

CS489: **Applied Software Development**

Author: Prof. O. Kalu
MIU Computer Science - November 2023

Lesson 10

Introduction to Containers and
Containerization using Docker

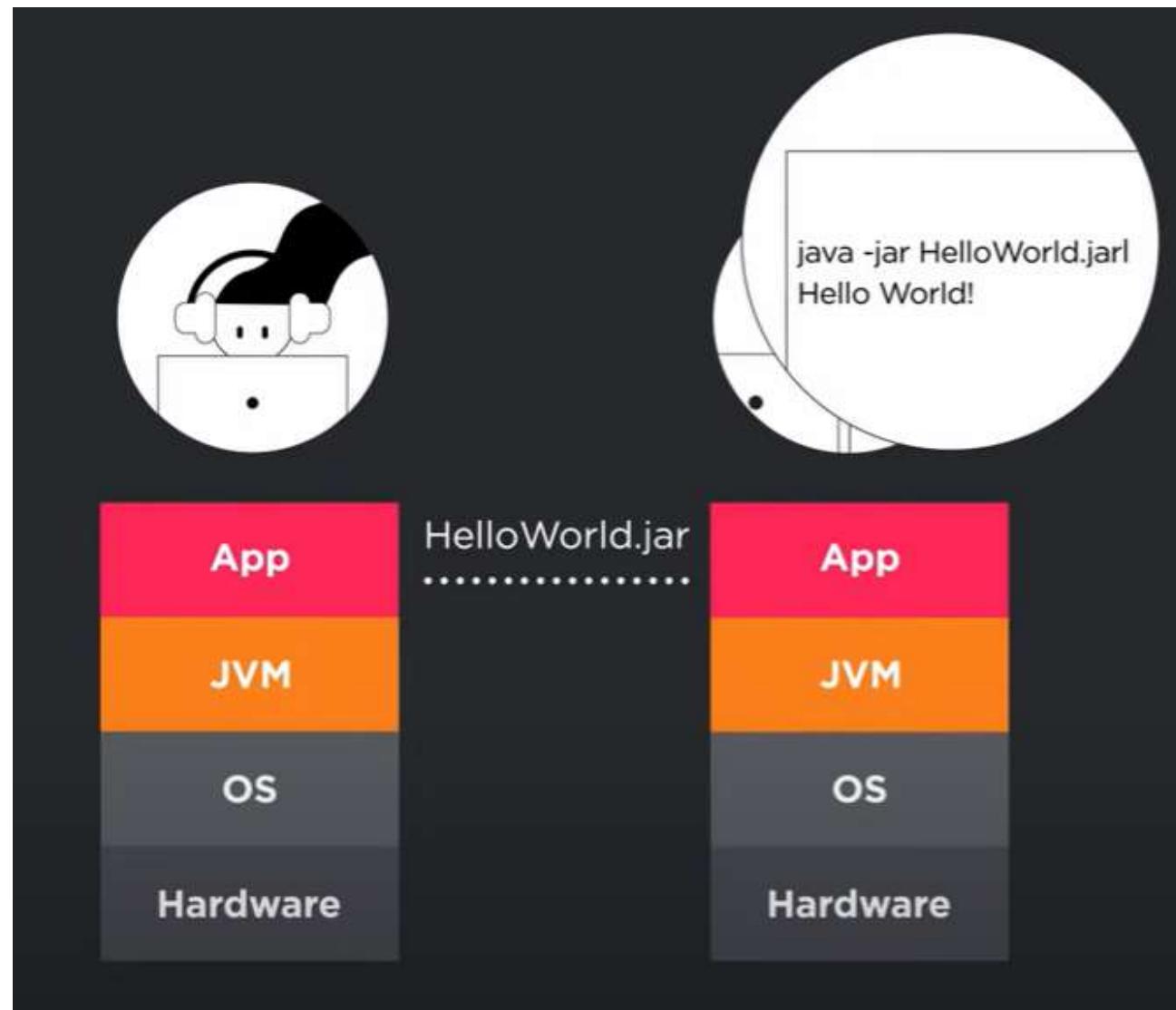
Wholeness

- Containerization is the practice of packaging software code together with the operating system (OS) libraries and any dependencies required to run the code, to create a single lightweight executable — called a container — that runs consistently on any infrastructure.
- Science of Consciousness: *The Whole is Contained in Every Part.*

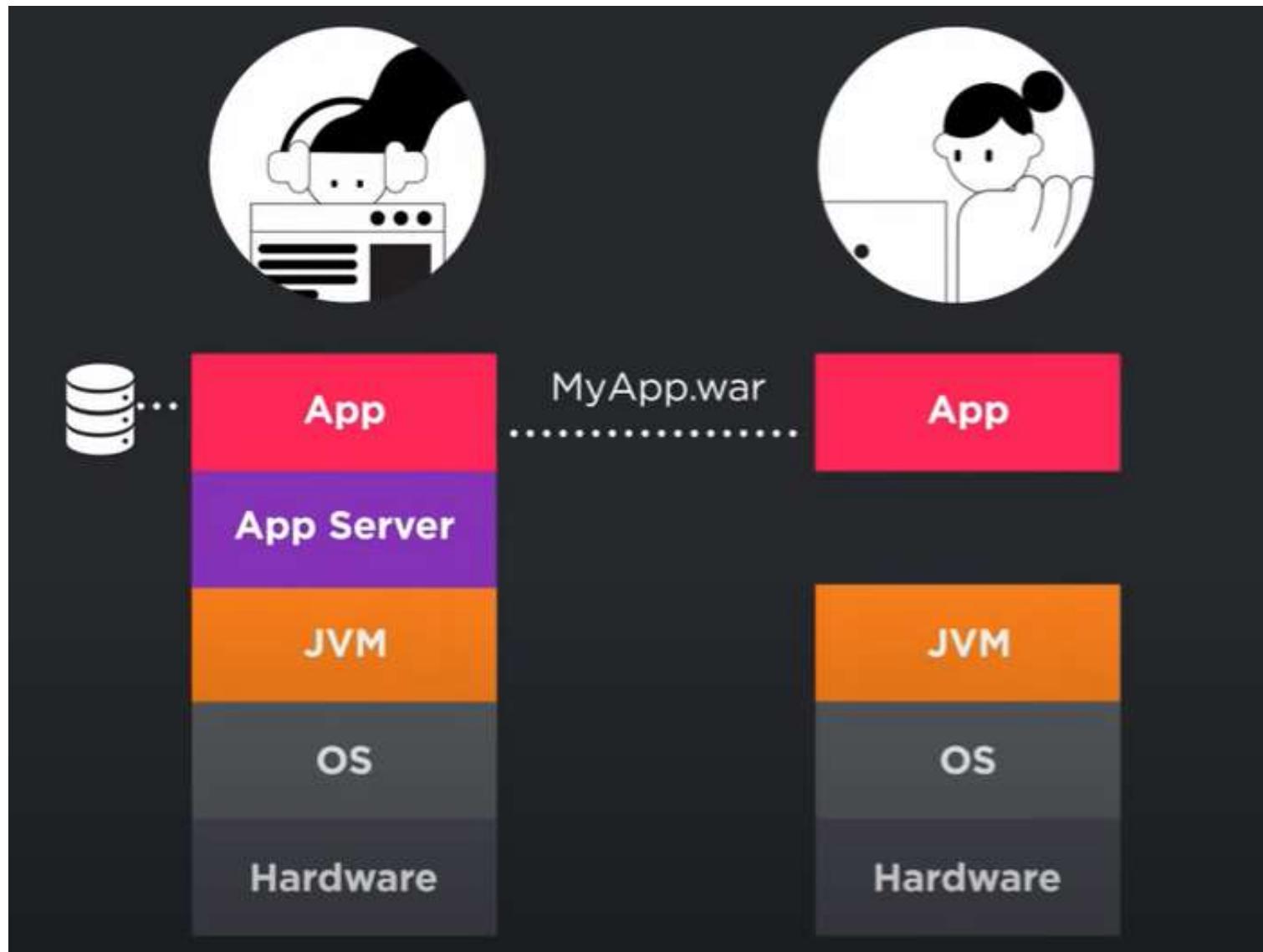
Why we need Containers

- Challenge of making application software run correctly across multiple execution environments/infrastructure
 - “the issue of works on one machine (development environment) but fails everywhere else (such as in production environment)

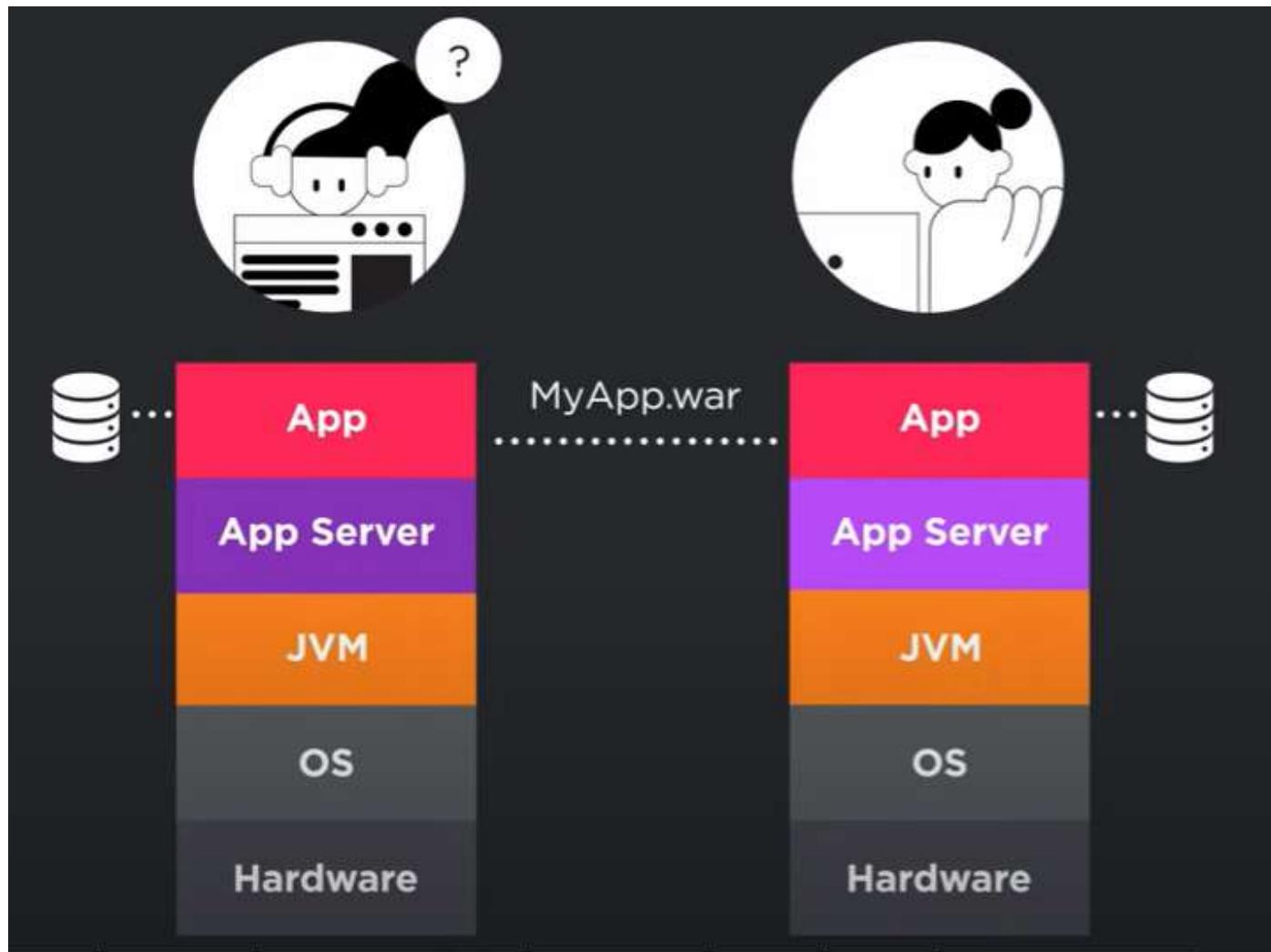
How to deploy/run app across Env



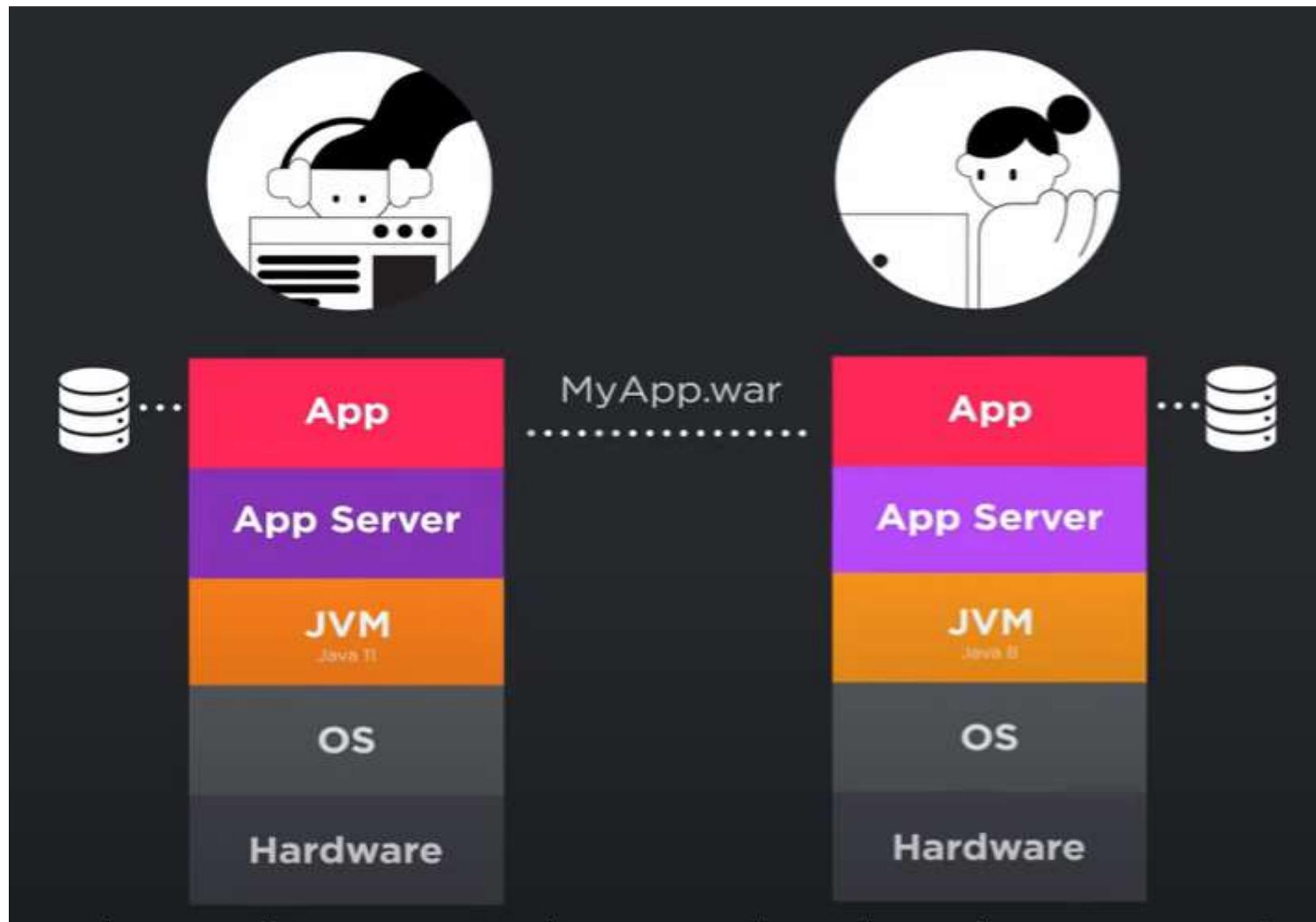
Deploy & run more complex App



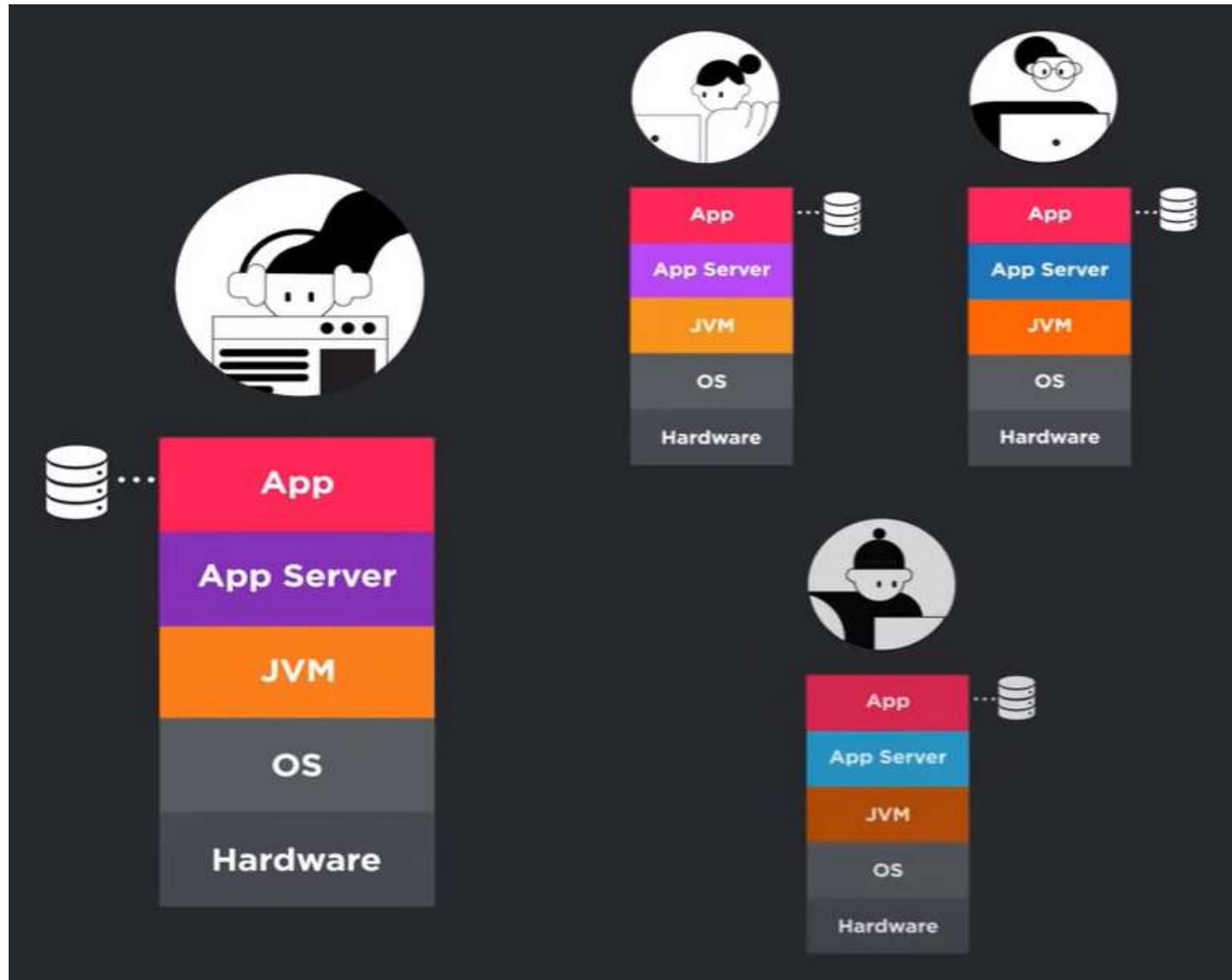
Deploy & run more complex app



Difference in the runtimes

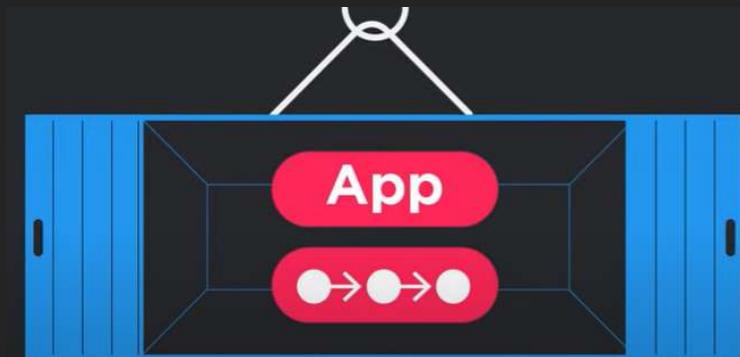


Need to deploy & run more instances



What is a Container?

- A Container is a software packaging mechanism which enables the encapsulation of an application together with all its dependencies into a single distributable unit
- It offers:
 - Lightweight solution (unlike VM)
 - Portability
 - Isolation



Containerization vs Virtualization

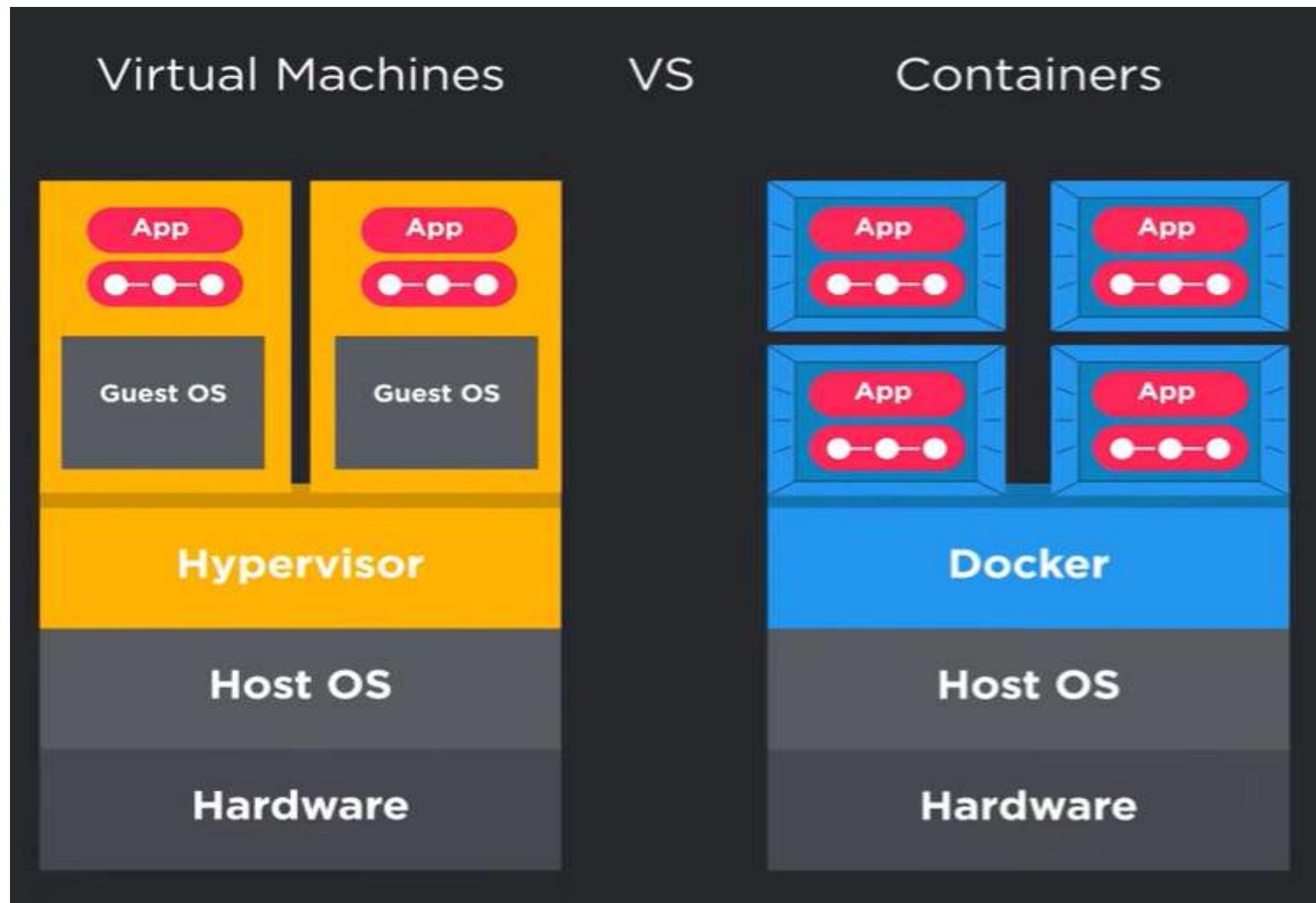
(Containers versus Virtual Machines)

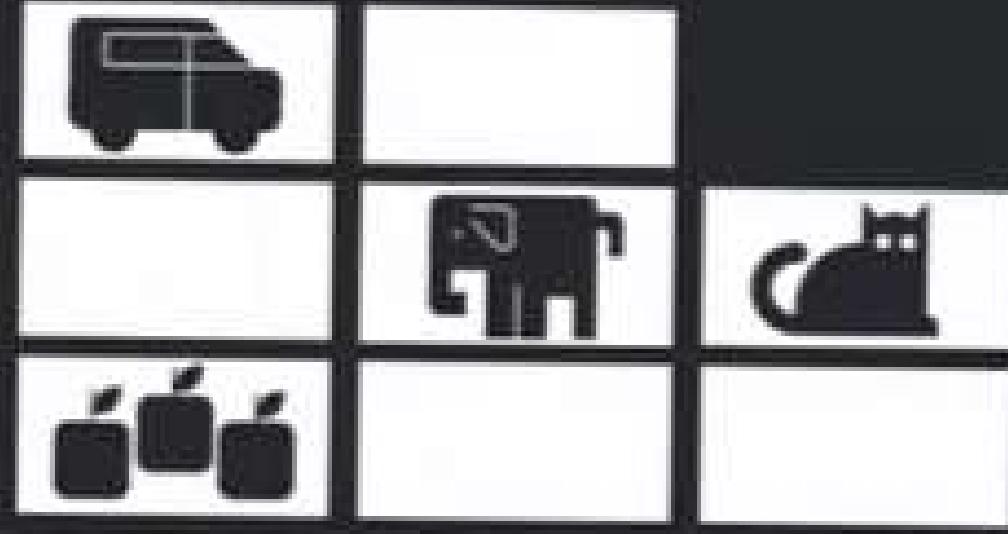
In contrast to Virtualization,
Containers are more portable and
more resource-efficient than virtual
machines (VMs).

Containers are ‘lightweight’ and
portable because they share the host
Operating system’s kernel

In contrast to Virtualization,
Containers are more portable and
more resource-efficient than virtual
machines (VMs).

