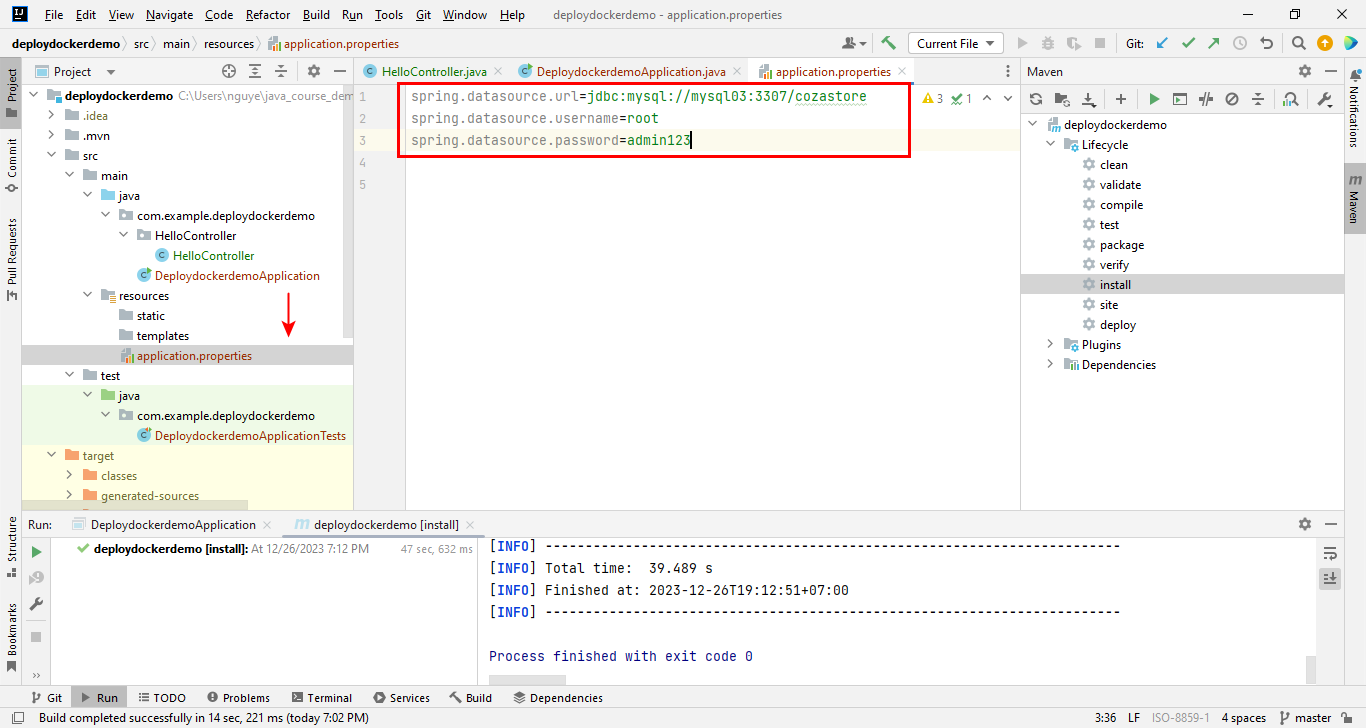
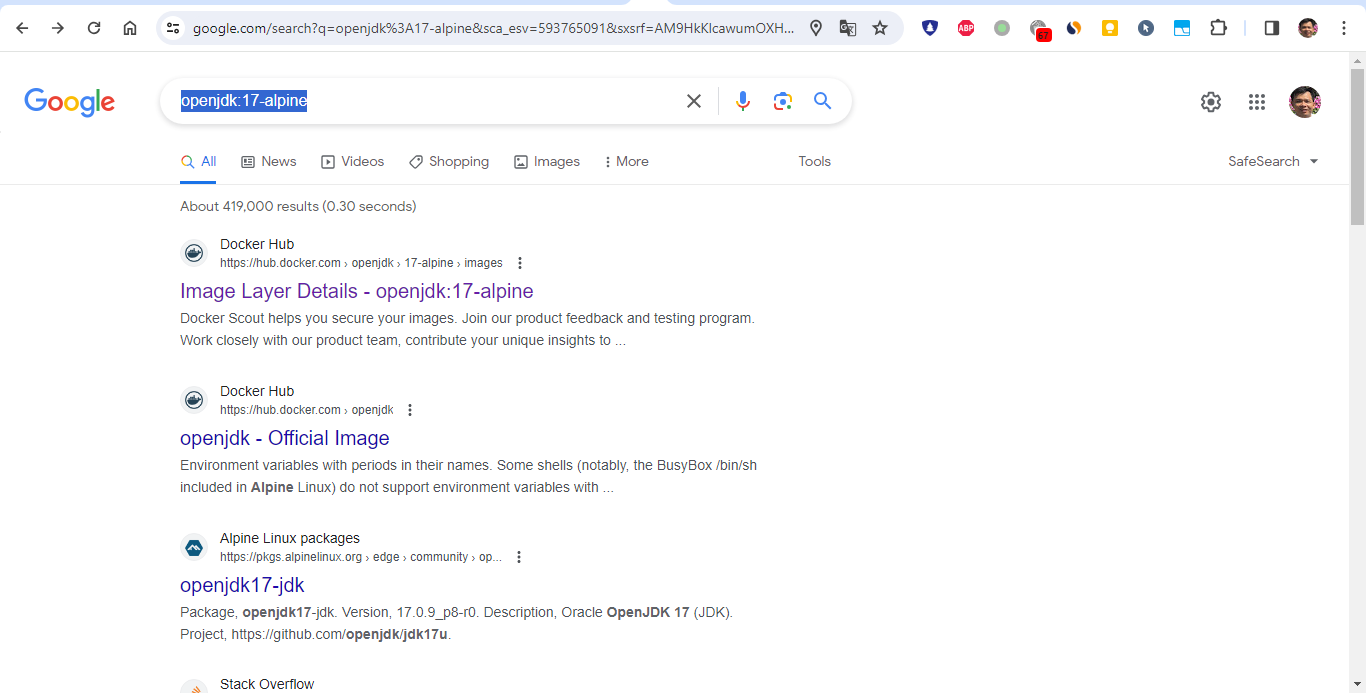


