Lab 20: Manually Run Multiple Containers with Kubernetes on Docker Desktop

Lab overview

In this lab activity, you will deploy and manage a multi-container application using Kubernetes on Docker Desktop. You will containerize a simple frontend and backend application, push the images to Docker Hub, and orchestrate them using Kubernetes manifests.

In this lab, you will:

- Create a Kubernetes cluster in Docker Desktop
- Build and push container images for a frontend and backend application to Docker Hub
- Develop Kubernetes manifests for deploying and exposing the application components
- Deploy the multi-container application to Kubernetes on Docker Desktop and verify its functionality through service interaction

Estimated completion time

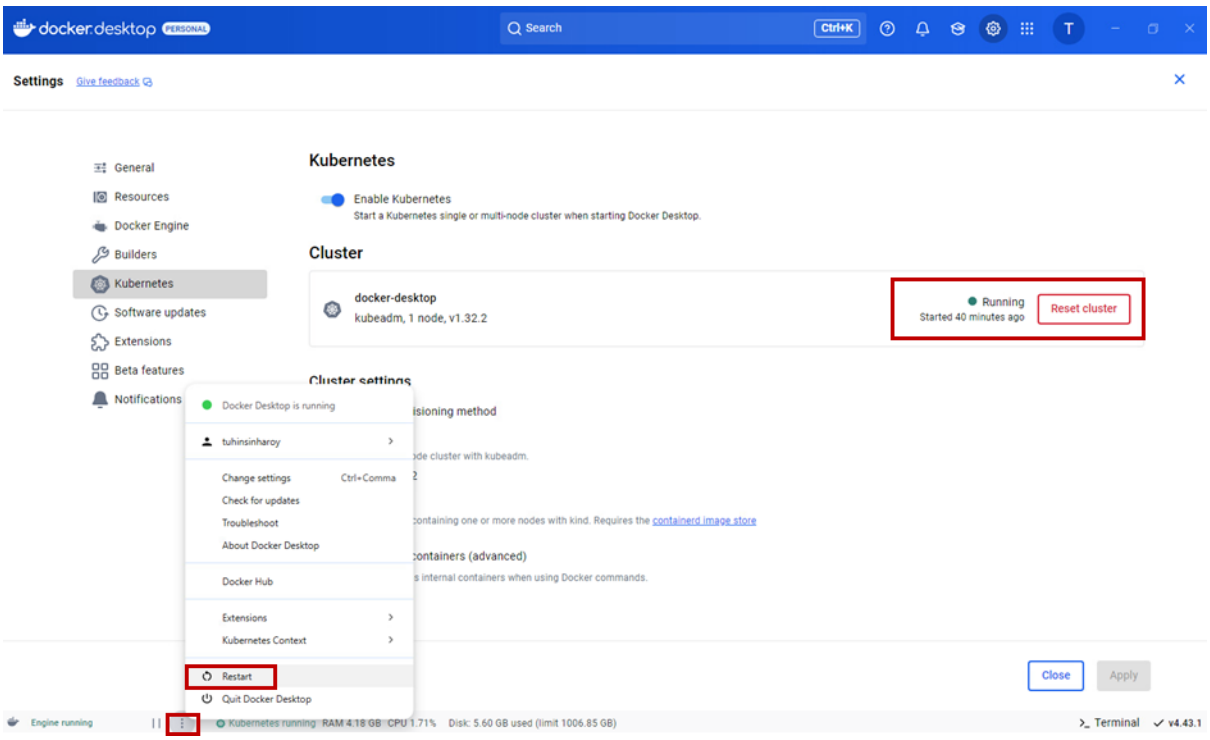
45 minutes

Task 1: Setting up Kubernetes on Docker Desktop

Docker Desktop includes a standalone Kubernetes server and client, that enables local Kubernetes development and testing directly on your machine. In this task, you will set up Kubernetes on Docker Desktop.

