Working with Databases

olsen software

Contents

1. Running MySQL in a Container

- Overview
- Defining a Dockerfile for a containerized MySQL
- Building a Docker image
- Running a Docker container

Overview

- In a "traditional" production environment (before the advent of containers)…
  - There would be a server machine running MySQL
  - Client applications would connect to that same MySQL instance

- In a "containerized" production environment…
  - You can run MySQL in a container
  - You can spin up any number of instances of the container
  - Client applications connect to a MySQL in any container

Defining a Dockerfile for a Containerized MySQL

- Here's a simple Dockerfile
  - Specifies a Docker image that runs MySQL

```
# Pull MySQL from Docker Hub (if not  already in local Docker
registry).
FROM mysql:5.7.19

# MySQL will run on port 3306 within the container.
EXPOSE 3306


# Set an environment variable, which MySQL will look for.
ENV MYSQL_ROOT_PASSWORD=c0nygre

# Copy a SQL script into the container.
COPY myschema.sql /docker-entrypoint-initdb.d
```

Dockerfile1-mysql

- Notes:
  - `mysql:5.7.19` is a standard Docker image on Docker Hub
  - You must set the `MYSQL_ROOT_PASSWORD` environment variable
  - MySQL auto-runs scripts in `/docker-entrypoint-initdb.d` the first time you start the container

Building a Docker Image

- Build the image as follows:

```
docker build -f Dockerfile1-mysql -t mysql1 .
```

- List images as follows, to verify the image has been created successfully:

```
docker image ls mysql1
```

Running a Docker Container

- Run a Docker container, based on the image you created on the previous slide

```
docker run -d --name mysql1container -p 3306:3306 mysql1
```

- List Docker container processes as follows, to verify your container is running:

```
docker container ps -a
```

2. Interacting with the Containerized MySQL

- Opening a Linux shell into the container
- Opening a MySQL prompt
- A word about persistence

Opening a Linux Shell into the Container

- In the previous section, you ran a Docker container named `mysql1container`
  - The container is running MySQL on port 3306
- You can open a Linux shell into the container

```
docker exec -it mysql1container bash
```

- You can then run Linux commands inside the container, to poke around its file system
  - What happens when you run the following command?

```
ls /var/lib/mysql/MYSCHEMA
```

  - What does this tell you?

Opening a MySQL Prompt

- You can connect to the containerized MySQL and open a MySQL prompt into it, as follows
  - You'll be prompted for the password - it's c0nygre

```
docker exec -it mysql1container mysql -u root -p
```

- A MySQL prompt appears, allowing you to execute SQL commands against the containerized database

```
USE MYSCHEMA;

SHOW TABLES;

SELECT * FROM EMPLOYEES;

UPDATE EMPLOYEES SET Salary=Salary*2;

SELECT * FROM EMPLOYEES;


EXIT
```

A Word about Persistence

- Stop/remove your container, and then run it again

```
docker rm -f mysql1container

docker run -d --name mysql1container -p 3306:3306 mysql1
```

- Connect a MySQL prompt to it again:

```
docker exec -it mysql1container mysql -u root -p
```

- See what data is in the database - what do you see, and what does this imply?

```
USE MYSCHEMA;
SELECT * FROM EMPLOYEES;
EXIT
```

3. Persisting Data in a Volume

- Overview
- Volumes
- Creating and managing volumes
- How to mount a volume in a container
- Mounting a volume into your MySQL container
- Updating data
- Verifying the data was persisted in the volume

Overview

- In the previous section, you saw what happens when you stop/remove a container
  - Its file system is wiped

- This isn't a great feature for containerized databases!
  - When the container goes away, so does its file system
  - So you lose your data!

- How do we achieve real persistence…?

