Running Spring PetClinic on Kubernetes

In this exercise you will learn how to Dockerise a Java application: - Use the Maven base image to build and run tests. - Use the JRE base image for creating a production image. - Create Kubernetes Manifest files to run both the application and it's database. - Optimize application resource usage to run optimally on Kubernetes.

You don't need to have a JDK or JRE installed to complete this exercise!

1. Clone PetClinic

```
git clone https://github.com/spring-projects/spring-petclinic.git
cd spring-petclinic
```

2. Create a Dockerfile

We will use a two-stage build to create our Docker image for PetClinic. This allows us to use Maven in the first stage by just extending the default Maven image. This stage produces the jar file which is then copied over to the JRE Alpine image in the second stage. This way our production image will be as small as possible without any bloat from build tools and we can use an optimal base image in both stages. This makes our Dockerfile very simple.

Note you might need to change the version of PetClinic. You will get a file copy error if you have a newer version.

```
FROM maven:3.5.0-jdk-8 as build
WORKDIR /app
COPY src /app/src
COPY pom.xml /app
RUN mvn package

FROM java:8u111-jre-alpine
WORKDIR /app
COPY --from=build /app/target/spring-petclinic-1.5.1.jar /app
CMD ["java", "-jar", "spring-petclinic-1.5.1.jar"]
```

3. Build your Docker image

```
docker build -t eu.gcr.io/adam-k8s/petclinic -t petclinic .
```

You can use `docker images | grep petclinic` to see your created image. The image name has to container the name of the image registry we are pushing to and the gcloud project. We also created a version that is just tagged `petclinic` for local testing.

4. Run the application locally

By default PetClinic will run with an embedded HSQLDB database. This makes it convenient to test whether our image is working properly.

```
docker run -ti -p 8080:8080 petclinic
```

If you can access the application on `http://localhost:8080` then you can continue to the next step.

5. Push the image to the Google Container Registry

Use `gcloud` to push the image to the registry:

```
gcloud docker -- push eu.gcr.io/adam-k8s/petclinic
```

6. Run the application on Kubernetes

After we made sure our PetClinic container is working, we can wrap it in a manifest file and deploy to Kubernetes.

We will create a LoadBalancer Service for our Deployment which will in turn create a cloud LoadBalancer in GCE. Our deployment will configure the PetClinic app to activate the mysql Spring profile. It will also configure database access. Note the URL to the MySQL server (spring_datasource_url). MySQL is accessed using a DNS name, no need for IP addresses inside the cluster.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: petclinic
spec:
  ports:
    - port: 80
      targetPort: 8080
  selector:
    app: petclinic
  type: LoadBalancer
---
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: petclinic
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: petclinic
    spec:
      containers:
      - image: eu.gcr.io/adam-k8s/petclinic
        name: petclinic
        env:
        - name: "spring_profiles_active"
```

```
      value: "mysql"
    - name: "spring_datasource_url"
      value: "jdbc:mysql://mysql/petclinic"
    - name: "spring_datasource_username"
      value: root
    - name: "spring_datasource_password"
      value: "password"
    - name: "spring_datasource_initialize"
      value: "true"
    ports:
    - containerPort: 8080
```

Create a file called petclinic.yaml with the above content and let `kubectl` apply it to your cluster.

```
kubectl apply -f petclinic.yaml
```

Observe the creation of you objects using

```
watch kubectl service,deployment,pod
```

You will see that your pod is failing to start and is getting restarted by Kubernetes. Use

```
kubectl get pods
kubectl logs petclinic-[insertyourpodidhere]
```

to see the logs of one of your pods. You will see it's trying to connect to MySQL which is not available as we have not started it yet.

7. Start MySQL on Kubernetes

Configure the standard storage class. Configuring a StorageClass will allow our deployment of MySQL to request persistent storage from GCE automatically.

```
kubectl apply -f resources/standard-storage-class-gcepd.yaml
```

We can now deploy MySQL and define a Service for it.

```
kubectl apply -f resources/mysql.yaml
```

Verify it gets started properly:

```
watch kubectl get pods
```

And create the `petclinic` database by execing into the MySQL pod (you will have to look up the exact name of the pod) and running the `mysql` client.

```
kubectl exec -ti mysql-340072548-zrj6f -- mysql -uroot -ppassword -e "create database petcli
```

After a while the PetClinic pods should restart, find MySQL and run successfully. Check your LoadBalancer Service for an external IP to load the page in your browser. Note that even though the Spring application doesn't support proper

waiting for the database to come online, Kubernetes can make up for it by restarting it when it crashes.