Note

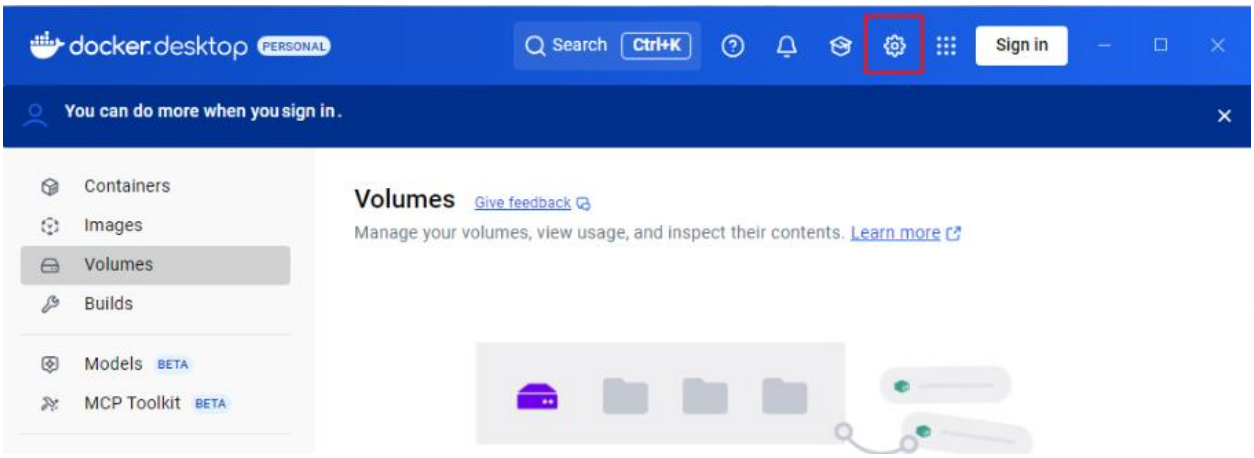
Kubernetes installation has been completed in advance to save time. There is a known issue with the Docker Desktop Application where Kubernetes may take some time to start. Please wait a few minutes, then restart the Docker engine. Allow additional time for Kubernetes to become fully operational.

