# SIT737 - Practical Task 6.1P: Deploying a Microservice Using Docker and Kubernetes

## Task Overview

This task focuses on containerizing a Node.js-based calculator microservice using Docker, and deploying it on a local Kubernetes cluster. The task includes creating a Docker image, setting up a Kubernetes deployment and service, and verifying the application's availability and health.

## Tools Used

- Git
- Visual Studio Code
- Node.js
- Docker
- Kubernetes
- Kubectl
- Docker CLI

## Prerequisites & Setup

1. **Cloned the Repository**
   Repo: *https://github.com/mustafaT96/sit737-2025-prac5p.git*

2. **Installed Required Tools**
   - Node.js
   - Docker Desktop (includes Kubernetes)
   - Kubectl CLI

3. **Enabled Kubernetes in Docker Desktop**
   Docker > Settings > Kubernetes > Enabled Kubernetes

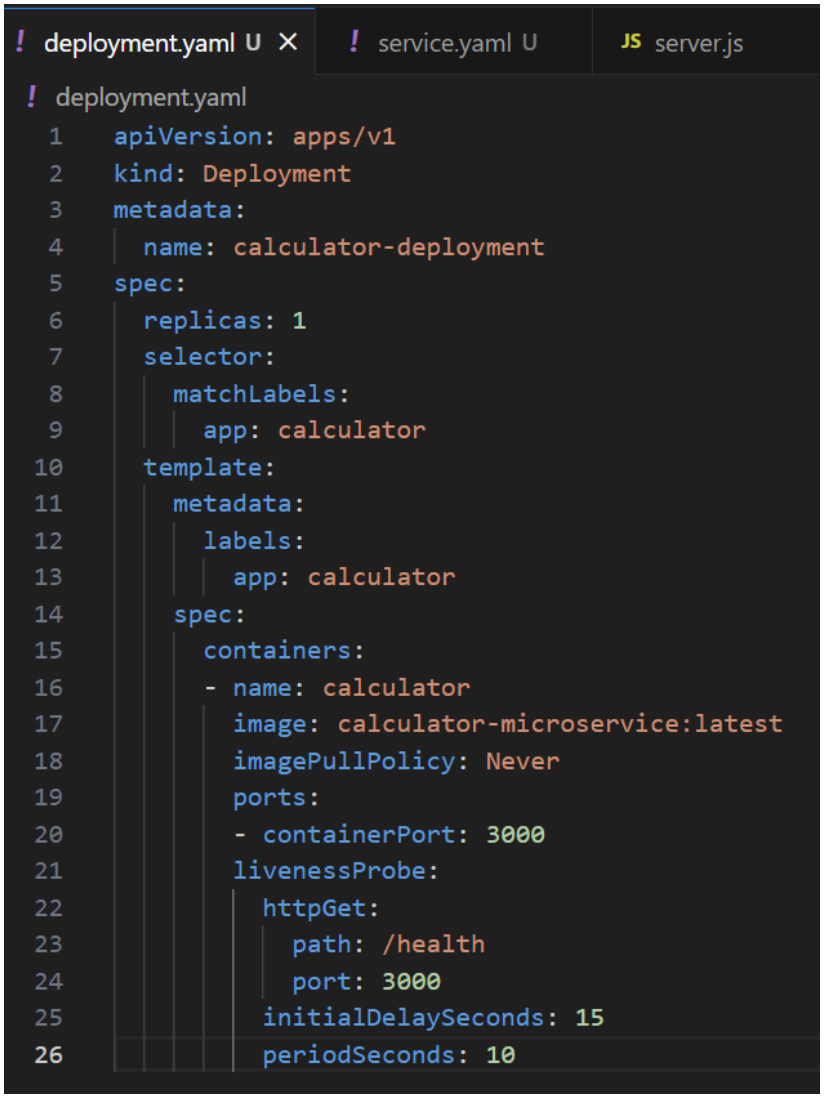
## Step-by-Step Instructions

1. **Create the Docker Image**
   Build the image locally:
   *docker build -t calculator-microservice:latest .*

2. **Kubernetes Cluster Setup**
   Confirmed the cluster is running:
   *kubectl cluster-info*

3. **Create Kubernetes Deployment**

The deployment manifest (*deployment.yml*) was created to manage the calculator microservice pod.




This YAML file creates a **Deployment**, which is responsible for managing Pods (containers) for our calculator app.

