Deployment Manual for Spring Boot Application using Kubernetes (k8s) and Docker

Contents

Prerequisites

1. **Install Java Development Kit (JDK)**
   - Ensure you have JDK installed (`openjdk:17-jdk-alpine`). You can download it from Oracle's JDK download page.
2. **Install Maven**
   - Maven is required to build the Spring Boot application. Download and install Maven from Apache Maven's download page.
3. **Install Docker**
   - Docker is required to containerize the application. Install Docker from Docker's official website.
4. **Install Kubernetes and Minikube**
   - Minikube is used to run a local Kubernetes cluster. Install Minikube from Minikube's installation page.
5. **Install kubectl**
   - kubectl is a command-line tool for interacting with Kubernetes clusters. Install kubectl from Kubernetes' installation page.

Creating Package File of Spring Boot Application

1. **Clean and Package the Application**
   - Open your terminal or command prompt.
   - Navigate to the root directory of your Spring Boot application.
   - Run the following command to clean and package your application:

     ```
     mvn clean package
     ```

   - This will generate a JAR file in the `target` directory of your project.

Building Docker Image

2. **Build the Docker Image**
   - Ensure your Docker daemon is running.
   - In the root directory of your Spring Boot application, create a Dockerfile with the following content:

     Dockerfile
     ```
     FROM openjdk:17-jdk-alpine

     EXPOSE 8080

     ADD target/twit-app-0.0.1-SNAPSHOT.jar twit-app.jar

     ENTRYPOINT ["java", "-jar", "/twit-app.jar"]
     ```

   - Build the Docker image with the following command:

```
docker build -t twit-app:1.0 .
```

Running Docker Container

3. **Run the Docker Container**
    - Start the Docker container and map port 8080 on your host to port 8080 in the container with the following command:

```
docker run -p 8080:8080 twit-app:1.0
```

    - Your application should now be accessible at `http://localhost:8080`.

Tagging and Pushing Docker Image

4. **Tag and Push the Docker Image**
    - Tag your Docker image for your Docker repository with the following command:

```
docker tag twit-app:1.0 <docker-id>/twit-app:1.0
```

    - Push the Docker image to your repository:

```
docker push <docker-id>/twit-app:1.0
```

Deploying to Kubernetes

5. **Apply Kubernetes Deployment**
    - Create a `deployment.yaml` file in the root directory with the following content:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: twit-k8s
spec:
  replicas: 2
  selector:
    matchLabels:
      app: twit-app
  template:
    metadata:
      labels:
        app: twit-app
    spec:
      containers:
        - name: twit-app
          image: binodleo/twit-app:1.0
          ports:
            - containerPort: 8080
```

- Apply the deployment configuration:

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

6. **Apply Kubernetes Service**
   - Create a `service.yaml` file in the root directory with the following content:

```
apiVersion: v1
kind: Service
metadata:
  name: twit-k8s-service
spec:
  type: LoadBalancer  # Exposes the service externally
using a cloud provider's load balancer
  selector:
    app: twit-app  # Matches the pods with this label
  ports:
    - protocol: TCP
      port: 80         # Port exposed by the service
      targetPort: 8080  # Port on the container to forward
traffic to
```

   - Apply the service configuration:

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

7. **Load Image in Minikube**
   - Load the Docker image into Minikube:

```
minikube image load twit-app:1.0
```

Verifying and Accessing the Application

8. **Verify Kubernetes Resources**
   - Check the status of all Kubernetes resources:

```
kubectl get all
```

   - Ensure that the pods, services, and deployments are running as expected.
9. **Access the Application**
   - Get the URL for your application service:

```
minikube service twit-app-service --url
```

   - This command will provide the URL where your application is accessible.

10. **Set Up Minikube Tunnel**
    - Start Minikube tunnel to expose services that use LoadBalancer type:

```
minikube tunnel
```

11. **Open Minikube Dashboard**
    - Access the Minikube dashboard for a graphical view of your Kubernetes cluster:

```
minikube dashboard
```

Scaling and Replicating Pods in Kubernetes

12. **Scaling Pods**
    - You can scale your application by increasing the number of replicas. To scale your deployment to 5 replicas, run

```
kubectl scale deployment twit-k8s --replicas=5
```

   - Verify the scaling operation by checking the status of your pods:

```
kubectl get pods
```

13. **Auto-scaling Pods**
    - Kubernetes supports horizontal pod auto-scaling based on resource utilization. To enable auto-scaling, use the following command:

```
kubectl autoscale deployment twit-k8s  --min=2 --max=10 --
cpu-percent=80
```

   - This command configures the Horizontal Pod Autoscaler to maintain between 2 and 10 replicas of the application, scaling based on CPU utilization.

Notes

- **Ensure Docker Daemon is Running**: Always ensure that your Docker daemon is running before building and running Docker images.
- **Customize Configurations**: Customize the Dockerfile and Kubernetes YAML configurations as per your application's requirements.

- **Production Deployment**: For deploying to a production Kubernetes cluster, consider using a cloud provider like AWS EKS, Google Kubernetes Engine (GKE), or Azure Kubernetes Service (AKS).