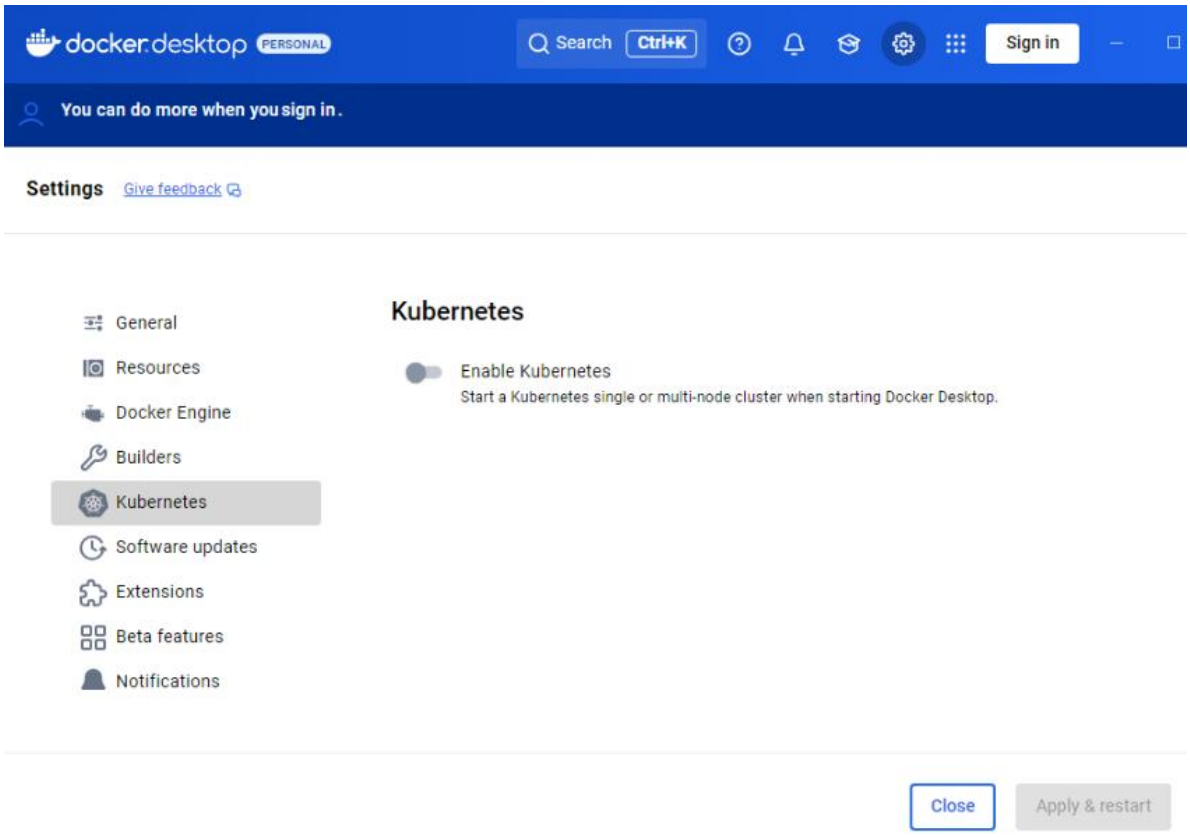


Note

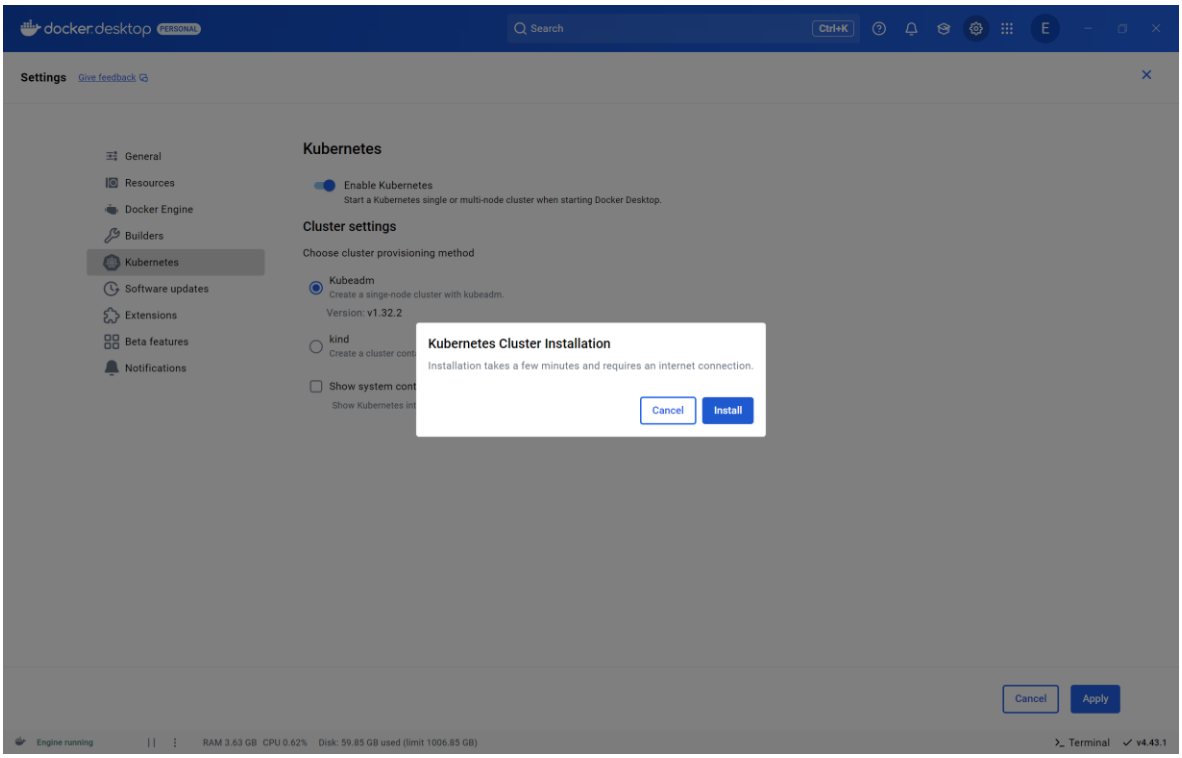
When Kubernetes shows Running, click **Sign In** and enter your credentials. If you don't have an account, create one on Docker Hub.