- **apiVersion**: apps/v1: Tells Kubernetes we're using the Deployment API.
- **kind: Deployment**: This is a Deployment object.
- **metadata.name**: Names the deployment as calculator-deployment.
- **spec.replicas**: 1: Tells Kubernetes to run **1 replica** (i.e., one instance) of our app.
- **selector.matchLabels**: Matches Pods that have the label app: calculator.
- **template**: Defines how the Pod should look.
    - **metadata.labels**: Labels the Pod with app: calculator so it matches the selector above
    - **spec.containers**: Describes the container to run:
        - **name**: Container name.
        - **image**: Uses the local Docker image calculator-microservice:latest.
        - **imagePullPolicy: Never**: Since the image is local, Kubernetes should not try to pull it from Docker Hub.
        - **ports.containerPort**: Exposes port 3000 inside the container.

- **livenessProbe**: A health check that pings /health every 10 seconds after a 15-second delay to make sure the app is running.

To apply the deployment:

*kubectl apply -f deployment.yml*

```
C:\My Work\data\Deakin\Semester 4\Cloud Native Applications\Task 6.1P\sit737-2025-prac5p>kubectl apply -f deployment.yaml
deployment.apps/calculator-deployment created
```

4. **Create Kubernetes Service**

The service manifest (*service.yml*) exposes the deployment through a NodePort:

deployment.yaml U ! service.yaml U

! service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: calculator-service
spec:
  type: NodePort
  selector:
    app: calculator
  ports:
    - protocol: TCP
      port: 3000
      targetPort: 3000
      nodePort: 30007
```

This YAML creates a **Service**, which acts as a gateway to our running Pod(s).

- **apiVersion: v1**: We're using the core Kubernetes API for services.
- **kind: Service:** This object is a Service.
- **metadata.name**: Names the service calculator-service.
- **spec.type: NodePort**: Exposes the service on a static port on the node (our local machine).
- **selector.app: calculator**: Tells Kubernetes to route traffic to Pods with label app: calculator (i.e., our deployed app).
- **ports**:
  - **port: 3000**: Port that the service exposes internally.
  - **targetPort: 3000:** Port inside the container where the app is running.
  - **nodePort: 30007**: Port on our machine to access the app, e.g., http://localhost:30007.

To apply the service:

*kubectl apply -f service.yml*

```
C:\My Work\data\Deakin\Semester 4\Cloud Native Applications\Task 6.1P\sit737-2025-prac5p>kubectl apply -f service.yaml
service/calculator-service created
```

5. **Verify Deployment**

Check pod status:

*kubectl get pods*

```
C:\My Work\data\Deakin\Semester 4\Cloud Native Applications\Task 6.1P\sit737-2025-prac5p>kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
calculator-deployment-5f9f597b6b-pf4b6   1/1     Running   0          14s
```

We can also check the service status by:

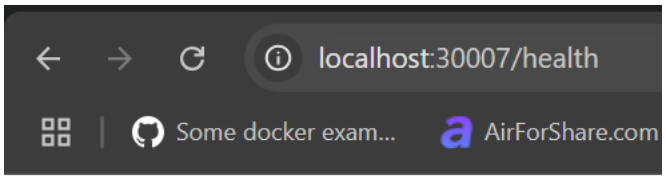
*kubectl get services*