Containers versus VMs





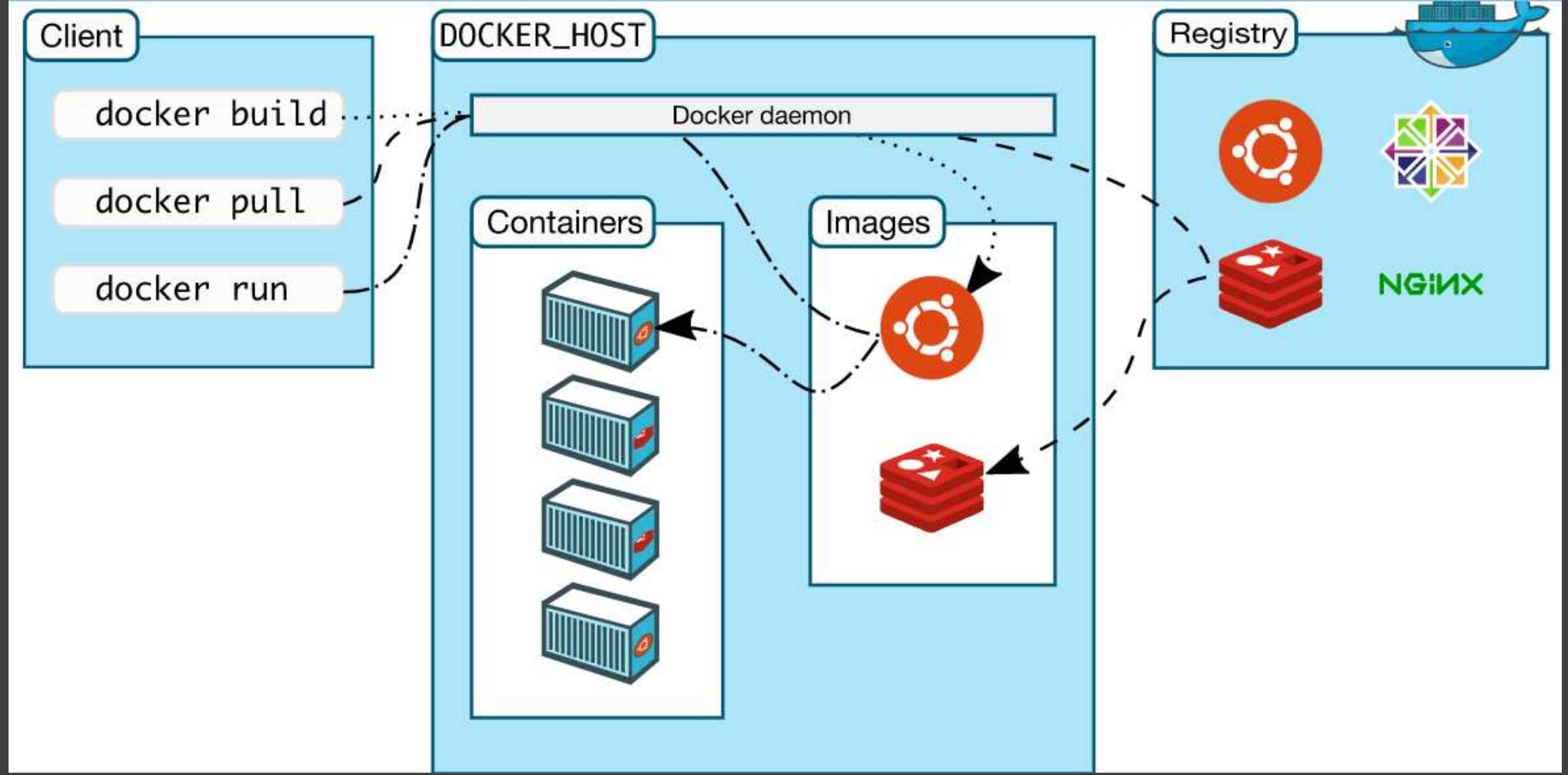
Origin of the
name,
“Container”

- Derived from the Shipping industry
- Shares quite similar characteristics

Container technologies

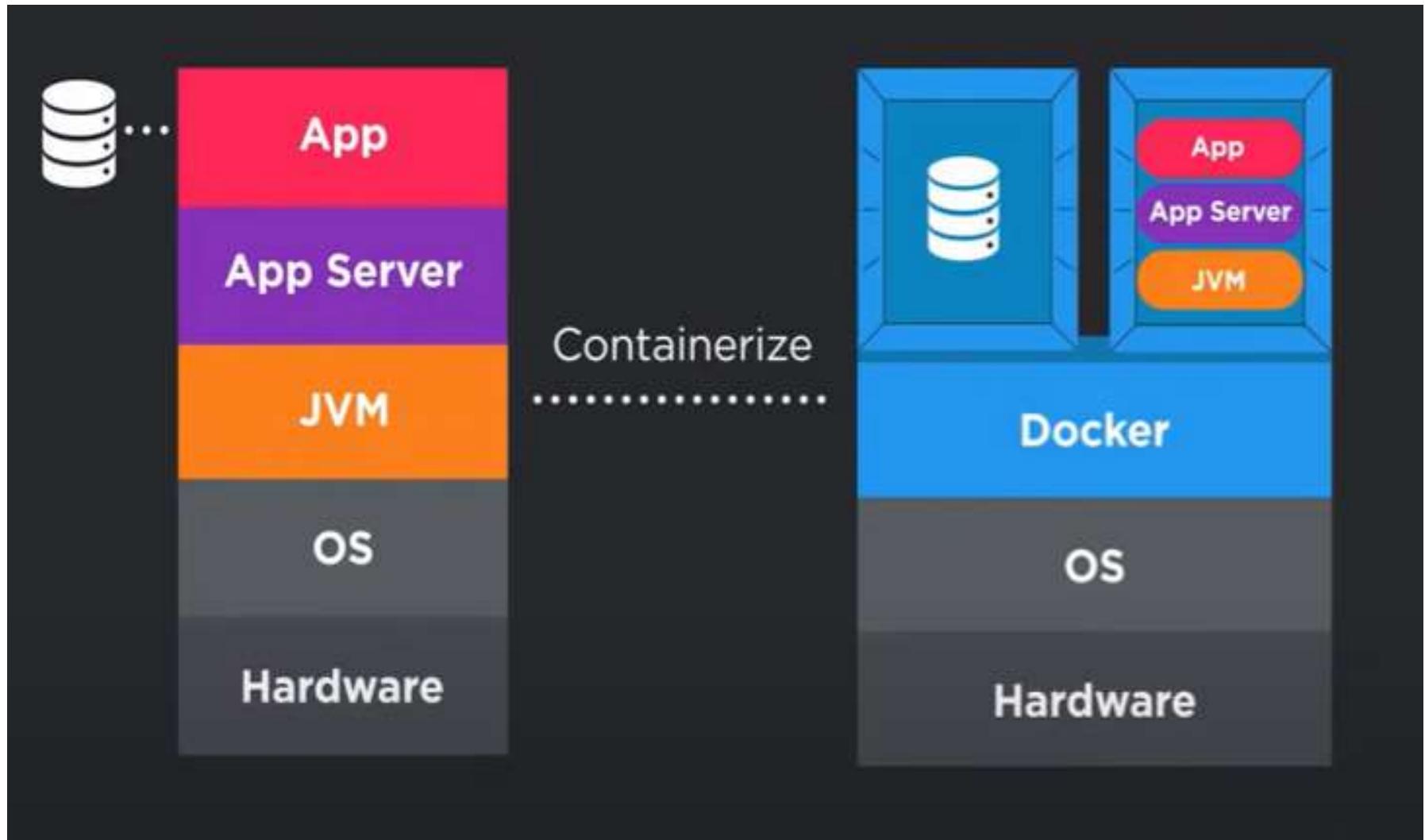
- Containers have become the de facto compute units of modern cloud-native applications.
- Docker is currently the premier/leading industry-standard containerization system
(<https://www.docker.com>)



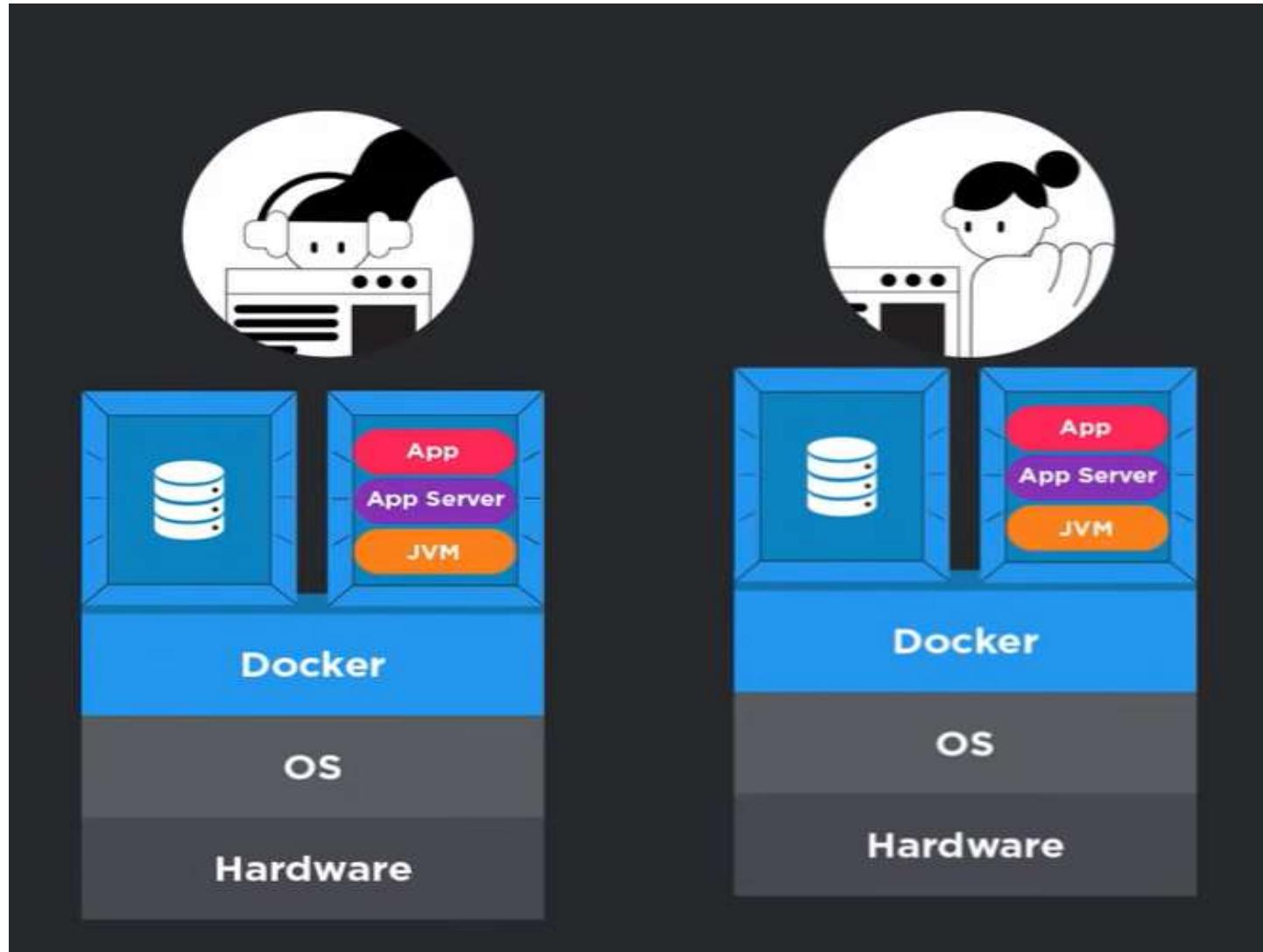


Architecture of Docker

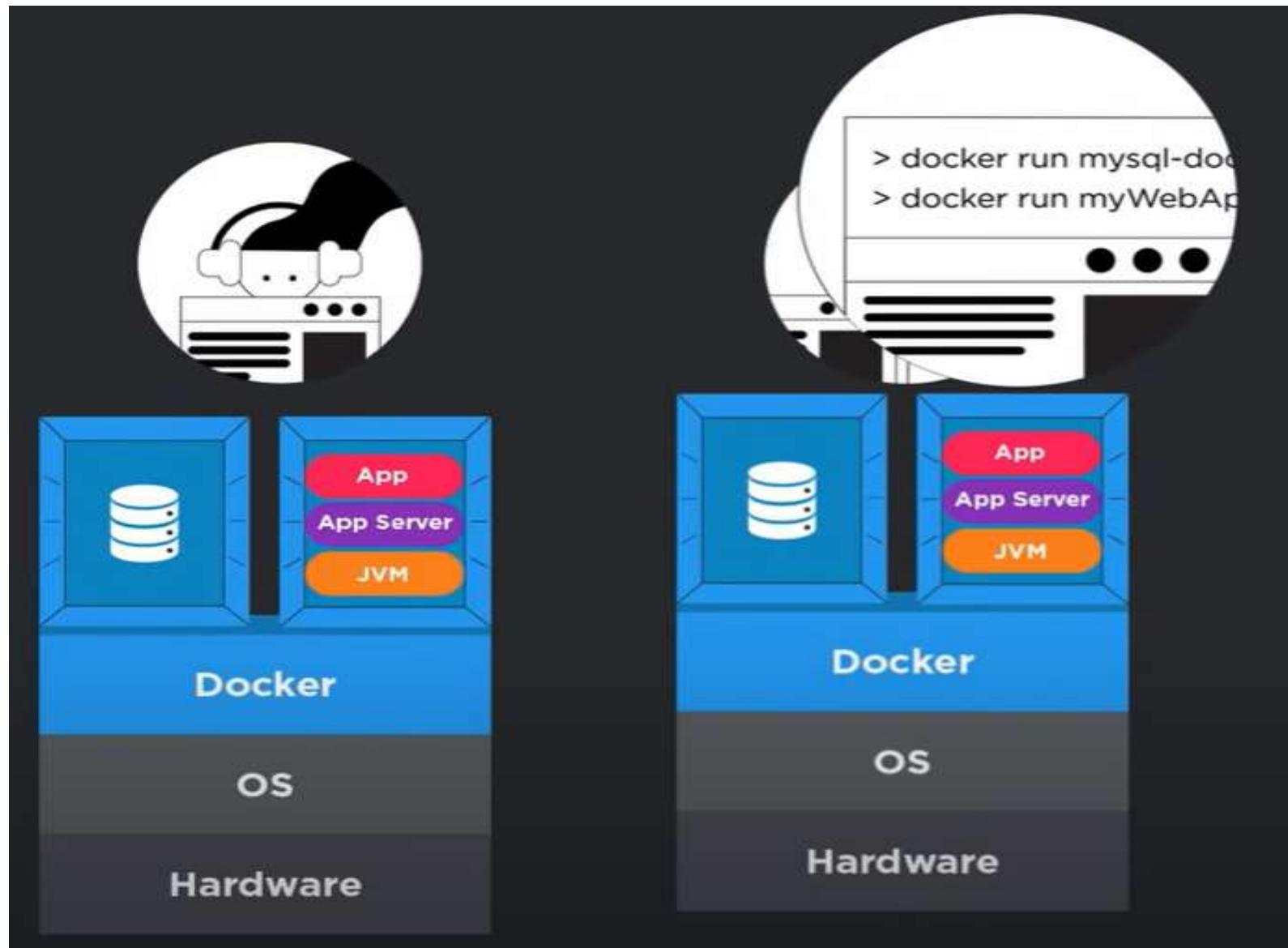
Containerize the app with docker



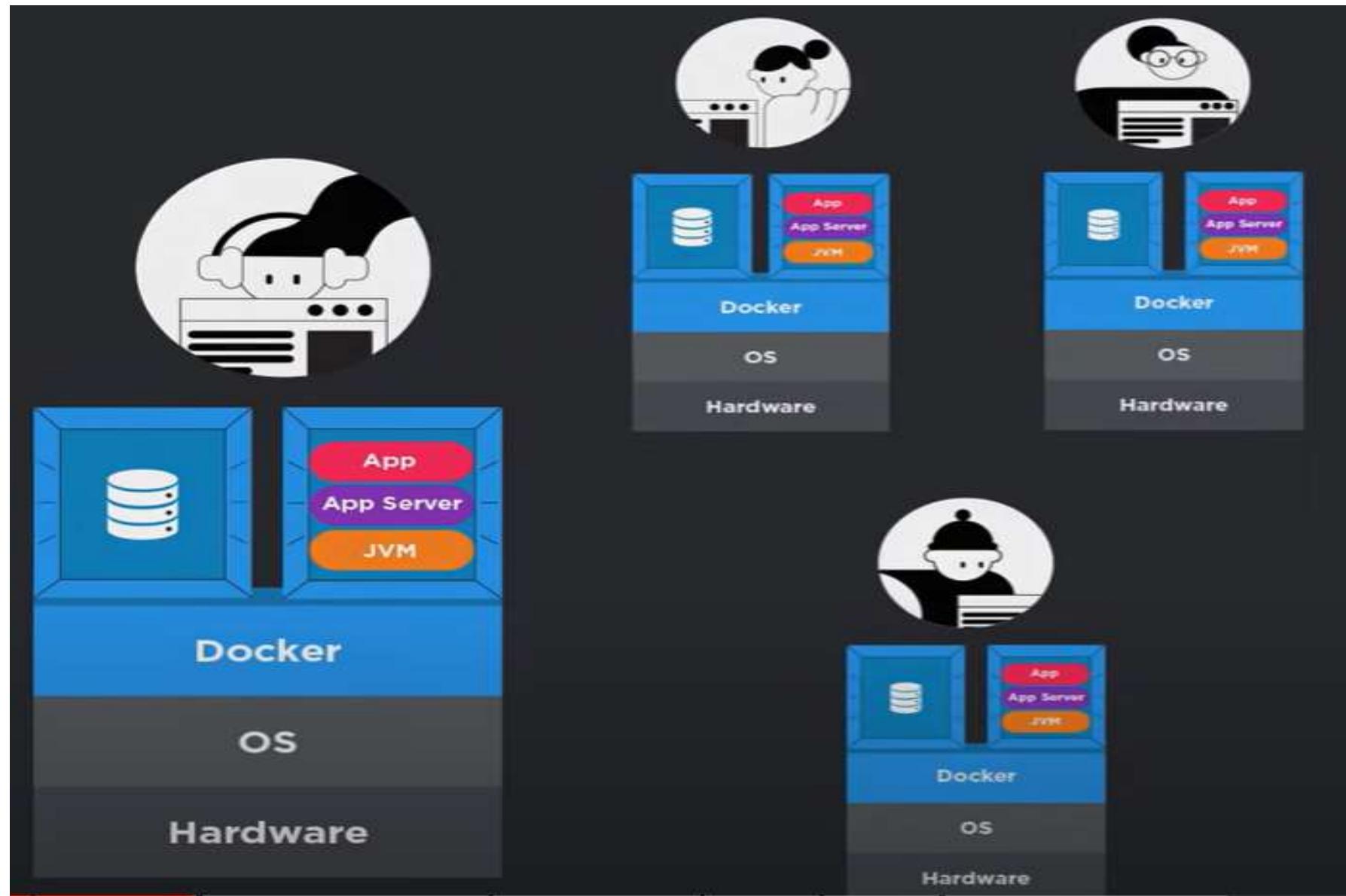
Replicate the configuration



Execute with docker run command

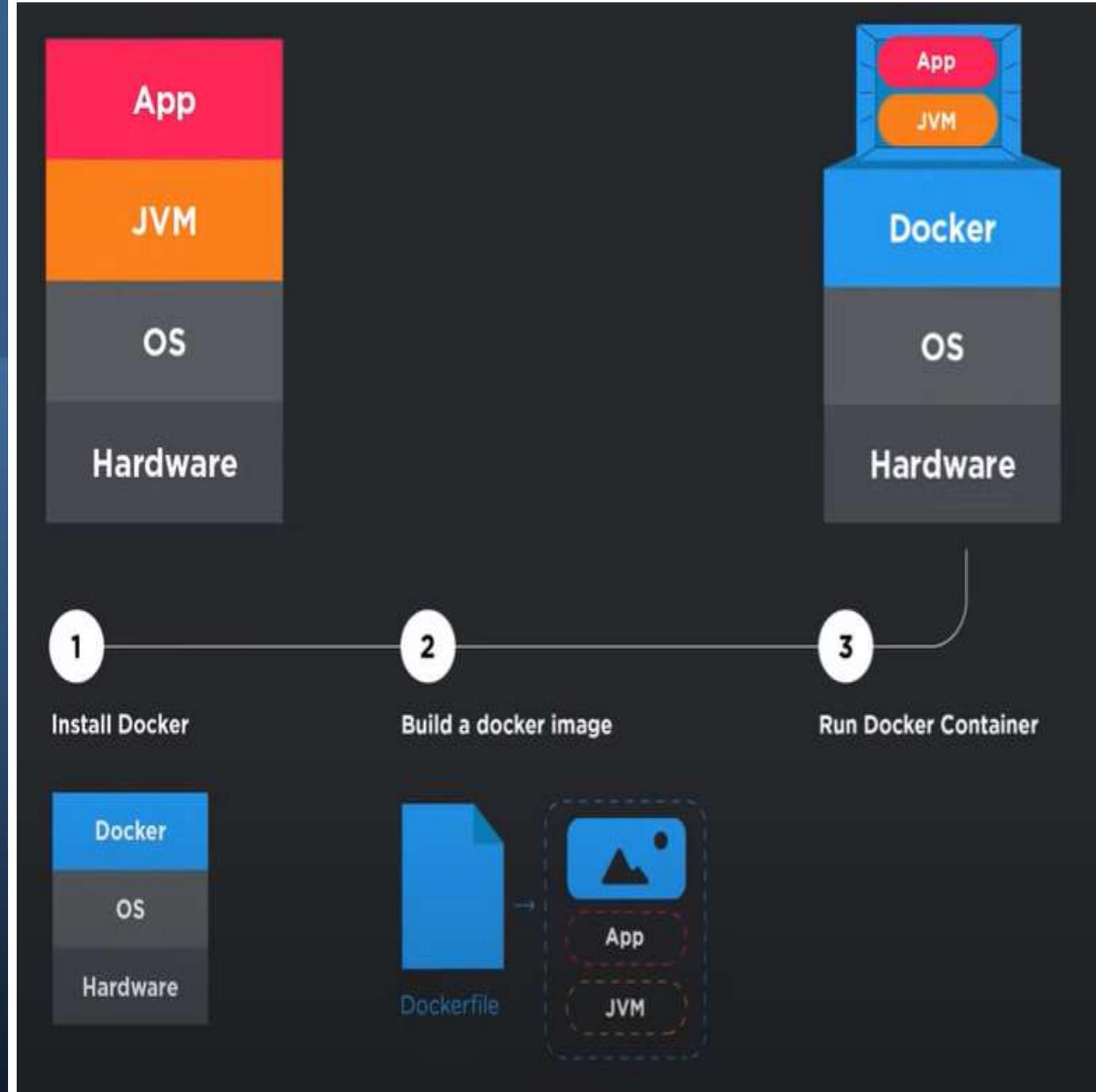


Containerized app – scales-out better

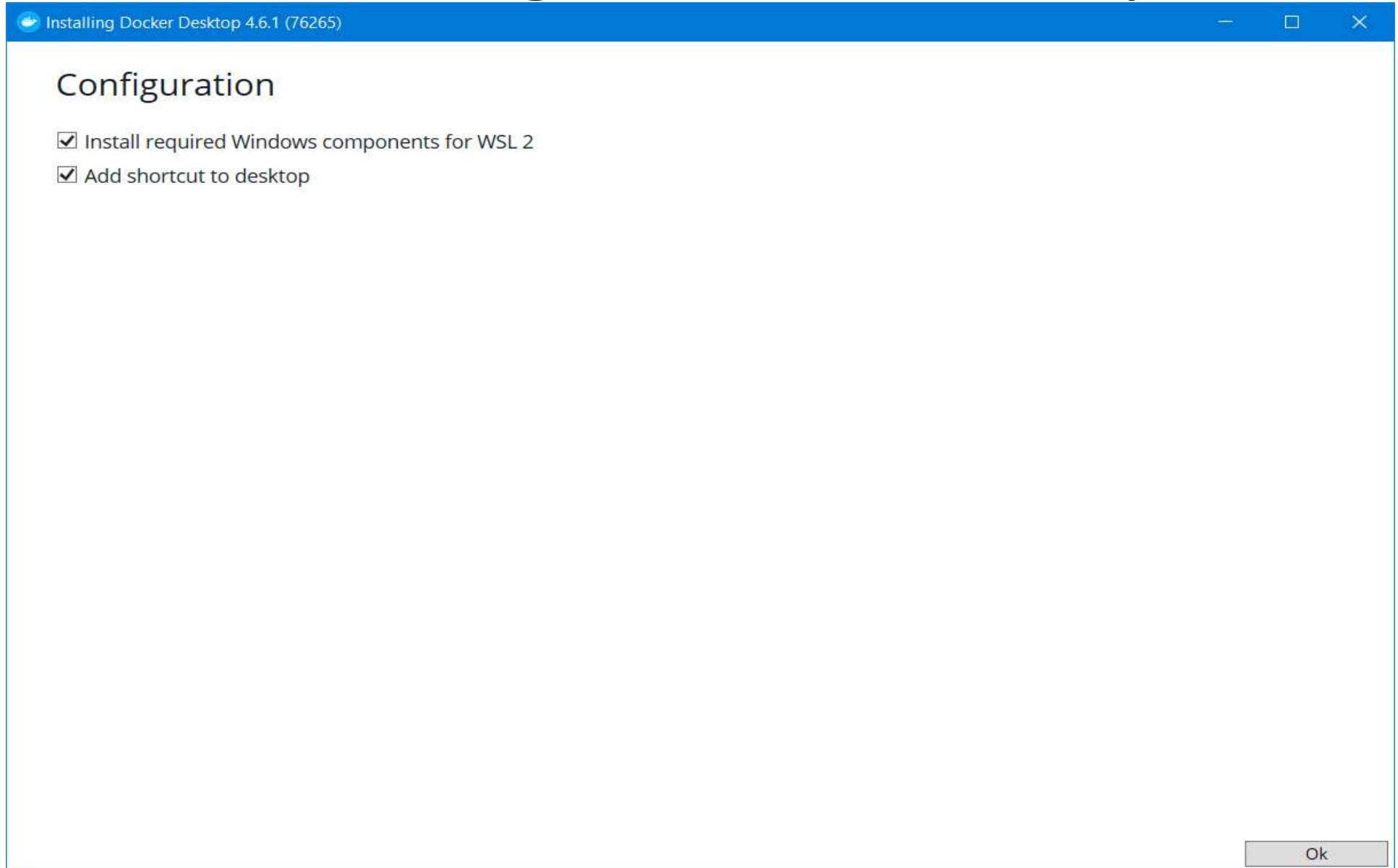


How to containerize software using Docker

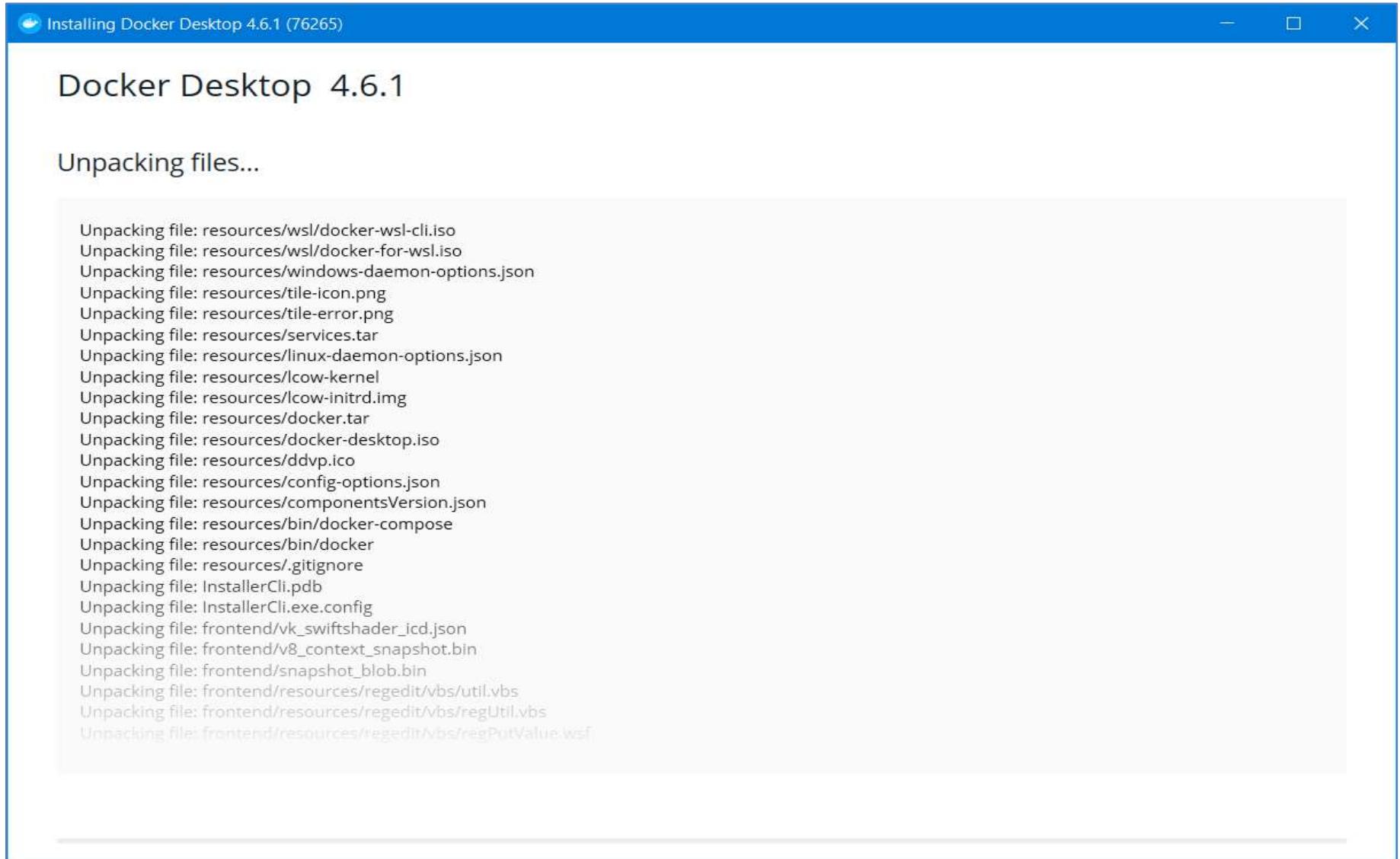
(<https://www.docker.com/get-started/>)