1. Open up Docker Desktop (if not already running). Navigate to **Settings** and then click on **Kubernetes**.

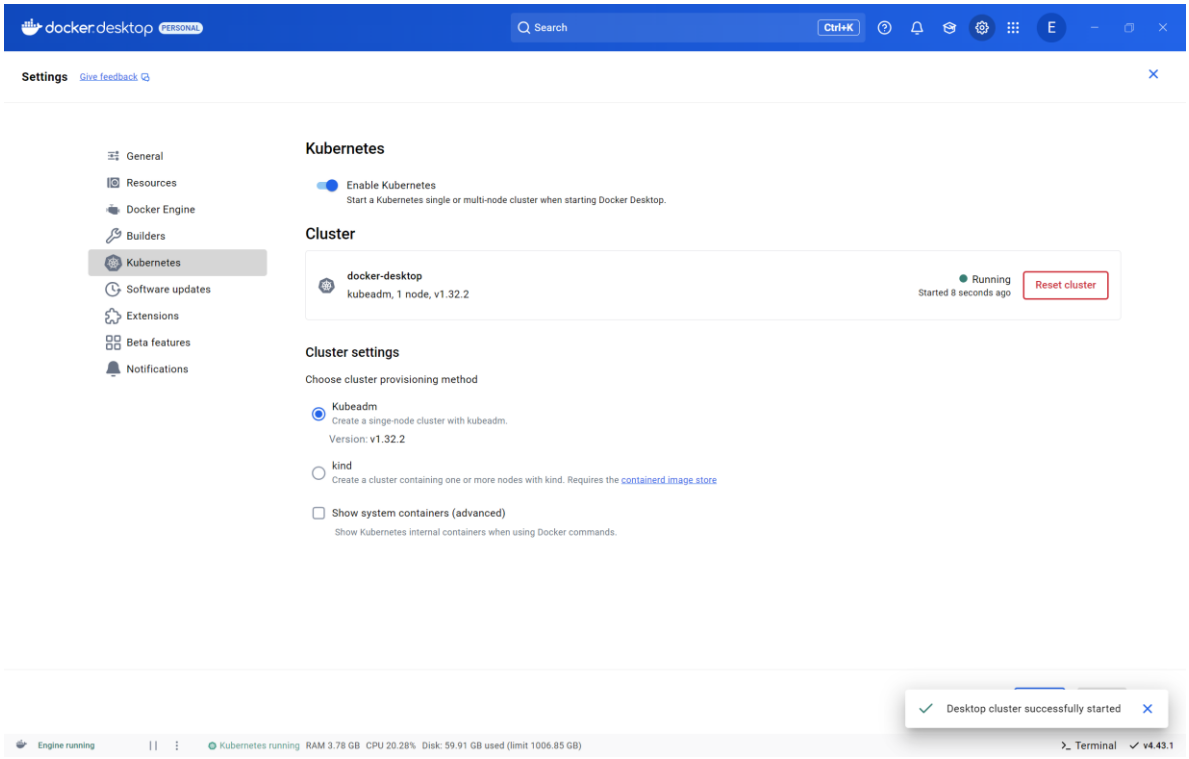




2. Enable Kubernetes with Kubeadmin (if not already enabled), and then click **Apply and Restart** followed by **Install**.