```
C:\My Work\data\Deakin\Semester 4\Cloud Native Applications\Task 6.1P\sit737-2025-prac5p>kubectl get services
NAME                TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
calculator-service  NodePort    10.111.173.242   <none>        3000:30007/TCP   9s
kubernetes          ClusterIP   10.96.0.1        <none>        443/TCP          9m49s
```

This shows that our service is hosted at the port 30007

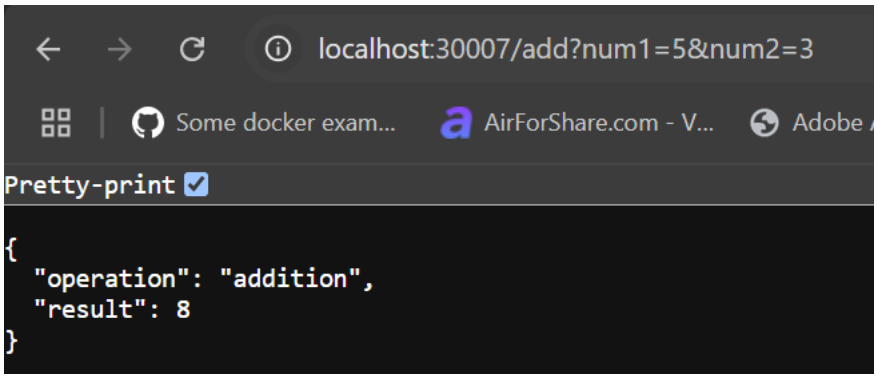
## 6. Testing the Service

- Health Check

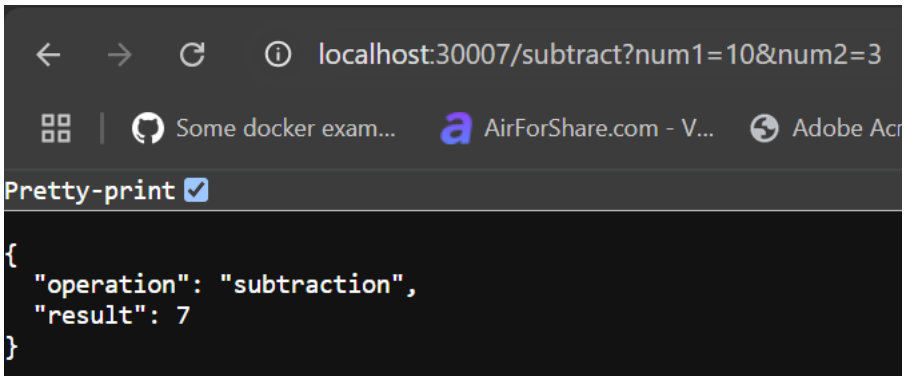



OK

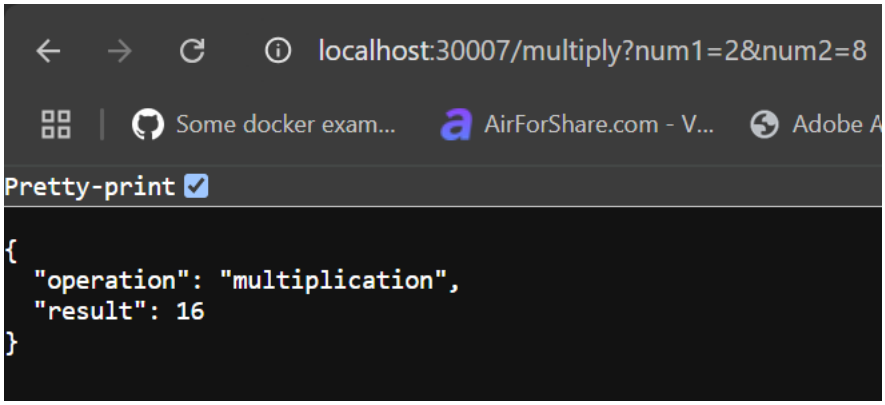
- Addition



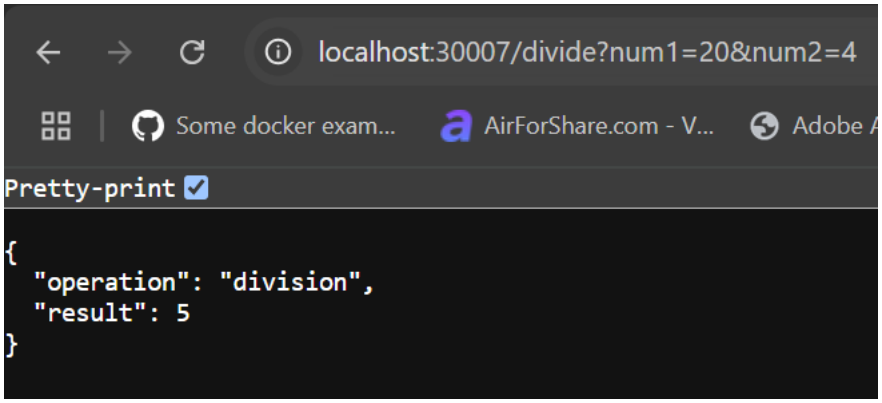

- Subtraction



- Multiplication




- Division




**Conclusion**

This task demonstrates a complete microservice deployment pipeline using Docker and Kubernetes. The service is successfully running on a local Kubernetes cluster, exposing basic calculator functionalities via HTTP endpoints. The system is production-ready for further scaling, container orchestration, and CI/CD integration.