SJSU SAN JOSÉ STATE UNIVERSITY

CMPE-272
Enterprise Software Platform

Assignment: Hello World Microservices Application

SJSU ID: 017992624
Kenil Gopani
kenilghanashyambhai.gopani@sjsu.edu

Setting Up the Development Environment

Programming language: Java
Web Framework: Spring boot

JDK installation: -

1. Download and install stable JDK version form the oracle (macOS) https://download.oracle.com/java/21/latest/jdk-21_macos-aarch64_bin.dmg
2. Verify JDK version To ensure that the JDK is installed correctly, verify the version by running the following command in your terminal:

```
$ java -version
```

Maven dependency installation: -

1. Visit the official site https://maven.apache.org/download.cgi and download binary file apache-maven-3.9.9-bin.tar.gz.
2. Extract tar file using below command:

```
$ tar -xvf apache-maven-3.6.3-bin.tar.gz
```

3. The next step is to set up the environment variables - M2_HOME and Path. We have to add the Maven bin directory to the Path variable. Open .bash_profile in text editor and add below lines to the end of it.

```
export M2_HOME="/path/to/apache-maven-3.9.9"
PATH="${M2_HOME}/bin:${PATH}"
export PATH
```

4. Verify maven installation:

```
      $ java -version
```

Download and Install IntelliJ IDEA

Visit the IntelliJ IDEA Website:

- Navigate to the official JetBrains website: IntelliJ IDEA Downloads and download community edition.

Creating Microservices using Spring Boot

Initialize a Spring Boot Project

- Visit Spring Initializr.
- Select below project settings
    - Project – Maven
    - Language – JAVA (version – 21)
    - Spring boot (version – 3.3.3)
    - Project Metadata:
        - Group: com.example
        - Artifact: hello-service
        - Name: hello-service
        - Package Name: com.example.hello-service
    - Packaging: JAR
- Download and unzip the generated project.

Implementing the Microservices

- Open the project in IntelliJ IDEA.
- Implement the HelloService and WorldService by adding controller classes that define the endpoints /hello and /world.

HelloController.java

```
package com.example.hello_service.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/hello")
public class HelloController {

    @GetMapping
    public String sayHello() {
        return "Hello";
    }
}
```

WorldController.java

```
package com.example.world_service.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/world")
public class WorldController {
```

```
    @GetMapping
    public String sayWorld() {
        return "World";
    }
}
```

Installing Docker

Download and Install Docker Desktop

- Download and Install Docker Desktop from the Docker website.
  https://www.docker.com/products/docker-desktop/

- Verify the installation by running:
  docker –version

Creating and Building Docker Images

- Create a Dockerfile in root directory of spring projects with following content:

/hello-service/Dockerfile

```
#helloService
FROM openjdk:21-jdk-slim

WORKDIR /helloService
COPY target/hello-service-0.0.1-SNAPSHOT.jar helloService.jar
EXPOSE 8081
ENTRYPOINT ["java", "-jar", "helloService.jar"]
```

/world-service/Dockerfile

```
#worldService
FROM openjdk:21-jdk-slim

WORKDIR /worldService
COPY target/world-service-0.0.1-SNAPSHOT.jar worldService.jar
EXPOSE 8082
ENTRYPOINT ["java", "-jar", "worldService.jar"]
```

Build the Docker Images

1. Build the Docker Image for Hello Service:
   ```
   docker build -t hello-service:latest .
   ```

2. Build the Docker Image for World Service:
   ```
   docker build -t world-service:latest .
   ```

Push Docker Images to Docker Hub

1. Use docker tag to rename the docker image with username

```
Docker tag hello-service:latest kenilgopanisjsu/hello-service:latest
Docker tag world-service:latest kenilgopanisjsu/world -servic:latest
```

2. Now run the docker push command.

```
docker push kenilgopanisjsu/hello-service:latest
docker push kenilgopanisjsu/world-service:latest
```