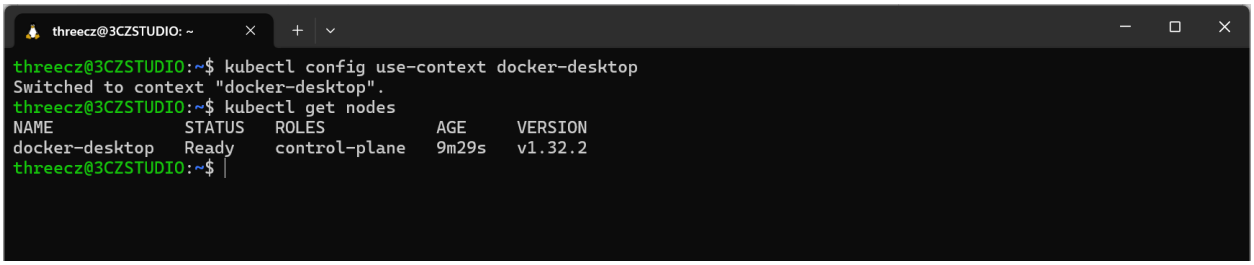
3. Wait for the Kubernetes cluster to provision successfully - this will take a few minutes. (Reset the cluster if it existed previously).



4. To confirm that Kubernetes is running, open up a PowerShell terminal and list the available nodes.

```
kubectl config use-context docker-desktop

kubectl get nodes
```



Task 2: Writing application code and building an image

1. Go to **https://github.com** and create a new **private** repository in your account called **DevOpsLab20**.

   Do not initialize a README file, **do not** add a .gitignore, and **do not** choose a license.

2. In your lab environment, create a folder with the name **DevOpsLab20** with the structure:
   - frontend
     - index.html
     - Dockerfile
     - default.conf
   - backend
     - index.php
     - Dockerfile
3. Inside the **frontend** folder, create a file called **index.html** and add the following code.

```
<!DOCTYPE html>

<html>

<head>

  <title>DevOps Lab 20 Frontend</title>

</head>

<body>

  <h1 style="color:black">What is your name?</h1>
```

```
  <form action="/api" method="POST">
    <label for="name">Name:</label>
    <input type="text" name="name" id="name" required>
    <button type="submit">Send</button>
  </form>
</body>
</html>
```

4. Inside the **frontend** folder, create a file called **Dockerfile** (with no file extension) with the following content.

```
FROM nginx:alpine
COPY default.conf /etc/nginx/conf.d/default.conf
COPY index.html /etc/nginx/html/index.html
EXPOSE 80
```

5. Create a file called **default.conf** to proxy traffic **/api** to the backend service.

```
server {
    listen 80;
    location /api {
        proxy_pass http://backend-service/;
    }
}
```

Note

The complete codes are also available in the DevOpsLab20 folder on the lab desktop.

6. Build, tag, and push the image to your Docker Hub account. Replace ***<docker_username>*** with your Docker Hub username.

```
docker build -t <docker_username>/frontend:v1 .
docker push <docker_username>/frontend:v1
```

7. Create a file called **index.php** in **backend** folder and add the following content.

```
<?php
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $name = htmlspecialchars($_POST['name']);
    echo "Hello, $name. Welcome to DevOps Lab 20!";
} else {
    echo "Please submit a POST request from the form.";
}
```

8. Create a file called **Dockerfile** in backend with the following content.

```
FROM php:8.2-apache
COPY index.php /var/www/html/
EXPOSE 80
```

Note

The complete codes are also available in the DevOpsLab20 folder on the lab desktop.

9. Build, tag, and push the image. Replace ***<docker_username>*** with your Docker Hub username.

```
docker build -t <docker_username>/backend:v1 .
docker push <docker_username>/backend:v1
```

10. Commit and push these changes to GitHub. From the **DevOpsLab20** lab root folder, run the following.

```
git config --global user.email "<you@example.com>"

git config --global user.name "<Your Name>"

git init

git add .

git commit -m "Initial commit with app"

git branch -M main

git remote add origin https://github.com/<your-
username>/DevOpsLab20.git

git push -u origin main
```

11. Sign in with your browser, if prompted.

Task 3: Creating Kubernetes Manifests

In this task, you will create Kubernetes Manifests for your two applications which will each contain a deployment and a service.