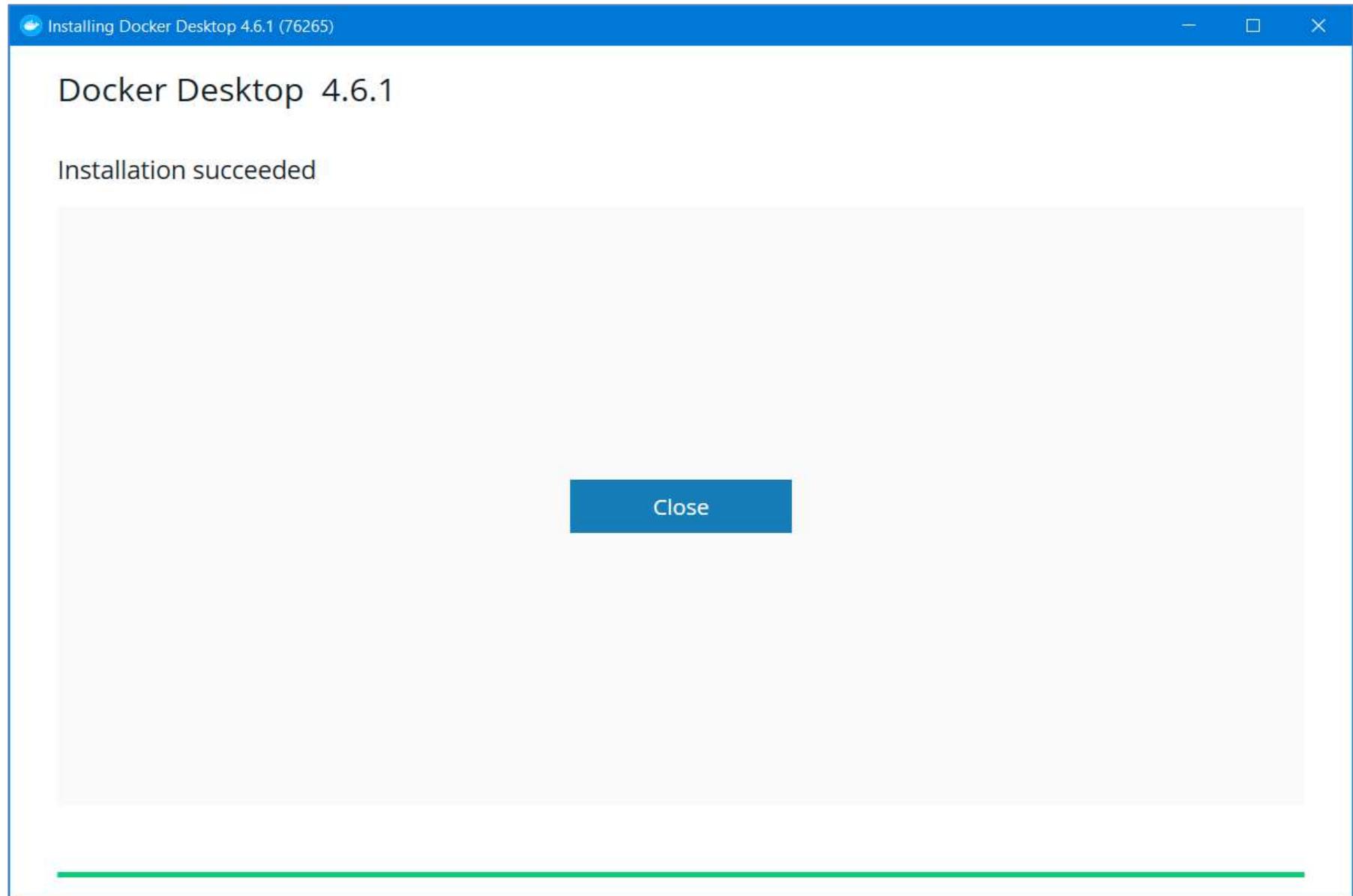
Installing Docker desktop



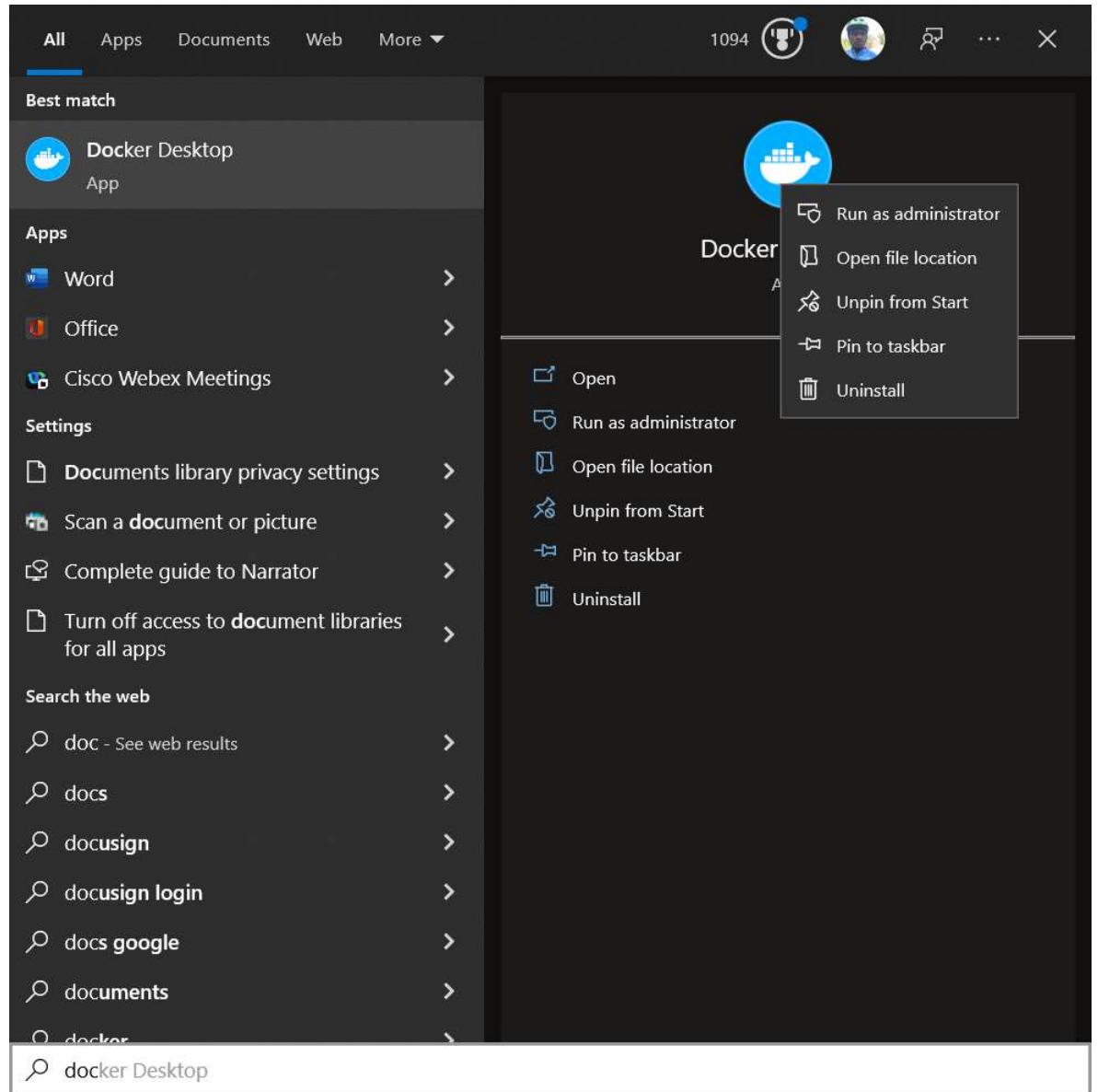
Installing Docker desktop



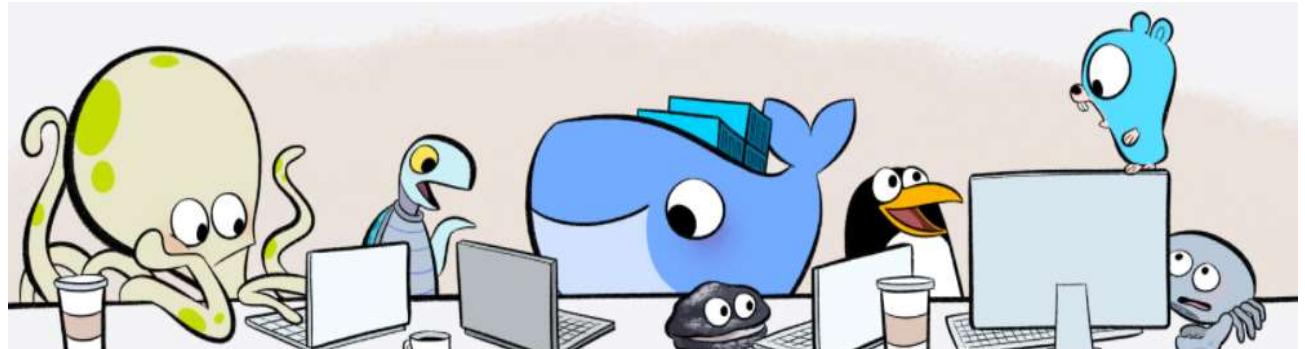
Docker desktop Installation complete



Start Docker Desktop engine



Accept the SLA



Our Service Agreement has Changed

We've updated the [Docker Subscription Service Agreement](#). Please read the [Blog](#) and [FAQs](#) to learn how companies using Docker Desktop may be affected. By checking "I accept the terms" you agree to the [Subscription Service Agreement](#), the [Data Processing Agreement](#), and the [Data Privacy Policy](#).

Here's a summary of key changes:

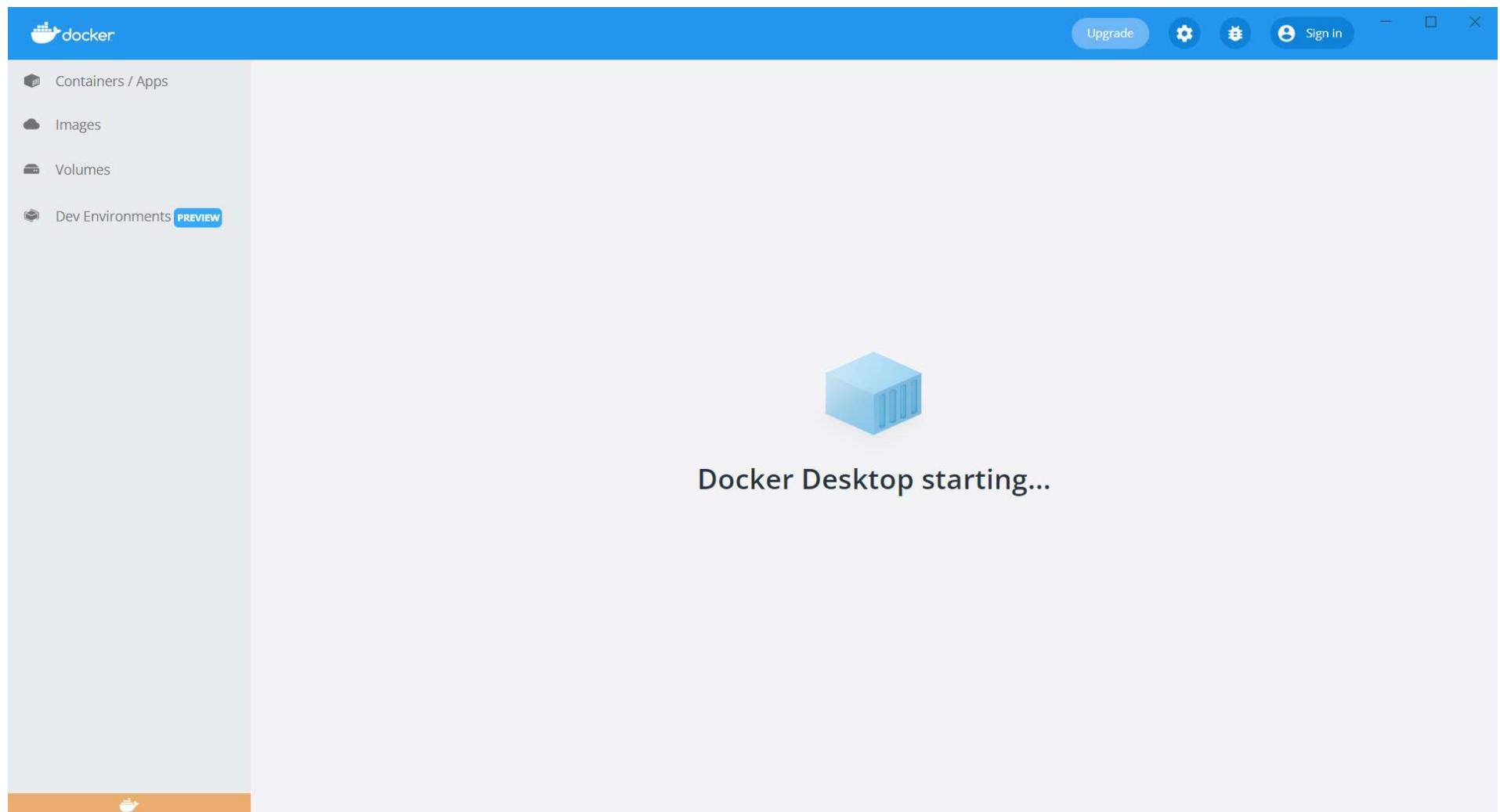
- Our Docker Subscription Service Agreement include a change to the terms of use for Docker Desktop.
 - It **remains free** for small businesses (fewer than 250 employees AND less than \$10 million in annual revenue), personal use, education, and non-commercial open source projects.
 - It requires a paid subscription for professional use in larger enterprises.
- The effective date of these terms is August 31, 2021. There was a **grace period** until January 31, 2022 for those that require a paid subscription to use Docker Desktop. Docker trusts our customers to be in compliance and Docker Desktop will continue to function normally after January 31st, but this is a

I accept the terms

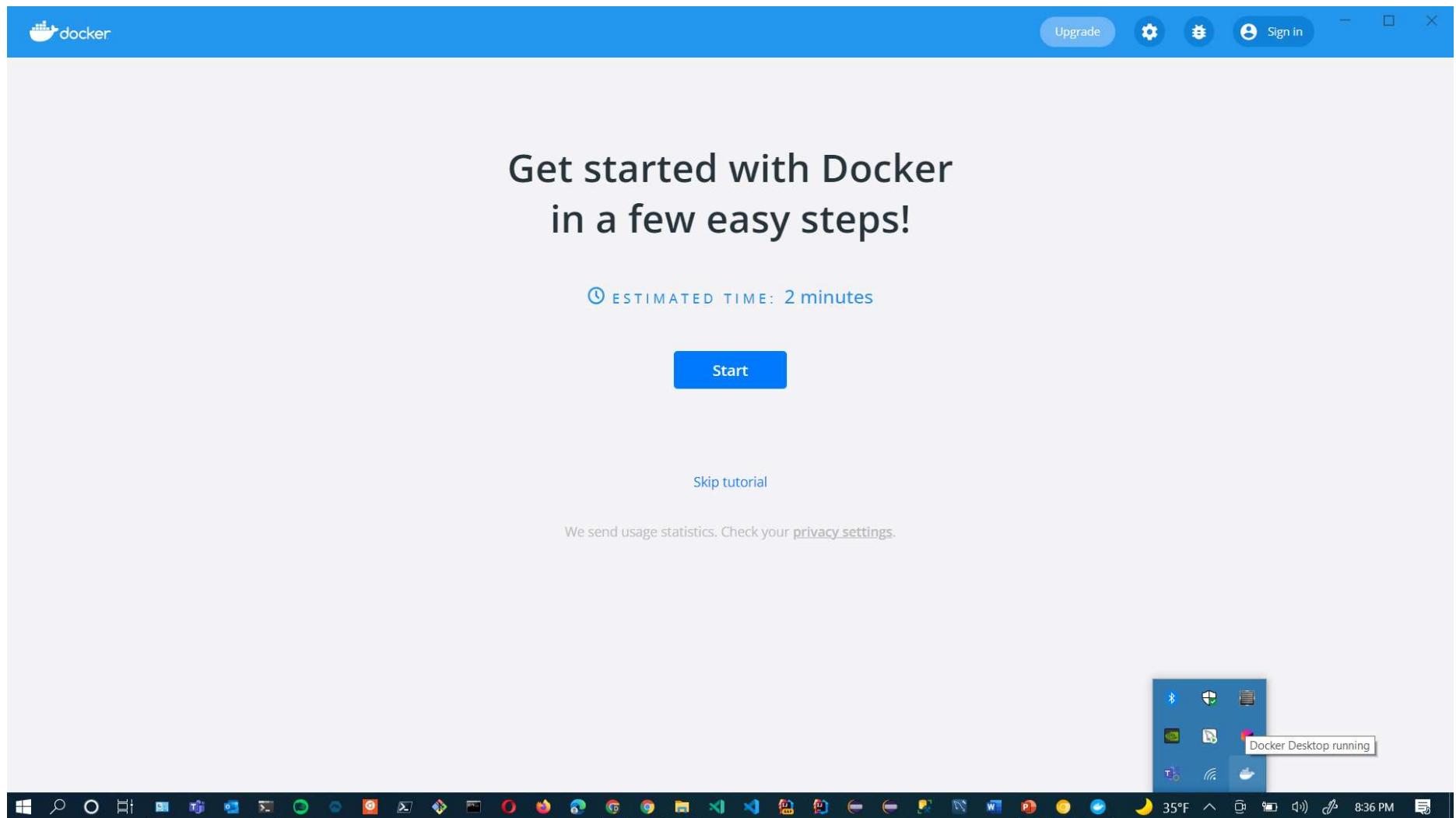
[View Full Terms](#) 

[Decline and Close Application](#) [Accept](#)

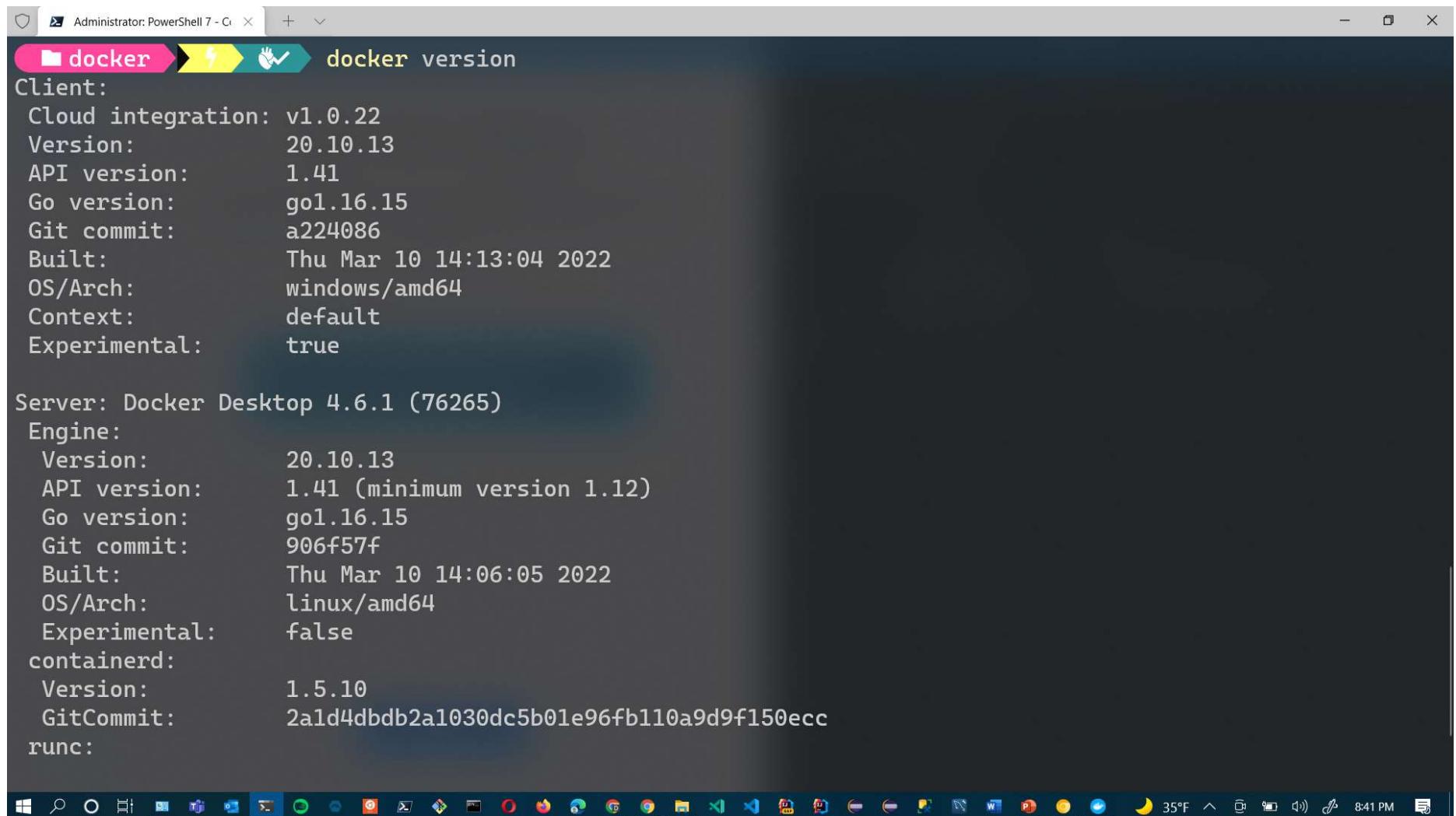
Starting...



Docker Desktop running



Verify running & version



```
Administrator: PowerShell 7 - C:\Users\... docker version
Client:
Cloud integration: v1.0.22
Version:          20.10.13
API version:      1.41
Go version:       go1.16.15
Git commit:       a224086
Built:            Thu Mar 10 14:13:04 2022
OS/Arch:          windows/amd64
Context:          default
Experimental:    true

Server: Docker Desktop 4.6.1 (76265)
Engine:
Version:          20.10.13
API version:      1.41 (minimum version 1.12)
Go version:       go1.16.15
Git commit:       906f57f
Built:            Thu Mar 10 14:06:05 2022
OS/Arch:          linux/amd64
Experimental:    false
containerd:
Version:          1.5.10
GitCommit:        2a1d4dbdb2a1030dc5b01e96fb110a9d9f150ecc
runc:
```

Practice with the ‘GettingStarted’

The screenshot shows a split-screen interface. On the left, a web-based Docker tutorial for 'Getting Started' is displayed. It features a sidebar with numbered steps: 1. Clone (highlighted), 2. Build, 3. Run, 4. Share. The main content area has a title 'First, clone a repository'. It explains that the Getting Started project is a GitHub repository containing everything needed to build and run a Docker image. Below this, a note says 'Clone the repository by running Git in a container.' A code block shows the command: `docker run --name repo alpine/git clone \ https://github.com/docker/getting-started.git >> docker cp repo:/git/getting-started/ .`. At the bottom of the tutorial window is a 'Next Step' button.

On the right, a Windows PowerShell window is open, showing the command-line process for cloning the repository. The terminal output is as follows:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\obi> d:
PS D:> cd D:\oak\MyLearning\docker\
PS D:\oak\MyLearning\docker> ls

Directory: D:\oak\MyLearning\docker

Mode                LastWriteTime         Length Name
----                <-----              ----- 
d----

The taskbar at the bottom of the screen shows various application icons, including a search bar, file explorer, and browser.


```

Getting Started

The screenshot shows a split-screen interface. On the left, a web-based Docker tutorial for 'Getting Started' is displayed. It features a sidebar with numbered steps: 1. Clone (highlighted), 2. Build, 3. Run, 4. Share. The main content area has a title 'First, clone a repository'. Below it, text explains that the project is a GitHub repository containing everything needed to build and run a container. A command-line snippet shows how to clone the repository using Docker:

```
docker run --name repo alpine/git clone \
https://github.com/docker/getting-started.git >>
docker cp repo:/git/getting-started/ .
```

Below this, a note says you can type the command directly in a command line interface. On the right, a Windows PowerShell window shows the execution of the command. It starts with a directory listing of 'D:\oak\MyLearning\docker' showing three folders: 'getting-started', 'helloworld-express-nodejs', and 'nodejs-docker-webstorm'. Then, it runs the Docker command to clone the repository, which fails because the 'alpine/git' image is not found locally. It then pulls the image from Docker Hub, showing the progress and final digest.

```
PS C:\Users\obi> d:
PS D:\> cd D:\oak\MyLearning\docker\
PS D:\oak\MyLearning\docker> ls

Directory: D:\oak\MyLearning\docker

Mode                LastWriteTime         Length Name
----                <-----              ----- 
d----

Next Step



Skip tutorial


```

docker

Upgrade

1 Clone

2 Build

3 Run

4 Share

First, clone a repository

The Getting Started project is a simple GitHub repository which contains everything you need to build an image and run it as a container.

Clone the repository by running Git in a container.

```
docker run --name repo alpine/git clone \
https://github.com/docker/getting-started.git  >>
docker cp repo:/git/getting-started/ .
```

You can also type the command directly in a command line interface.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\obi> d:
PS D:> cd D:\oak\MyLearning\docker\
PS D:\oak\MyLearning\docker> ls

Directory: D:\oak\MyLearning\docker

Mode                LastWriteTime         Length Name
----                -----        ----- 
d-----          7/3/2020   6:29 PM            getting-started
d-----         9/17/2019   3:40 PM           helloworld-express-nodejs
d-----         9/17/2019   3:40 PM           nodejs-docker-webstorm

PS D:\oak\MyLearning\docker> docker run --name repo alpine/git clone https://github.com/docker/getting-started.git
Unable to find image 'alpine/git:latest' locally
latest: Pulling from alpine/git
3d2430473443: Pull complete
3cede86c7d99: Pull complete
5d60e16cf3af: Pull complete
eed811f041bf: Pull complete
Digest: sha256:1283cf559e7fa83951f25b292394dc7bac783e12e2c0353ddda8e3c51583d10f
Status: Downloaded newer image for alpine/git:latest
Cloning into 'getting-started'...
PS D:\oak\MyLearning\docker> docker images --all
REPOSITORY      TAG      IMAGE ID      CREATED       SIZE
alpine/git      latest   b337a04161f7   6 days ago   38.2MB
PS D:\oak\MyLearning\docker> docker ps --all
CONTAINER ID    IMAGE      COMMAND      CREATED      STATUS      PORTS      NAMES
463f72e32d07   alpine/git   "git clone https://g..."   5 minutes ago   Exited (0) 5 minutes ago
PS D:\oak\MyLearning\docker>
```

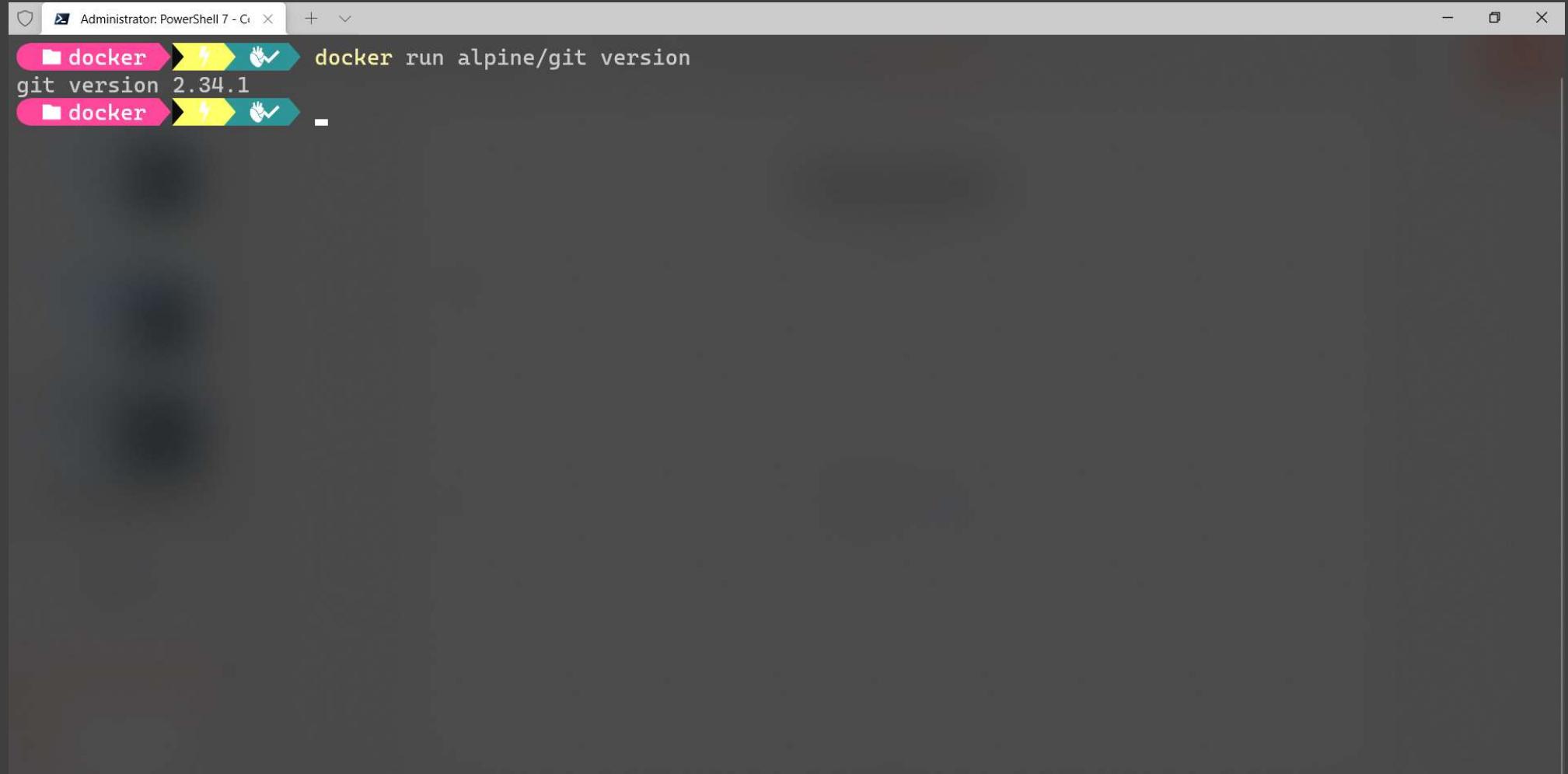
Next Step

Practicing with the 'Getting Started'

- The cmd: > docker run --name repo alpine/git ... fetches a docker image containing 'git on alpine linux' and it makes a container from this image, and runs/executes the git command in it, to clone a repository from github using the url
- > docker images –all : lists all images
- > docker ps –all : List all containers

Exercise - Try this:

- Execute a docker command to display what version of git is available on the container named, repo, which was created from the docker image named, ‘alpine/git’



A screenshot of a Windows PowerShell window titled "Administrator: PowerShell 7 - C:\Users\...". The window shows two terminal sessions. The top session has a pink header and shows the command "docker run alpine/git version" followed by the output "git version 2.34.1". The bottom session has a pink header and shows a single dash character "-". The background of the slide is dark grey.

Solution

- > 'docker run' first creates a container, then starts/runs it, passing it the specified command

Demo

- Create a simple Java CLI Application using Apache Maven quickstart archetype
- Add config in maven-jar-plugin to specify the mainClass



```
51 <plugin>
52   <artifactId>maven-jar-plugin</artifactId>
53   <version>3.0.2</version>
54   <configuration>
55     <archive>
56       <manifest>
57         <mainClass>edu.miu.cs.cs425.App</mainClass>
58       </manifest>
59     </archive>
60   </configuration>
61 </plugin>
```

The screenshot shows a code editor with a syntax-highlighted XML configuration for a Maven plugin. The code defines a single plugin configuration for the 'maven-jar-plugin'. It specifies the artifact ID as 'maven-jar-plugin' and the version as '3.0.2'. Inside the configuration, there is an 'archive' section which contains a 'manifest' section. Within the manifest section, the 'mainClass' is set to 'edu.miu.cs.cs425.App'. The code editor has vertical and horizontal scroll bars, and the status bar at the bottom right shows the page number '34'.

