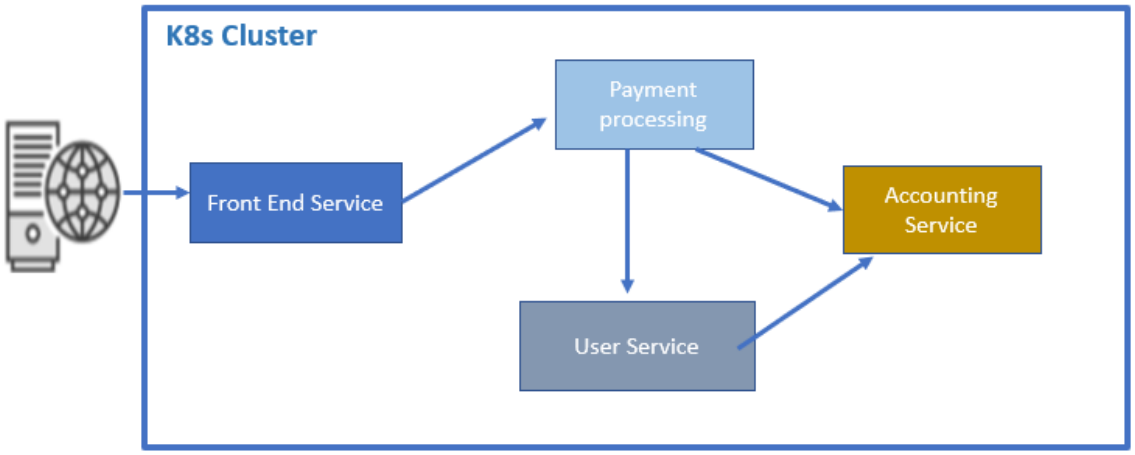
Contents

Kubernetes Case Study Submission by Chetan G Sanghi



High Level Design

UI

Docker

1. Docker image for UI Service uploaded at
    a. https://hub.docker.com/r/chetangsanghi/kube-ui-httpd
2. This is being used then by Kubernetes Deployment and Service for UI Service as per below details

Kubernetes

1. There is a UI Service based on httpd
2. It has index.html with an image
3. For index.html there is a form submission to initiate a payment which calls the Payment Service on Cluster IP
4. Public Repository where you can find the following files:
    a. https://github.com/chetansanghiwork/kube-ui-service
        i. **Dockerfile** for UI Service/Image
        ii. **Deployment file:** kube-ui-deployment.yaml
        iii. **Service file:** kube-ui-service.yaml
    b. Also attached in the Appendix A contents of each file
    c. You can look at other additional files I have attempted:
        i. uipod.yaml
        ii. Docker for nginx which for some reason did not work on AWS EC-2 instance

Payment Service

I have written code for Spring Boot based REST webservice for Payment Service. It has a POST operation available at /payment to take payments. It supports two modes:

- Cash
- Credit

Sample Invocation Usage:

```
curl --location --request POST 'http://<IP of Container or
Service>:8081/payment?user=cgs&reference=paycash&amount=1'
```

This payment service supports CPU restrictions and Horizontal POD Autoscaler.

This payment service supports Cluster IP

Docker

1. Docker image for Payment Service is available at
    a. https://hub.docker.com/r/chetangsanghi/kube-payment-service
2. This is being used then by Kubernetes Deployment and Service for UI Service as per below details

Kubernetes

1. Dockerfile for Payment Service image build up
2. Following Kubernetes Artifacts are available at repository https://github.com/chetansanghiwork/kube-payment-service as well as in Appendix B for reference
    - Deployment [kube-payment-service-deployment.yaml]
    - Service [kube-payment-service.yaml]
    - HPA [payment-hpa.yml]
3. Additional artifacts:
    - Source code for REST Service
    - paymentpod.yaml – Pod for payment service
    - Curl script for testing the service
    - Miscellaneous notes in Readme for what all I attempted

User Service

I have written code for Spring Boot based REST webservice for User Service. It has a GET operation available at /user/validate to validate an user.

Sample Invocation Usage

```
curl --location --request GET 'http://<IP of POD or
Service>:8090/user/validate?username=abc123'
```

This user service supports CPU restrictions and Horizontal POD Autoscaler.

This user service supports Cluster IP

Docker

1. Docker image for User Service is available at
    a. https://hub.docker.com/r/chetangsanghi/kube-user-service
2. This is being used then by Kubernetes Deployment and Service for UI Service as per below details

Kubernetes

1. Dockerfile for User Service image build up
2. Following Kubernetes Artifacts are available at repository https://github.com/chetansanghiwork/kube-user-service as well as in Appendix C for reference
    1. Deployment [kube-user-service-deployment.yaml]
    2. Service [kube-user-service.yaml]
    3. HPA [user-hpa.yml]
3. Additional artifacts:
    1. Source code for REST Service
    2. userpod.yaml – Pod for User service
    3. Curl script for testing the service

Accounting Service

I have written code for Spring Boot based REST webservice for User Service. It has a GET operation available at /account/validate to validate an account.

