

A large, light gray play button icon with a white triangle pointing right, centered within a circle. The circle has a thick white border and is set against a dark gray background.

# Docker Compose

1. Linking containers manually
2. Using Docker Compose



# 1. Linking Containers Manually

- Overview
- Non-containerized application
- Containerized application
- Building the MySQL image
- Building the application image
- Running containers

# Overview

- In a realistic application, you'll have lots of containers. For example:
  - Container #1 running MySQL
  - Container #2 running an application that talks to container #1
- We're going to show how to link these containers together
  - So the application can get data out of the MySQL database

# A Non-Containerized Application

- Take a look at `demo-docker-compose-before`
  - It's a Spring Boot app that accesses data in a MySQL database
  - It's not containerized yet
- `application.properties` specifies connectivity info
  - It assumes MySQL is running on `localhost`
  - We'll need to change this property if MySQL is in a container

```
spring.datasource.url=jdbc:mysql://localhost:3306/MYSCHEMA?serverTimezone=UTC  
spring.datasource.username=root  
spring.datasource.password=c0nygre
```

`application.properties`

# Containerized Application

- Now take a look at `demo-docker-compose-after`
  - This is a containerized version of the app, with 4 new files...
- `Dockerfile-mysql` and `myschema.sql`
  - Builds a Docker image for MySQL, using the SQL script to create and populate tables
- `Dockerfile-app`
  - Builds a Docker image for the Spring Boot app
- `docker-compose.yaml`
  - Uses Docker Compose to simplify things (see later)

# Building the MySQL image

- `Dockerfile-mysql` builds an image for MySQL
  - Based on the standard MySQL image, with our database schema

```
FROM mysql:8.0.28
```

```
EXPOSE 3306
```

```
ENV MYSQL_ROOT_PASSWORD=c0nygre
```

```
COPY myschema.sql /docker-entrypoint-initdb.d
```

`Dockerfile-mysql`

- To build this image, run the following command:

```
docker build -f Dockerfile-mysql -t emps/mysql .
```

# Building the Application Image (1 of 2)

- Dockerfile-app builds an image for the Spring app

```
FROM openjdk:17

ADD target/employee-app-0.0.1.jar app.jar

RUN sh -c 'echo spring.datasource.url=jdbc:mysql://mysql:3306/MYSCHEMA?serverTimezone=UTC > application.properties'

RUN sh -c 'echo spring.datasource.username=root >> application.properties'

RUN sh -c 'echo spring.datasource.password=c0nygre >> application.properties'

ENTRYPOINT ["java","-jar","/app.jar"]
```

Dockerfile-app

- Note it creates a new `application.properties` file, which overrides the one embedded in the JAR
- Connects to a machine named **mysql** (not localhost)

## Building the Application Image (2 of 2)

- You can build the "Spring Boot app" Docker image as follows:

```
docker build -f Dockerfile-app -t emps/app .
```



# Running Containers

- Run a MySQL container as follows:

```
docker run --name mysql -d -p 3306:3306 emps/mysql
```

- Run an application container as follows:

```
docker run --name app --link mysql:mysql emps/app
```

- Note the **--link** option
- 1st **mysql** is the name of the container we want to link to
- 2nd **mysql** is the alias by which we'll refer to it in our container

## 2. Using Docker Compose

- Overview
- How Docker Compose works
- Defining a Docker Compose configuration file
- Building images and running containers

# Overview

- In the previous section, you ran each container individually
  - This is quite a manual process
  - You have to remember to get the ports and names correct
  - This is very error-prone!
- A better approach would be to automate the creation of Docker images and containers via a configuration file
  - You can achieve this using a tool called Docker Compose

# How Docker Compose Works

- You supply a configuration file
  - By default, you name the file `docker-compose.yaml`
- The configuration file specifies:
  - A list of services (how to build an image and run a container)
  - What volumes or mount points are needed by the containers
  - How the containers are linked together

# Defining a Docker Compose Config File (1)

- Here's the structure of the Docker Compose configuration file for our example:

```
version: '3.0'

services:

  mysql:
    # Details for the MySQL service ...

  app:
    # Details for the Spring Boot app service ...
```

`docker-compose.yaml`

# Defining a Docker Compose Config File (2)

Here's how we configure the MySQL service

```
mysql:
  container_name: mysql

  build:
    context: .
    dockerfile: Dockerfile-mysql

  image: emps/mysql

  ports:
    - "3306:3306"

  volumes:
    - /docker/emps/mysql:/var/lib/mysql

  restart: always

  environment:
    MYSQL_ROOT_PASSWORD: c0nygre

  command: --explicit_defaults_for_timestamp
```

`docker-compose.yaml`

# Defining a Docker Compose Config File (3)

app:

container\_name: app

build:

context: .

dockerfile: Dockerfile-app

image: emps/app

depends\_on:

- mysql

**links:**

- **mysql:mysql**

Here's how we configure the App service

- Note the link to the mysql container

docker-compose.yaml

# Building Images and Running Containers

- You can run the Docker Compose file as follows
  - Builds images if they don't already exist
  - Runs container instances

```
docker-compose up
```



A large, light gray play button icon is positioned on the left side of the slide. It consists of a white triangle pointing right, centered within a series of concentric circles that create a sense of depth and shadow.

# Summary

- Linking containers manually
- Using Docker Compose