1. Create a folder outside the DevOpsLab20 folder called **k8s** and add the following to a file called **frontend-deployment.yaml**. Replace ***<docker_username>*** with your Docker Hub username.

```
apiVersion: apps/v1

kind: Deployment

metadata:

  name: frontend

spec:

  replicas: 1

  selector:

    matchLabels:

      app: frontend
```



```
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
        - name: frontend-app
          image: <docker_username>/frontend:v1
          ports:
            - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: frontend-service
spec:
  selector:
    app: frontend
  ports:
    - port: 8080
      targetPort: 80
  type: LoadBalancer
```

2. Create a file called **backend-deployment.yaml** inside **k8s** and add the following. Replace *<docker_username>* with your Docker Hub username.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend
spec:
  replicas: 1
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
        - name: backend-app
          image: <docker_username>/backend:v1
          ports:
            - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: backend-service
spec:
```

```
  selector:
    app: backend
  ports:
    - port: 80
      targetPort: 80
  type: ClusterIP
```

Note

The complete codes are also available in the DevOpsLab20 folder on the lab desktop.

Task 4: Manually deploying to Kubernetes

In this task, you will connect to your local Kubernetes cluster and create your deployments.

1. Ensure that you can authenticate to your local Kubernetes cluster.

```
kubectl config use-context docker-desktop
kubectl get nodes
```

Expected output:

```
NAME             STATUS   ROLES           AGE     VERSION
docker-desktop   Ready    control-plane   9m29s   v1.32.2
```

2. From the k8s folder, the apply command to deploy the frontend.

```
kubectl apply -f frontend-deployment.yaml
```

Note

Should a Windows Defender window pop-up, leave as default and click **Allow Access**.

3. From the k8s folder, run the apply command to deploy the backend.

```
kubectl apply -f backend-deployment.yaml
```

4. From the k8s folder, run the following command to confirm that the deployments were successful.

```
kubectl get pods
```

```
threecz@3CZSTUDIO: ~
threecz@3CZSTUDIO:~$ kubectl get pods
NAME                        READY   STATUS    RESTARTS   AGE
backend-7547fc5667-tpw87    1/1     Running   0          5s
frontend-769d94bc5c-zhv7h   1/1     Running   0          14s
threecz@3CZSTUDIO:~$
```

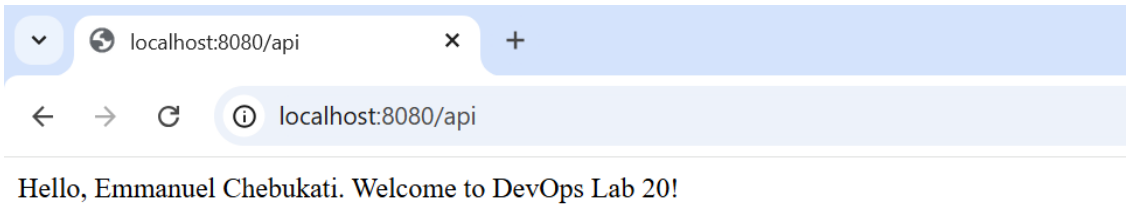
5. Now run the following command to get the External IP of your frontend service.

```
kubectl get svc
```

```
C:\Users\student>kubectl get svc
NAME               TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
backend-service    ClusterIP      10.97.100.145   <none>        80/TCP         29s
frontend-service   LoadBalancer   10.98.28.17     localhost     80:31741/TCP   54s
kubernetes         ClusterIP      10.96.0.1       <none>        443/TCP        3h45m
```

6. Navigate to the External IP address, **http://localhost:8080** in this case.
7. Your application is running. Enter your name and click **Send**.

8. The application responds. This indicates that the backend is working too!



9. From the k8s folder, run the following commands to delete the deployments.

```
kubectl delete -f frontend-deployment.yaml
```

```
kubectl delete -f backend-deployment.yaml
```

Important

Retain the repositories in your GitHub and Docker Hub stores. They will be required for upcoming labs.

Lab review

1. What does **kubectl get svc** display?
   A. Running deployments
   B. Services and their cluster/external IPs
   C. Pod logs
   D. Current Kubernetes context
2. What type of Kubernetes service is used to expose the backend application internally within the cluster, and not publicly like the frontend?
   A. LoadBalancer
   B. NodePort
   C. ClusterIP
   D. ExternalIP

STOP

You have successfully completed this lab.