Docker hub URLs of hello-service and world-service images

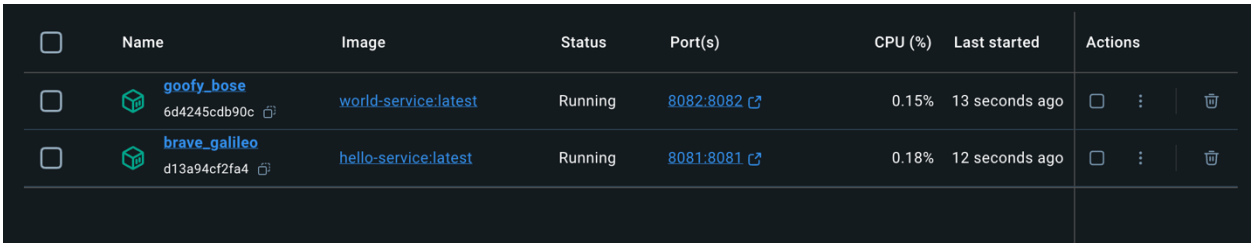
https://hub.docker.com/repository/docker/kenilgopanisjsu/hello-service/general
https://hub.docker.com/repository/docker/kenilgopanisjsu/world-service/general

Run the Docker Containers

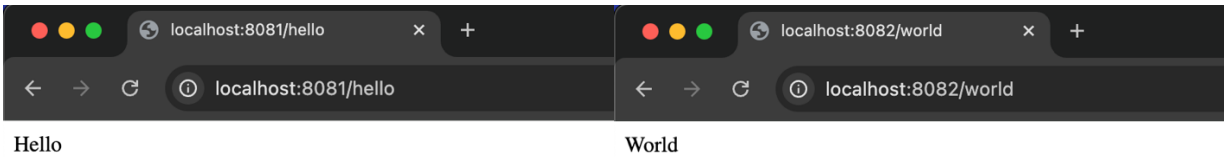
1. Run the Docker Container for Hello Service:
   - Start a container from the hello-service image:
     docker run -d -p 8081:8081 hello-service:latest
2. Run the Docker Container for World Service:
   - Similarly, start a container from the world-service image:
     docker run -d -p 8082:8082 world-service:latest

3. Verify the Running Containers
   - Use the following command to see if your containers are running:
     $ docker ps

2. Access the Services:
   - Verify that the services are accessible by navigating to the following URLs in your web browser:
     - http://localhost:8081/hello
     - http://localhost:8082/world



Name	Image	Status	Port(s)	CPU (%)	Last started	Actions
goofy_bose 6d4245cdb90c	world-service:latest	Running	8082:8082	0.15%	13 seconds ago	
brave_galileo d13a94cf2fa4	hello-service:latest	Running	8081:8081	0.18%	12 seconds ago	

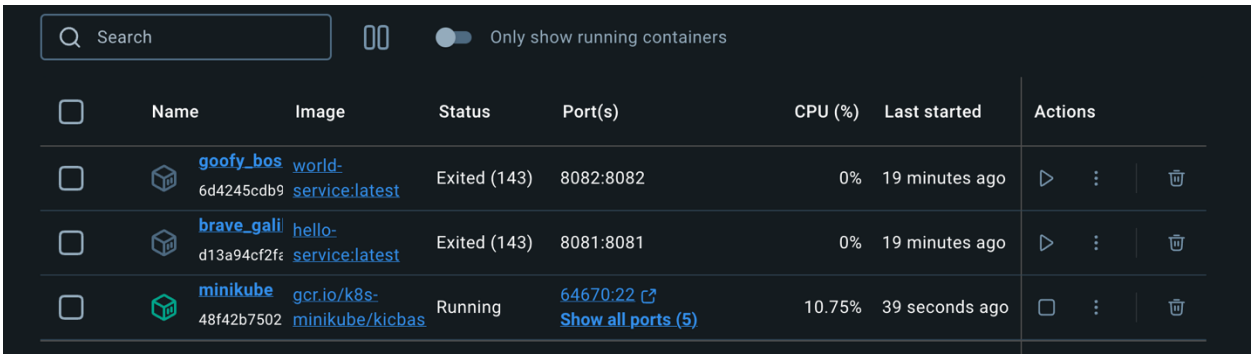


Installing Minikube and initialize a cluster

- Run the following command to install Minikube:
  - brew install minikube

- Initialize a Minikube Cluster:
  - minikube start

- Verify Minikube Status:
  - minikube status



Writing Kubernetes Manifests

Create Deployment and Service Manifests

1. Create a Deployment Manifest for Hello Service:
   - Create a file named hello-deployment.yaml with the following content:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-deployment
spec:
  replicas: 2
```

