SE-441 Continuous Delivery and DevOps
Winter 2018-2019
Homework 8
Due On: March 14, 2019
50 points

Please submit your answers to D2L by March 14, 2019.

Assignment

This week you will automate the setup of the Spring PetClinic locally using Docker. You will also use docker compose to setup a 2-container system with the Spring PetClinic application in one container and a MySQL database in the other.

Docker

You will first use Docker to build a container image that runs Spring PetClinic with the embedded, in-memory database.

Install Docker

If you don't already have Docker installed, download and install it from: `https://www.docker.com`.

Note: If you're doing this on windows, you might run into some difficulties when Docker tries to start the default VM. The only solution I found that helped was to uninstall and re-install Docker.

Create a Dockerfile

Like the `Vagrantfile`, the `Dockerfile` is used to tell Docker how to perform its work. While it's possible to manually configure a Docker image, this is not considered good practice. Dockerfiles can be source-code controlled and manipulated as part of the build process.

Create a `Dockerfile` with the following properties:

1. A base image of "openjdk:8u191-jre-alpine".
2. Expose TCP port 8080.
3. The Spring PetClinic JAR file copied to the root directory.
4. An entrypoint to run the JAR when the container is started.

Once you have created the `Dockerfile`, build it to your local Docker repository. Give it a tag of "se441/spring-petclinic:standalone".

Run Your Dockerfile Locally

Run your Docker container and bind it to port 8484:

```
docker run --rm -p 8484:8080/tcp se441/spring-petclinic:standalone
```

Note: The `--rm` option ensures that the container is removed once it stops. It isn't strictly required, but it helps keep things tidy.

Docker Compose - MySQL Only

In this part of the assignment you will reconfigure the Spring PetClinic application to use `docker-compose` to activate a MySQL container.

Note: To do this, we're going to stores some of the container files on your local machine using Docker's `VOLUME` concept. This ensures that the data won't be lost if we need to rebuild the container and will improve startup times.

Update the Docker Compose File

Update the content of the `docker-compose.yml` file to look as follows:

```
version: '3'
services:
  mysql:
    image: mysql:5.7
    ports:
      - "3306:3306/tcp"
    environment:
      - MYSQL_ROOT_PASSWORD=petclinic
      - MYSQL_DATABASE=petclinic
    volumes:
      - "./conf.d:/etc/mysql/conf.d:ro"
      - "./data.d:/var/lib/mysql"
```

Reconfigure the Application

The first time we run Spring PetClinic with MySQL, we need to instruct it to create and populate the MySQL database.

1. Edit the file, `src/main/resources/application-mysql.properties`, by uncommenting the last property in the file. **Note:** This instructs the application to create and populate the database on each run of the application.
2. Rebuild the application using the command `mvn clean package`.
3. Start the database using the command `docker-compose up`.

   Note: On windows you may be required to approve Docker Compose's attempt to access the local drive.
4. In another command window, run the Spring PetClinic with the following command:

   `java -jar target/spring-petclinic-2.1.0.BUILD-SNAPSHOT.jar ↩ --spring.profiles.active=mysql`.
5. Once the application has started, visit the "Veterinarians" page. This will validate that the data was actually added to the MySQL database.
6. Stop the application and re-comment the property you uncommented in step #1. Then rebuild the application and re-run it as you did in step #4. You shouldn't notice any difference in operations, but now the database will not be re-initialized on application startup.

Update your Docker Image

Update your Docker image to add the following argument to the `ENTRYPOINT`:

```
--spring.profiles.active=mysql
```

Rebuild the Docker image and give it a new tag of "se441/spring-petclinic:mysql". Once the build has completed, run the container:

```
docker run --rm -p 8484:8080/tcp se441/spring-petclinic:mysql
```

Notice that the container start fails, ultimately because of a `ConnectException`, indicating that it is unable to open a connection to MySQL. We could address this with some fancy networking, but we'd rather let Docker handle it for us, and we do in the next section.

Docker Compose - Application and MySQL

In this part of the assignment you will reconfigure the Spring PetClinic application to use `docker-compose` to activate a MySQL container as well as the application server itself. We'll again use Docker Compose, but we'll need to make some changes to the existing configuration.

Update the Docker Compose File

Update the content of the `docker-compose.yml` file to look as follows:

```
version: '3'
services:
  mysql:
    image: mysql:5.7
    ports:
      - "3306:3306/tcp"
    environment:
      - MYSQL_ROOT_PASSWORD=petclinic
      - MYSQL_DATABASE=petclinic
    volumes:
      - "./conf.d:/etc/mysql/conf.d:ro"
      - "./data.d:/var/lib/mysql"
  webapp:
    image: se441/spring-petclinic:both
  ports:
    - "8484:8080/tcp"
  depends_on:
    - mysql
```

Reconfigure the Application – Again

Now that we want the application to be started via Docker Compose, we need to change it's configuration so that it can locate the MySQL server.

1. Edit the file, `src/main/resources/application-mysql.properties` and change the property, `spring.datasource.url` to have the value `jdbc:mysql://mysql/petclinic`.

   Note: Since the container is running in isolation, it no longer has a path to the MySQL server using `localhost` the way that it did in the first part of the previous exercise. Instead, we have to use the server name defined for the mysql service in the `docker-compose.yml` file. This name will automatically be exposed by Docker Compose when it creates the virtual network for the services within the file.

2. Rebuild the application using the command `mvn clean package`.

3. Rebuild your application container image and give it the tag "se441/spring-petclinic:both".

   Note: This is the same name we referenced in the image in the changes we made to the `docker-compose.yml` file in the previous section.

4. Startup the containers using Docker Compose: `docker-compose up`

   Note: There is the potential for a race condition here. Docker Compose does not provide a way for one application container to wait until another container's internal application has started although it is possible for it wait until a container has started. For example, the updated `docker-compose.yml` identifies a dependency from the `webapp` service on the `mysql` service. However, that does not mean that Docker Compose will wait until MySQL itself has completely started before it starts the `webapp` container – it only means that it waits until the `mysql` container has started, and that usually won't take very long. So it is possible that the `docker-compose up` will fail if the `webapp` container starts up faster than the `mysql` container. If that happens, simply stop the containers using `docker-compose down` and try again. There are other ways of dealing with this issue, but they're beyond what I'd like to cover this term.

Deliverables [50 pts]

For this week, please provide screen captures uploaded and embedded into the SUBMISSIONS.md file the show:

DOCKER

- 5 pts Your dockerfile. Please provide a link to this file rather than a screen capture.
- 5 pts Your running docker instance as shown by a ps command.
- 5 pts Your browser accessing the main page of the website from your local container.

DOCKER COMPOSE - MYSQL ONLY

- 5 pts The output from the `docker-compose up` command.
- 5 pts Your browser accessing the "Veterinarians" page of the website from your local container when you run the application from the host system.
- 5 pts A section of the stack trace generated when you attempt to run the application container that has been updated to use MySQL.

DOCKER COMPOSE - APP SERVER AND MYSQL

- 5 pts Your updated `docker-compose.yml` file containing the application server, built from your local Dockerfile, and the existing MySQL configuration. Please provide a link to this file rather than a screen capture.
- 5 pts Your updated `application-mysql.properties` file containing the URL change for the database server. Please provide a link to this file rather than a screen capture.
- 5 pts The output from the `docker-compose up` command.
- 5 pts Your browser accessing the "Veterinarians" page of the website from your local container.