

CS489: **Applied Software Development**

Author: Prof. O. Kalu

MIU Computer Science - November 2023

Lesson 10

Introduction to Containers and
Containerization using Docker

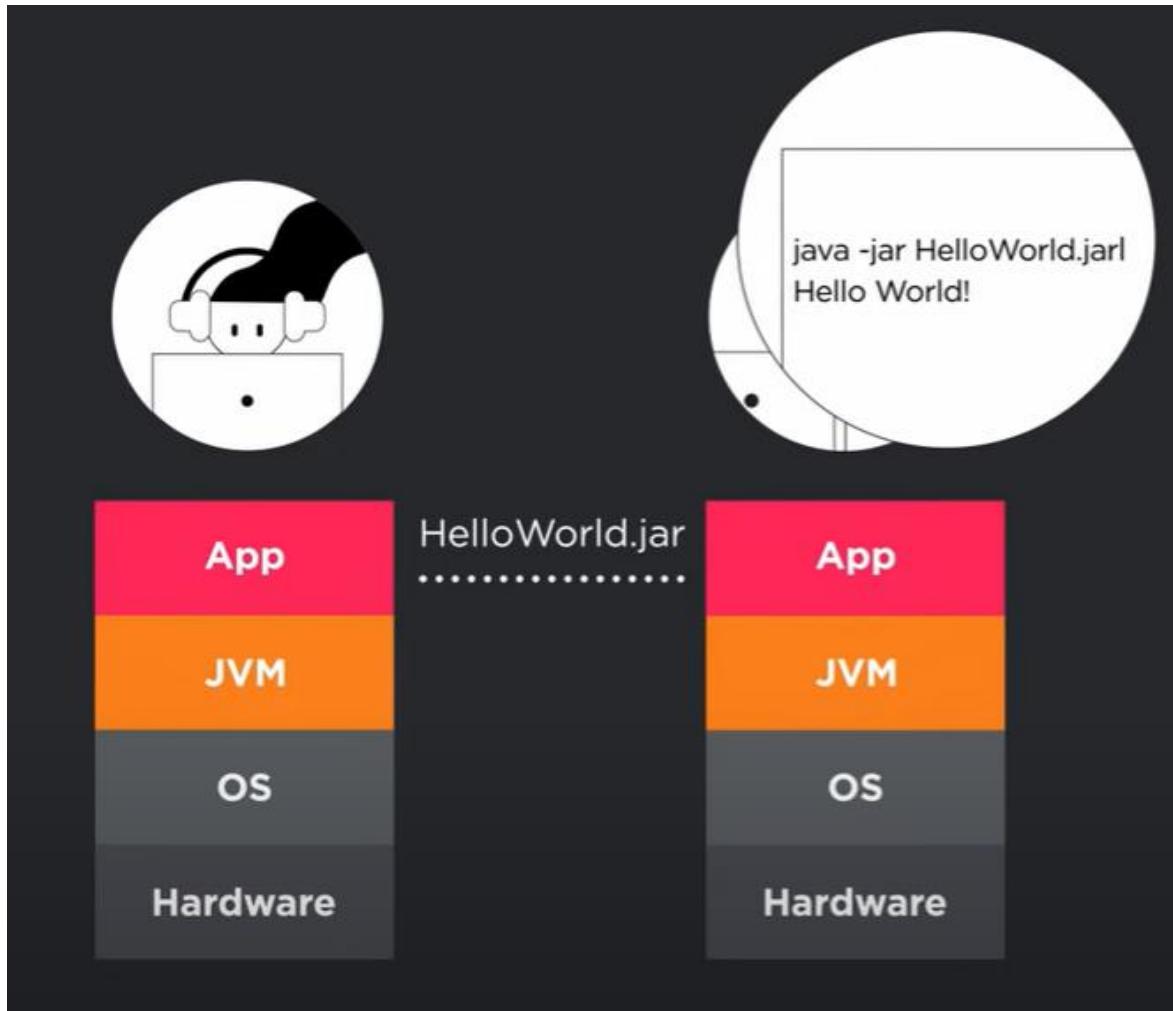
Wholeness

- 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.*

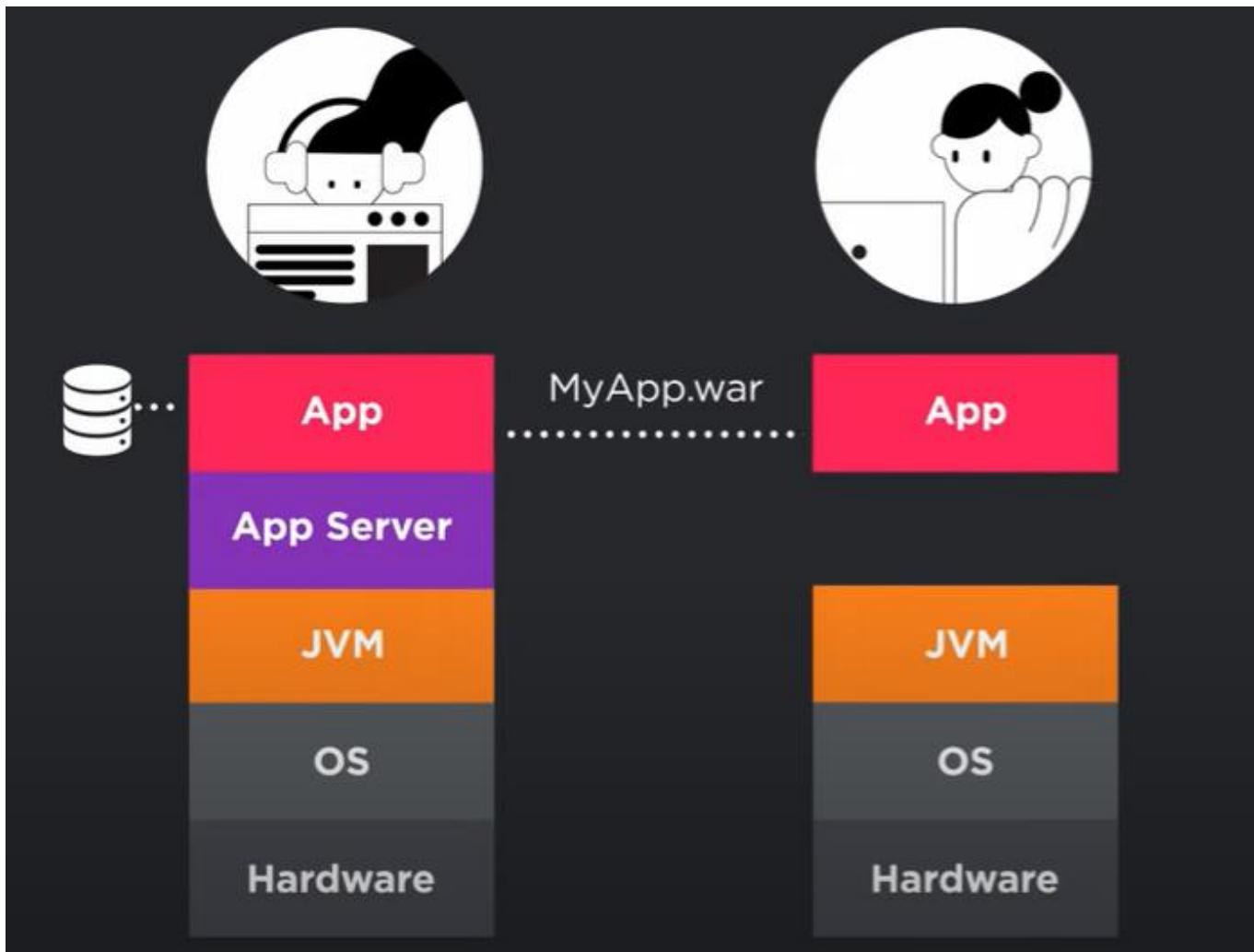
Why we need Containers

- 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)