Demo

- Package the app by executing the maven command: > mvn clean package
- Create a Dockerfile:

Demo

The screenshot shows the IntelliJ IDEA interface with a Java project named "helloworlddocker". The project structure on the left includes ".idea", "src" (with "main" and "java" subfolders), "test", "target" (containing "classes", "generated-sources", "generated-test-sources", "maven-archiver", "maven-status", "surefire-reports", "test-classes", and "helloworlddocker-1.0-SNAPSHOT.jar"), and "Dockerfile" and "pom.xml". The "Dockerfile" tab is selected in the top navigation bar, displaying the following Dockerfile content:

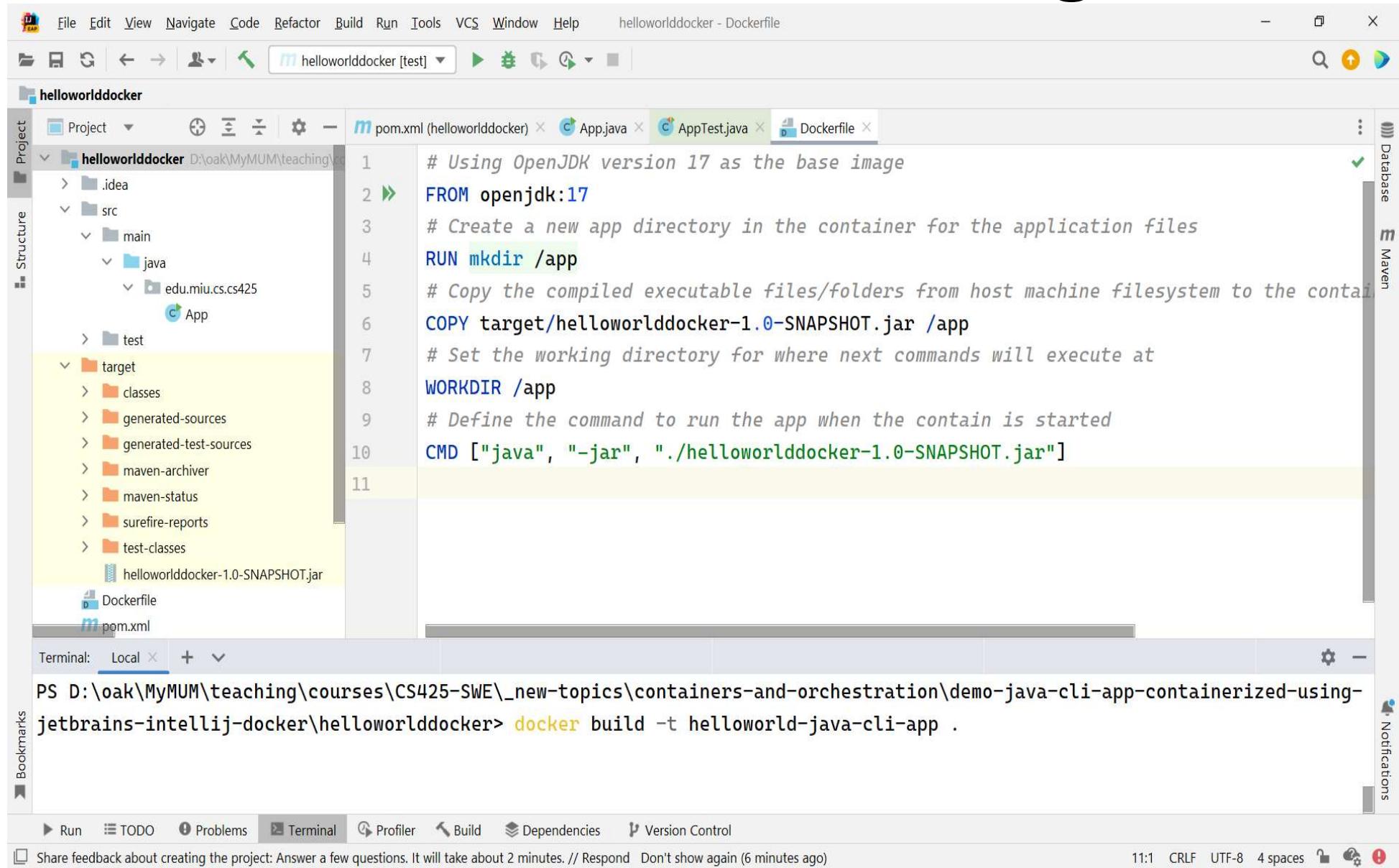
```
# Using OpenJDK version 17 as the base image
FROM openjdk:17
# Create a new app directory in the container for the application files
RUN mkdir /app
# Copy the compiled executable files/folders from host machine filesystem to the container
COPY target/helloworlddocker-1.0-SNAPSHOT.jar /app
# Set the working directory for where next commands will execute at
WORKDIR /app
# Define the command to run the app when the container is started
CMD ["java", "-jar", "./helloworlddocker-1.0-SNAPSHOT.jar"]
```

The terminal at the bottom shows the command "docker scan" being run against the image.

```
Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
PS D:\oak\MyMUM\teaching\courses\CS425-SWE\_new-topics\containers-and-orchestration\demo-java-cli-app-containerized-using-jetbrains-intellij-docker\helloworlddocker> docker scan
```

IntelliJ IDEA status bar: 11:1 CRLF UTF-8 4 spaces

Demo – build the image



The screenshot shows the IntelliJ IDEA interface with a Java project named "helloworlddocker". The project structure on the left includes ".idea", "src" (with "main" and "java" subfolders containing "edu.mi.u.cs.cs425" and "App.java"), "test", and "target" (containing "classes", "generated-sources", "generated-test-sources", "maven-archiver", "maven-status", "surefire-reports", "test-classes", and "helloworlddocker-1.0-SNAPSHOT.jar"). The "Dockerfile" tab is selected in the top navigation bar, showing the following Dockerfile content:

```
# Using OpenJDK version 17 as the base image
FROM openjdk:17
# Create a new app directory in the container for the application files
RUN mkdir /app
# Copy the compiled executable files/folders from host machine filesystem to the container
COPY target/helloworlddocker-1.0-SNAPSHOT.jar /app
# Set the working directory for where next commands will execute at
WORKDIR /app
# Define the command to run the app when the container is started
CMD ["java", "-jar", "./helloworlddocker-1.0-SNAPSHOT.jar"]
```

The terminal at the bottom shows the command being run:

```
PS D:\oak\MyMUM\teaching\courses\CS425-SWE\_new-topics\containers-and-orchestration\demo-java-cli-app-containerized-using-jetbrains-intellij-docker\helloworlddocker> docker build -t helloworld-java-cli-app .
```

Run the container

```
Administrator: PowerShell 7 - C:\ > docker images --all
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
alpine/git      latest   b337a04161f7  7 days ago   38.2MB

Administrator: PowerShell 7 - C:\ > docker ps --all
CONTAINER ID      IMAGE      COMMAND      CREATED      STATUS      PORTS      NAMES
463f72e32d07    alpine/git  "git clone https://g..."  4 hours ago  Exited (128)  3 hours ago  repo

Administrator: PowerShell 7 - C:\ > docker ps --all
CONTAINER ID      IMAGE      COMMAND      CREATED      STATUS      PORTS      NAMES
463f72e32d07    alpine/git  "git clone https://g..."  4 hours ago  Exited (128)  3 hours ago  repo

Administrator: PowerShell 7 - C:\ > docker images --all
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
helloworld-java-cli-app  latest   ef5052b490c5  15 minutes ago  471MB
alpine/git      latest   b337a04161f7  7 days ago   38.2MB

Administrator: PowerShell 7 - C:\ > docker run helloworld-java-cli-app:latest
Hello World Docker!
Administrator: PowerShell 7 - C:\ >
```

Simulate a running server-like app

The screenshot shows the IntelliJ IDEA interface with a Java project named "helloworlddocker". The code editor displays the `App.java` file, which contains the following code:

```
public class App {  
    public static void main( String[] args ) {  
        System.out.println( "Hello World Docker!" );  
        int count = 0;  
        try {  
            while(true) {  
                Thread.sleep( millis: 2000 );  
                System.out.printf("App is still running. Iteration #: %d\n", ++count);  
            }  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
            throw new RuntimeException(e);  
        }  
    }  
}
```

The terminal window at the bottom shows the command being run:

```
PS D:\oak\MyMUM\teaching\courses\CS425-SWE\_new-topics\containers-and-orchestration\demo-java-cli-app-containerized-using-jetbrains-intellij-docker\helloworlddocker> java -jar .\target\helloworlddocker-1.0-SNAPSHOT.jar
```

Build another docker image

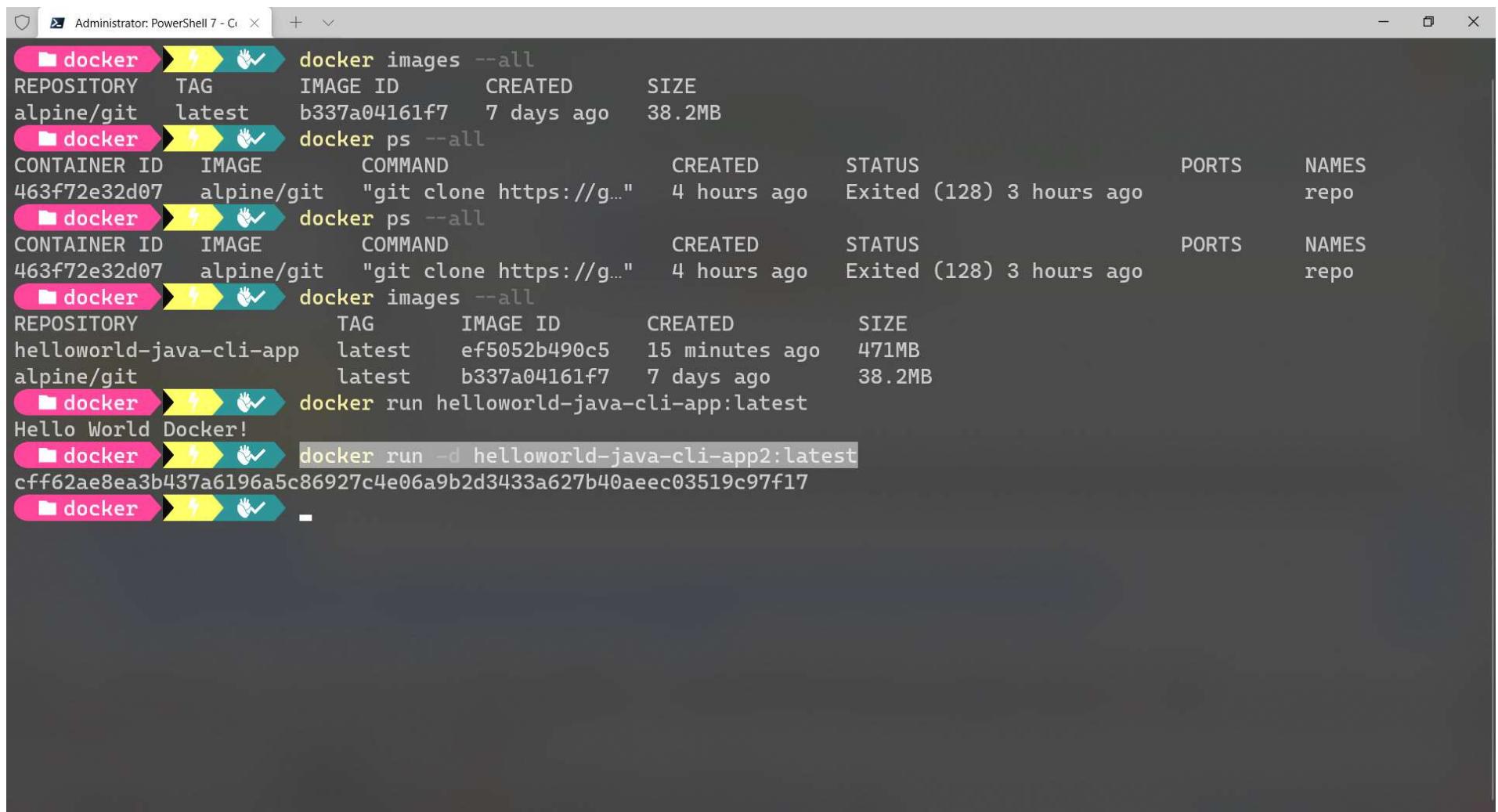
The screenshot shows the IntelliJ IDEA interface with a Java project named "helloworlddocker". The code editor displays the following Java code:

```
public class App {  
    public static void main( String[] args ) {  
        System.out.println( "Hello World Docker!" );  
        int count = 0;  
        try {  
            while(true) {  
                Thread.sleep( millis: 2000 );  
                System.out.printf("App is still running. Iteration #: %d\n", ++count);  
            }  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
            throw new RuntimeException(e);  
        }  
    }  
}
```

The IntelliJ interface includes a Project tool window, a Structure tool window, and a Terminal window at the bottom. The Terminal window shows the command being run:

```
PS D:\oak\MyMUM\teaching\courses\CS425-SWE\_new-topics\containers-and-orchestration\demo-java-cli-app-containerized-using-jetbrains-intellij-docker\helloworlddocker> docker build -t helloworld-java-cli-app2 .
```

Execute/start/run the container (in detached mode)



```
Administrator: PowerShell 7 - C:\Users\Public\Documents> docker images --all
REPOSITORY TAG IMAGE ID CREATED SIZE
alpine/git latest b337a04161f7 7 days ago 38.2MB

Administrator: PowerShell 7 - C:\Users\Public\Documents> docker ps --all
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
463f72e32d07 alpine/git "git clone https://g..." 4 hours ago Exited (128) 3 hours ago
repo

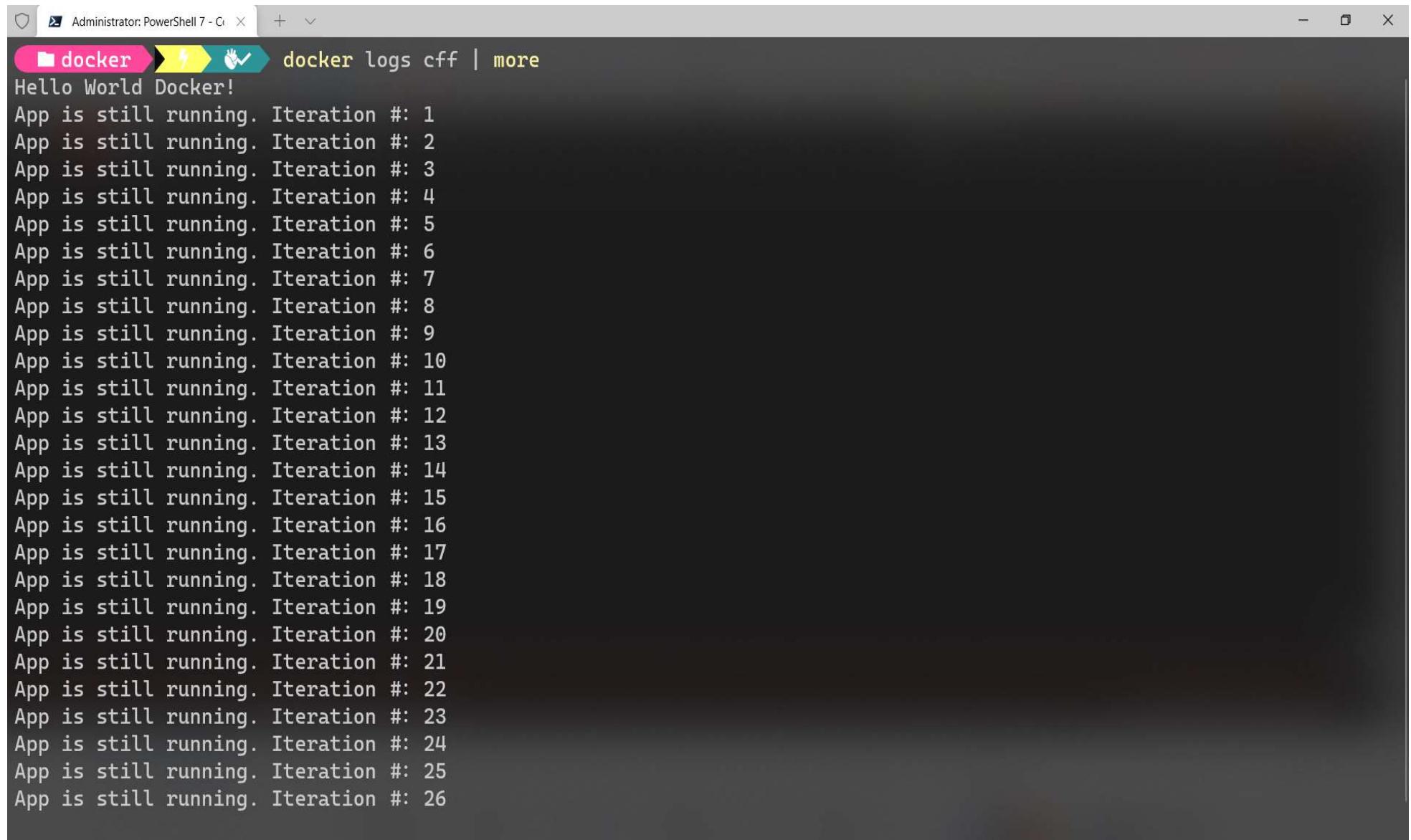
Administrator: PowerShell 7 - C:\Users\Public\Documents> docker ps --all
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
463f72e32d07 alpine/git "git clone https://g..." 4 hours ago Exited (128) 3 hours ago
repo

Administrator: PowerShell 7 - C:\Users\Public\Documents> docker images --all
REPOSITORY TAG IMAGE ID CREATED SIZE
helloworld-java-cli-app latest ef5052b490c5 15 minutes ago 471MB
alpine/git latest b337a04161f7 7 days ago 38.2MB

Administrator: PowerShell 7 - C:\Users\Public\Documents> docker run helloworld-java-cli-app:latest
Hello World Docker!
Administrator: PowerShell 7 - C:\Users\Public\Documents> docker run -d helloworld-java-cli-app2:latest
cff62ae8ea3b437a6196a5c86927c4e06a9b2d3433a627b40aeeec03519c97f17

Administrator: PowerShell 7 - C:\Users\Public\Documents>
```

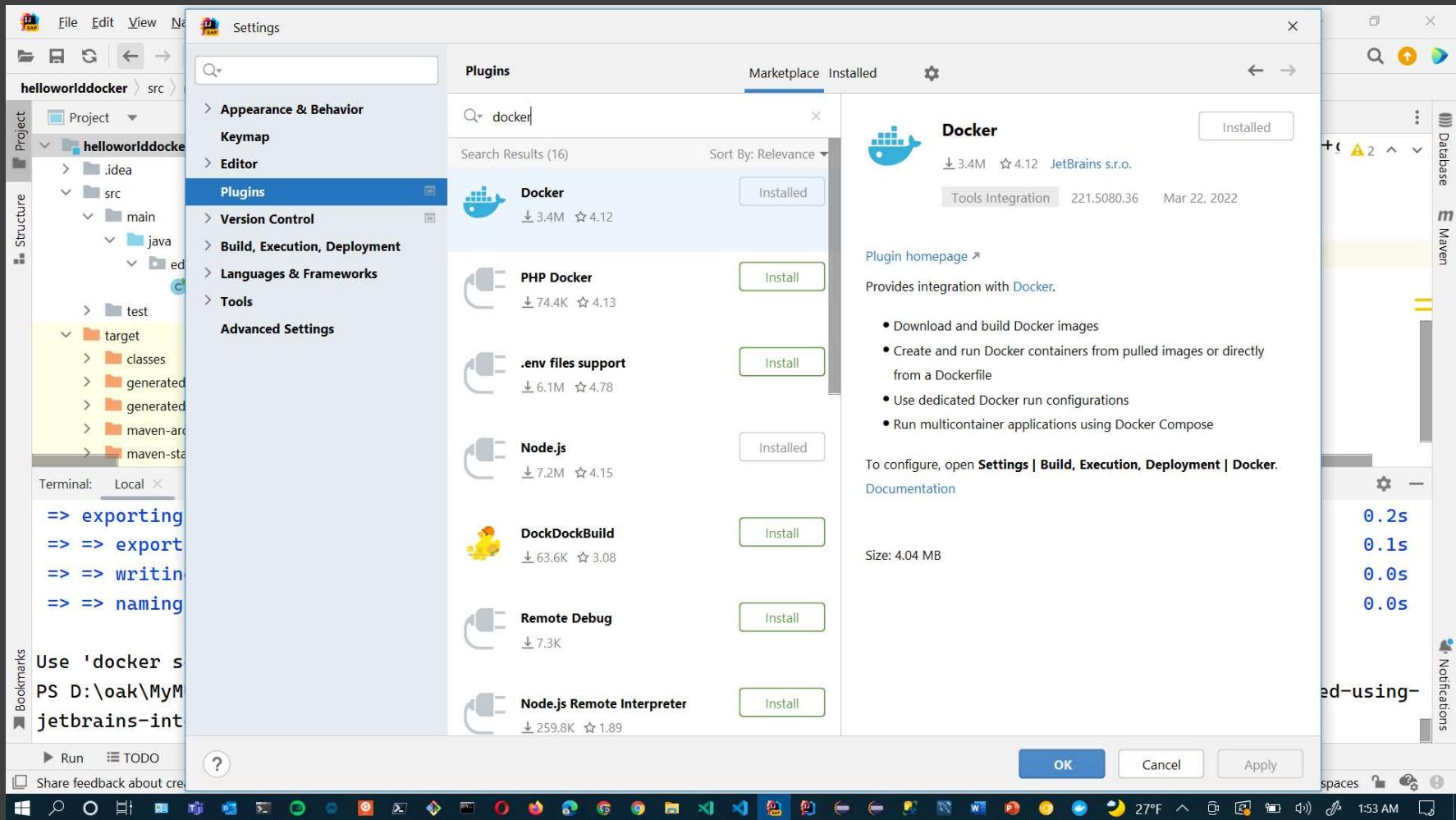
Display the container's log



```
Administrator: PowerShell 7 - C:\Users\... docker logs cff | more
Hello World Docker!
App is still running. Iteration #: 1
App is still running. Iteration #: 2
App is still running. Iteration #: 3
App is still running. Iteration #: 4
App is still running. Iteration #: 5
App is still running. Iteration #: 6
App is still running. Iteration #: 7
App is still running. Iteration #: 8
App is still running. Iteration #: 9
App is still running. Iteration #: 10
App is still running. Iteration #: 11
App is still running. Iteration #: 12
App is still running. Iteration #: 13
App is still running. Iteration #: 14
App is still running. Iteration #: 15
App is still running. Iteration #: 16
App is still running. Iteration #: 17
App is still running. Iteration #: 18
App is still running. Iteration #: 19
App is still running. Iteration #: 20
App is still running. Iteration #: 21
App is still running. Iteration #: 22
App is still running. Iteration #: 23
App is still running. Iteration #: 24
App is still running. Iteration #: 25
App is still running. Iteration #: 26
```

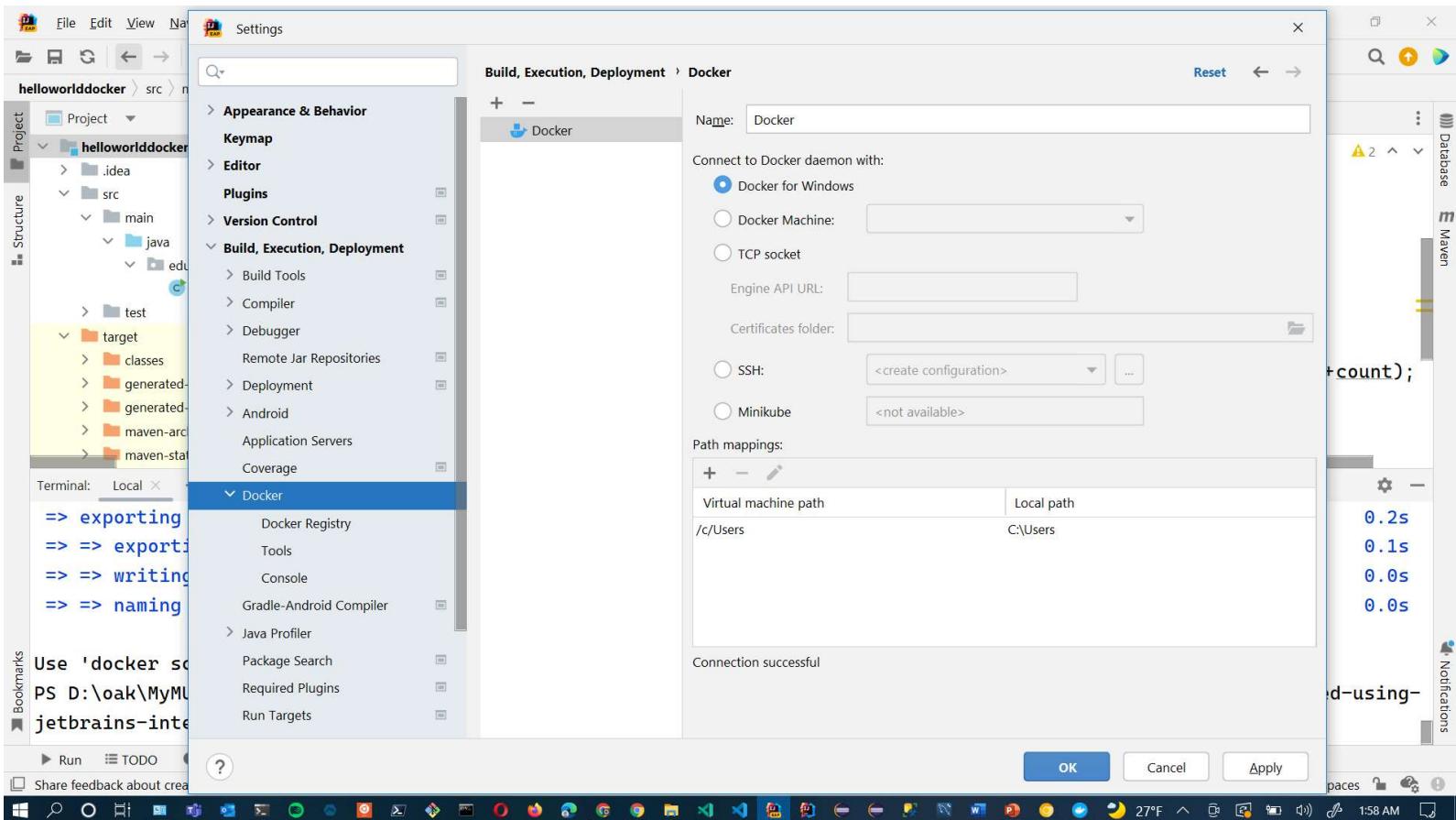
Check the status and stop the running container

```
Administrator: PowerShell 7 - C:\Users\mes... + - X
docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
MES
cff62ae8ea3b helloworld-java-cli-app2:latest "java -jar ./hellowo..." 9 minutes ago Up 9 minutes fe
stive_hellman
docker container stopcff
cff
docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
docker ps --all
CONTAINER ID IMAGE COMMAND CREATED STATUS
PORTS NAMES
cff62ae8ea3b helloworld-java-cli-app2:latest "java -jar ./hellowo..." 10 minutes ago Exited (143) 13 seconds ago
o
festive_hellman
f55eadfbfa8 helloworld-java-cli-app:latest "java -jar ./hellowo..." 41 minutes ago Exited (0) 41 minutes ago
elastic_zhukovsky
463f72e32d07 alpine/git
repo
git clone https://g...
"git clone https://g..." 5 hours ago Exited (128) 4 hours ago
repo
docker -
```



Operating Docker with IntelliJ IDEA

- Jetbrains IntelliJ IDEA Ultimate has the Docker plugin pre-bundled.
- Jetbrains IntelliJ IDEA Community Edition requires you to add the Docker plugin



Configure IntelliJ IDEA for Docker

- Open Settings dialog, go to Build, Exec, Deployment; Select “Docker” and click the + button
- IntelliJ IDEA auto-detects a running Docker for your OS and connects to the docker daemon successfully
- Click “Ok” and Docker appears in the Services tab

The screenshot shows the IntelliJ IDEA interface with the following details:

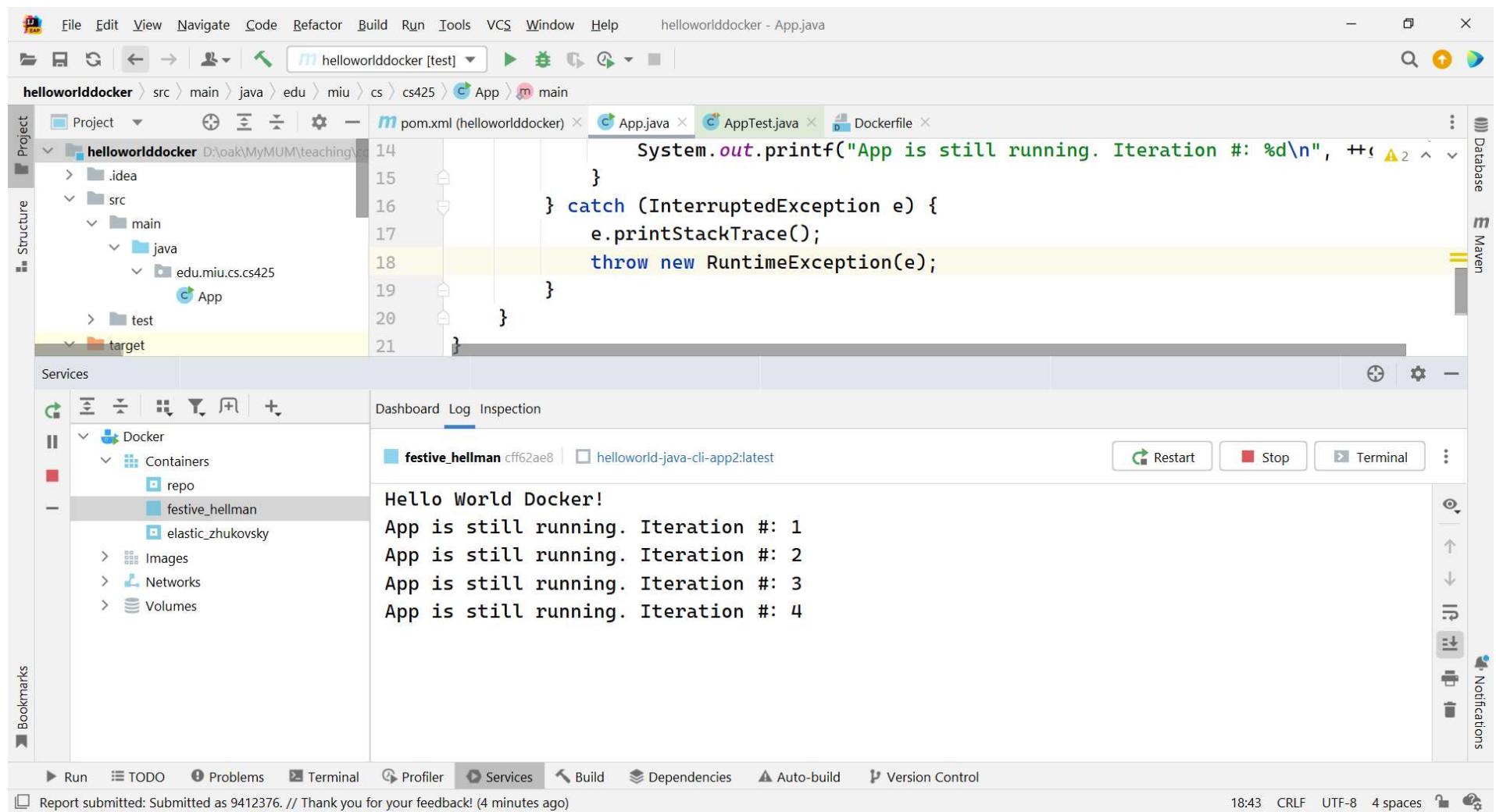
- Project Structure:** The project is named "helloworlddocker". The "src" directory contains "main", "java", and "edu.miu.cs.cs425". "edu.miu.cs.cs425" contains "App.java".
- Code Editor:** The "App.java" file is open, showing the following code:

```
public class App {  
    public static void main( String[] args ) {  
        System.out.println( "Hello World Docker!" );  
        int count = 0;  
        try {  
            while(true) {  
                Thread.sleep( millis: 2000 );  
                System.out.printf("App is still running. Iteration #: %d\n", ++count);  
            }  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
            throw new RuntimeException(e);  
        }  
    }  
}
```
- Services Panel:** The "Docker" service is selected.
- Bottom Bar:** Includes buttons for Run, TODO, Problems, Terminal, Profiler, Services, Build, Dependencies, Auto-build, Version Control, and a feedback message about creating the project.