Sample Invocation Usage

```
curl --location --request GET 'http://<IP of POD or
Service>:9000/account/validate?username=cgs200&account=10000'
```

This accounting service supports CPU restrictions and Horizontal POD Autoscaler.

This accounting service supports Cluster IP

Docker

1. Docker image for User Service is available at
    a. https://hub.docker.com/r/chetangsanghi/kube-accounting-service
2. This is being used then by Kubernetes Deployment and Service for UI Service as per below details

Kubernetes

1. Dockerfile for Accounting Service image build up
2. Following Kubernetes Artifacts are available at repository https://github.com/chetansanghiwork/kube-accounting-service as well as in Appendix D for reference
    - Deployment [kube-accounting-service-deployment.yaml]
    - Service [kube-accounting-service.yaml]
    - HPA [accounting-hpa.yml]
3. Additional artifacts:
    - Source code for REST Service
    - accountingpod.yaml – Pod for Accounting service
    - Curl script for testing the service

Database

I attempted creating database first for maria db and then for mysql and that too with persistence volume configuration. Due to space restrictions on AWS EC2 instance could not do as desired.

However, I successfully accomplished following:

1. **Created a generic OPAQUE secret containing username.txt and password.txt**

kubectl create secret generic dbsecret --from-file=username.txt --from-file=password.txt

2. Used direct Docker to run it as container

docker run --name some-mariadb -e MYSQL_ROOT_PASSWORD=my-secret-pw -d mariadb:latest

3. Then test this to connect using another container to mariadb and create a database

   docker run -it --network bridge --rm mariadb mysql -h <IP of POD above with mariadb> -u root -p

4. Used this secret to create a POD for mariadb to use the password from secret and run mariadb in that POD –
   a. **kubectl apply -f mariadb.yml**
   b. *Contents of mariadb.yml file:*

```
apiVersion: v1
kind: Pod
metadata:
 name: my-database
spec:
 containers:
 - name: some-mariadb
  image: mariadb:latest
  env:
  - name: MYSQL_ROOT_PASSWORD
   valueFrom:
    secretKeyRef:
     name: dbsecret
     key: password.txt
```

Git Hub Repository – Database

Please find below Git Hub Repository for Artifacts related to Database & also available in Appendix E

1. Repository Location
    a. https://github.com/chetansanghiwork/kube-database
2. File for DB Pod
    a. mariadb.yml
3. There are other miscellaneous artefacts where I tried following:
    a. Running this DB as deployment - kube-database-deployment.yml
    b. Running mysql with persistence volume and without persistence volume

Appendix A – UI Service

Dockerfile

```
FROM httpd:2.4
MAINTAINER Chetan
COPY ./public-html /usr/local/apache2/htdocs
EXPOSE 80
```

Kubernetes Deployment For UI Service

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kube-ui-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      run: my-ui
  template:
    metadata:
      labels:
        run: my-ui
    spec:
      containers:
      - name: ui00
        image: chetangsanghi/kube-ui-httpd:0.1
        ports:
        - containerPort: 80
```

Kubernetes Service for UI Service

```
kind: Service
apiVersion: v1
metadata:
  name: uiservice
spec:
  ports:
    - port: 80
      targetPort: 8080
  selector:
    run: my-ui
  type: NodePort
```

Appendix B – Payment Service

Dockerfile

```
# Use the official maven/Java 8 image to create a build artifact.
# https://hub.docker.com/_/maven
FROM maven:3.5-jdk-8-alpine as builder

# Copy local code to the container image.
WORKDIR /app
COPY pom.xml .
COPY src ./src

# Build a release artifact.
RUN mvn package -DskipTests

# Use AdoptOpenJDK for base image.
# It's important to use OpenJDK 8u191 or above that has container support enabled.
# https://hub.docker.com/r/adoptopenjdk/openjdk8
# https://docs.docker.com/develop/develop-images/multistage-build/#use-multi-stage-builds
FROM adoptopenjdk/openjdk8:jdk8u202-b08-alpine-slim

# Copy the jar to the production image from the builder stage.
COPY --from=builder /app/target/PaymentService-1.0.0.jar /PaymentService.jar

# Run the web service on container startup.
CMD ["java", "-Djava.security.egd=file:/dev/./urandom", "-jar", "/PaymentService.jar"]
```

Kubernetes Deployment for Payment Service

```
apiVersion: apps/v1
```

```
kind: Deployment
metadata:
  name: my-payment-deployment
spec:
  selector:
    matchLabels:
      run: my-payment
  replicas: 2
  template:
    metadata:
      name: my-payment-pod
      labels:
        run: my-payment
    spec:
      containers:
      - name: payment01
        image: chetangsanghi/kube-payment-service:0.1
        ports:
        - containerPort: 8081
        resources:
          limits:
            cpu: 500m
          requests:
            cpu: 200m
```

Kubernetes Service for Payment Service