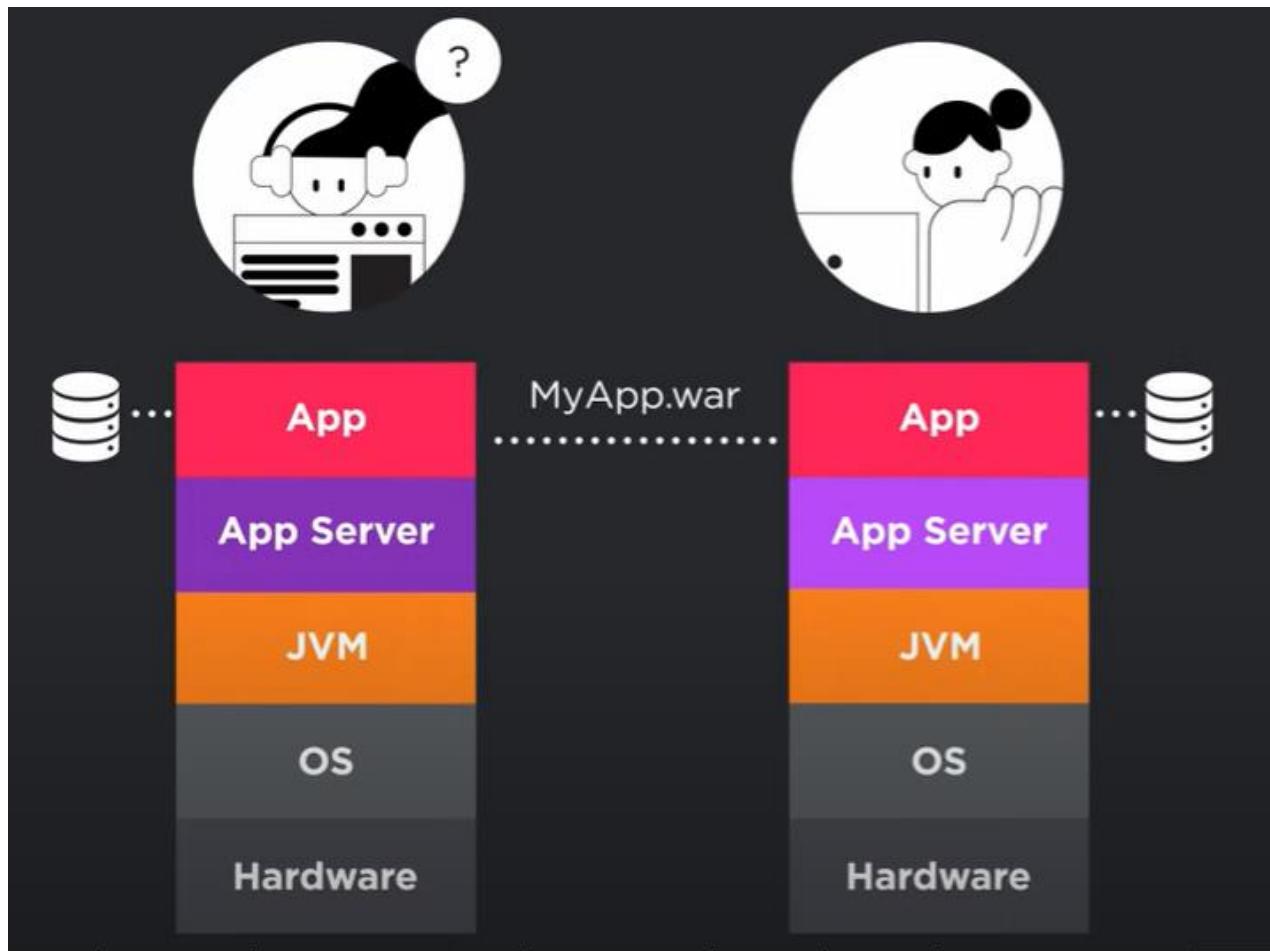
How to deploy/run app across Env



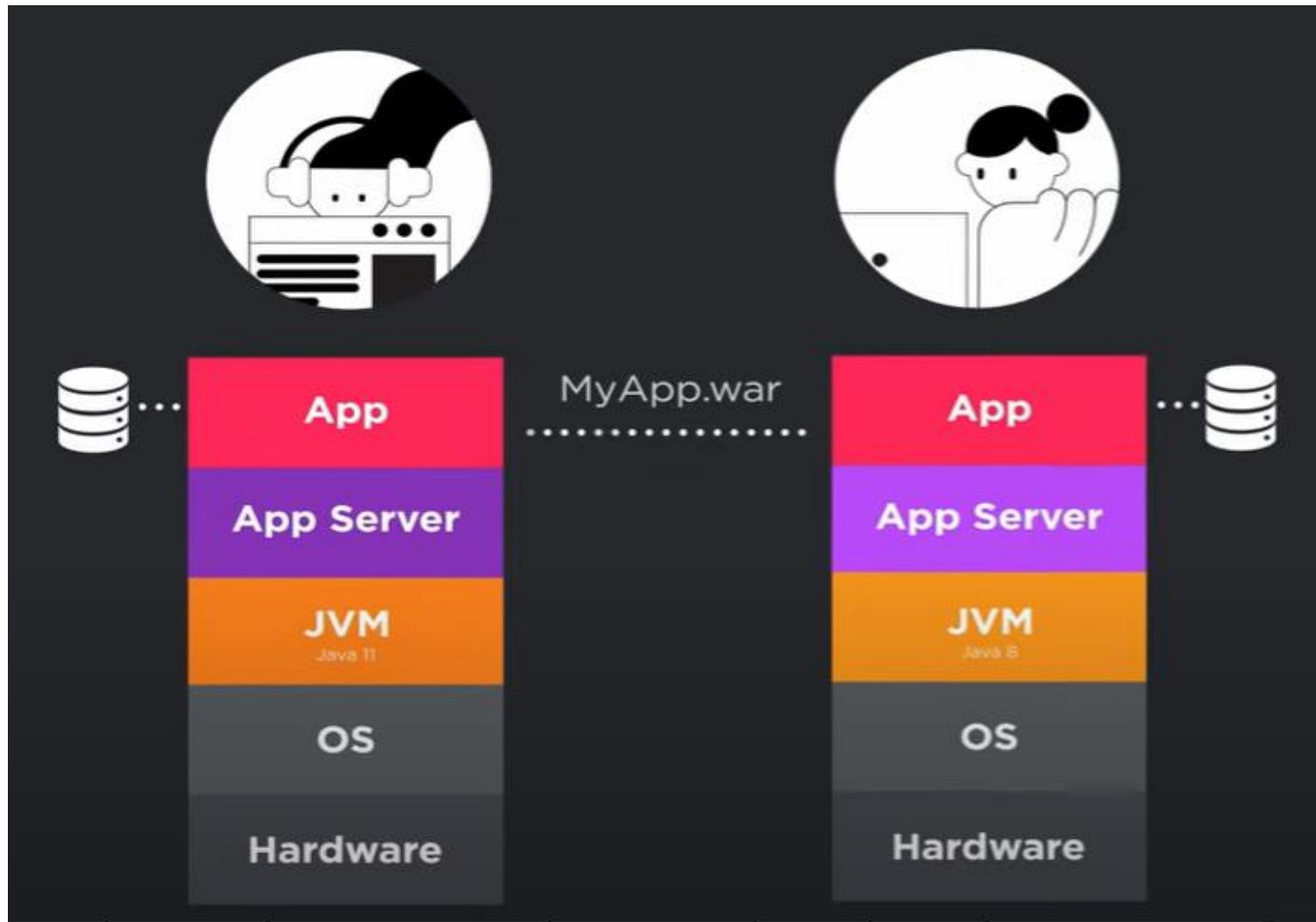
Deploy & run more complex App



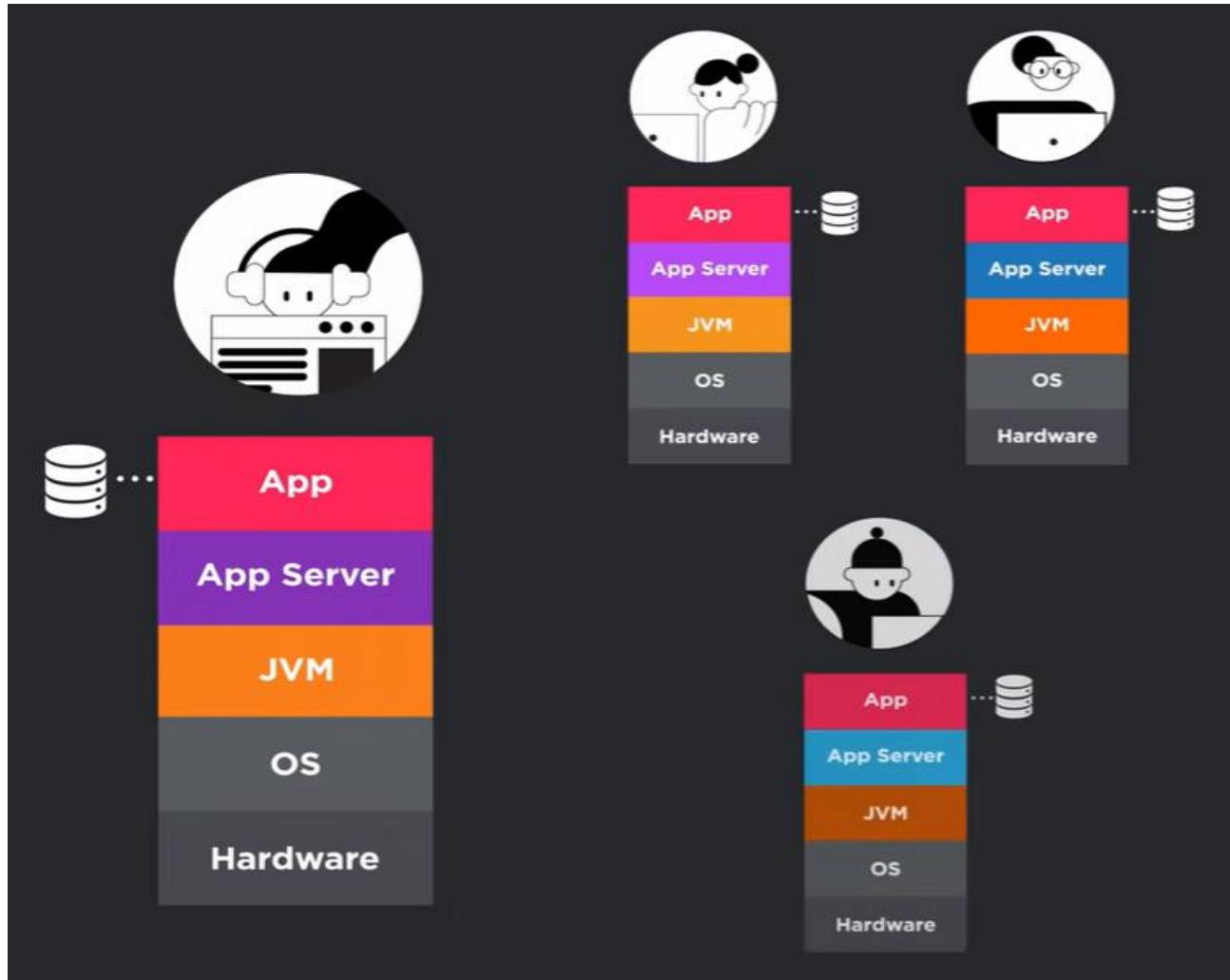
Deploy & run more complex app



Difference in the runtimes