Operating Docker in IntelliJ

- Select the Docker service and click the Green button to connect to it
- Next, expand the containers node and select a container and start/run it and view the log

Working with Docker in IntelliJ



```
System.out.printf("App is still running. Iteration #: %d\n", ++i);
}
} catch (InterruptedException e) {
    e.printStackTrace();
    throw new RuntimeException(e);
}
```

Docker

Images

- alpine/git:latest
- helloworld-java-cli-app2:latest
- helloworld-java-cli-app:latest

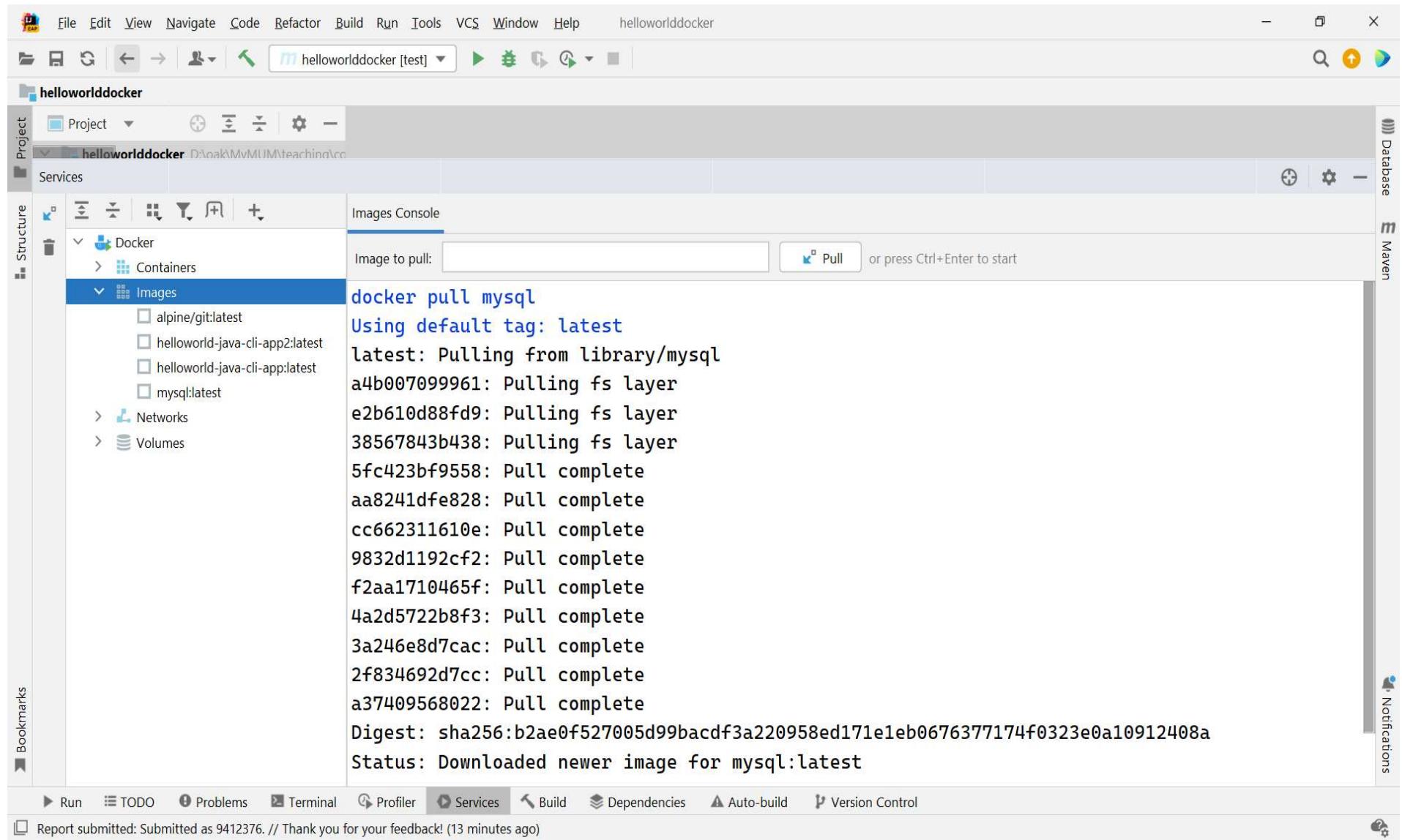
mysql

Press Enter to insert, Tab to replace [Next Tip](#)

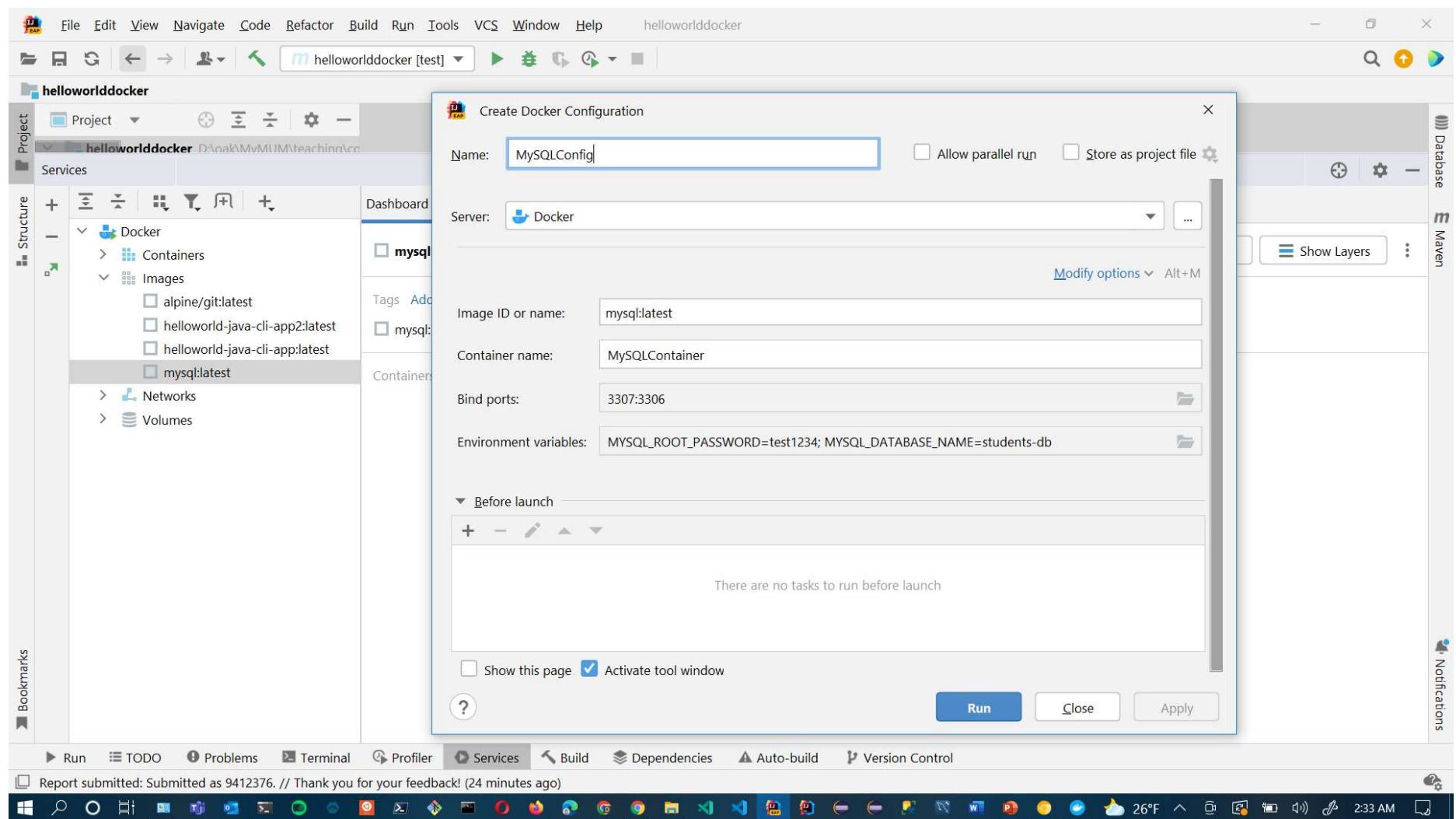
Working with Docker in IntelliJ IDEA

- Select the “Images” node and start typing name of an Image to pull (IntelliJ IDEA offers auto-completion using suggestions from Docker Hub)
- Select “MySQL” database image and click the “Pull” button.

