...
* Configuring bridge CNI (Container Networking Interface) ...
* Verifying Kubernetes components...
  - Using image docker.io/kubernetesui/dashboard:v2.7.0
  - Using image docker.io/kubernetesui/metrics-scraper:v1.0.8
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
* Some dashboard features require the metrics-server addon. To enable all featur
es please run:

        minikube addons enable metrics-server


* Enabled addons: storage-provisioner, default-storageclass, dashboard
* Done! kubectl is now configured to use "minikube" cluster and "default" namesp
ace by default

user@LAPTOP-26ATSE3V MINGW64 ~
$ minikube dashboard
W0216 15:25:48.874495    2700 main.go:291] Unable to resolve the current Docker
CLI context "default": context "default": context not found: open C:\Users\user\
.docker\contexts\meta\37a8eec1ce19687d132fe29051dca629d164e2c4958ba141d5f4133a33
f0688f\meta.json: The system cannot find the path specified.
* Verifying dashboard health ...
* Launching proxy ...
* Verifying proxy health ...
* Opening http://127.0.0.1:53270/api/v1/namespaces/kubernetes-dashboard/services
/http:kubernetes-dashboard:/proxy/ in your default browser...
```

Here Docker is taken by default, so no need to mention –driver in the CLI

8. Create a database directory

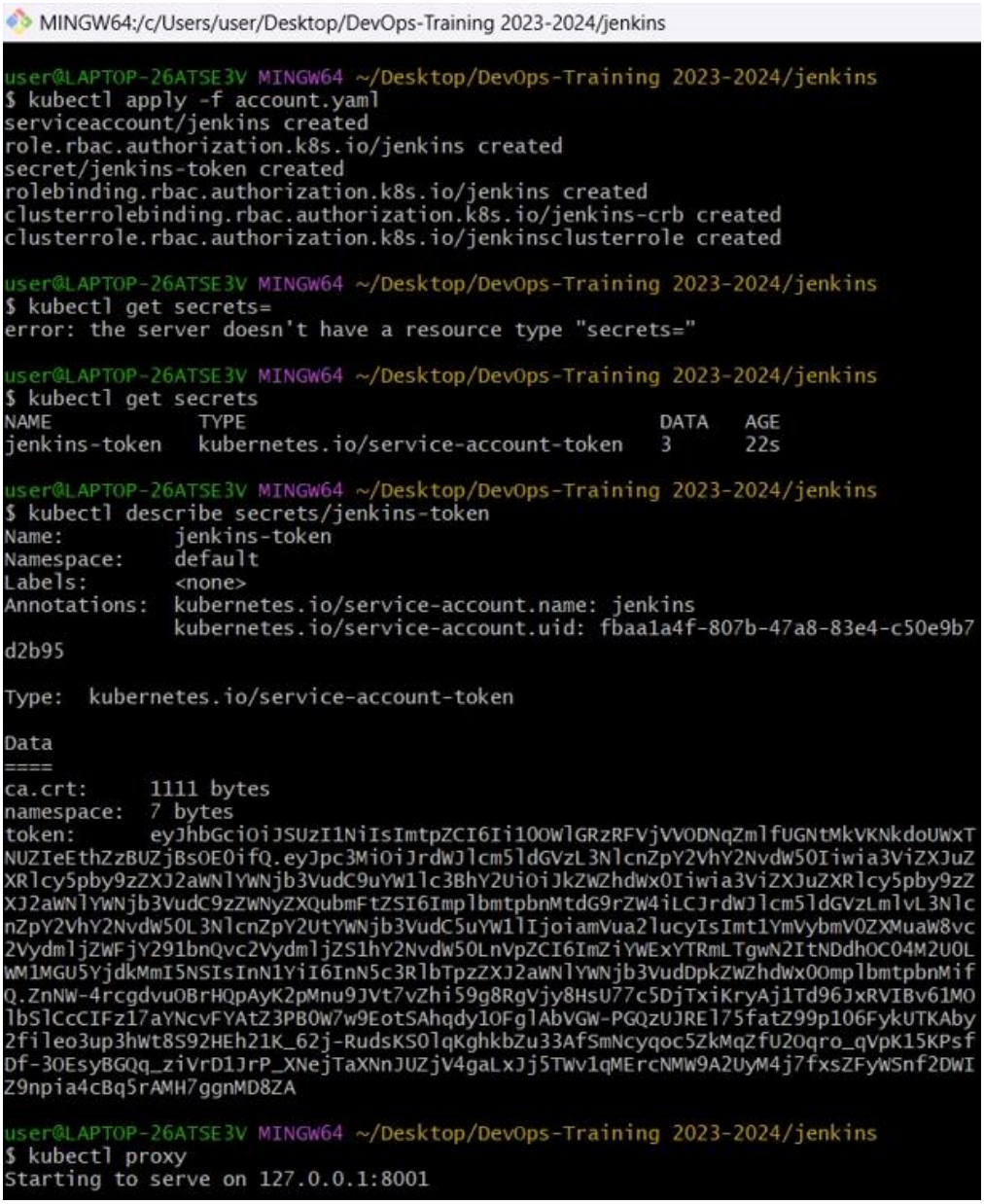
- Navigate to minikube container on docker desktop
- Navigate to exec



- Create a mysql directory, as appears in the above image

9. Jenkins Pipeline execution

10.Apply Kubernetes Configuration for Account Resources



This command applies the configuration defined in the "account.yaml" file to Kubernetes, creating or updating resources related to user accounts according to the specifications provided in the YAML file.

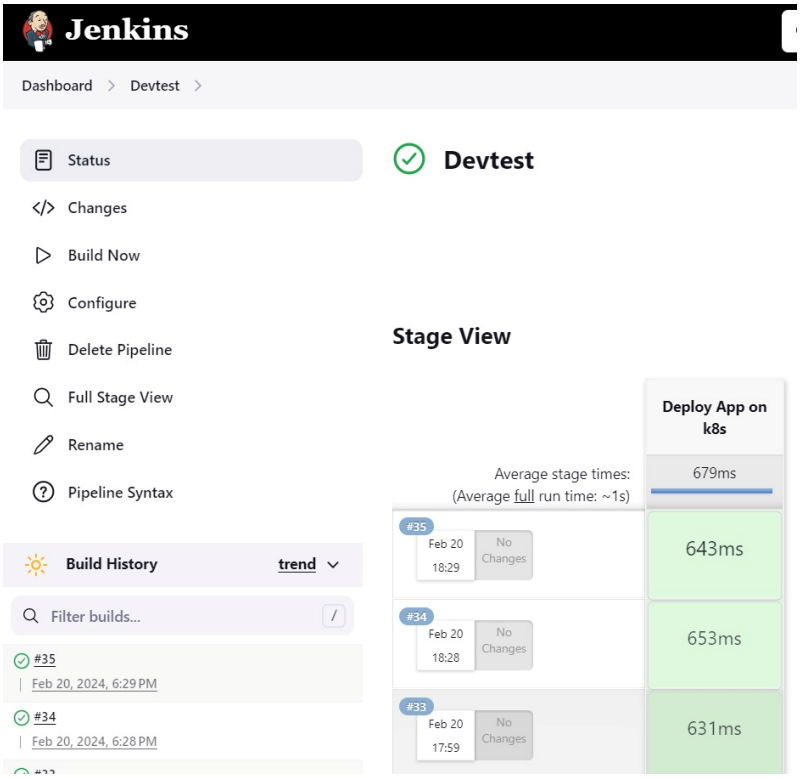
apiVersion: v1
kind: ServiceAccount