spring.datasource.url=jdbc:mysql://localhost:3307/cozastore  
spring.datasource.username=root  
spring.datasource.password=admin123

spring.datasource.url=jdbc:mysql://mysql03:3307/cozastore  
spring.datasource.username=root  
spring.datasource.password=admin123



openjdk:17-alpine



# Use an official Java runtime with Java 17 as a parent image

FROM openjdk:17-alpine

# Set the working directory in the container

WORKDIR /usr/app

# Copy the JAR file into the container at /usr/app

COPY ./deploydockerdemo-0.0.1-SNAPSHOT.jar /usr/app

# Make port 8080 available to the world outside this container

EXPOSE 8080

# Run the JAR file

ENTRYPOINT ["java", "-jar", "deploydockerdemo-0.0.1-SNAPSHOT.jar"]

Không nhất thiết mở port 8080 mà có mở bất kì port nào cũng được

# Use an official Java runtime with Java 17 as a parent image

FROM openjdk:17-alpine

# Set the working directory in the container

WORKDIR /usr/app

# Copy the JAR file into the container at /usr/app

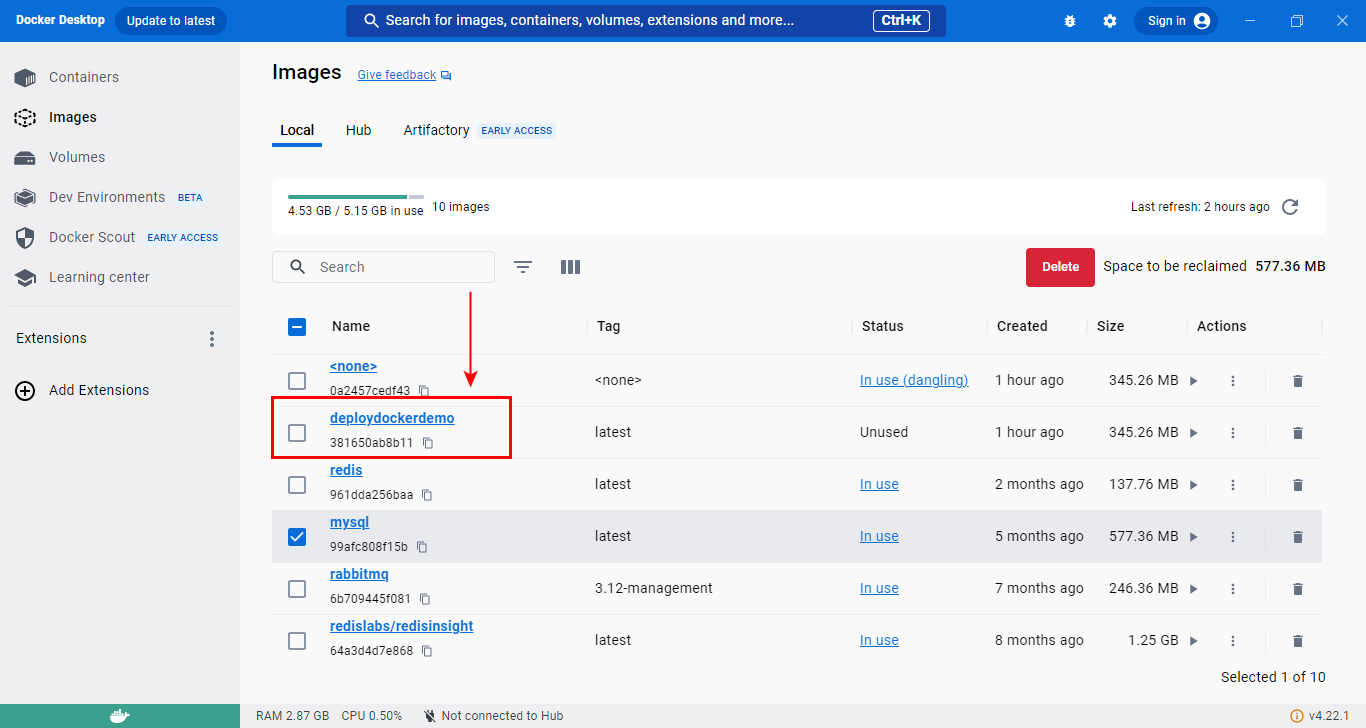
COPY ./deploydockerdemo-0.0.1-SNAPSHOT.jar /usr/app