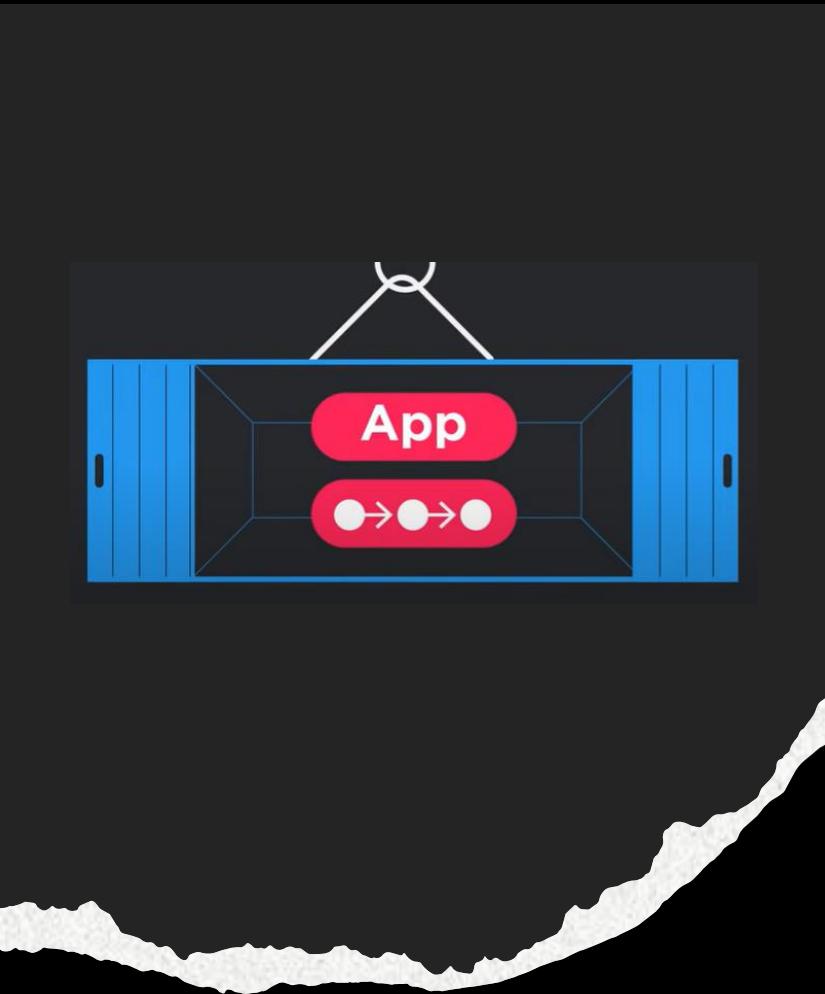


Need to deploy & run more instances



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



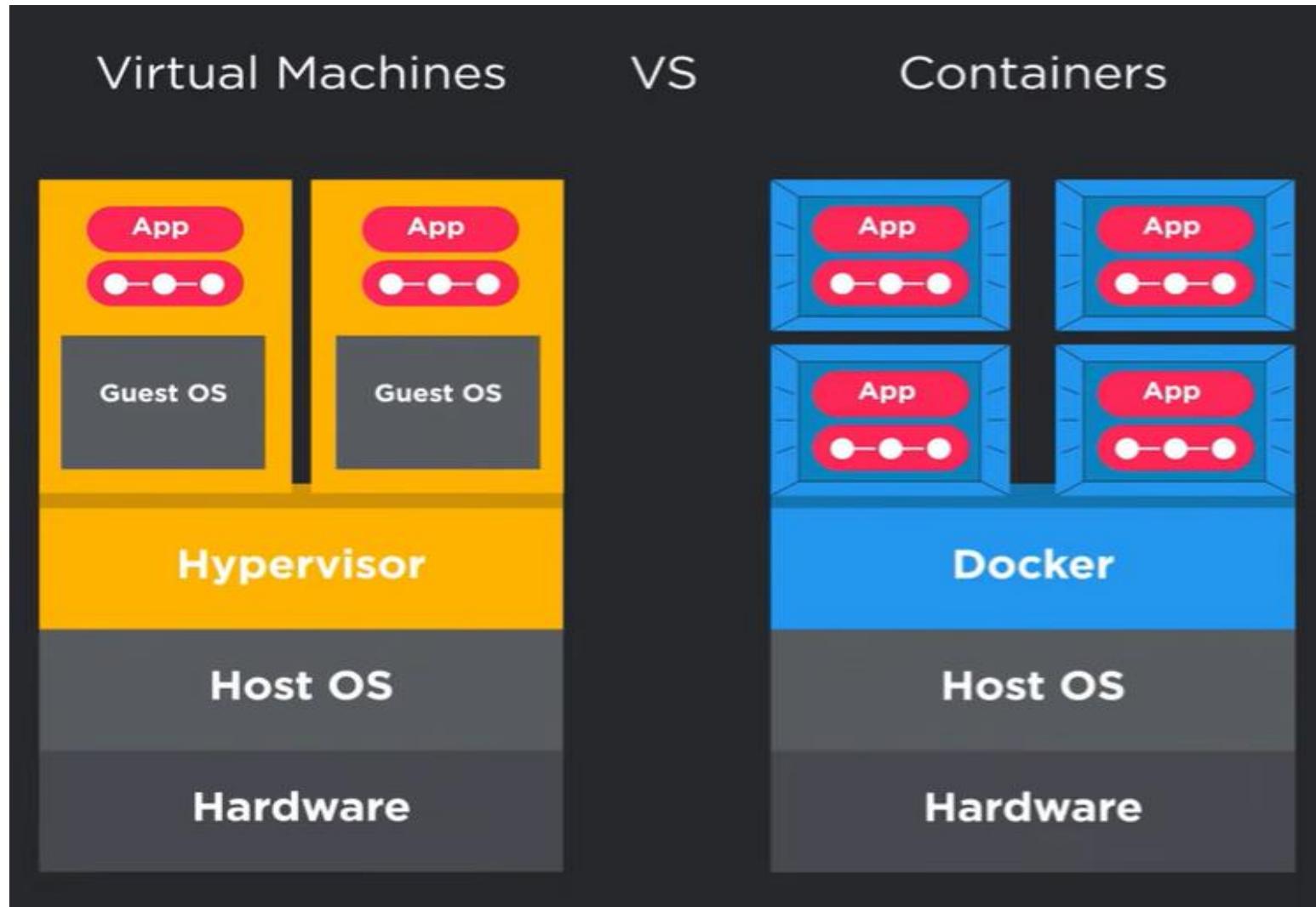
Containerization vs Virtualization (Containers versus Virtual Machines)