Volumes

- Volumes are the preferred persistence mechanism for Docker containers
  - A volume is a persistent storage area, completely managed by Docker in a dedicated directory on the host computer

- A volume is external to, and independent of, any particular container
  - A volume can exist before any containers are created
  - A volume continues to exist after containers are removed

- Volumes also help you minimize the size of containers
  - Store data in a volume, rather than in the writable layer in a container's filesystem

Creating and Managing Volumes

- You use the Docker CLI to create and manage volumes…

- To create a volume:

```
docker volume create myvol1
```

- To list volumes:

```
docker volume ls
```

- To inspect a volume:

```
docker volume inspect myvol1
```

- To remove a volume (don’t do this just yet!):

```
docker volume rm myvol1
```

How to Mount a Volume in a Container (1 of 3)

- When you run a container, you can mount a volume into the container
  - You map the volume to a directory in the container filesystem

- You can use either of the following two flags to mount a volume in a container:
  - `--volume` (or `-v` for short)
  - `--mount`

- We'll show examples of how to use both of these flags in the following slides

How to Mount a Volume in a Container (2 of 3)

- General syntax for mounting a volume via `-v`

```
-v aVolume:aMountPoint:options
```

- aVolume
  - The name of the volume to mount
  - If the volume doesn't exist, Docker creates it for you
  - If omitted, Docker creates a volume with a unique name

- aMountPoint
  - The path where Docker mounts the volume within the container

- options
  - Comma-separated options, e.g. `ro` to mount volume as read-only

How to Mount a Volume in a Container (3 of 3)

- General syntax for mounting a volume via `--mount`

```
--mount type=volume,source=aVolume,destination=aMountPoint,readonly
```

- `type`
  - `type=volume` – Mount a Docker volume
  - `type=bind` – Bind to an existing directory on host machine
  - `type=tmpfs` – Bind to a tmpfs directory in memory

- `source` (or `src`)
  - The name of the volume to mount (must already exist)

- `destination` (or `dest`, or `target`)
  - The path where Docker mounts the volume in the container

Mounting a Volume into your MySQL Container

- First of all, make sure your previous MySQL container is stopped/removed

```
docker container rm -f mysql1container
```

- Then run your MySQL container again, and this time map a volume to its MySQL data storage directory

```
docker run -d --name mysql1container \
-v myvol1:/var/lib/mysql \
-p 3306:3306 \
mysql1
```

Updating Data

- Run the following command to open a MySQL prompt into the database (the password is c0nygre)

```
docker exec -it mysql1container mysql -u root -p
```

- In the MySQL prompt, enter the following SQL commands to update some data

```
USE MYSCHEMA;
SELECT * FROM EMPLOYEES;
UPDATE EMPLOYEES SET Salary=Salary*2;
SELECT * FROM EMPLOYEES;
EXIT
```

- Then stop/remove the container - this stops the container, but the data lives on in the volume

```
docker container rm –f mysql1container
```

Verifying the Data was Persisted in the Volume

- Run another instance of your MySQL container

```
docker run -d --name mysql1container \
-v myvol1:/var/lib/mysql \
-p 3306:3306 \
mysql1
```

- Open a MySQL prompt into the database again

```
docker exec -it mysql1container mysql –u root -p
```

- In the MySQL prompt, enter the following SQL commands

```
USE MYSCHEMA;
SELECT * FROM EMPLOYEES;
EXIT
```

4. Binding to a Native Directory

- Overview
- How to bind to a native directory

Overview

- The previous section showed how to mount a volume into a Docker container
  - Docker manages the volume
  - The volume persists across container starts/stops

- It's also possible to mount a host-machine directory into a Docker container
  - The host-machine directory is completely external to Docker
  - Docker doesn't manage the host-machine directory

How to Bind to a Native Directory

- The native directory must exist on your host computer
  - So run the following command, to create a native directory

```
mkdir /root/DoshDev/MyData
```

- You can then mount the native directory into a container directory as follows

```
docker run -d --name mysql1container \
--mount type=bind,source=/root/DoshDev/MyData,destination=/var/lib/mysql \
-p 3306:3306 mysql1
```

- MySQL will now use the native directory to store its data, you can see it as follows:

```
ls -l /root/DoshDev/MyData
```

Any Questions?