# Make port 9091 available to the world outside this container

EXPOSE 9091

# Run the JAR file

ENTRYPOINT ["java", "-jar", "deploydockerdemo-0.0.1-SNAPSHOT.jar"]



C:\Users\nguye\java\_course\_demo\deployJAVAdocker>docker run --name deploydockerdemo03 -p 9092:9091 -d deploydockerdemo

e69e4e3c1045e8a26ea61fc7c955b037ef85e486aee760ccd9ea183c7bff9b6c

Explain command:

--name deploydockerdemo03: tên container image khi chạy

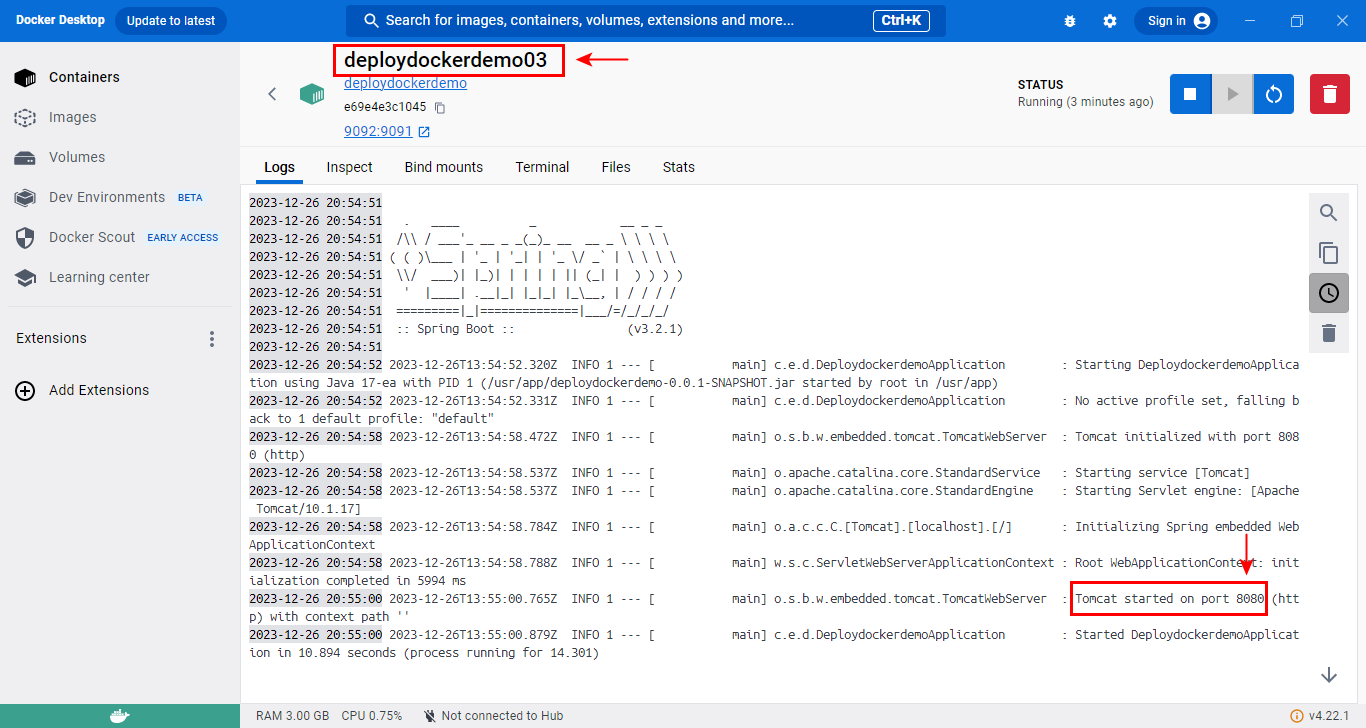
-p 9092:9091: WRONG

-p 9092:8080: RIGHT

-p<public port: private port>

# Make port 9091 available to the world outside this container

EXPOSE 9091 : WRONG



# Make port 8080 available to the world outside this container

EXPOSE 8080

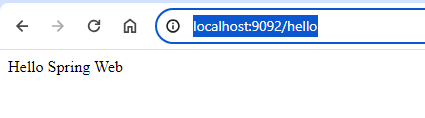
-> CORRECT

C:\Users\nguye\java\_course\_demo\deployJAVAdocker>docker run --name deploydockerdemo -p 9092:8080 -d deploydockerdemo03