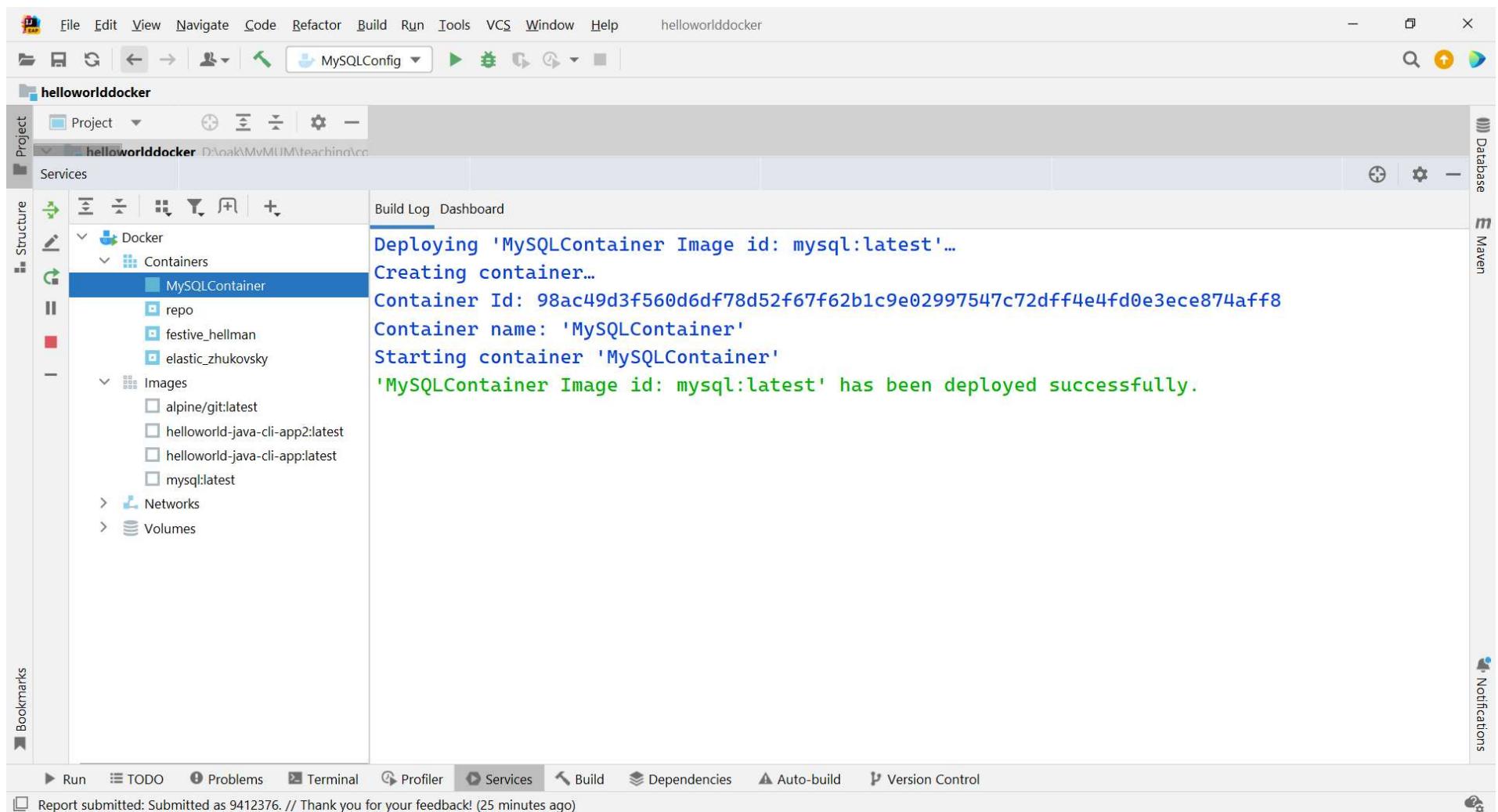
Setting-up a MySQL database container



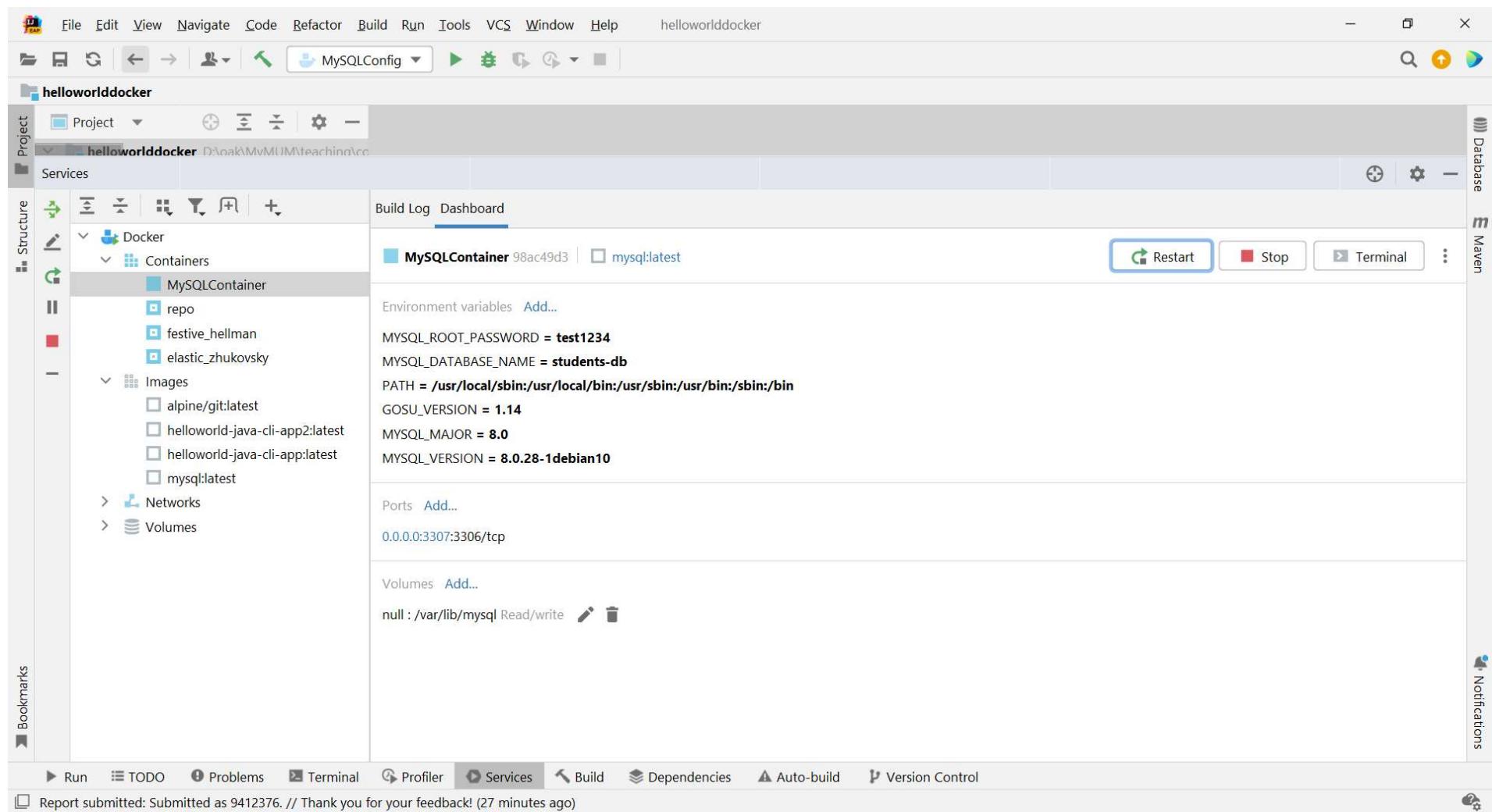
Create a MySQL DB container



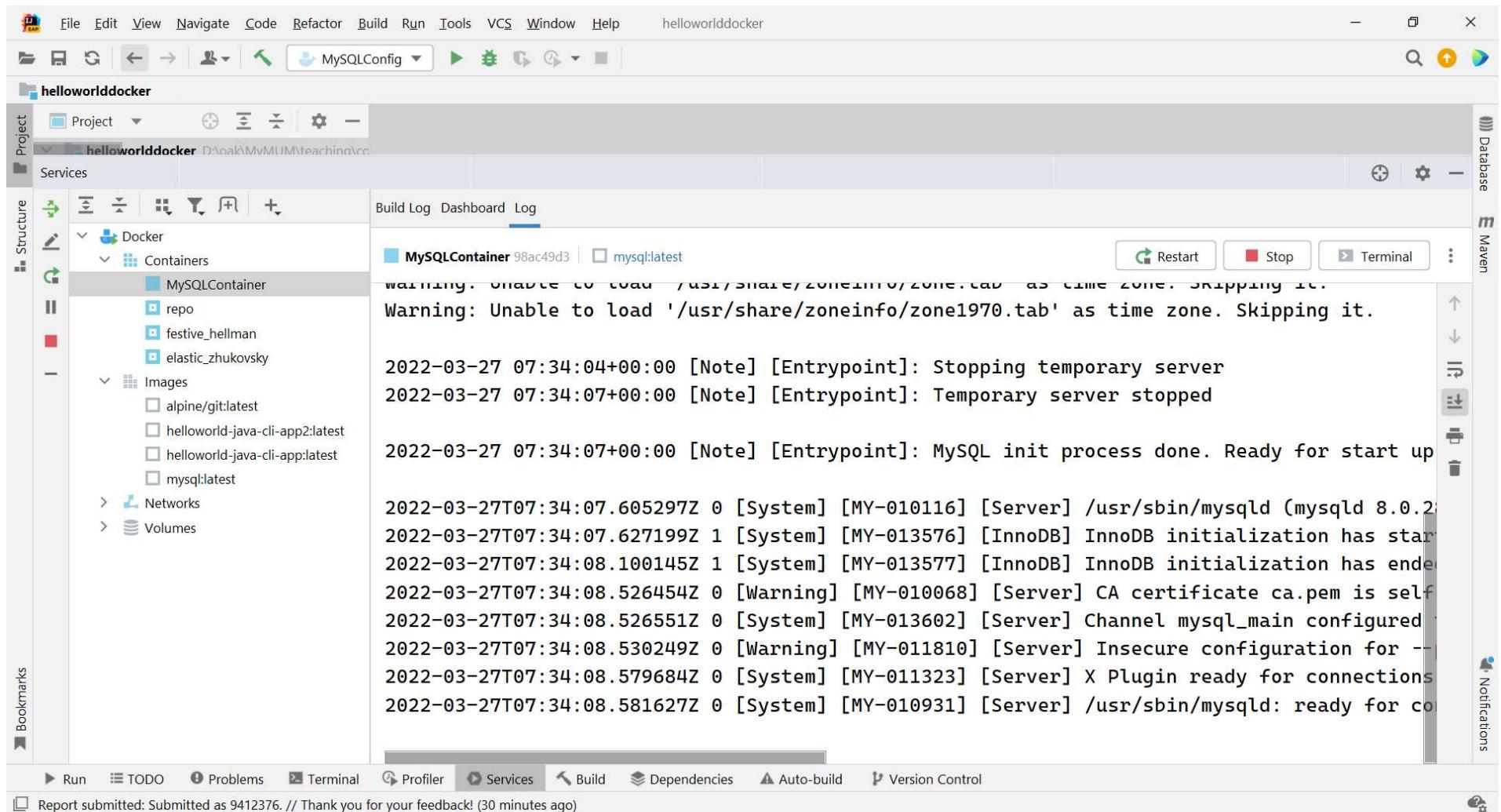
MySQL Database container created



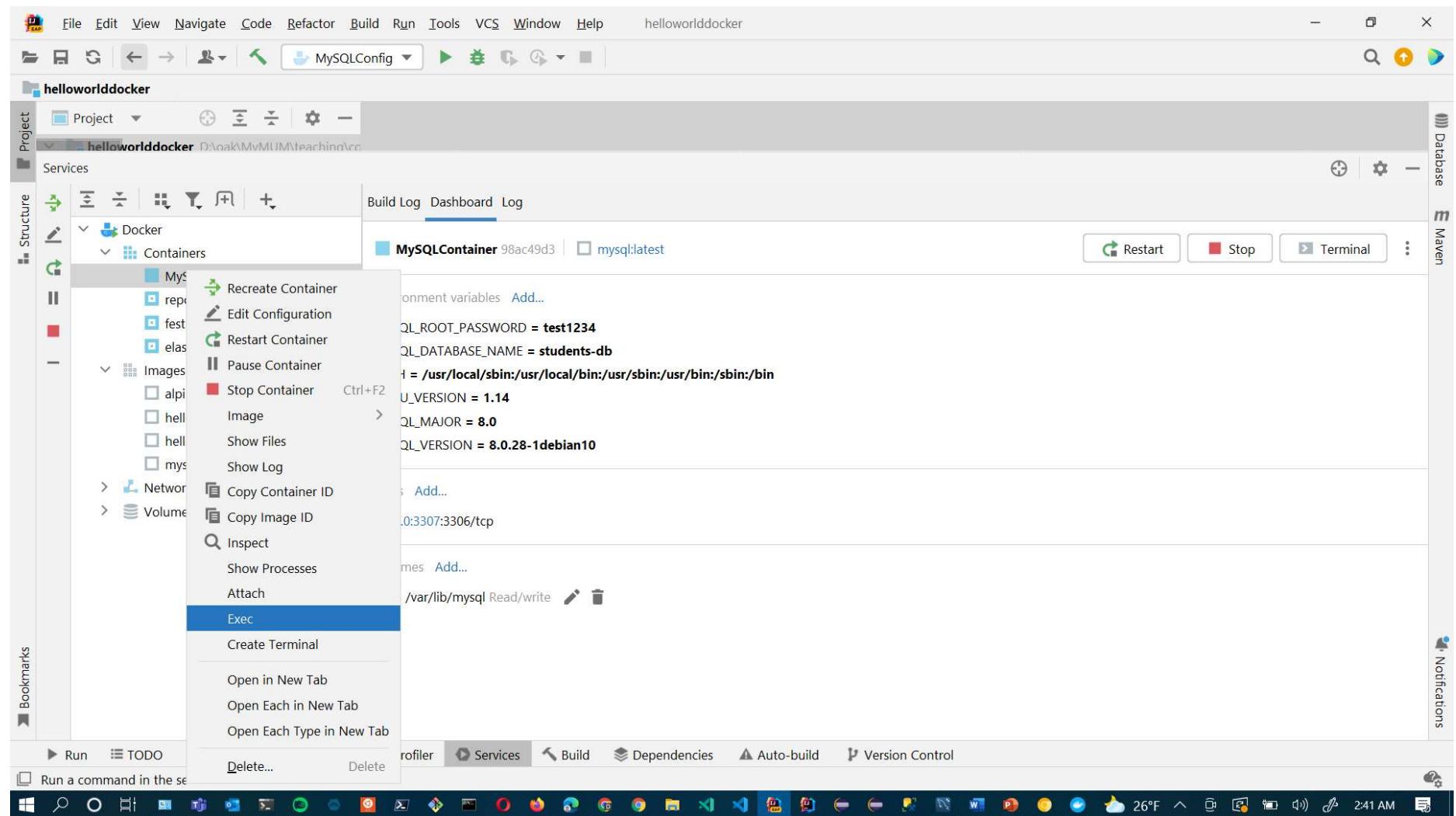
MySQL Database container running



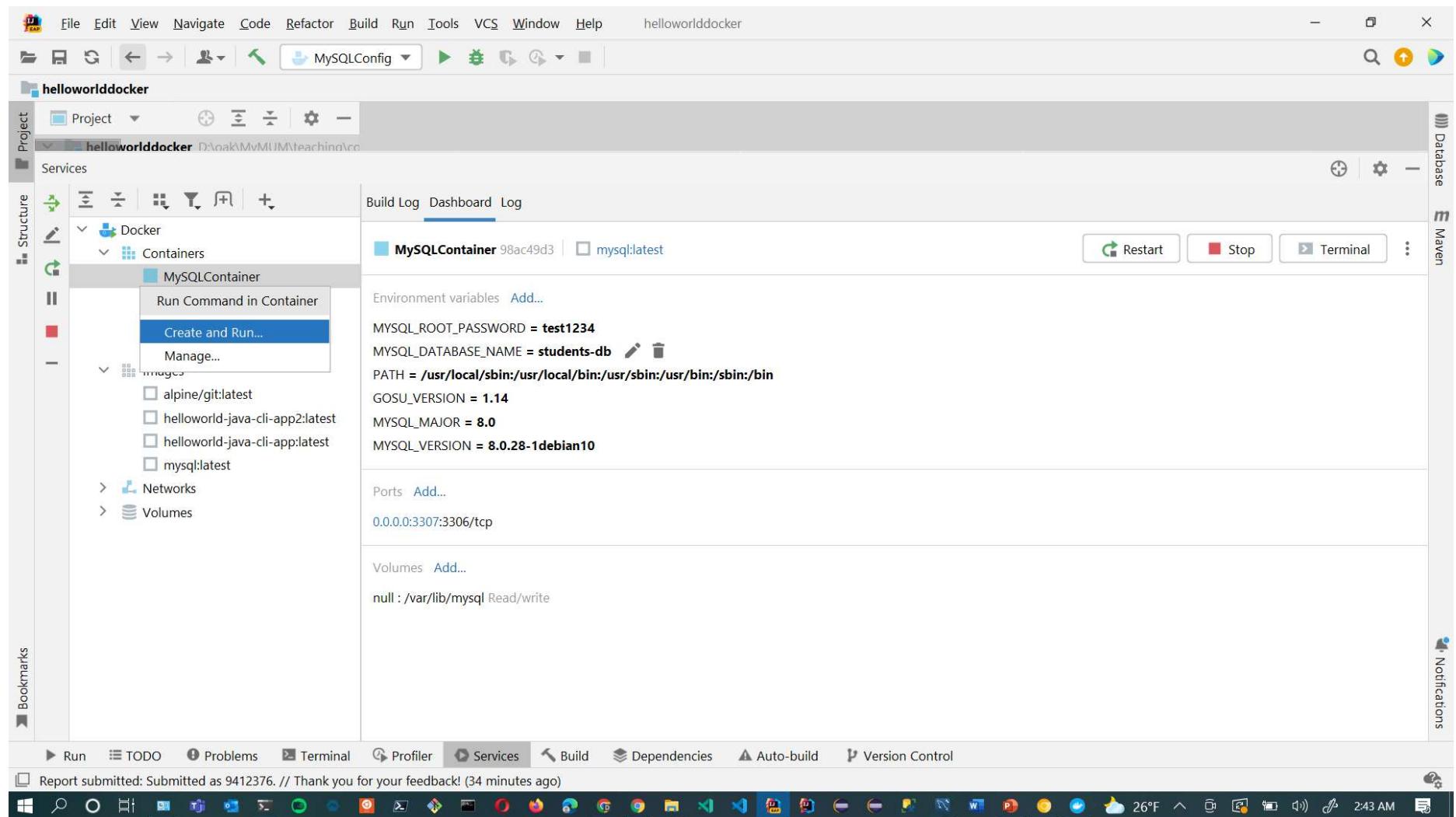
View the log



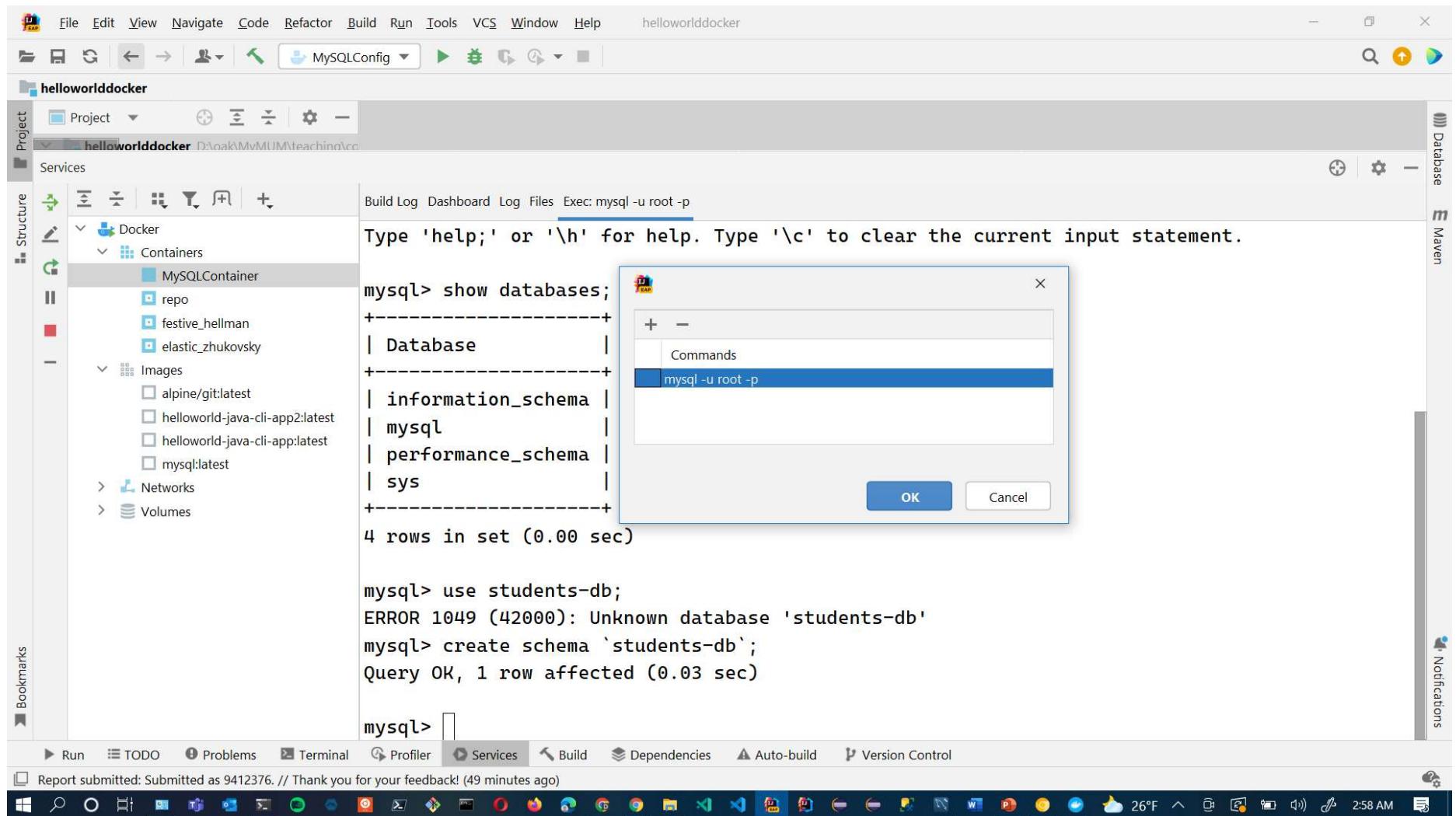
Connect with MySQL shell or client



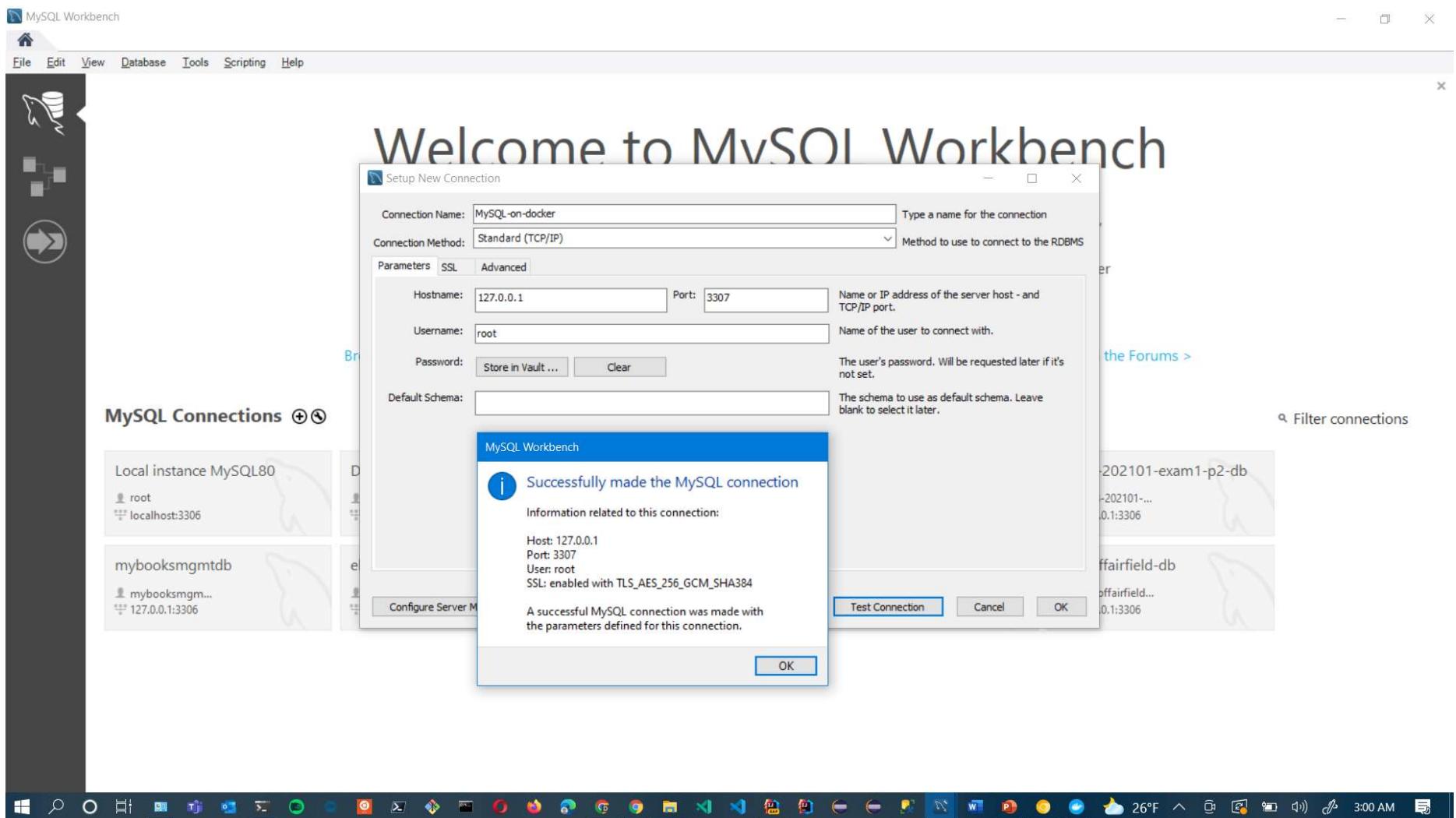
Connect with MySQL shell or client



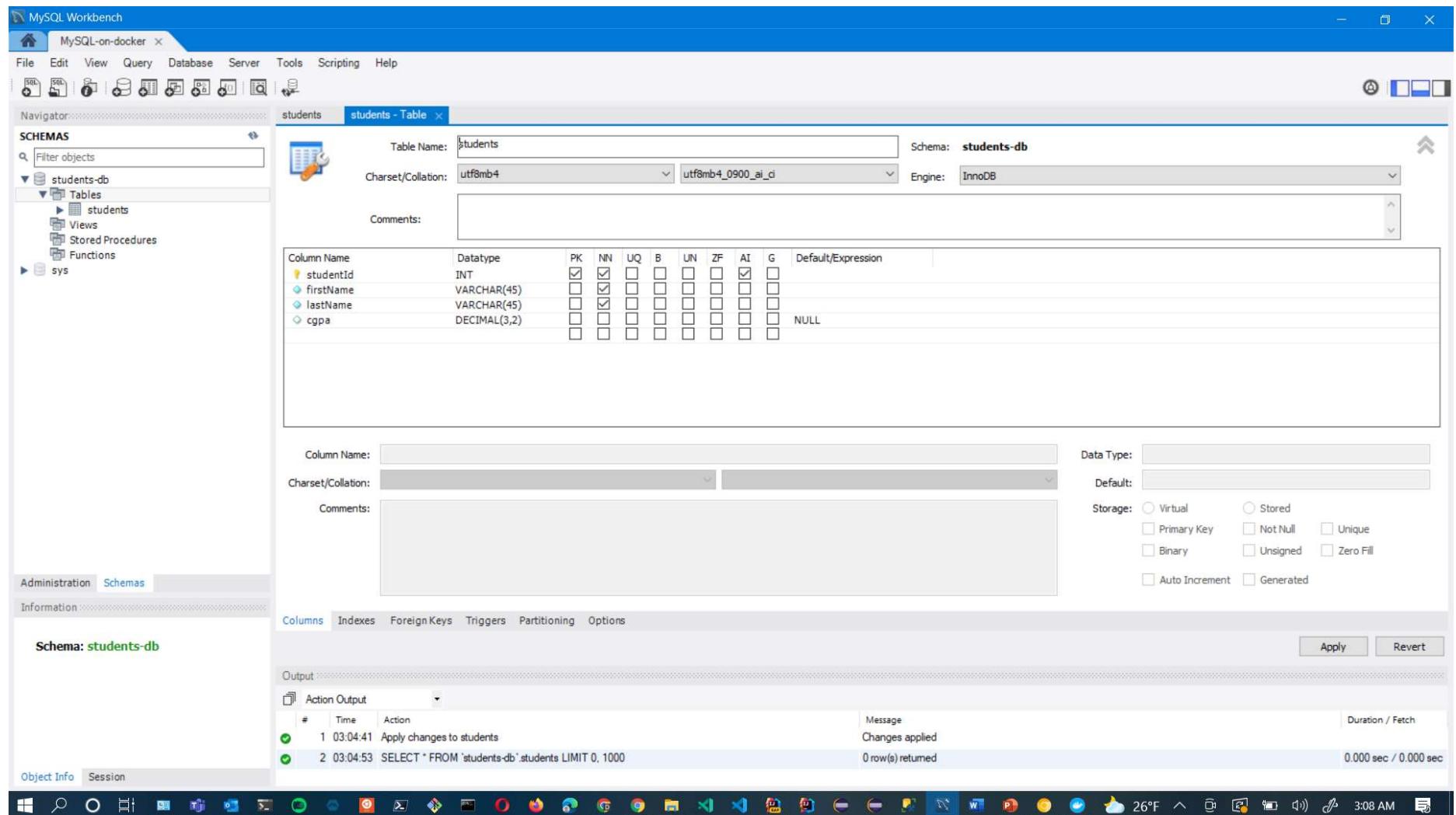
Connect and create a new database



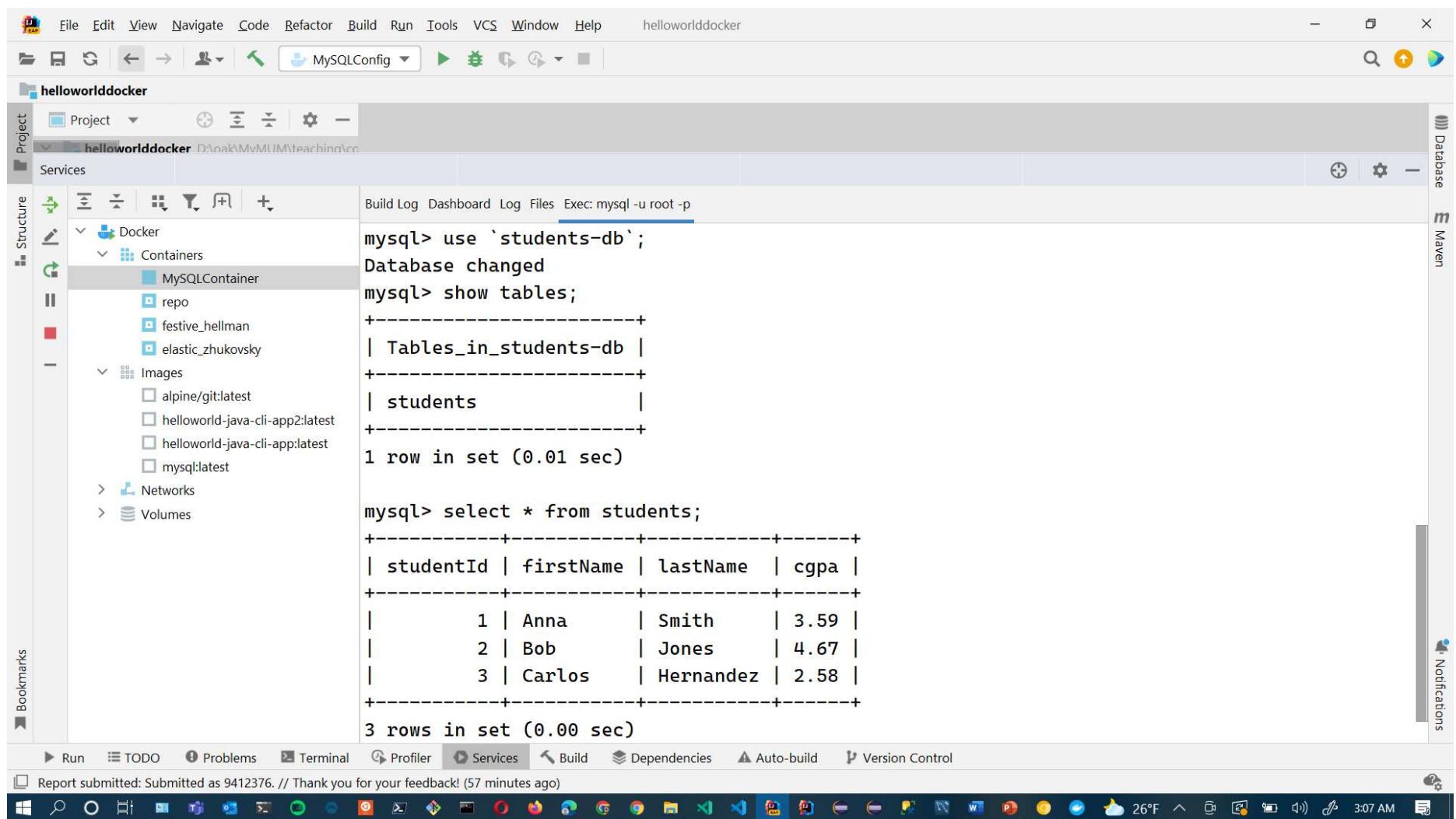
Connecting via MySQL Workbench from the host



Using MySQL Workbench



Working with mysql client in IntelliJ



Containerizing a Spring Boot Application

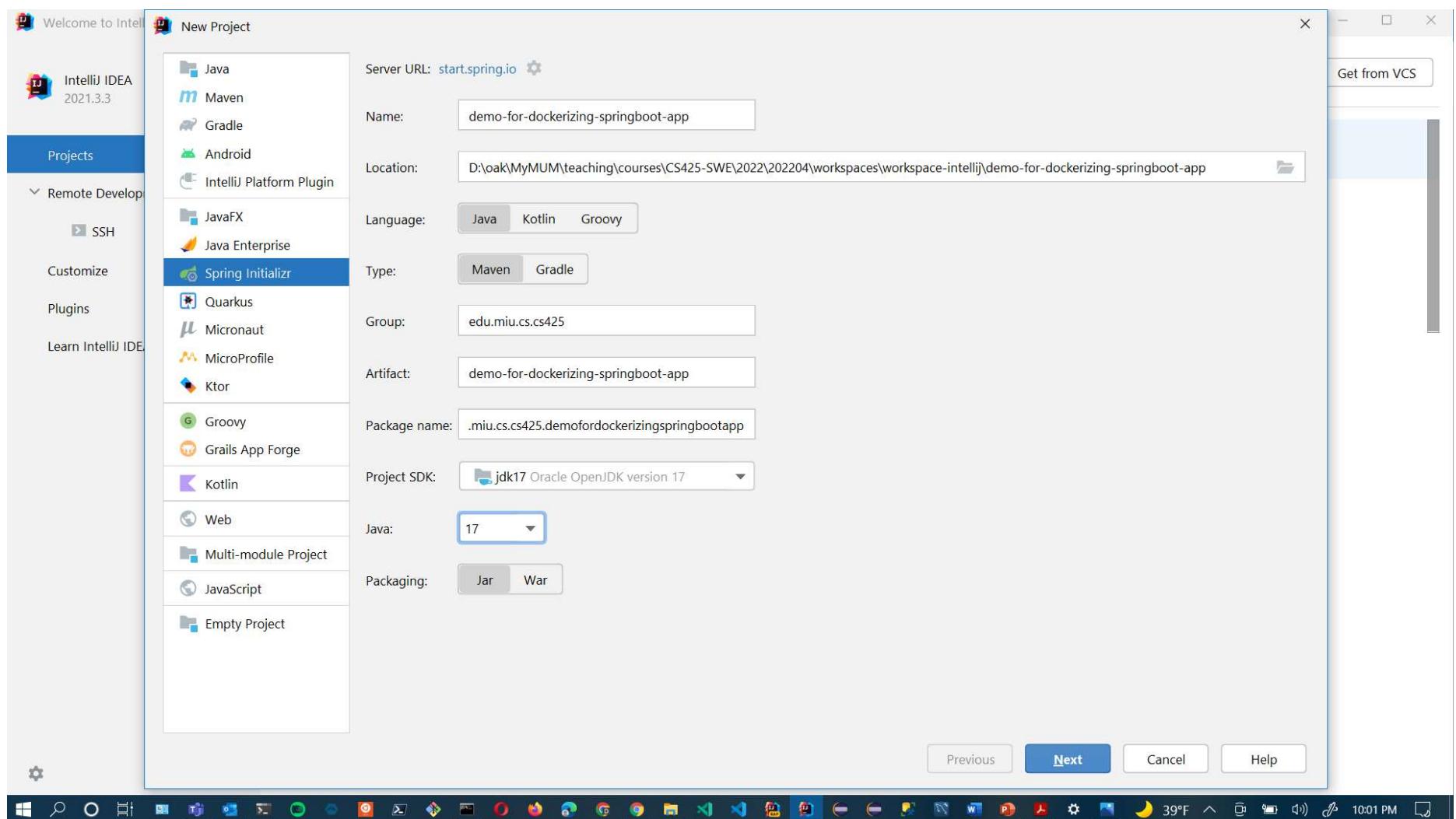
Best Practices for building a Docker Image of your Spring Boot Application

Containerize Spring Boot App with Docker

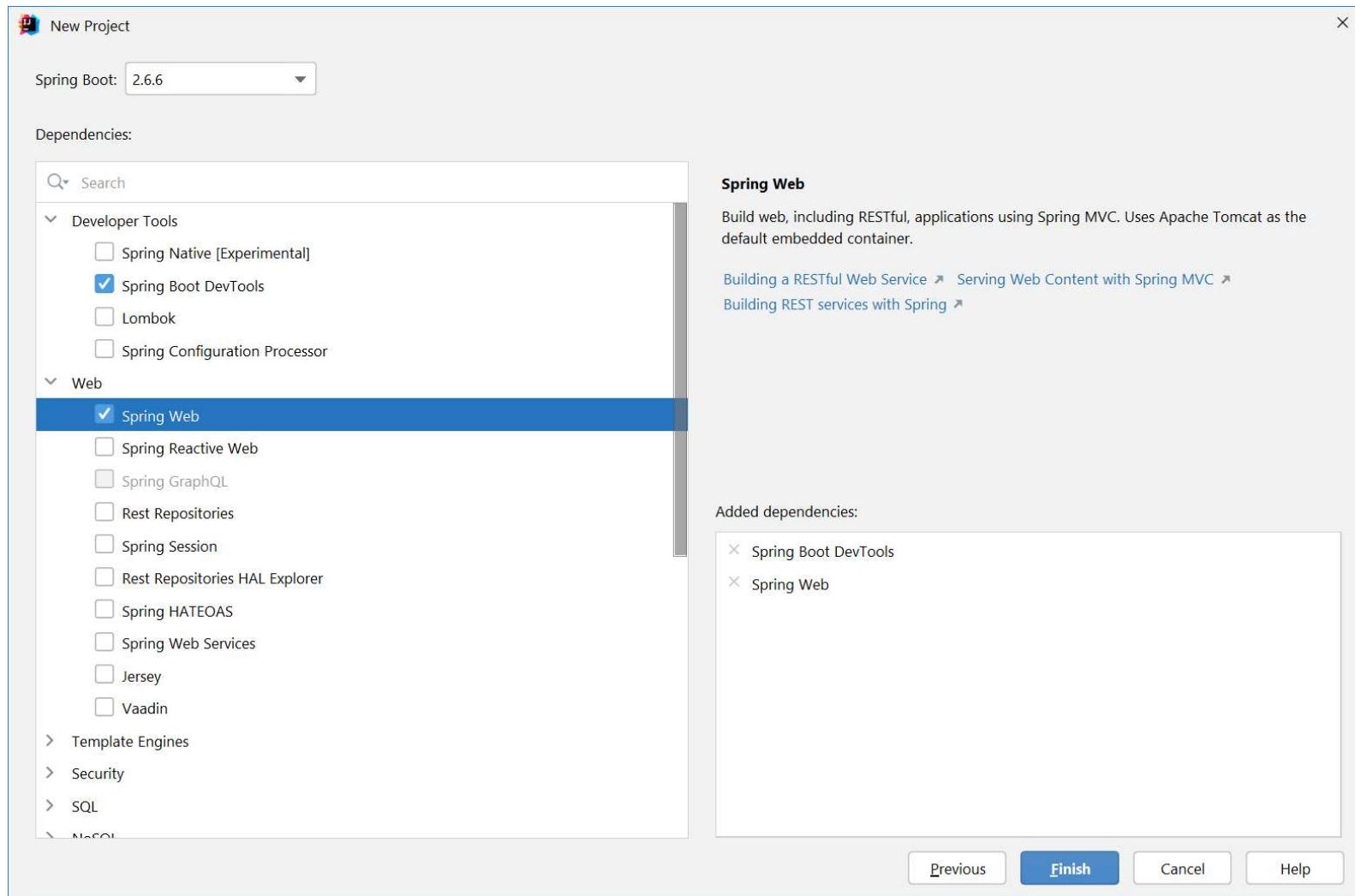
- Docker Image creation functionality was introduced in Spring Boot version 2.3.
- The main dependency required for this is Spring-Boot-DevTools



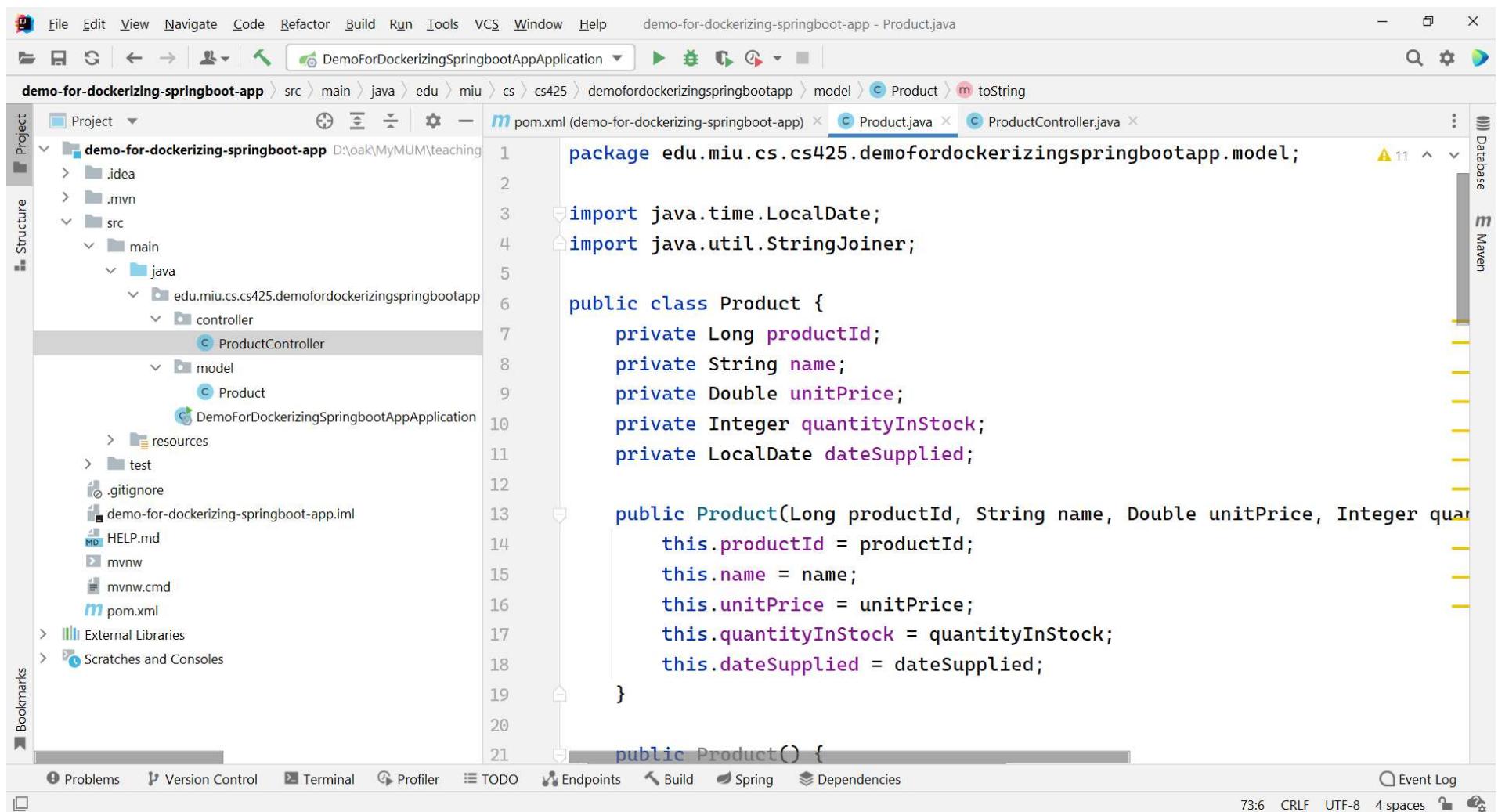
Create a new Spring Initializer Project in IntelliJ IDEA



For the Dependencies: Spring Boot DevTools and Spring Web



Code model class: product.java



The screenshot shows the IntelliJ IDEA interface with the code editor open to the `Product.java` file. The project structure on the left shows a Maven project named `demo-for-dockerizing-springboot-app`. The code editor displays the following Java code:

```
package edu.miu.cs.cs425.demofordockerizingspringbootapp.model;

import java.time.LocalDate;
import java.util.StringJoiner;

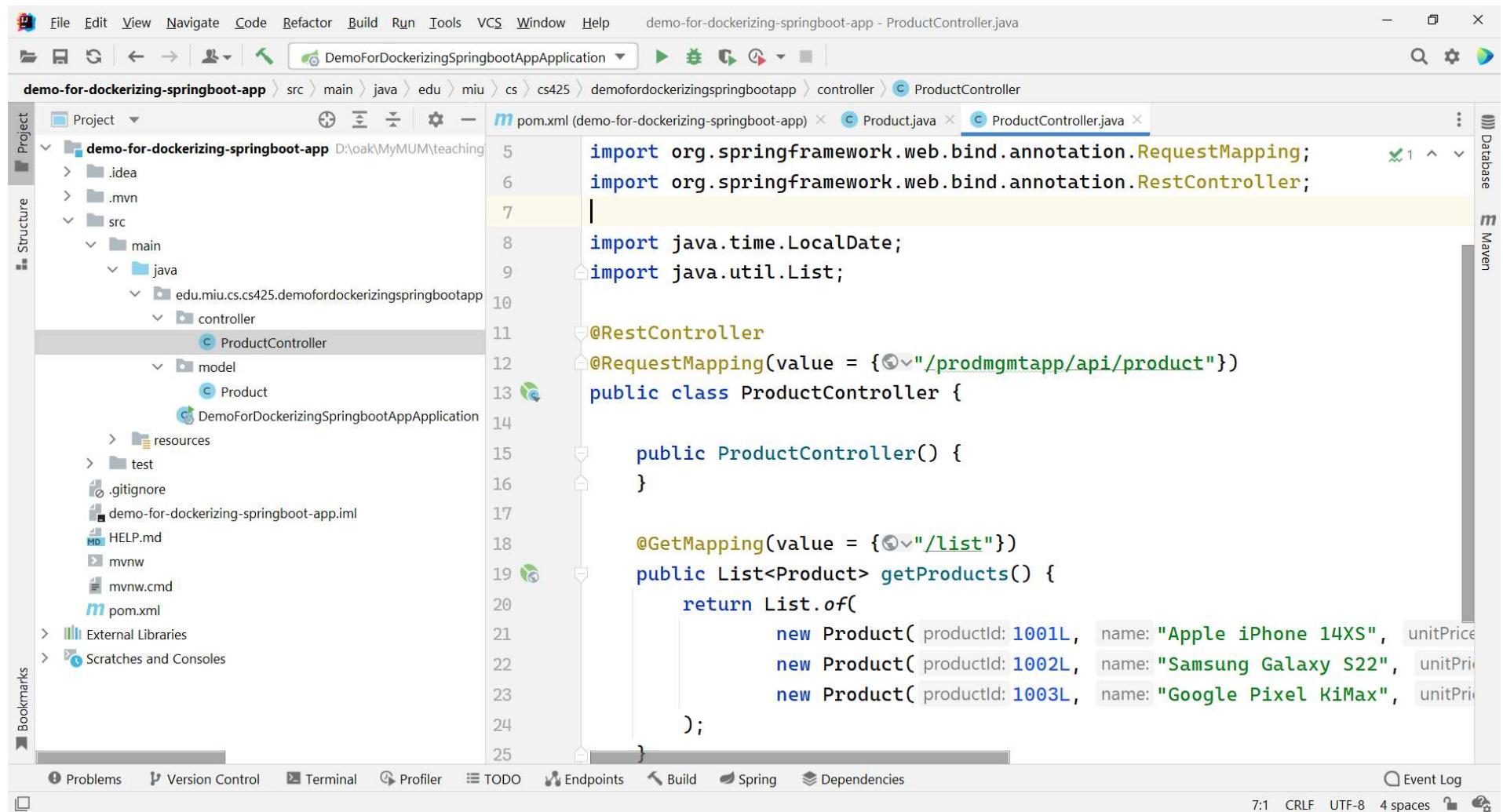
public class Product {
    private Long productId;
    private String name;
    private Double unitPrice;
    private Integer quantityInStock;
    private LocalDate dateSupplied;

    public Product(Long productId, String name, Double unitPrice, Integer quantityInStock, LocalDate dateSupplied) {
        this.productId = productId;
        this.name = name;
        this.unitPrice = unitPrice;
        this.quantityInStock = quantityInStock;
        this.dateSupplied = dateSupplied;
    }

    public Product() {
    }
}
```

The code editor includes syntax highlighting for Java keywords and annotations. The status bar at the bottom shows the file is 73:6 CRLF UTF-8 4 spaces.