```
  selector:
    matchLabels:
      app: hello
  template:
    metadata:
      labels:
        app: hello
    spec:
      containers:
        - name: hello-service
          image: hello-service:latest
          imagePullPolicy: Never
          ports:
            - containerPort: 8081
```

2. Create a Service Manifest for Hello Service:
   - Create a file named hello-service.yaml with the following content:

```
apiVersion: v1
kind: Service
metadata:
  name: hello-service
spec:
  selector:
    app: hello
  ports:
    - protocol: TCP
      port: 8081
      targetPort: 8081
  type: NodePort
```

3. Create a Deployment Manifest for World Service:
   - Create a file named world-deployment.yaml with the following content:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: world-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: world
  template:
    metadata:
      labels:
        app: world
    spec:
      containers:
        - name: world-service
          image: world-service:latest
          imagePullPolicy: Never
          ports:
            - containerPort: 8082
```

4. Create a Service Manifest for World Service:

- Create a file named world-service.yaml with the following content:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: world-service
spec:
  selector:
    app: world
  ports:
    - protocol: TCP
      port: 8082
      targetPort: 8082
  type: NodePort
```

Applying Kubernetes Manifests

- Apply the deployment and service manifests for hello-service and world-service:

```
spartan@IMS-048MBA kubernetes % kubectl apply -f hello-deployment.yaml
kubectl apply -f hello-service.yaml
kubectl apply -f world-deployment.yaml
kubectl apply -f world-service.yaml
deployment.apps/hello-deployment created
service/hello-service created
deployment.apps/world-deployment created
service/world-service created
spartan@IMS-048MBA kubernetes %
```

Verify Deployments and Services

1. Check Pods:
   - Verify that the pods are running:

   kubectl get pods

```
spartan@IMS-048MBA kubernetes % kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
hello-deployment-689b788dfb-52sqh   1/1     Running   0          111s
hello-deployment-689b788dfb-mljhp   1/1     Running   0          111s
world-deployment-66b59c5b97-bg8cp   1/1     Running   0          110s
world-deployment-66b59c5b97-qx2jz   1/1     Running   0          110s
spartan@IMS-048MBA kubernetes %
```

2. Check Services:
   - Verify the services are created and have NodePorts:

   kubectl get services

```
spartan@IMS-048MBA kubernetes % kubectl get services
NAME            TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
hello-service   NodePort    10.104.152.17   <none>        8081:30862/TCP   3m5s
kubernetes      ClusterIP   10.96.0.1       <none>        443/TCP          7m40s
world-service   NodePort    10.96.78.73     <none>        8082:30519/TCP   3m4s
spartan@IMS-048MBA kubernetes %
```

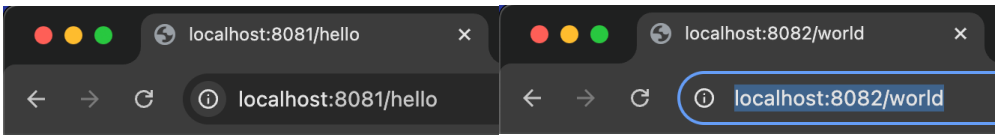
3. Get Service Details:
    - Describe the service to see the NodePort and other details:

        kubectl describe service hello-service
        kubectl describe service world-service

Accessing Services

1. Forward Ports for Testing:
    - Forward ports to access services from your local machine:

        kubectl port-forward service/hello-service 8081:8081
        kubectl port-forward service/world-service 8082:8082
2. Access Services:
    - Open your web browser and navigate to:
        - http://localhost:8081 (Hello Service)
        - http://localhost:8082 (World Service)



Hello World

Testing and integration of Services with combined.sh

The script forwards ports for hello-service and world-service, tests both services, prints "Hello World" if both are running, and cleans up port forwarding.

Run the Script:

- /combined.sh