C:\Users\nguye\java\_course\_demo\deployJAVAdocker>docker run --name deploydockerdemo03 -p 9092:8080 -d deploydockerdemo

01d69860302e7d6c632f0b96072e2490f69f8855a78bd77655d441aa8ff1a9c0

<http://localhost:9092/hello>



<https://spring.io/guides/topicals/spring-boot-docker/>

# Spring Boot Docker

Many people use containers to wrap their Spring Boot applications, and building containers is not a simple thing to do. This is a guide for developers of Spring Boot applications, and containers are not always a good abstraction for developers. They force you to learn about and think about low-level concerns. However, you may on occasion be called on to create or use a container, so it pays to understand the building blocks. In this guide, we aim to show you some of the choices you can make if you are faced with the prospect of needing to create your own container.

We assume that you know how to create and build a basic Spring Boot application. If not, go to one of the [Getting Started Guides](https://spring.io/guides) — for example, the one on building a [REST Service](https://spring.io/guides/gs/rest-service/). Copy the code from there and practice with some of the ideas contained in this guide.

|  |  |
| --- | --- |
|  | There is also a Getting Started Guide on [Docker](https://spring.io/guides/gs/spring-boot-docker), which would also be a good starting point, but it does not cover the range of choices that we cover here or cover them in as much detail. |

## A Basic Dockerfile

A Spring Boot application is easy to convert into an executable JAR file. All the [Getting Started Guides](https://spring.io/guides) do this, and every application that you download from [Spring Initializr](https://start.spring.io/) has a build step to create an executable JAR. With Maven, you run ./mvnw install, With Gradle, you run ./gradlew build. A basic Dockerfile to run that JAR would then look like this, at the top level of your project:

Dockerfile

COPYFROM eclipse-temurin:17-jdk-alpine

VOLUME /tmp

ARG JAR\_FILE

COPY ${JAR\_FILE} app.jar

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

You could pass in the JAR\_FILE as part of the docker command (it differs for Maven and Gradle). For Maven, the following command works:

COPYdocker build --build-arg JAR\_FILE=target/\*.jar -t myorg/myapp .

For Gradle, the following command works:

COPYdocker build --build-arg JAR\_FILE=build/libs/\*.jar -t myorg/myapp .

Once you have chosen a build system, you don’t need the ARG. You can hard code the JAR location. For Maven, that would be as follows:

Dockerfile

COPYFROM eclipse-temurin:17-jdk-alpine

VOLUME /tmp

COPY target/\*.jar app.jar

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

Then we can build an image with the following command:

COPYdocker build -t myorg/myapp .

Then we can run it by running the following command:

COPYdocker run -p 8080:8080 myorg/myapp

The output resembles the following sample output:

COPY. \_\_\_\_ \_ \_\_ \_ \_

/\\ / \_\_\_'\_ \_\_ \_ \_(\_)\_ \_\_ \_\_ \_ \ \ \ \

( ( )\\_\_\_ | '\_ | '\_| | '\_ \/ \_` | \ \ \ \

\\/ \_\_\_)| |\_)| | | | | || (\_| | ) ) ) )

' |\_\_\_\_| .\_\_|\_| |\_|\_| |\_\\_\_, | / / / /

=========|\_|==============|\_\_\_/=/\_/\_/\_/

:: Spring Boot :: (v2.7.4)

Nov 06, 2018 2:45:16 PM org.springframework.boot.StartupInfoLogger logStarting

INFO: Starting Application v0.1.0 on b8469cdc9b87 with PID 1 (/app.jar started by root in /)

Nov 06, 2018 2:45:16 PM org.springframework.boot.SpringApplication logStartupProfileInfo

...

If you want to poke around inside the image, you can open a shell in it by running the following command (note that the base image does not have bash):

COPYdocker run -ti --entrypoint /bin/sh myorg/myapp

The output is similar to the following sample output:

COPY/ # ls

app.jar dev home media proc run srv tmp var

bin etc lib mnt root sbin sys usr

/ #

|  |  |
| --- | --- |
|  | The alpine base container we used in the example does not have bash, so this is an ash shell. It has some but not all of the features of bash. |