```
metadata:
  name: jenkins
  namespace: default
---

kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: jenkins
  namespace: default
rules:
- apiGroups: [""]
  resources: ["pods","services"]
  verbs: ["create","delete","get","list","patch","update","watch"]
- apiGroups: ["apps"]
  resources: ["deployments"]
  verbs: ["create","delete","get","list","patch","update","watch"]
- apiGroups: [""]
  resources: ["pods/exec"]
  verbs: ["create","delete","get","list","patch","update","watch"]
- apiGroups: [""]
  resources: ["pods/log"]
  verbs: ["get","list","watch"]
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get"]
- apiGroups: [""]
  resources: ["persistentvolumeclaims"]
  verbs: ["create","delete","get","list","patch","update","watch"]

---
apiVersion: v1
kind: Secret
metadata:
  name: jenkins-token
  annotations:
    kubernetes.io/service-account.name: jenkins
type: kubernetes.io/service-account-token

---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: jenkins
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: jenkins
```

```
subjects:
- kind: ServiceAccount
  name: jenkins
---
# Allows jenkins to create persistent volumes
# This cluster role binding allows anyone in the "manager" group to read secrets in any namespace.
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: jenkins-crb
subjects:
- kind: ServiceAccount
  namespace: default
  name: jenkins
roleRef:
  kind: ClusterRole
  name: jenkinsclusterrole
  apiGroup: rbac.authorization.k8s.io
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: jenkinsclusterrole
rules:
- apiGroups: [""]
  resources: ["persistentvolumes"]
  verbs: ["create","delete","get","list","patch","update","watch"]
```

These Kubernetes manifests define resources and permissions necessary for Jenkins to interact with the Kubernetes cluster. It includes a ServiceAccount, Role, Secret, RoleBinding, ClusterRoleBinding, and ClusterRole. These resources grant Jenkins appropriate permissions for managing pods, services, deployments, secrets, and persistent volume claims within the default namespace. Additionally, it allows Jenkins to create and manage persistent volumes across the entire cluster.

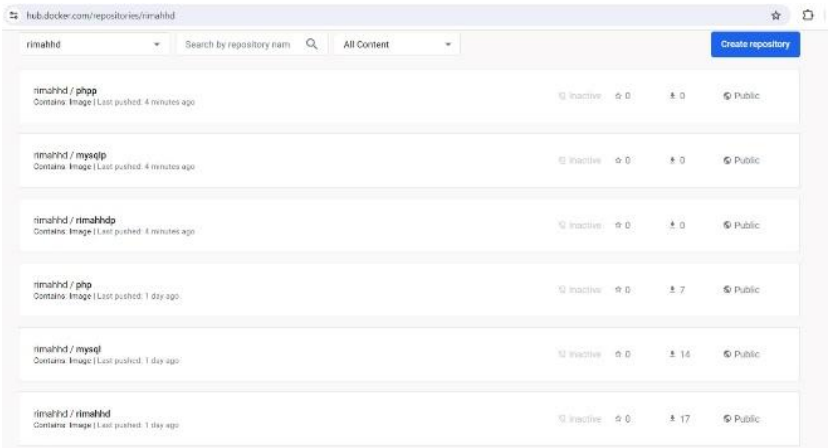
Then run kubectl proxy that initiates a proxy server, facilitating a connection between your local system and the Kubernetes API server. This enables you to interact with the Kubernetes API directly from your local machine, enhancing accessibility and ease of management.



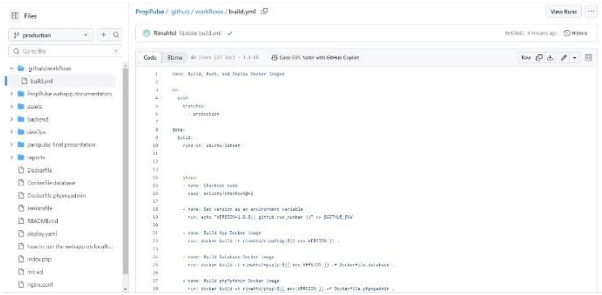
Then wait till the pods finish running

```
user@LAPTOP-26ATSE3V MINGW64 ~
$ kubectl get pods
NAME                                          READY   STATUS    RESTARTS   A
rimahhd-phpmyadmin-deployment-645bdd4b5d-9btw5   1/1     Running   0          8
rimahhd-web-deployment-79c669c99d-9c5d9          1/1     Running   0          8
wissam-db-statefulset-0                          1/1     Running   0          8
```

11.Creating a Production Environment




```
12
13
14        steps:
15        - name: Checkout code
16          uses: actions/checkout@v2
17
18        - name: Set version as an environment variable
19          run: echo "VERSION=1.0.${{ github.run_number }}" >> $GITHUB_ENV
20
21        - name: Build App Docker image
22          run: docker build -t rimahhd/rimahhdp:${{ env.VERSION }} .
23
24        - name: Build Database Docker image
25          run: docker build -t rimahhd/mysqlp:${{ env.VERSION }} -f Dockerfile.database .
26
27        - name: Build phpMyAdmin Docker image
28          run: docker build -t rimahhd/phpp:${{ env.VERSION }} -f Dockerfile.phpmyadmin .
29
30        - name: Log in to Docker
31          run: echo "${{ secrets.DOCKERHUB_TOKEN }}" | docker login -u "${{ secrets.DOCKERHUB_USERNAME }}" --password-stdin
32
33        - name: Push App Docker image
34          run: docker push rimahhd/rimahhdp:${{ env.VERSION }}
35
36        - name: Push Database Docker image
37          run: docker push rimahhd/mysqlp:${{ env.VERSION }}
38
39        - name: Push phpMyAdmin Docker image
40          run: docker push rimahhd/phpp:${{ env.VERSION }}
```



- Creating a production branch

S	W	Name ↓	Last Success	Last Failure	Last Duration	
✓	☀	Devtest	2 days 21 hr #37	10 days #11	17 sec	▷
…	☀	ititit	N/A	N/A	N/A	▷
✗	🌧	Jenkins Pipeline	N/A	1 mo 25 days #19	18 sec	▷
…	☀	Jenkins Pipelinet	N/A	N/A	N/A	▷
✓	☁	prodution	1 day 22 hr #2	1 day 22 hr #1	12 sec	▷
✗	🌧	projetfinal	N/A	10 days #2	12 sec	▷
✗	🌧	Rimah Houssameldine	N/A	1 mo 24 days #6	27 sec	▷
✓	☀	test	1 mo 10 days #8	N/A	0.56 sec	▷

```
  ports:
    - protocol: TCP
      port: 3306
      targetPort: 3306
  type: ClusterIP # Change to ClusterIP, as you don't need external access
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mysql-pv
spec:
  capacity:
    storage: 10Gi  # Adjust the storage capacity as needed
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: standard  # Update with the appropriate storage class name
  hostPath:
    path: /var/lib/mysql  # Update with the appropriate host path


---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: database-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  # Reference the previously defined PersistentVolume by name
  volumeName: mysql-pv

---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: rimahhd-db-statefulset
spec:
  replicas: 1
```