Design and Develop Database Tier

The architecture of pages microservice consists of two major components, database tier and service tier. In this lab, we will focus only on the database tier. The database tier will be developed and deployed first, which paves the way for developing the application/service tier.

We will use database migrations for versioning our schema changes as our database evolves. There are various automation tools for implementing the database migrations. We will be using `Flyway` , as it is well adopted and widely accepted in the spring boot community.

Learning Outcomes

After completing the lab, you will be able to:

1. Develop the Database Tier using MySql
2. Create MySql deployment in K8s
3. Expose MySql to be discovered by other services within the cluster
4. Implement Database Migrations using Flyway locally and on the production cluster
5. Leverage the concept of K8s Jobs for background processing

Before starting the lab, checkout the `database-tier-start` task.

```
   git status
   # Ensure the source code is checked in to github
   # You can take a back up of your codebase, to keep your depl
oyment files from being overwritten
   # Checkout into a feature branch
   git checkout database-tier-start -b db
   # You are on branch db
```

In case you get an error when you cherry-pick/check-out, open `intellij` , right-click on the `project` , select `git → resolve-conflicts → View changes`

`and merge them based on the differences`

Creating the database locally

1. The updated codebase contains the refactored and re-organized pipeline. It also brings in database folder containing the DDL and DML scripts for creating the database and schema migrations. Take some time to walk-through them.
2. `database/create_database.sql` contains sql script for creating the database `pages` and user `pages_user` for developing and testing locally.
3. `database/migrations/V1__initial_schema.sql` contains the script for creating the schema for the tables.
4. Create the database locally.

   ```
   mysql -uroot < database/create_database.sql
   ```

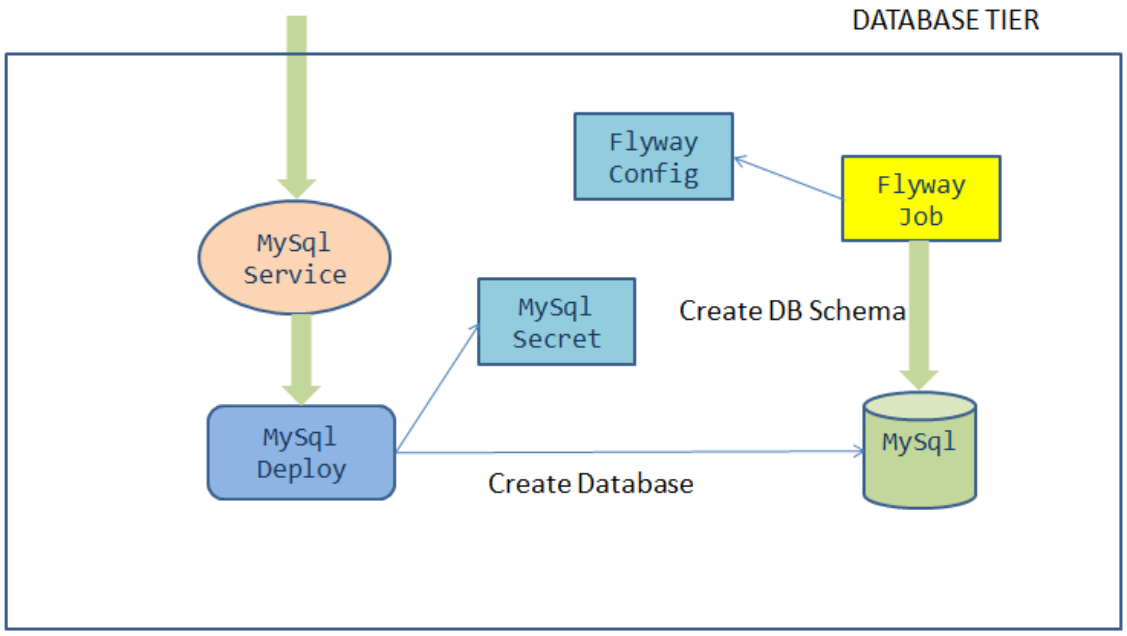
   They do not have any tables yet.
5. Run the flyway database migration command to create the schema

   `flyway -url="jdbc:mysql://localhost:3306/pages" -user=pages_user -password=password -locations=filesystem:database migrate`
6. Inspect the database created locally using mysql client.

   ```
   mysql -u pages_user -p
   Enter password: ********
   use pages;
   describe pages;
   ```

Kubernetize the Database Tier

Database Tier - Deployment Architecture



1. Create the file `deployment/mysql-secret.yaml` containing the manifest for `secret` in K8s with the following specs:

```
name-> mysql-pass
namespace -> [your namespace]
data ->
spring.datasource.password: [base64 encoded password]
password: [base64 encoded password]

For generating base64 encoded password use one of the 2 me
thods below: (Use the first method)
1.You can directly generate the secret in imperative way.

kubectl create secret generic mysql-pass --namespace=[stud
ent-name] --from-literal=spring.datasource.password=passwo
rd --from-literal=password=password --dry-run=client -o ya
ml

2. Use the command: echo -n password | base64 and copy tha
t into the yaml file if you want to write the yaml from sc
ratch.

For more information refer to http://kubernetes.io/docs
```