```
kind: Service
apiVersion: v1
metadata:
  name: paymentservice
```

```
spec:
  ports:
    - port: 8081
      targetPort: 8081
  selector:
    run: my-payment
  type: ClusterIP
```

Kubernetes HPA for Payment Service

```
kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v1
metadata:
  name: my-payment-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: my-payment-deployment
  minReplicas: 1
  maxReplicas: 5
  targetCPUUtilizationPercentage: 50
```

Appendix C – User Service

Dockerfile

```
# Use the official maven/Java 8 image to create a build artifact.
# https://hub.docker.com/_/maven
FROM maven:3.5-jdk-8-alpine as builder

# Copy local code to the container image.
WORKDIR /app
COPY pom.xml .
COPY src ./src

# Build a release artifact.
RUN mvn package -DskipTests

# Use AdoptOpenJDK for base image.
# It's important to use OpenJDK 8u191 or above that has container support enabled.
# https://hub.docker.com/r/adoptopenjdk/openjdk8
# https://docs.docker.com/develop/develop-images/multistage-build/#use-multi-stage-builds
FROM adoptopenjdk/openjdk8:jdk8u202-b08-alpine-slim

# Copy the jar to the production image from the builder stage.
COPY --from=builder /app/target/UserService-1.0.0.jar /UserService.jar

# Run the web service on container startup.
CMD ["java", "-Djava.security.egd=file:/dev/./urandom", "-jar", "/UserService.jar"]
```

Kubernetes Deployment for User Service

```
apiVersion: apps/v1
```

```
kind: Deployment
metadata:
  name: my-user-deployment
spec:
  selector:
    matchLabels:
      run: my-user
  replicas: 2
  template:
    metadata:
      name: my-user-pod
      labels:
        run: my-user
    spec:
      containers:
      - name: user01
        image: chetangsanghi/kube-user-service:0.1
        ports:
        - containerPort: 8090
        resources:
          limits:
            cpu: 500m
          requests:
            cpu: 200m
```

Kubernetes Service for User Service

```
kind: Service
apiVersion: v1
metadata:
  name: userservice
```

```
spec:
  ports:
    - port: 8090
      targetPort: 8090
  selector:
    run: my-user
  type: ClusterIP
```

Kubernetes HPA for User Service

```
kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v1
metadata:
  name: my-user-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: my-user-deployment
  minReplicas: 1
  maxReplicas: 3
  targetCPUUtilizationPercentage: 50
```

Appendix D – Accounting Service

Dockerfile

```
# Use the official maven/Java 8 image to create a build artifact.
# https://hub.docker.com/_/maven
FROM maven:3.5-jdk-8-alpine as builder

# Copy local code to the container image.
WORKDIR /app
COPY pom.xml .
COPY src ./src

# Build a release artifact.
RUN mvn package -DskipTests

# Use AdoptOpenJDK for base image.
# It's important to use OpenJDK 8u191 or above that has container support enabled.
# https://hub.docker.com/r/adoptopenjdk/openjdk8
# https://docs.docker.com/develop/develop-images/multistage-build/#use-multi-stage-builds
FROM adoptopenjdk/openjdk8:jdk8u202-b08-alpine-slim

# Copy the jar to the production image from the builder stage.
COPY --from=builder /app/target/AccountingService-1.0.0.jar /AccountingService.jar

# Run the web service on container startup.
CMD ["java", "-Djava.security.egd=file:/dev/./urandom", "-jar", "/AccountingService.jar"]
```

Kubernetes Deployment for Accounting Service

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-accounting-deployment
spec:
  selector:
    matchLabels:
      run: my-accounting
  replicas: 2
  template:
    metadata:
      name: my-accounting-pod
      labels:
        run: my-accounting
    spec:
      containers:
      - name: accounting01
        image: chetangsanghi/kube-accounting-service:0.1
        ports:
        - containerPort: 9000
        resources:
          limits:
            cpu: 500m
          requests:
            cpu: 200m
```

Kubernetes Service for Accounting Service

```
kind: Service
apiVersion: v1
metadata:
  name: accountingservice
spec:
  ports:
    - port: 9000
      targetPort: 9000
  selector:
    run: my-accounting
  type: ClusterIP
```

Kubernetes HPA for Accounting Service

```
kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v1
metadata:
  name: my-accounting-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: my-accounting-deployment
  minReplicas: 1
  maxReplicas: 3
  targetCPUUtilizationPercentage: 50
```

Appendix E – Database

Kubernetes POD for Database

```
apiVersion: v1
kind: Pod
metadata:
  name: my-database
spec:
  containers:
  - name: some-mariadb
    image: mariadb:latest
    env:
    - name: MYSQL_ROOT_PASSWORD
      valueFrom:
        secretKeyRef:
          name: dbsecret
          key: password.txt
```