Hosting a Calculator Application in Kubernetes Using kind

June 11, 2025

1 Introduction

This document provides a comprehensive guide to hosting a simple calculator web application in a Kubernetes cluster using the `kind` tool. The steps include creating the calculator application, containerizing it with Docker, setting up a local Kubernetes cluster with `kind`, and deploying the application.

2 Prerequisites

Ensure the following tools are installed:

- Docker: For building and running containers.
- kind: For creating a local Kubernetes cluster.
- kubectl: For interacting with the Kubernetes cluster.
- Node.js and npm: For developing the calculator web application.

3 Step 1: Creating the Calculator Application

Creating a simple calculator web application using Node.js and Express.

3.1 Directory Structure

Create a directory for the project and set up the following structure:

```
calculator-app/
   index.html
   script.js
   styles.css
   server.js
   package.json
   Dockerfile
```

3.2 Application Code

Defining the HTML, CSS, JavaScript, and server code for the calculator.

Listing 1: index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Calculator</title>
  <link rel="stylesheet" href="/styles.css">
</head>
<body>
  <div class="calculator">
    <input type="text" id="display" readonly>
    <div class="buttons">
      <button onclick="clearDisplay()">C</button>
      <button onclick="appendToDisplay('7')">7</button>
      <button onclick="appendToDisplay('8')">8</button>
      <button onclick="appendToDisplay('9')">9</button>
      <button onclick="appendToDisplay('/')">/</button>
      <button onclick="appendToDisplay('4')">4</button>
      <button onclick="appendToDisplay('5')">5</button>
      <button onclick="appendToDisplay('6')">6</button>
      <button onclick="appendToDisplay('*')">*</button>
      <button onclick="appendToDisplay('1')">1</button>
      <button onclick="appendToDisplay('2')">2</button>
      <button onclick="appendToDisplay('3')">3</button>
      <button onclick="appendToDisplay('-')">-</button>
      <button onclick="appendToDisplay('0')">0</button>
      <button onclick="appendToDisplay('.')">.</button>
      <button onclick="calculate()">=</button>
      <button onclick="appendToDisplay('+')">+</button>
    </div>
  </div>
  <script src="/script.js"></script>
</body>
</html>
```

Listing 2: script.js

```
function appendToDisplay(value) {
  document.getElementById('display').value += value;
}

function clearDisplay() {
  document.getElementById('display').value = '';
}

function calculate() {
  try {
    const result = eval(document.getElementById('display').value)
      ;
```

```
    document.getElementById('display').value = result;
  } catch (error) {
    document.getElementById('display').value = 'Error';
  }
}
```

Listing 3: styles.css

```
body {
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
  margin: 0;
  background-color: #f0f0f0;
}
.calculator {
  border: 1px solid #ccc;
  padding: 20px;
  background-color: #fff;
}
#display {
  width: 100%;
  padding: 10px;
  font-size: 20px;
  margin-bottom: 10px;
}
.buttons {
  display: grid;
  grid-template-columns: repeat(4, 1fr);
  gap: 10px;
}
button {
  padding: 20px;
  font-size: 18px;
  cursor: pointer;
}
```

Listing 4: server.js

```
const express = require('express');
const path = require('path');
const app = express();

app.use(express.static(path.join(__dirname, '.')));

app.get('/', (req, res) => {
  res.sendFile(path.join(__dirname, 'index.html'));
});

const port = process.env.PORT || 3000;
app.listen(port, () => {
```

```
  console.log(`Server running on port ${port}`);
});
```

Listing 5: package.json

```
{
  "name": "calculator-app",
  "version": "1.0.0",
  "description": "A simple calculator web app",
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },
  "dependencies": {
    "express": "^4.17.1"
  }
}
```

3.3 Installing Dependencies

Run the following command to install dependencies:

```
npm install
```

4 Step 2: Containerizing the Application

Creating a Dockerfile to containerize the calculator application.

```
FROM node:14

WORKDIR /app

COPY package*.json ./
RUN npm install

COPY . .

EXPOSE 3000

CMD ["npm", "start"]
```

4.1 Building the Docker Image

Build and test the Docker image:

```
docker build -t calculator-app:latest .
docker run -p 3000:3000 calculator-app:latest
```

Access the application at `http://localhost:3000` to verify it works.

5 Step 3: Setting Up a kind Cluster

Installing and configuring `kind` to create a local Kubernetes cluster.

5.1 Installing kind

Follow the official `kind` installation guide at `https://kind.sigs.k8s.io/docs/user/quick-start/#installation`.

5.2 Creating a kind Cluster

Create a cluster with a custom configuration to allow external access:

Listing 6: kind-config.yaml

```
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
  extraPortMappings:
  - containerPort: 30080
    hostPort: 30080
    protocol: TCP
```

Run the following command to create the cluster:

```
kind create cluster --config kind-config.yaml
```

5.3 Loading the Docker Image into kind

Load the Docker image into the `kind` cluster:

```
kind load docker-image calculator-app:latest
```

6 Step 4: Deploying to Kubernetes

Creating Kubernetes manifests to deploy the calculator application.

Listing 7: deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: calculator-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: calculator
  template:
    metadata:
```

```
      labels:
        app: calculator
    spec:
      containers:
      - name: calculator
        image: calculator-app:latest
        ports:
        - containerPort: 3000
```

Listing 8: service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: calculator-service
spec:
  selector:
    app: calculator
  ports:
  - protocol: TCP
    port: 80
    targetPort: 3000
    nodePort: 30080
  type: NodePort
```

6.1 Applying the Manifests

Apply the Kubernetes configurations:

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

6.2 Accessing the Application

Access the application at `http://localhost:30080`. If the port is not accessible, verify the cluster's IP:

```
kubectl get nodes -o wide
```

Use the external IP of the control-plane node with port 30080.

7 Step 5: Testing and Scaling

Testing the deployment and scaling the application.

7.1 Verifying the Deployment

Check the status of the pods:

```
kubectl get pods
```

View logs if needed:

```
kubectl logs <pod-name>
```

7.2 Scaling the Application

Scale the deployment to 3 replicas:

```
kubectl scale deployment calculator-deployment --replicas=3
```

8 Cleaning Up

Deleting the `kind` cluster when done:

```
kind delete cluster
```

9 Conclusion

You have successfully hosted a calculator application in a Kubernetes cluster using `kind`. The application is accessible via a browser, and you can scale it as needed.