2. Create the file `deployment/mysql-deployment.yaml` containing the manifest for mysql deployment in K8s with the following specs:

```
name -> mysql
namespace -> [your namespace]
labels->   app: pages
           tier: database
selectors->   app: pages
              tier: database
image -> mysql:8.0
container name -> mysql
expose port -> 3306
Create a volume called mysql-persistent-storage of type em
ptyDir.
Mount the volume at path -> mountPath: /docker-entrypoint-
initdb.d
Add 5 environment variables to the container with key-valu
e pair as follows:

MYSQL_SERVICE_HOST: "pages-mysql"
MYSQL_SERVICE_PORT: 3306
MYSQL_DATABASE: pages
MYSQL_USER: root
MYSQL_ROOT_PASSWORD: value from a secret called mysql-pass
with key-name as password.
```

Constructing the deployment object in yaml file, as given in the above specification.

```
SOLUTION:

Generate the yaml file
kubectl create deploy mysql --image=mysql:8.0 --namespace=
student-name  --dry-run=client -o yaml

Copy the output onto the yaml file.

Edit/Update the labels & selectors as given in the specs.

Define volume under containers sections with name mysql-pe
rsistence-volume
and type -
emptyDir: {}

Mount the volume in the containers section. Use the name m
ysql-persistence-volume and mounthPath as given in the spe
c.

snippet for env section under containers:
          env:
```

```
            - name: MYSQL_ROOT_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mysql-pass
                  key: password
            - name: MYSQL_SERVICE_HOST
              value: "pages-mysql"
            - name: MYSQL_SERVICE_PORT
              value: "3306"
            - name: MYSQL_DATABASE
              value: "pages"
            - name: MYSQL_USER
              value: "root"

If you get stuck, read the documentation for creating a de
ployment, labels & selectors,  volumes, volumemounts, conf
igmaps, secrets and environment variables from http://kube
rnetes.io/docs
```

3. Create the `deployment/mysql-service.yaml` with the manifest for exposing the mysql deployment with following specs:

```
apiVersion: v1
kind: Service
metadata:
  name: pages-mysql
  namespace: student-name
  labels:
    app: pages
    tier: database
spec:
  ports:
    - port: 3306
  selector:
    app: pages
    tier: database
  type: ClusterIP
```

4. Create the file `deployment/flyway-configmap.yaml` with thethe manifest for generating the config map in K8s with the following specs:

```
name-> flyway-configmap
namespace -> [your namespace]
data ->
spring.datasource.username -> root
V1__inital_schema.sql -> This should be the content of you
```

```
r database/migrations/V1__inital_schema.sql

Refer to https://kubernetes.io/docs/concepts/configuratio
n/configmap/ on how k8s treats multiline values for a sing
le key.
```

5. Create the file `deployment/flyway-job.yaml` with manifest for running a K8s job with the following specs:

```
apiVersion: batch/v1
kind: Job
metadata:
  name: flyway-job
  namespace: [student-name]
  labels:
    app: pages
spec:
  template:
    spec:
      containers:
        - name: flyway
          image: flyway/flyway:6.4.4
          args:
            - info
            - migrate
            - info
          env:
            - name: FLYWAY_URL
              value: jdbc:mysql://pages-mysql/pages
            - name: FLYWAY_USER
              value: root
            - name: FLYWAY_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mysql-pass
                  key: password
            - name: FLYWAY_PLACEHOLDER_REPLACEMENT
              value: "true"
            - name: FLYWAY_PLACEHOLDERS_USERNAME
              valueFrom:
                configMapKeyRef:
                  name: flyway-configmap
                  key: spring.datasource.username
            - name: FLYWAY_PLACEHOLDERS_PASSWORD
              valueFrom:
                secretKeyRef:
```

```
                name: mysql-pass
                key: spring.datasource.password
        volumeMounts:
          - mountPath: /flyway/sql
            name: sql
    volumes:
      - name: sql
        configMap:
          name: flyway-configmap
    restartPolicy: Never
```

Testing locally

1. Switch the `kubectl` context to `minikube` and set the context to point to your namespace.
2. Create all the 5 resources inside your namespace. Wait for some time for the migration job to complete. Verify the resources were created without errors.
3. To verify the database was created with the table `pages` use `kubectl exec` to get a shell to the `mysql` container

```
kubectl get pods
#copy the name of mysql pod
kubectl exec -it <pod-name> -- sh
mysql -uroot -p
password

show databases;
use pages;
show tables;
describe pages;
exit ;
```

4. Verify `scripts/deploy-to-k8s.sh` is updated to create all the 5 resources
5. Update the docker image tag in `pipeline.yaml` and `pages-deployment.yaml` and verify they are same

Deploy to the production cluster

1. In the `pipeline.yml` file update the branch to pick up the commit from the feature branch. Replace it with the branch name `db`
2. Pushing the souce code to github

```
git status
git add .
git commit -m "database-tier"
git push -u origin db --tags
```

3. Ci/CD pipeline will trigger the build and deployment. Wait for it to succeed.
4. Switch the `kubectl` context to `production cluster` pointing to your namespace
5. Verify the database is provisioned as per the deployment architecture. Use `kubectl exec` into mysql pod through the terminal and verify the table `pages` is created with 1 successful migration. Also, you can check the logs from the flyway job.

The background job will not be deleted automatically. You will have to manually delete the flyway job by running `kubectl delete job flyway-job`

The database tier is now ready to accept requests.