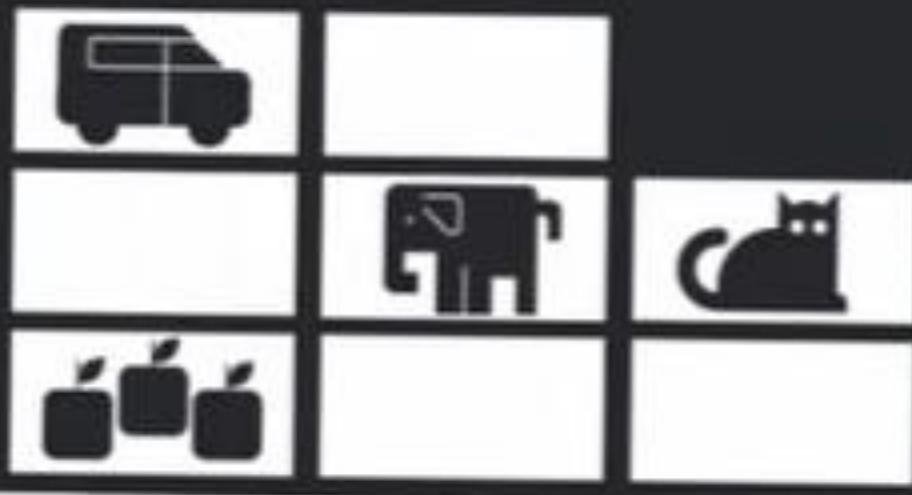
In contrast to Virtualization,
Containers are more portable and
more resource-efficient than virtual
machines (VMs).

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





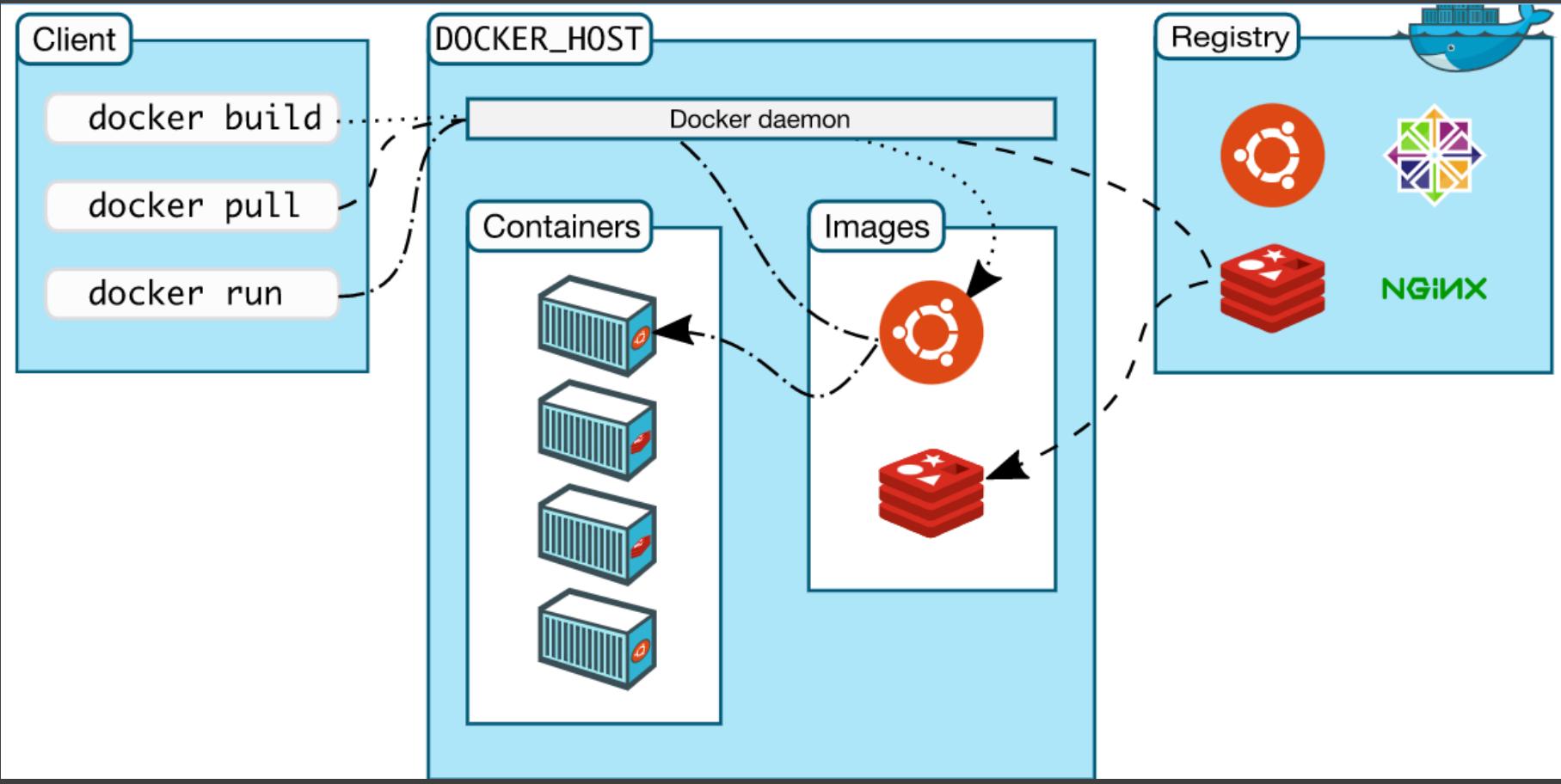
Origin of the
name,
“Container”

- Derived from the Shipping industry
- Shares quite similar characteristics

Container technologies

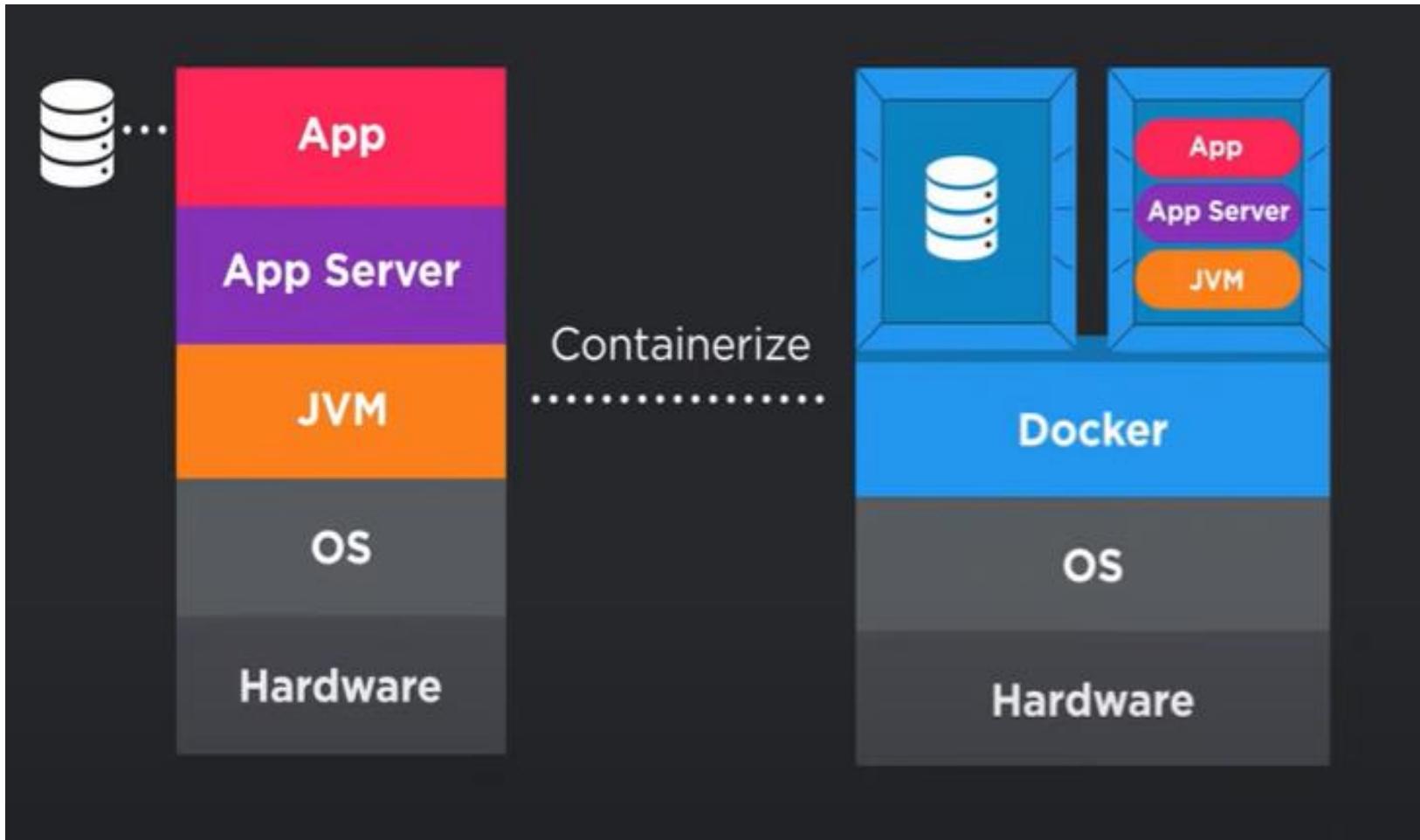
- Containers have become the de facto compute units of modern cloud-native applications.
- Docker is currently the premier/leading industry-standard containerization system
(<https://www.docker.com>)