Code the RestController class



The screenshot shows the IntelliJ IDEA interface with the code editor displaying `ProductController.java`. The code implements a REST controller for products.

```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import java.time.LocalDate;
import java.util.List;

@RestController
@RequestMapping(value = {"${prodmgmtapp}/api/product"})
public class ProductController {

    public ProductController() {
    }

    @GetMapping(value = {"${prodmgmtapp}/list"})
    public List<Product> getProducts() {
        return List.of(
            new Product( productId: "1001L", name: "Apple iPhone 14XS", unitPrice: 1000 ),
            new Product( productId: "1002L", name: "Samsung Galaxy S22", unitPrice: 1200 ),
            new Product( productId: "1003L", name: "Google Pixel Kimax", unitPrice: 1500 )
        );
    }
}
```

The project structure on the left shows the directory `demo-for-dockerizing-springboot-app` containing `pom.xml`, `src/main/java` with `edu.miu.cs.cs425.demofordockerizingspringbootapp.controller`, and `model`, `Product`, and `DemoForDockerizingSpringbootAppApplication` files.

Run the App

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The project is named "demo-for-dockerizing-springboot-app". The code editor displays the main class, `DemoForDockerizingSpringbootAppApplication.java`, which contains the following code:

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DemoForDockerizingSpringbootAppApplication {

    public static void main(String[] args) {
        SpringApplication.run(DemoForDockerizingSpringbootAppApplication.class, args);
    }
}
```

- Run Console:** The "Run" tab is selected, showing the application's startup logs:

```
restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 13ms
restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path
restartedMain] moForDockerizingSpringbootAppApplication : Started DemoForDockerizingSpringbootAppApplication in 2.809s (Process completed in 2.809s)
```

- Bottom Status Bar:** The status bar indicates: "Build completed successfully in 6 sec, 99 ms (a minute ago)".

Using curl or browser run HTTP GET url

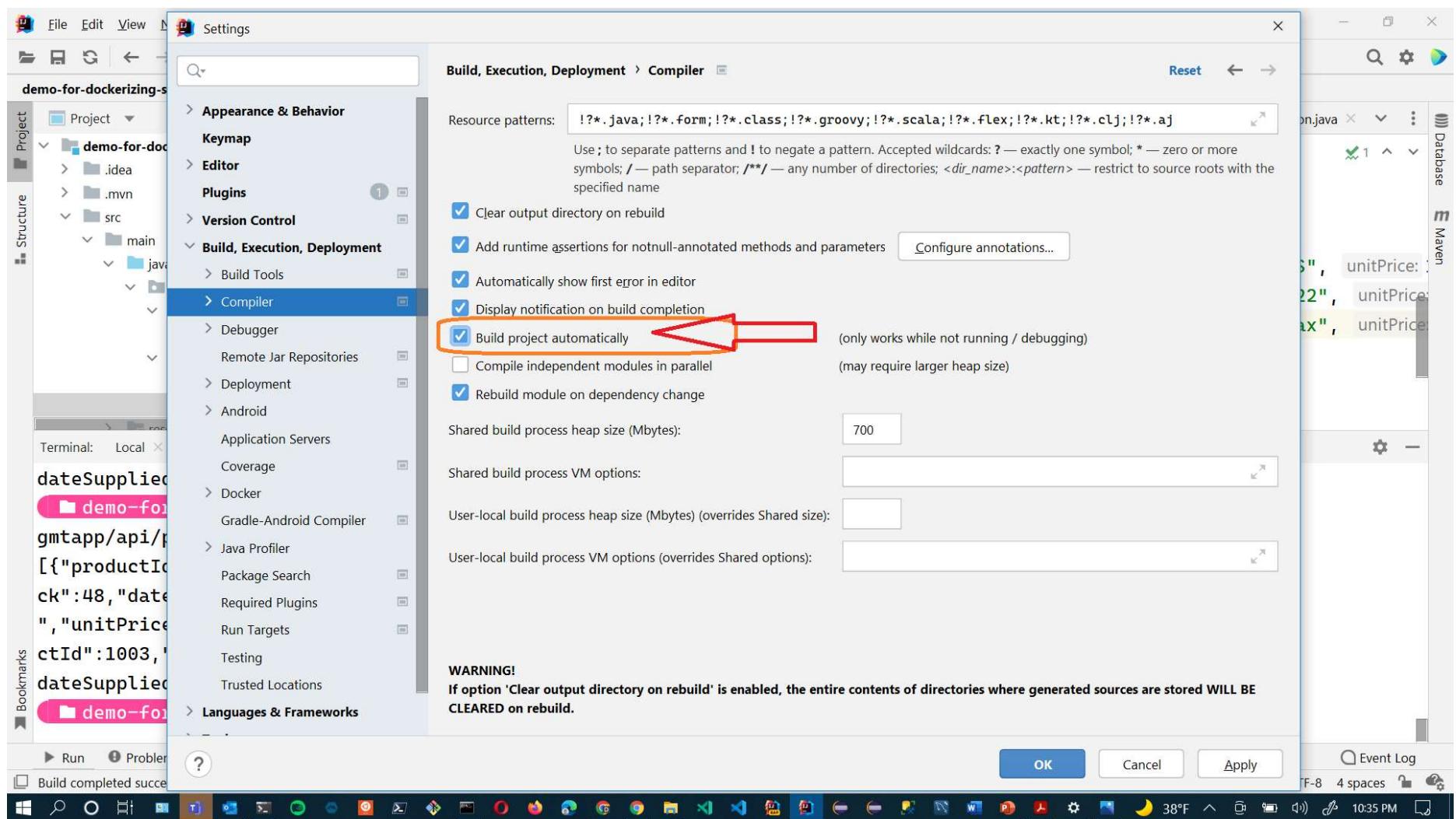
The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The project is named "demo-for-dockerizing-springboot-app". The code editor displays the `ProductController.java` file, which contains a REST controller for products.
- Code Editor:** The code for `ProductController` is shown, including annotations like `@RestController`, `@RequestMapping`, and `@GetMapping`.
- Terminal:** The terminal window shows two curl commands being run:

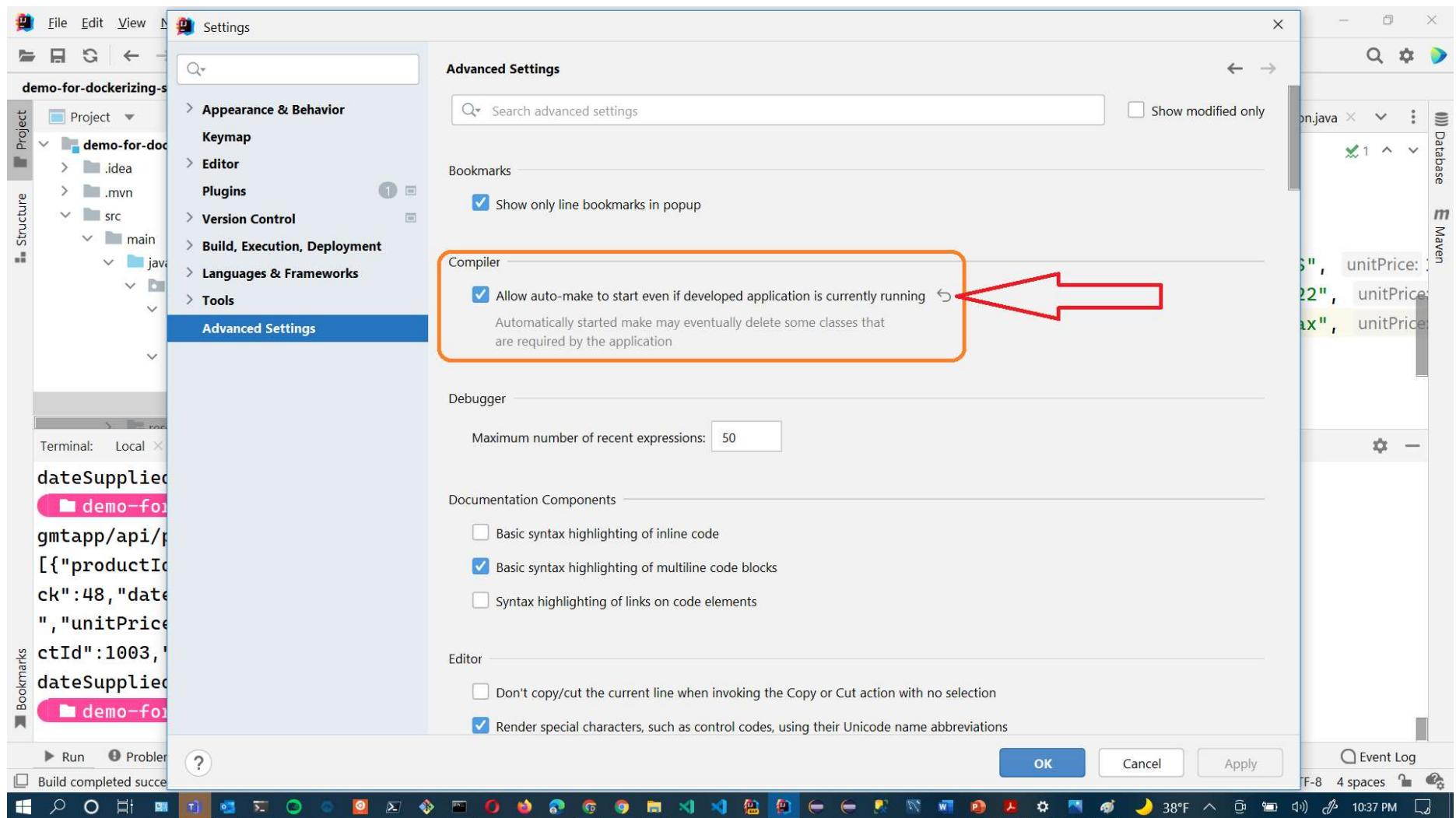
```
curl http://localhost:8080/prodmgmtapp/api/product
curl http://localhost:8080/prodmgmtapp/api/product/list
```
- Output:** The terminal output shows the JSON response for the product list:

```
[{"productId":1001,"name":"Apple iPhone 14XS","unitPrice":1850.95,"quantityInStock":48,"dateSupplied":"2022-01-15"}, {"productId":1002,"name":"Samsung Galaxy S22","unitPrice":1700.99,"quantityInStock":115,"dateSupplied":"2021-10-27"}, {"productId":1003,"name":"Google Pixel Kimax","unitPrice":1350.5,"quantityInStock":82,"dateSupplied":"2022-03-19"}]
```
- Bottom Navigation:** The navigation bar includes tabs for Run, Problems, Version Control, Terminal, Profiler, TODO, Endpoints, Build, Spring, Dependencies, and Event Log.
- Build Status:** A message at the bottom indicates "Build completed successfully in 6 sec, 99 ms (5 minutes ago)".

Open IntelliJ Settings and enable this:



Open IntelliJ Settings and enable this:



Re-run the Application and check that the container auto-reloads on any code changes (i.e. DevTools works)

The screenshot shows an IDE interface with the following details:

- File Menu:** File, Edit, View, Navigate, Code, Refactor, Build, Run, Tools, VCS, Window, Help.
- Title Bar:** demo-for-dockerizing-springboot-app - ProductController.java
- Toolbar:** Standard icons for file operations.
- Project Explorer:** Shows the project structure under "demo-for-dockerizing-springboot-app".
- Code Editor:** Displays `ProductController.java` with code for returning a list of products.
- Terminal:** Local PowerShell tab showing three consecutive curl requests to `http://localhost:8080/prodmgmtapp/api/product/list`. Each request returns a JSON array of products, with the third request showing a slight variation in product data.

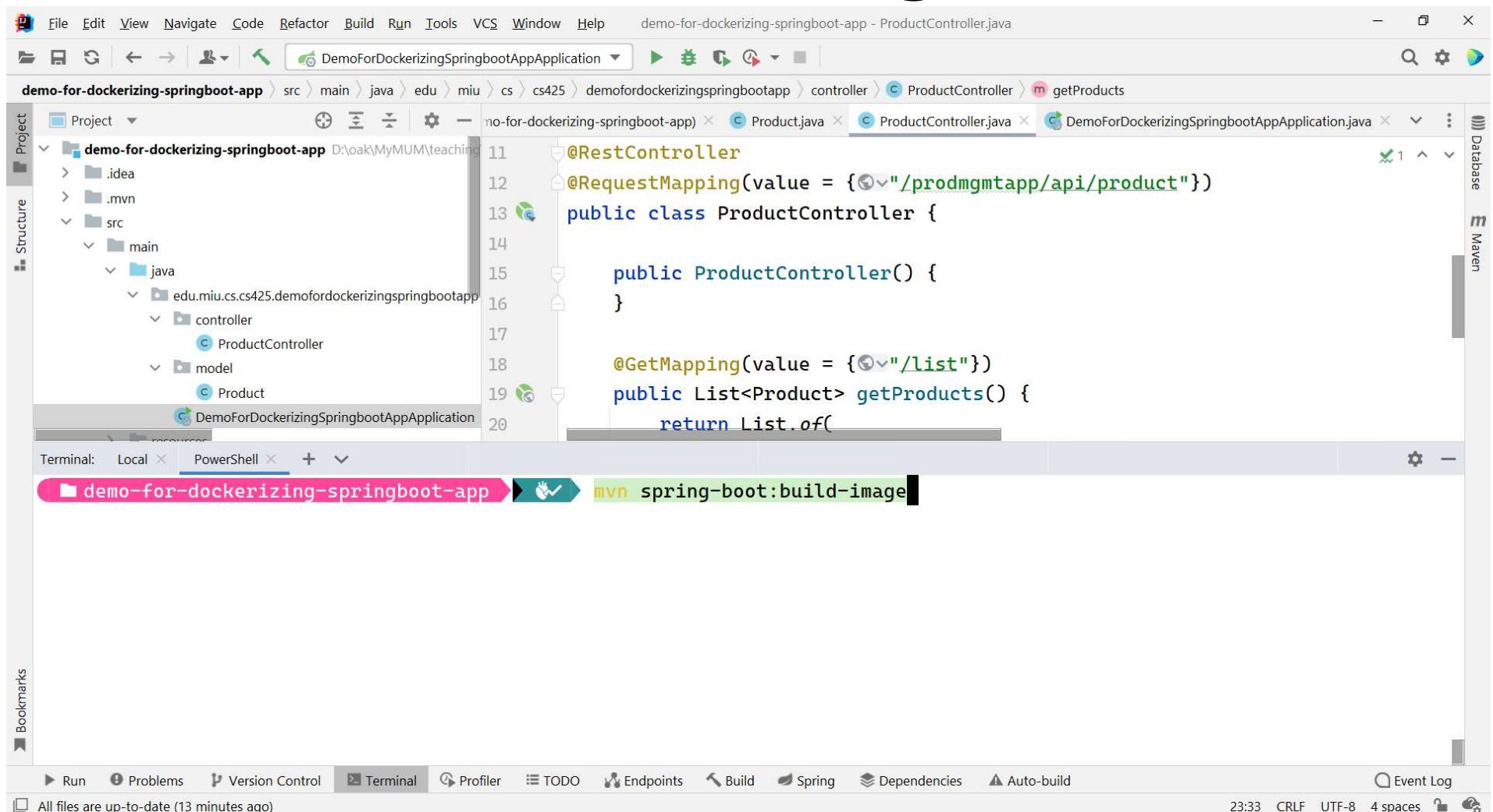
```
curl http://localhost:8080/prodmgmtapp/api/product/list
[{"productId":1002,"name":"Samsung Galaxy S22","unitPrice":1700.99,"quantityInStock":115,"dateSupplied":"2021-10-27"}, {"productId":1009,"name":"Google Pixel KiMax","unitPrice":1350.5,"quantityInStock":82,"dateSupplied":"2022-03-19"}]
curl http://localhost:8080/prodmgmtapp/api/product/list
[{"productId":1001,"name":"Apple iPhone 14XS","unitPrice":1850.95,"quantityInStock":48,"dateSupplied":"2022-01-15"}, {"productId":1002,"name":"Samsung Galaxy S22","unitPrice":1700.99,"quantityInStock":115,"dateSupplied":"2021-10-27"}, {"productId":1003,"name":"Google Pixel KiMax","unitPrice":1350.5,"quantityInStock":82,"dateSupplied":"2022-03-19"}]
curl http://localhost:8080/prodmgmtapp/api/product/list
[{"productId":1001,"name":"Apple iPhone 14XS","unitPrice":1850.95,"quantityInStock":48,"dateSupplied":"2022-01-15"}, {"productId":1002,"name":"Samsung Galaxy S22","unitPrice":1700.99,"quantityInStock":115,"dateSupplied":"2021-10-27"}, {"productId":1009,"name":"Google Pixel KiMax","unitPrice":1350.5,"quantityInStock":82,"dateSupplied":"2022-03-19"}]
```

- Bottom Bar:** Run, Problems, Version Control, Terminal, Profiler, TODO, Endpoints, Build, Spring, Dependencies, Auto-build, Event Log.
- Status Bar:** All files are up-to-date (2 minutes ago), 23:33, CRLF, UTF-8, 4 spaces.

How to Build Docker Image of the Spring Boot Application

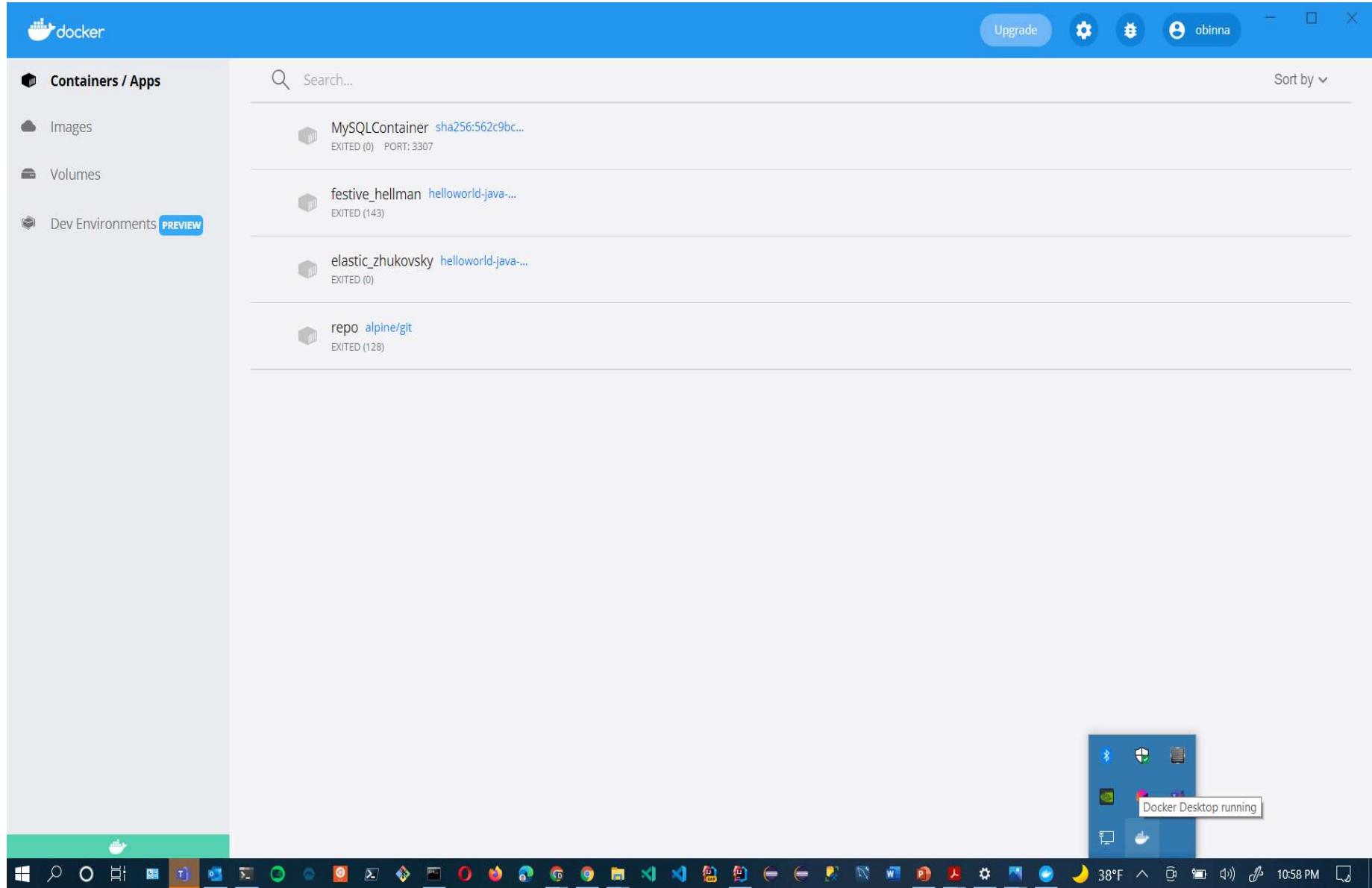
- Beginning with Spring Boot version 2.3, the `spring-boot-maven-plugin` can create an OCI (e.g. docker) image from a `.jar` or `.war` package file using CNCF's Cloud Native Buildpacks
- Images can be built by using the `mvn build-image` goal
- Open terminal and run:
 > `mvn spring-boot:build-image`

Build Image



Note: Make sure docker daemon is running before executing build-image

Docker is up and running



Building Docker Image

The screenshot shows the IntelliJ IDEA interface with a Java Spring Boot application named "demo-for-dockerizing-springboot-app". The code editor displays the `ProductController.java` file, which contains the following code:

```
11  @RestController
12  @RequestMapping(value = {"prodmgmtapp/api/product"})
13  public class ProductController {
14
15      public ProductController() {
16  }
17
18      @GetMapping(value = {"list"})
19      public List<Product> getProducts() {
20          return List.of(
21              new Product("Laptop", "A high-end laptop with a large screen and fast processor.", 1200),
22              new Product("Smartphone", "A sleek smartphone with a triple-camera system and long battery life.", 800),
23              new Product("Tablet", "A compact tablet with a responsive screen and portable design.", 600)
24          );
25      }
26  }
```

The terminal window at the bottom shows the Maven command being run to build the Docker image:

```
[INFO] --- spring-boot-maven-plugin:2.6.6:build-image (default-cli) @ demo-for-dockerizing-springboot-app ---
[INFO] Building image 'docker.io/library/demo-for-dockerizing-springboot-app:0.0.1-SNAPSHOT'
[INFO]
[INFO] > Pulling builder image 'docker.io/paketobuildpacks/builder:base' 0%
[INFO] > Pulling builder image 'docker.io/paketobuildpacks/builder:base' 0%
[INFO] > Pulling builder image 'docker.io/paketobuildpacks/builder:base' 1%
[INFO] > Pulling builder image 'docker.io/paketobuildpacks/builder:base' 2%
```

Successfully Built Image

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The project is named "demo-for-dockerizing-springboot-app". It contains a "src" directory with "main" and "java" sub-directories. The "java" directory contains "edu.miu.cs.cs425.demofordockerizingspringbootapp" package, which has "controller" and "model" sub-packages. "ProductController.java" is open in the editor.
- Code Editor:** The code for ProductController.java is displayed:

```
11  @RestController
12  @RequestMapping(value = {"prodmgmtapp/api/product"})
13  public class ProductController {
14
15      public ProductController() {
16  }
17
18      @GetMapping(value = {"list"})
19      public List<Product> getProducts() {
20          return List.of()
```
- Terminal:** The terminal output shows the results of a Docker build:

```
[INFO] Successfully built image 'docker.io/library/demo-for-dockerizing-springboot-app:0.0.1-SNAPSHOT'
[INFO]
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 01:37 min
[INFO] Finished at: 2022-04-01T23:01:53-05:00
[INFO]
```
- Status Bar:** The status bar at the bottom indicates "All files are up-to-date (23 minutes ago)" and shows system information like "23:33 CRLF UTF-8 4 spaces".

Start the Application in Docker container

- To spin-up a container running the application, execute the command:
- > docker run –tty –publish 8080:8080 demo-for-dockerizing-springboot-app:0.0.1-SNAPSHOT

Creates and Run a container

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The project is named "demo-for-dockerizing-springboot-app". It contains a "src" directory with "main" and "java" sub-directories. The "java" directory contains "edu.miu.cs.cs425.demofordockerizingspringbootapp" package, which has "controller" and "model" sub-packages. "ProductController.java" is open in the editor.
- Code Editor:** The code for ProductController.java is displayed:

```
11  @RestController
12  @RequestMapping(value = {"prodmgmtapp/api/product"})
13  public class ProductController {
14
15      public ProductController() {
16
17
18      @GetMapping(value = {"list"})
19      public List<Product> getProducts() {
20          return List.of()
```
- Terminal:** A PowerShell terminal window is open, showing the command: `docker run --tty --publish 8080:8080 demo-for-dockerizing-springboot-app:0.1-SNAPSHOT`.
- Bottom Navigation:** The navigation bar includes "Run", "Problems", "Version Control", "Terminal", "Profiler", "TODO", "Endpoints", "Build", "Spring", "Dependencies", "Auto-build", and "Event Log".
- Status Bar:** The status bar at the bottom indicates "All files are up-to-date (33 minutes ago)" and shows system information: "23:33 CRLF UTF-8 4 spaces".

Test the container endpoint

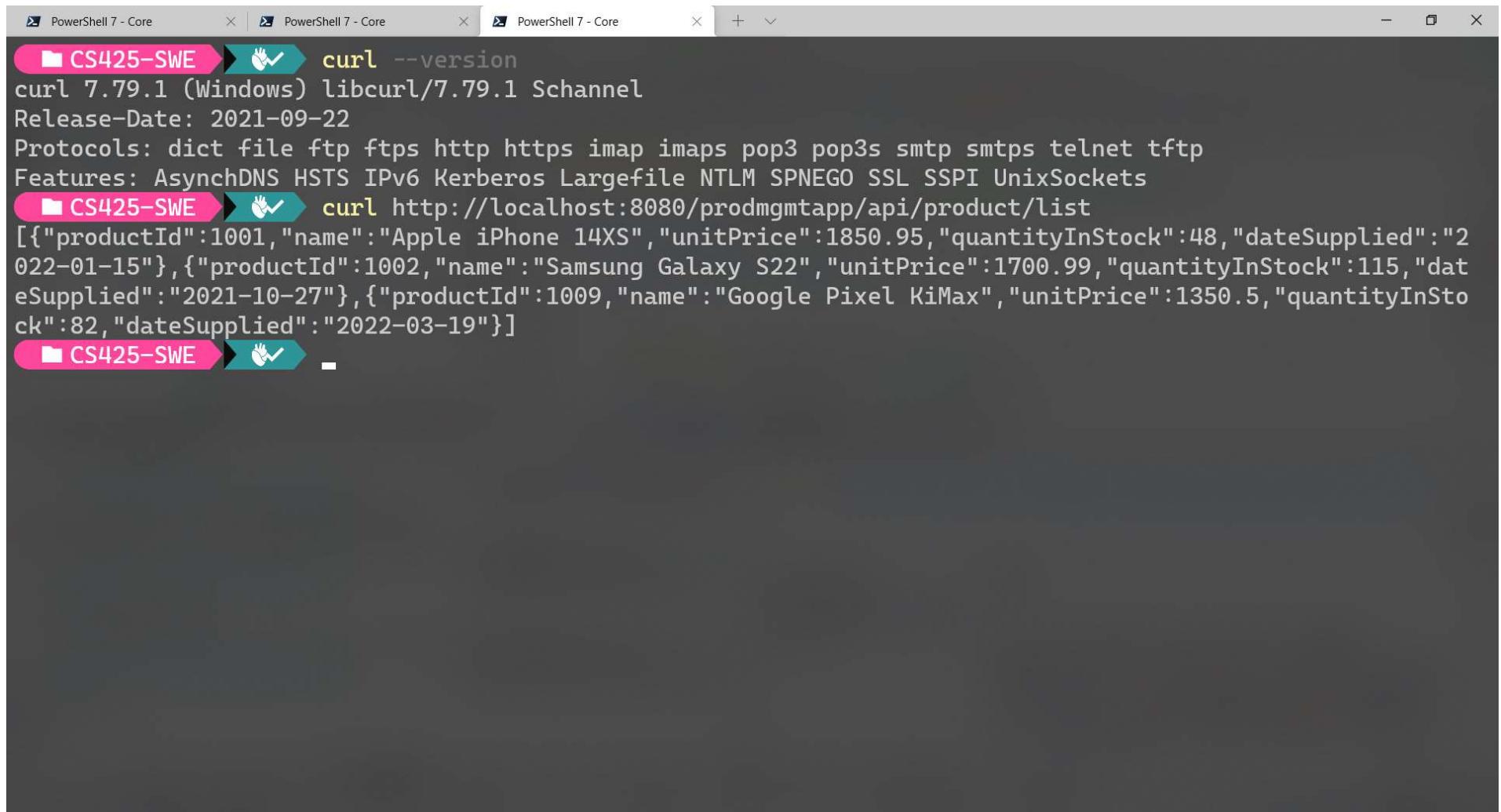
The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The project is named "demo-for-dockerizing-springboot-app". The code editor displays the `ProductController.java` file, which contains a `@RestController` and a `@GetMapping` method for listing products.
- Terminal:** The terminal window shows the application's logs and a command to test the endpoint.

