**Option 1: Use a Shared Docker Network**

This is the **recommended and easiest** method.

1. **Create a user-defined bridge network:**

docker network create my\_shared\_network

1. **Start both containers on the same network:**

docker run --name spring-app --network my\_shared\_network my-spring-image

docker run --name mysql-db --network my\_shared\_network mysql

1. **In the Spring app config:**

Use the **container name** as the hostname:

spring.datasource.url=jdbc:mysql://mysql-db:3306/mydb

**✅ Option 2: Connect Containers from Different Networks**

If the containers are already on **different networks**, you can **connect them manually**.

**1. Connect one container to the other's network:**

Assume:

* Spring Boot container is on spring\_net
* DB container is on db\_net

To allow Spring Boot to access DB:

docker network connect db\_net spring-app

Now spring-app is part of both spring\_net and db\_net, so it can reach services in db\_net.

You can confirm with:  
docker inspect spring-app | grep -A 20 Networks

**2. Use the service/container name of the target container:**

spring.datasource.url=jdbc:mysql://mysql-db:3306/mydb

**✅ Option 3: Use Docker Compose with Multiple Networks**

If you use **Docker Compose**, define multiple networks and attach services accordingly:

version: '3.8'

services:

spring-app:

image: my-spring-app

networks:

- app\_net

- shared\_net

mysql-db:

image: mysql

networks:

- db\_net

- shared\_net

networks:

app\_net:

db\_net:

shared\_net:

Now both services share shared\_net, and you can use:

spring.datasource.url=jdbc:mysql://mysql-db:3306/mydb

**🧠 Important Notes:**

* **Always use user-defined bridge networks** (not default bridge) to allow container name-based DNS.
* **Avoid using localhost** inside containers to refer to other containers — it only refers to the same container.
* **Test connectivity** using:  
  docker exec -it spring-app ping mysql-db