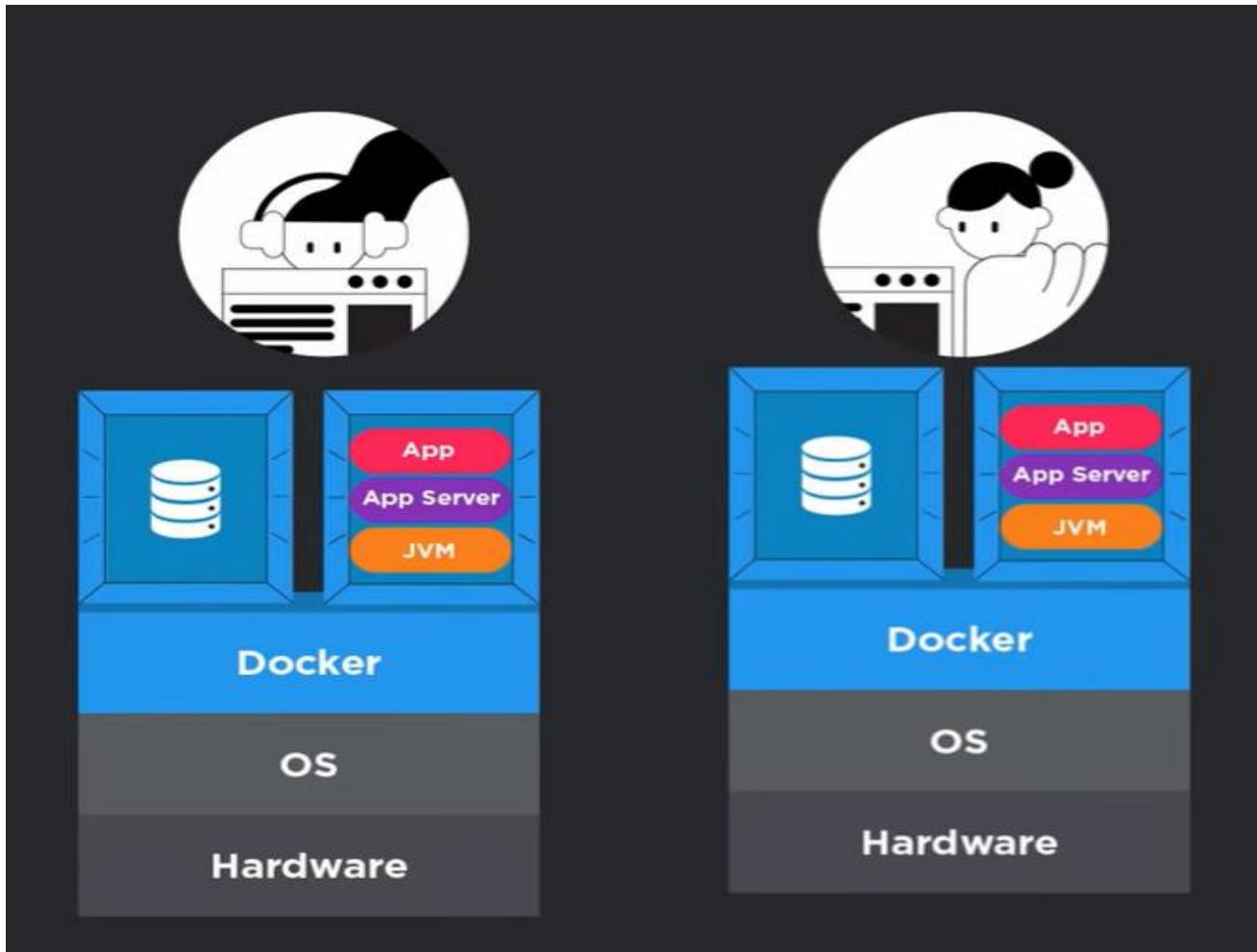


Architecture of Docker

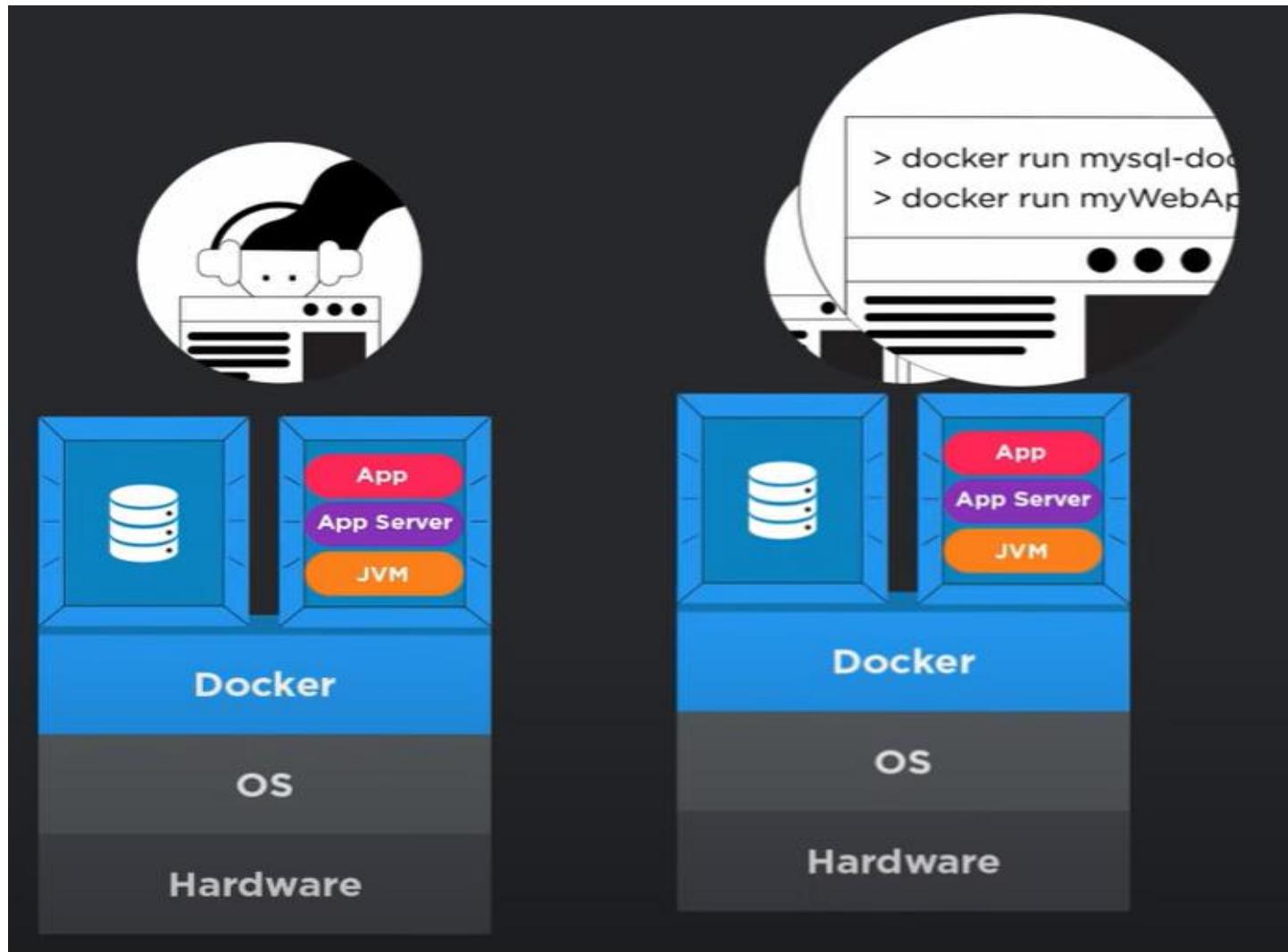
Containerize the app with docker



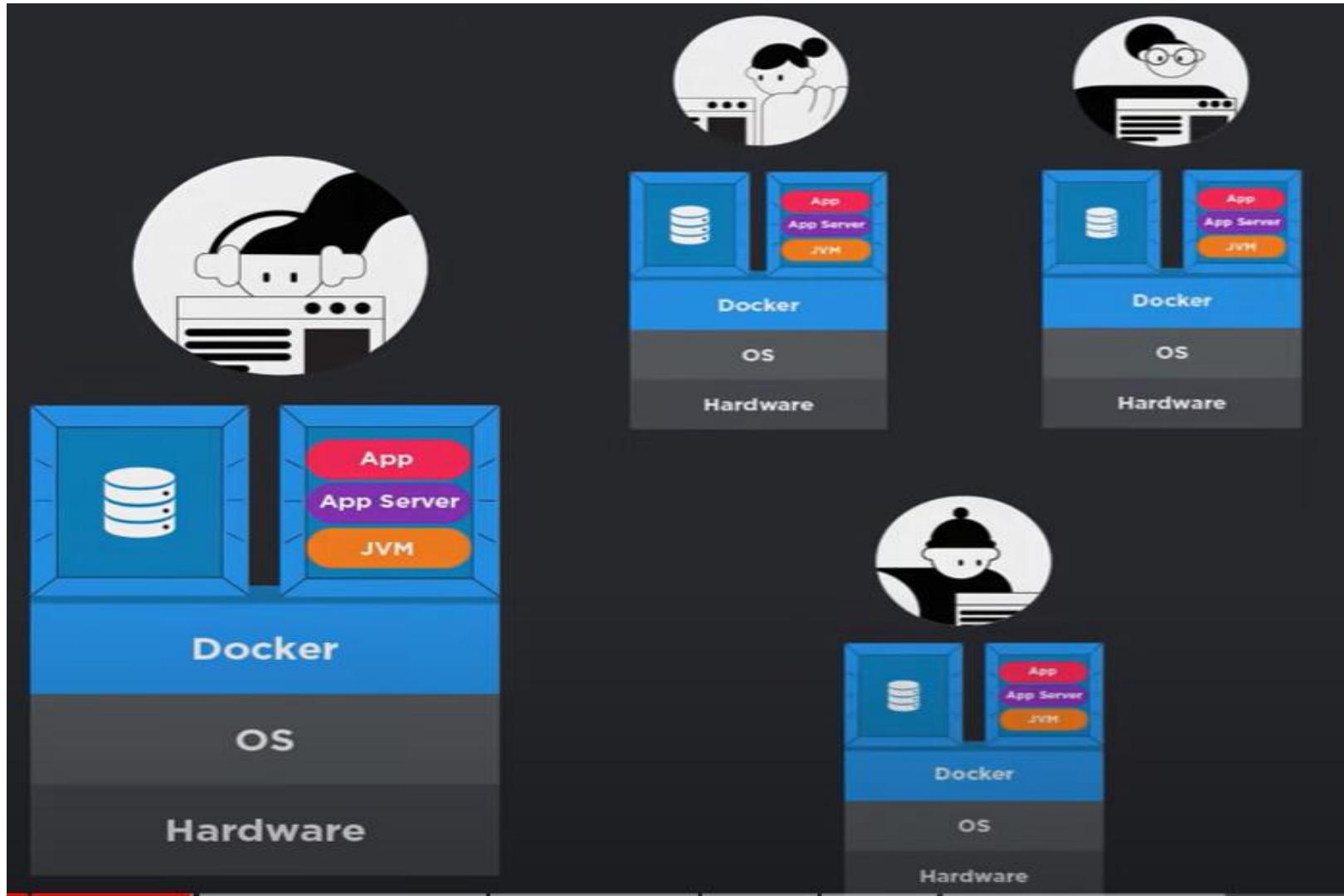
Replicate the configuration