```
c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'  
2022-04-02 04:17:35.585 INFO 1 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'  
2022-04-02 04:17:35.587 INFO 1 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 2 ms
```

```
> curl http://localhost:8080/prodmgmtapp/api/product/list
```
- Logs:** The right side of the terminal window shows the response from the curl command, including status code, status description, and content.
- Bottom Status Bar:** Shows "All files are up-to-date (40 minutes ago)" and other system information like time and encoding.

Test the Container Endpoint



A screenshot of a Windows Taskbar showing three PowerShell 7 - Core windows. The active window displays the output of a curl command:

```
curl 7.79.1 (Windows) libcurl/7.79.1 Schannel
Release-Date: 2021-09-22
Protocols: dict file ftp ftps http https imap imaps pop3 pop3s smtp smtps telnet tftp
Features: AsynchDNS HSTS IPv6 Kerberos Largefile NTLM SPNEGO SSL SSPI UnixSockets
curl http://localhost:8080/prodmgmtapp/api/product/list
[{"productId":1001,"name":"Apple iPhone 14XS","unitPrice":1850.95,"quantityInStock":48,"dateSupplied":"2022-01-15"}, {"productId":1002,"name":"Samsung Galaxy S22","unitPrice":1700.99,"quantityInStock":115,"dateSupplied":"2021-10-27"}, {"productId":1009,"name":"Google Pixel KiMax","unitPrice":1350.5,"quantityInStock":82,"dateSupplied":"2022-03-19"}]
```

In Split Screen

The screenshot shows a Windows desktop with a split-screen feature. On the left side, there is a PowerShell window titled "PowerShell 7 - Core". The command entered is:

```
CS425-SWE ➔ 🐳 ➔ docker run --tty --publish 8080:8080 demo-for-dockerizing-springboot-app:0.0.1-SNAPSHOT
```

The output of the command is displayed, showing the Docker container starting up and configuring the Java Virtual Machine (JVM) with specific memory settings. It also mentions enabling Java Native Memory Tracking and adding certificates to the JVM trust store.

On the right side, there is another PowerShell window titled "PowerShell 7.2.2". The command entered is:

```
curl http://localhost:8080/prodmgmt/app/api/product/list
```

The output of this command is a JSON array of product details, including their names, unit prices, quantities in stock, and dates supplied. The products listed are Apple iPhone 14XS, Samsung Galaxy S22, and Google Pixel Kimax.

```
[{"productId":1001, "name": "Apple iPhone 14XS", "unitPrice": 1850.95, "quantityInStock": 48, "dateSupplied": "2022-01-15"}, {"productId":1002, "name": "Samsung Galaxy S22", "unitPrice": 1700.99, "quantityInStock": 115, "dateSupplied": "2021-10-27"}, {"productId":1009, "name": "Google Pixel Kimax", "unitPrice": 1350.5, "quantityInStock": 82, "dateSupplied": "2022-03-19"}]
```

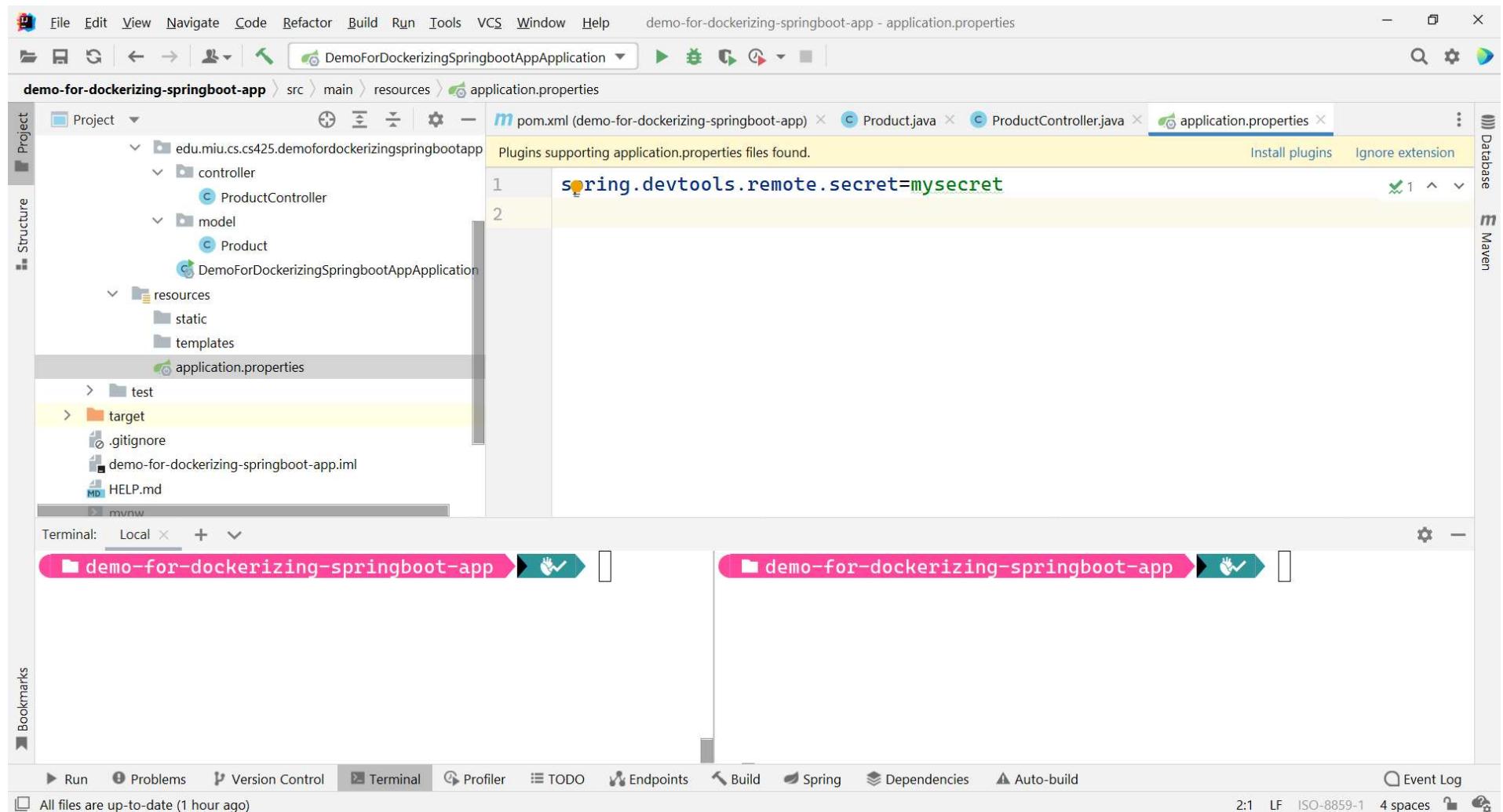
Next: Enable DevTools auto-reload in the container

- Open application.properties file and add the setting:
 - spring.devtools.remote.secret=mysecret
- Open the pom.xml file, add the config below



```
pom.xml (demo-for-dockerizing-springboot-app) × C Product.java × C ProductController.java × application.properties ×  
38 <build>  
39   <plugins>  
40     <plugin>  
41       <groupId>org.springframework.boot</groupId>  
42       <artifactId>spring-boot-maven-plugin</artifactId>  
43       <configuration>  
44         <excludeDevtools>false</excludeDevtools>  
45       </configuration>  
46     </plugin>  
47   </plugins>  
48 </build>
```

application.properties



pom.xml

The screenshot shows the IntelliJ IDEA interface with the following details:

- File Bar:** File, Edit, View, Navigate, Code, Refactor, Build, Run, Tools, VCS, Window, Help.
- Title Bar:** demo-for-dockerizing-springboot-app - pom.xml (demo-for-dockerizing-springboot-app)
- Toolbars:** Standard toolbar with icons for file operations.
- Project Tool Window:** Shows the project structure with packages like edu.miu.cs.cs425.demofordockerizingspringbootapp, controller, model, and resources.
- Code Editor:** The pom.xml file is open at line 42, showing configuration for the spring-boot-maven-plugin. The code includes:

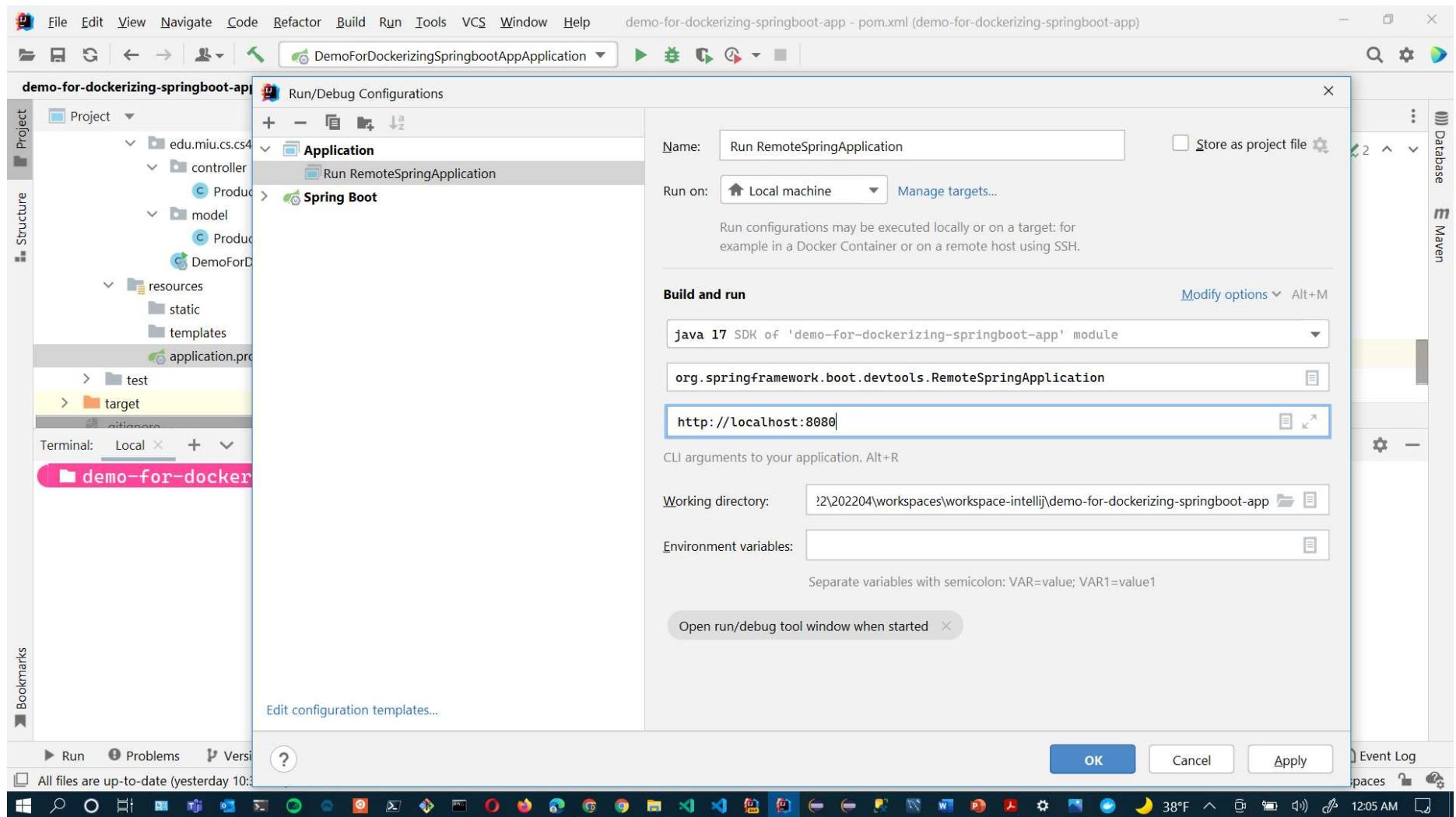
```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <configuration>
                <excludeDevtools>false</excludeDevtools>
            </configuration>
        </plugin>
    </plugins>
</build>
```
- Terminal:** Two terminals are shown, both titled "demo-for-dockerizing-springboot-app".
- Bottom Navigation:** Run, Problems, Version Control, Terminal, Profiler, TODO, Endpoints, Build, Spring, Dependencies, Auto-build, Event Log.
- Status Bar:** All files are up-to-date (today 10:39 PM), 44:43 LF, UTF-8, 4 spaces.

Stop the container, delete it and re-build image again

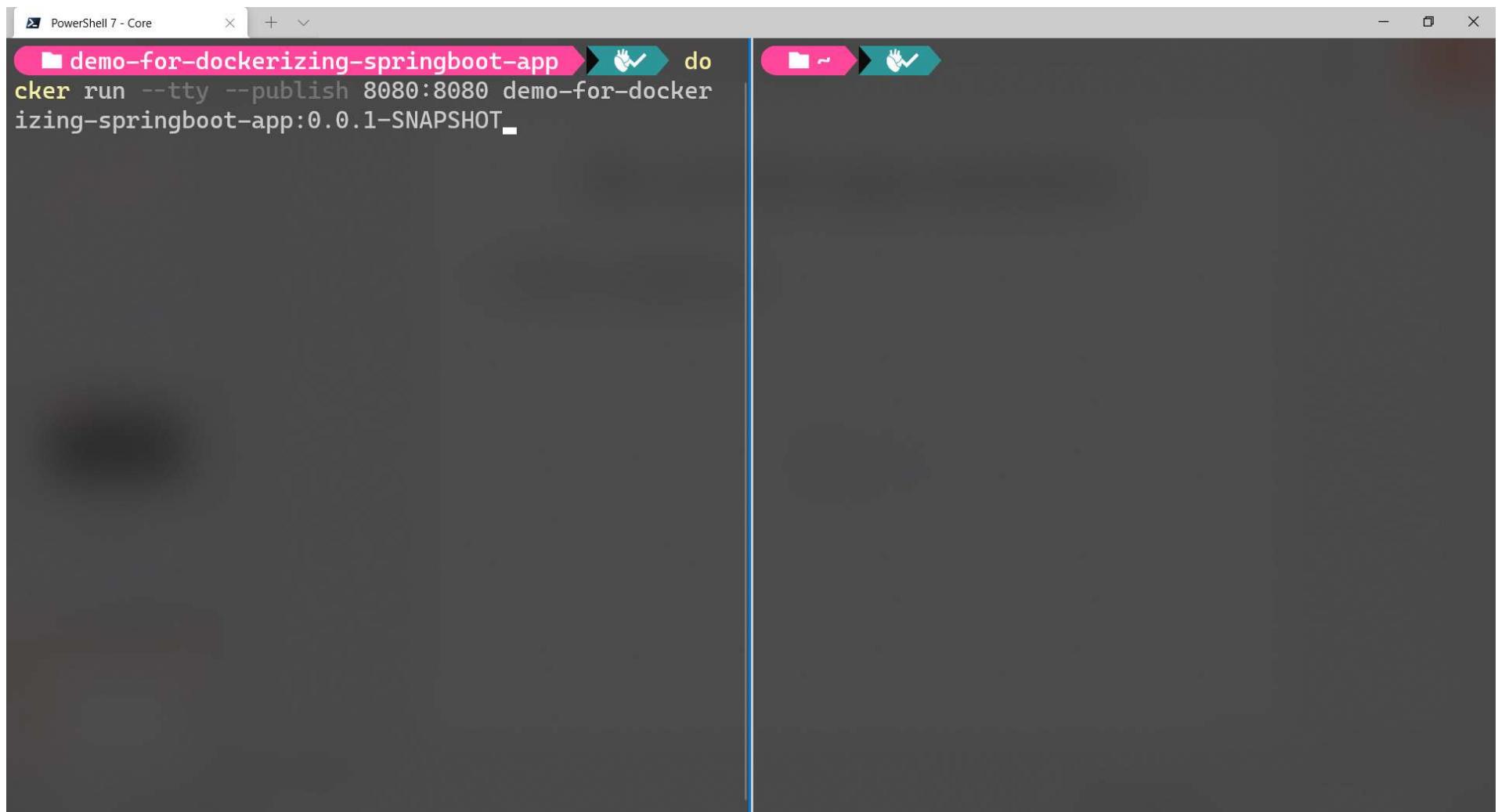
The screenshot shows a PowerShell window with two tabs. The left tab is titled 'demo-for-dockerizing-springboot-app' and contains the command 'mv n spring-boot:build-image_'. The right tab is titled 'PowerShell 7 - Core' and contains the following sequence of commands:

```
docker ps --all
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
98ac49d3f560        562c9bc24a08      docker-entrypoint.s... " 5 days ago   Exited (0) 5 days ago          MySQLContainer
cff62ae8ea3b        helloworld-java-cli-app2:latest "java -jar ./hellowo..." 5 days ago   Exited (143) 5 days ago          festive_hellman
f55eadfbfa8        helloworld-java-cli-app:latest "java -jar ./hellowo..." 5 days ago   Exited (0) 5 days ago          elastic_zhukovsky
463f72e32d07        alpine/git           git clone https://g... " 6 days ago   Exited (128) 6 days ago          repo
docker container rm 728
```

Create a Run Config for a RemoteSpringApplication



Re-run the app container

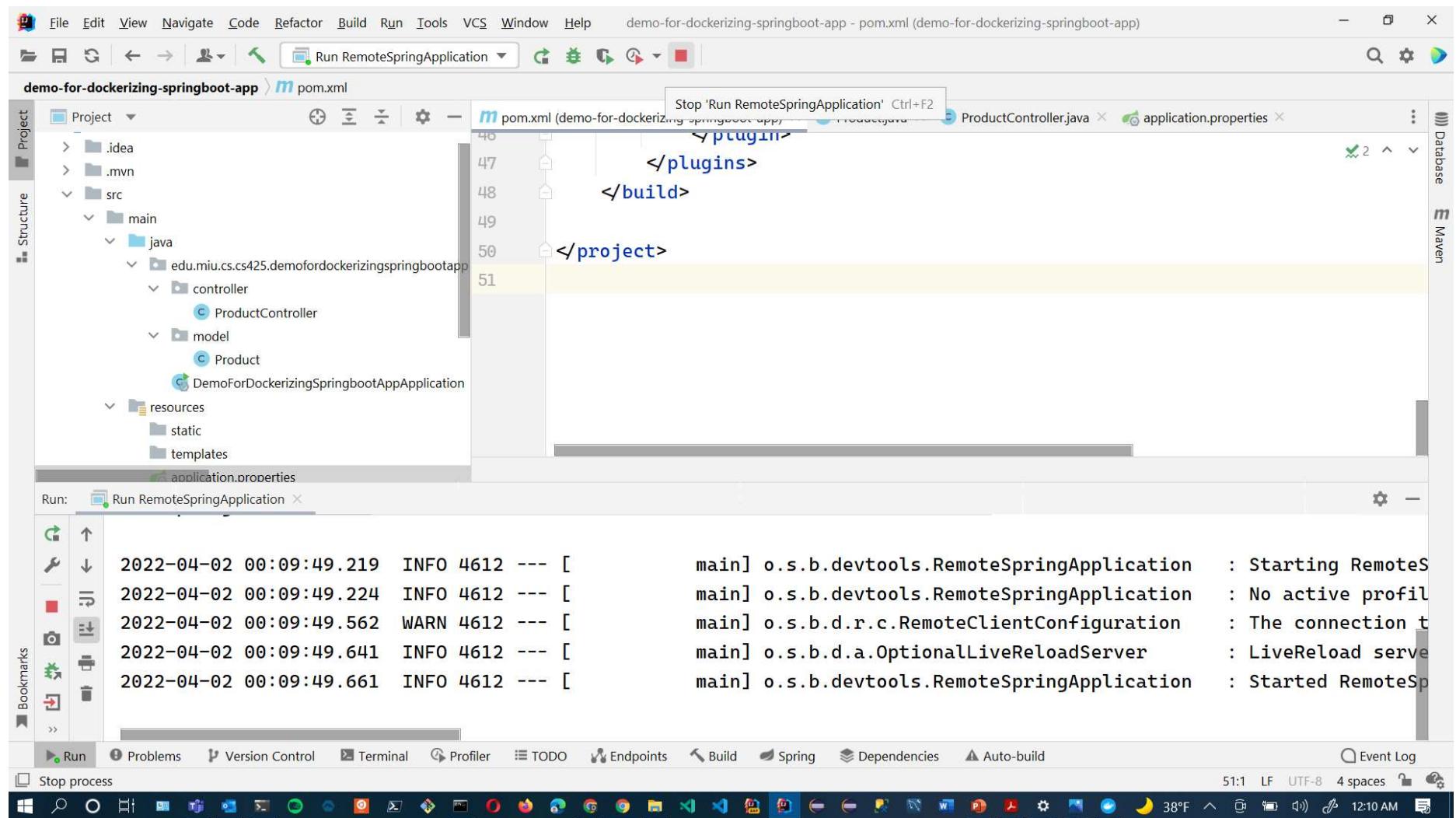


A screenshot of a Windows PowerShell window titled "PowerShell 7 - Core". The window contains a single command:

```
demo-for-dockerizing-springboot-app ➔ do
cker run --tty --publish 8080:8080 demo-for-docker
izing-springboot-app:0.0.1-SNAPSHOT
```

The command uses the Docker Compose command (`do`) to run the `demo-for-dockerizing-springboot-app` service from the current directory, publishing port 8080 to 8080. The Docker image is specified as `demo-for-dockerizing-springboot-app:0.0.1-SNAPSHOT`.

Run the RemoteSpringApplication



Modify the code and see it reload

The screenshot shows the IntelliJ IDEA IDE interface. The main window displays the code for `ProductController.java`, which contains a `@GetMapping` annotation and a `getProducts` method that returns a list of `Product` objects. The code is annotated with various IDE features like `GetMapping`, `List`, and `Product`. The `Run` tool window at the bottom shows logs for a `Run RemoteSpringApplication` run, with several INFO messages from the application's startup and configuration. The status bar at the bottom right indicates the current time is 24:11, the date is April 2, 2022, and the system temperature is 38°F.

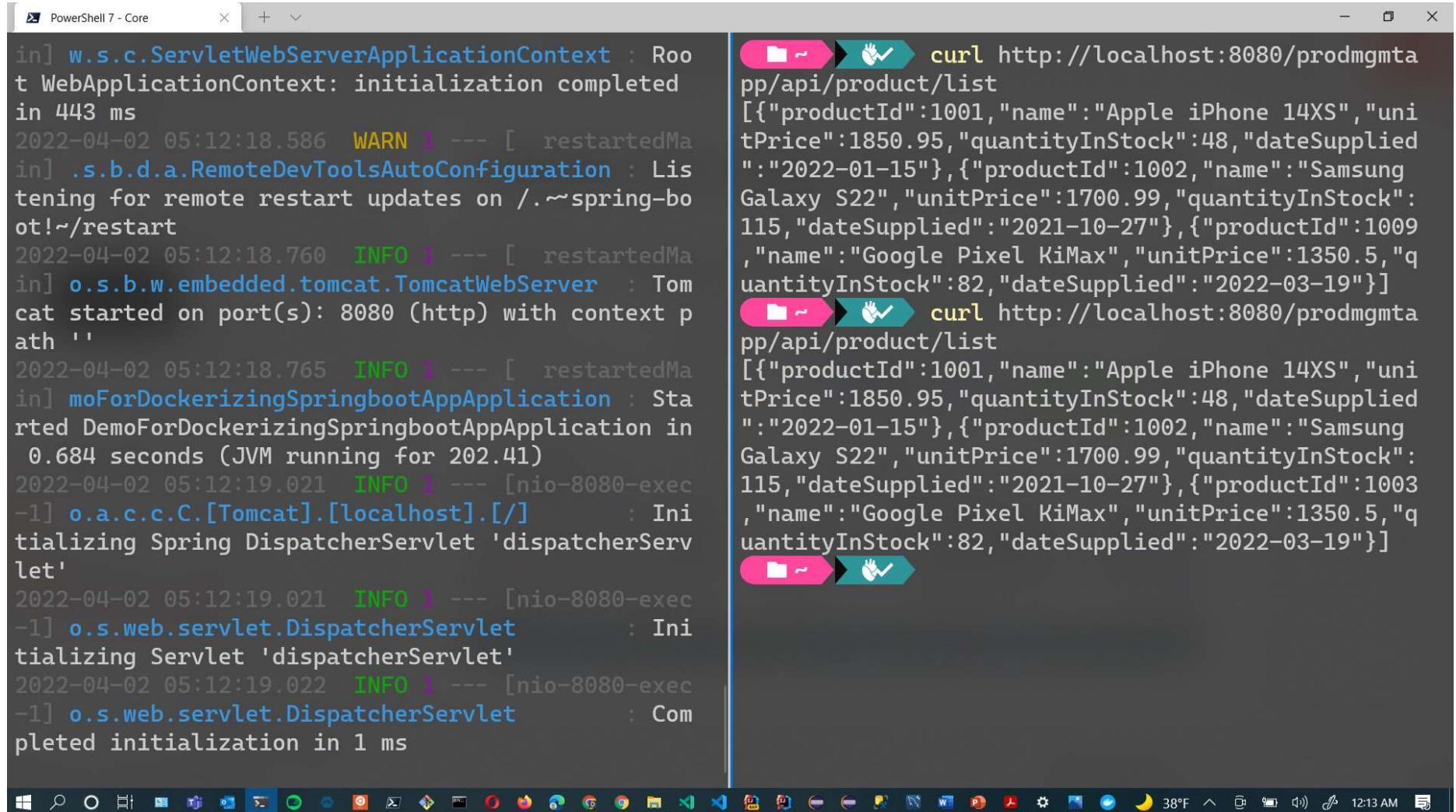
```
17
18
19 @GetMapping(value = {"/list"})
20 public List<Product> getProducts() {
21     return List.of(
22         new Product(productId: 1001L, name: "Apple iPhone 14XS", unitPrice: 1000),
23         new Product(productId: 1002L, name: "Samsung Galaxy S22", unitPrice: 900),
24         new Product(productId: 1003L, name: "Google Pixel KiMax", unitPrice: 800)
25     );
26 }
27 }
```

Run: Run RemoteSpringApplication

```
2022-04-02 00:09:49.641 INFO 4612 --- [           main] o.s.b.d.a.OptionalLiveReloadServer      : LiveReload server
2022-04-02 00:09:49.661 INFO 4612 --- [           main] o.s.b.devtools.RemoteSpringApplication : Started RemoteSp
2022-04-02 00:12:16.400 INFO 4612 --- [File Watcher] o.s.b.d.r.c.ClassPathChangeUploader : Uploaded 1 class
2022-04-02 00:12:17.936 INFO 4612 --- [File Watcher] o.s.b.d.r.c.ClassPathChangeUploader : Uploaded 1 class
2022-04-02 00:12:19.029 INFO 4612 --- [pool-1-thread-1] o.s.b.d.r.c.DelayedLiveReloadTrigger : Remote server ha
2022-04-02 00:12:20.053 INFO 4612 --- [pool-1-thread-1] o.s.b.d.r.c.DelayedLiveReloadTrigger : Remote server ha
```

All files are up-to-date (3 minutes ago)

Verify the output



```
in] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 443 ms
2022-04-02 05:12:18.586  WARN [ restartedMain] .s.b.d.a.RemoteDevToolsAutoConfiguration : Listening for remote restart updates on /./spring-boot!~/restart
2022-04-02 05:12:18.760  INFO [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2022-04-02 05:12:18.765  INFO [ restartedMain] moForDockerizingSpringbootAppApplication : Started DemoForDockerizingSpringbootAppApplication in 0.684 seconds (JVM running for 202.41)
2022-04-02 05:12:19.021  INFO [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2022-04-02 05:12:19.021  INFO [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2022-04-02 05:12:19.022  INFO [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
```

```
curl http://localhost:8080/prodmgmtapp/api/product/list
[{"productId":1001,"name":"Apple iPhone 14XS","unitPrice":1850.95,"quantityInStock":48,"dateSupplied":"2022-01-15"}, {"productId":1002,"name":"Samsung Galaxy S22","unitPrice":1700.99,"quantityInStock":115,"dateSupplied":"2021-10-27"}, {"productId":1009,"name":"Google Pixel KiMax","unitPrice":1350.5,"quantityInStock":82,"dateSupplied":"2022-03-19"}]
```

```
curl http://localhost:8080/prodmgmtapp/api/product/list
[{"productId":1001,"name":"Apple iPhone 14XS","unitPrice":1850.95,"quantityInStock":48,"dateSupplied":"2022-01-15"}, {"productId":1002,"name":"Samsung Galaxy S22","unitPrice":1700.99,"quantityInStock":115,"dateSupplied":"2021-10-27"}, {"productId":1003,"name":"Google Pixel KiMax","unitPrice":1350.5,"quantityInStock":82,"dateSupplied":"2022-03-19"}]
```

CS489: **Software Development**