Execute with docker run command



Containerized app – scales-out better

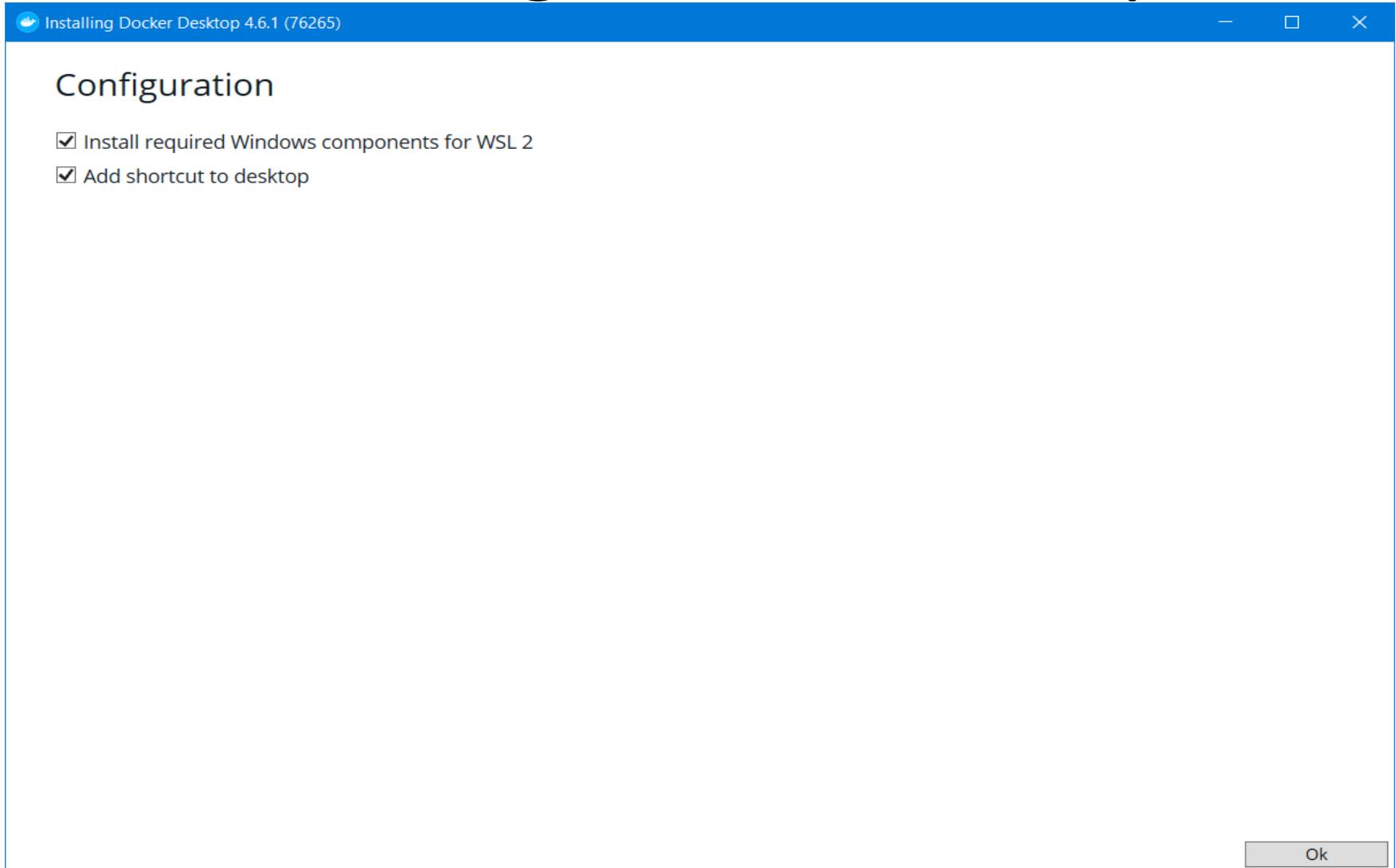


How to containerize software using Docker

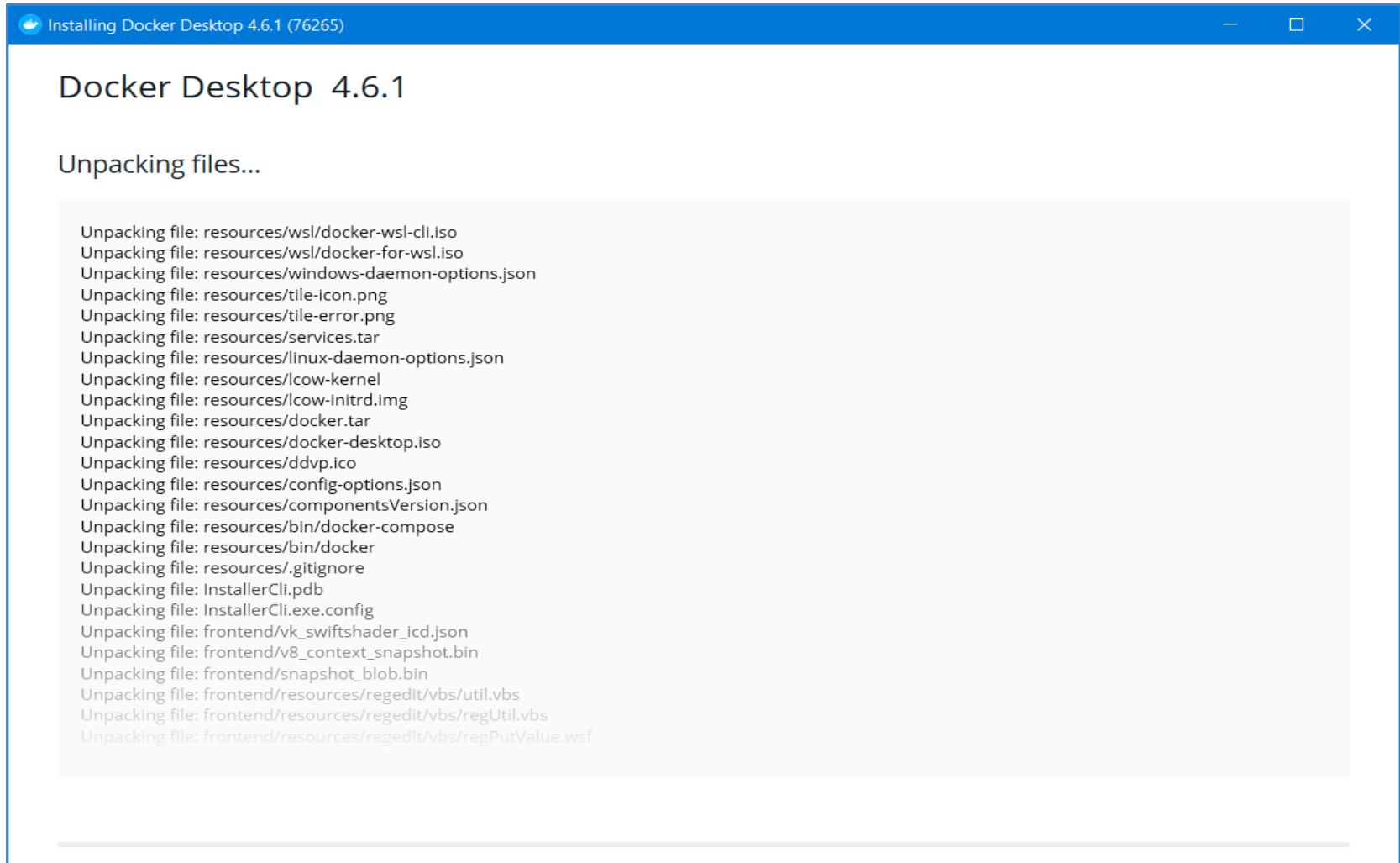
(<https://www.docker.com/get-started/>)



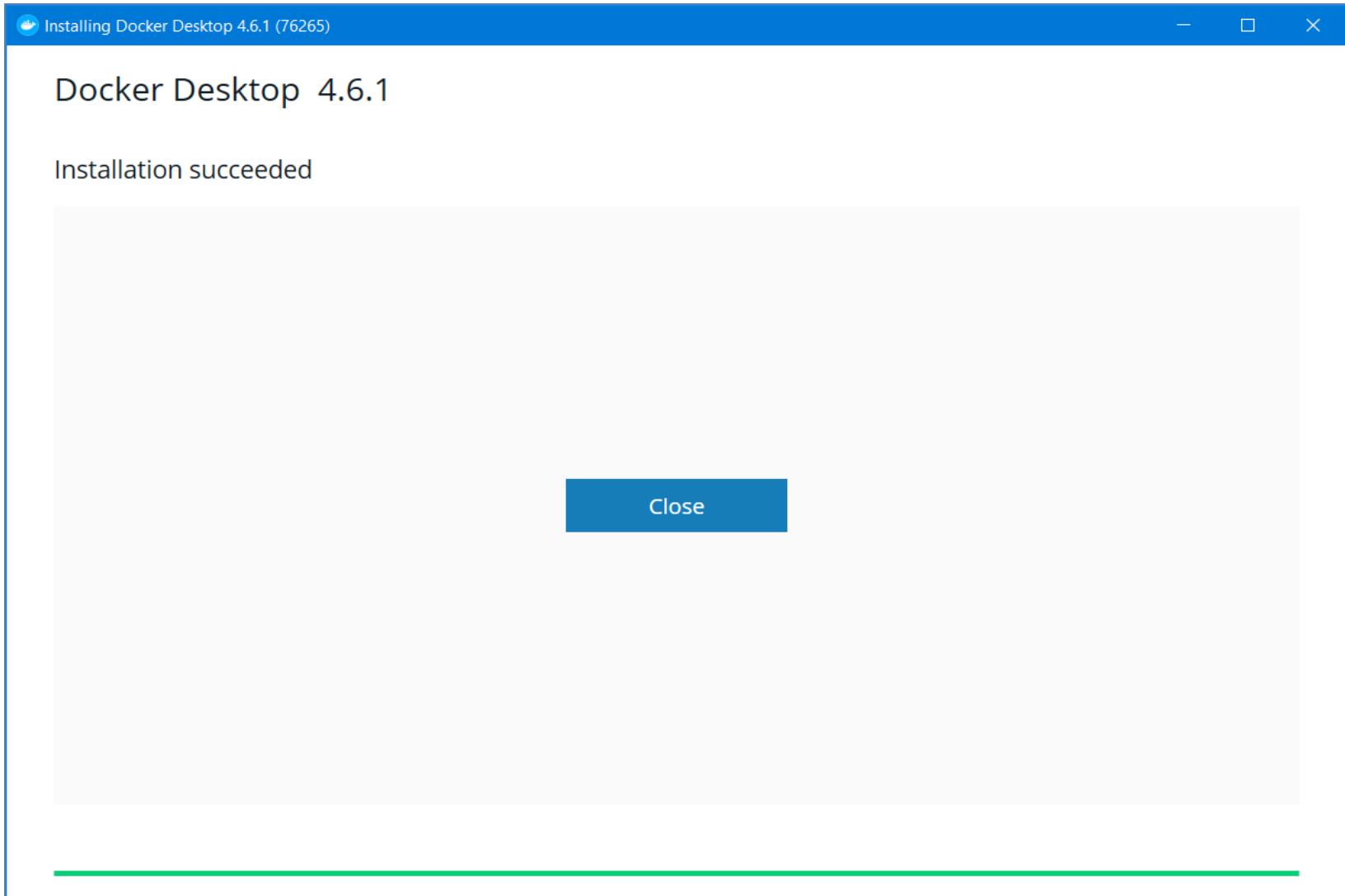
Installing Docker desktop



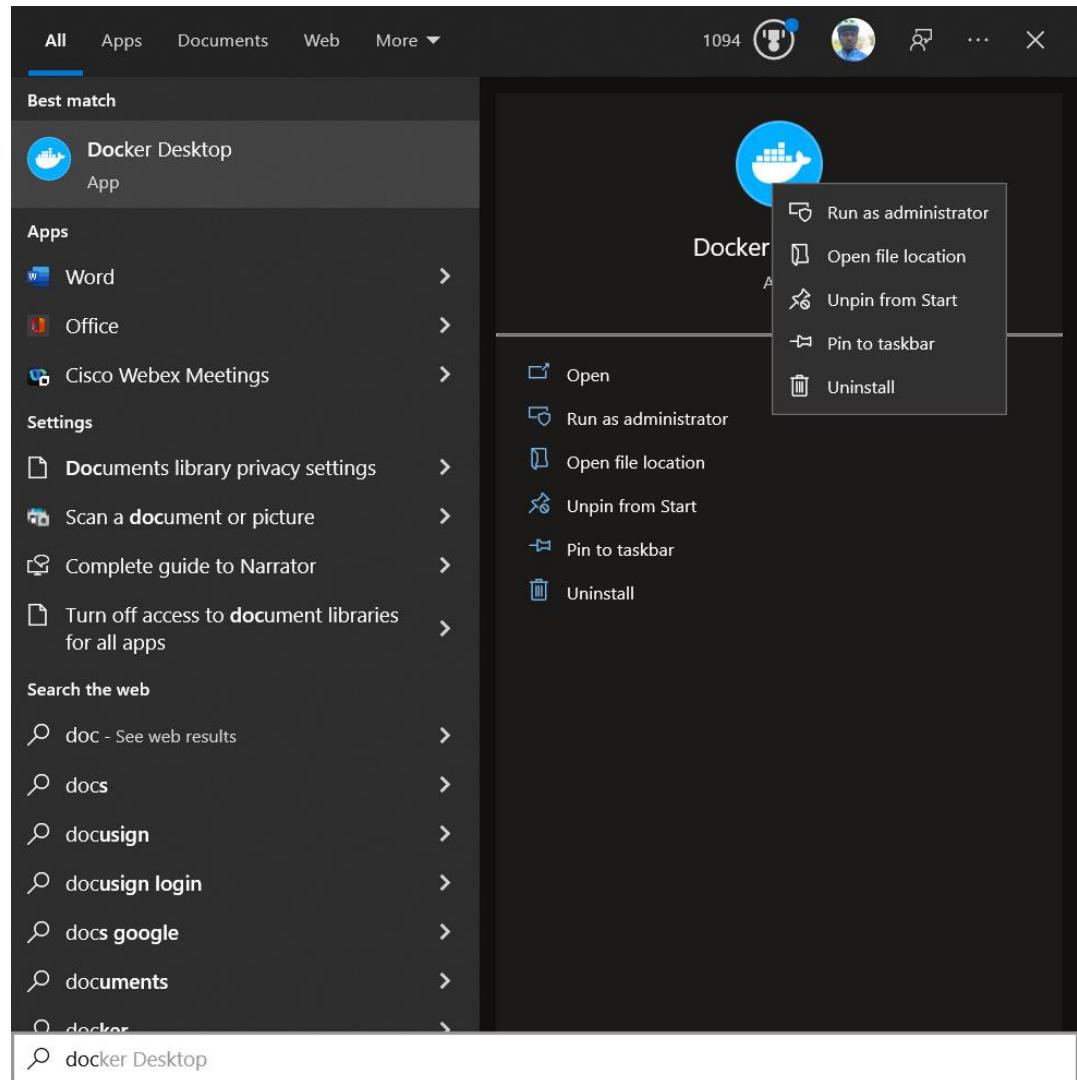
Installing Docker desktop



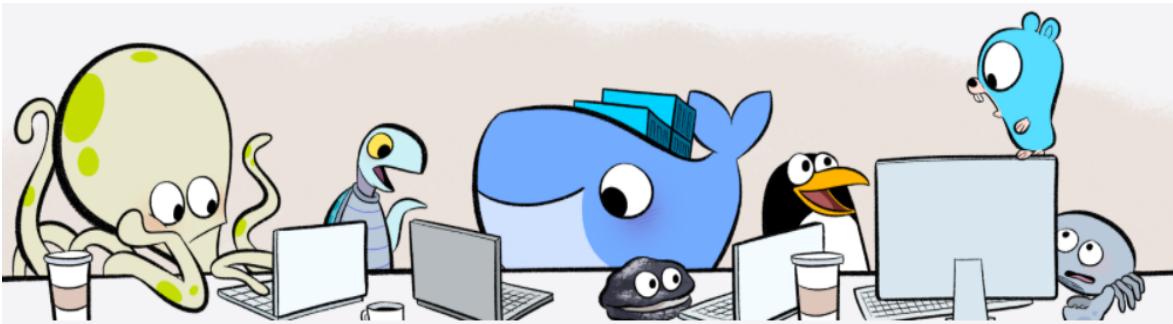
Docker desktop Installation complete



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

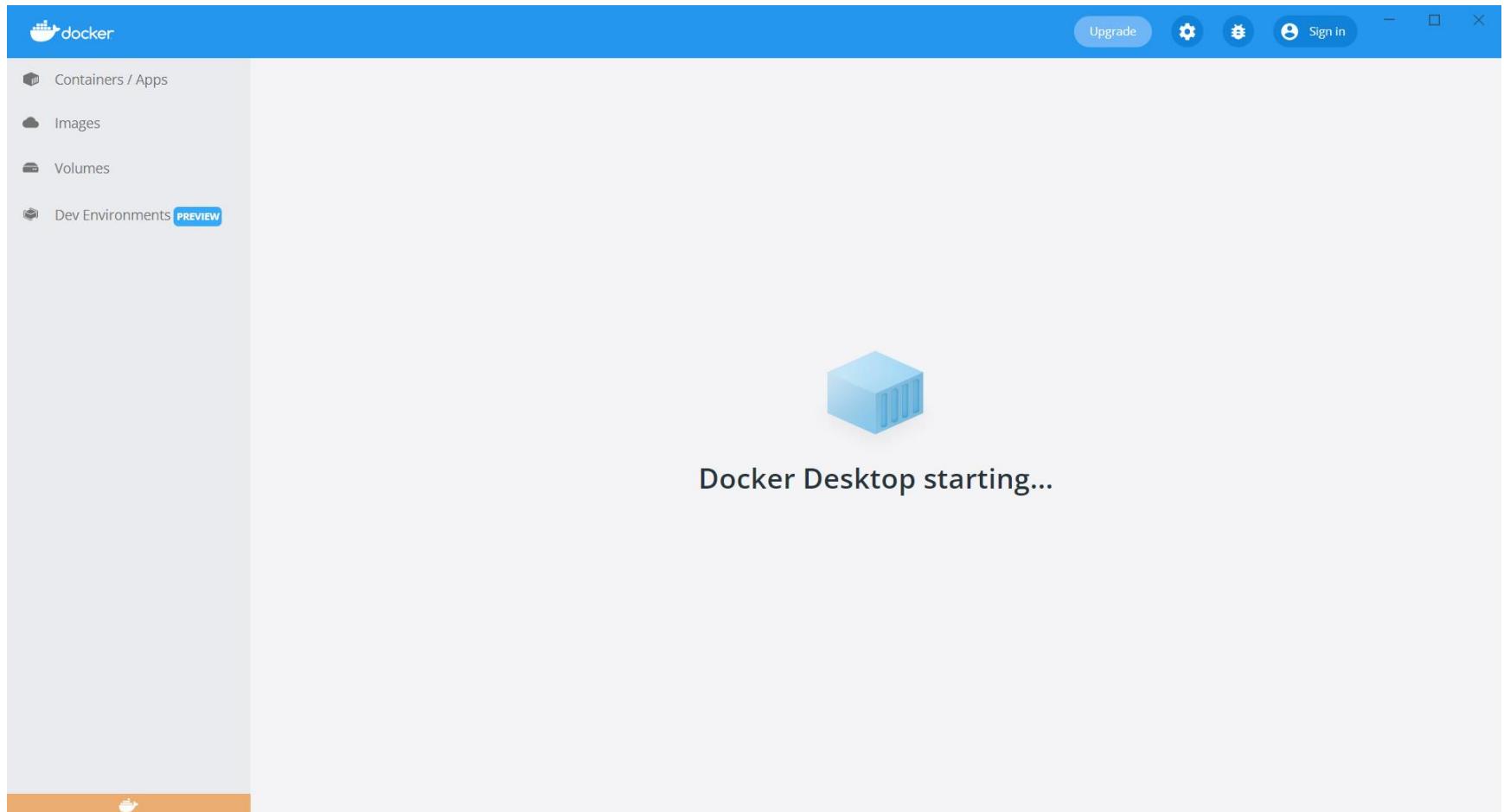
I accept the terms

[View Full Terms](#) (opens in new tab)

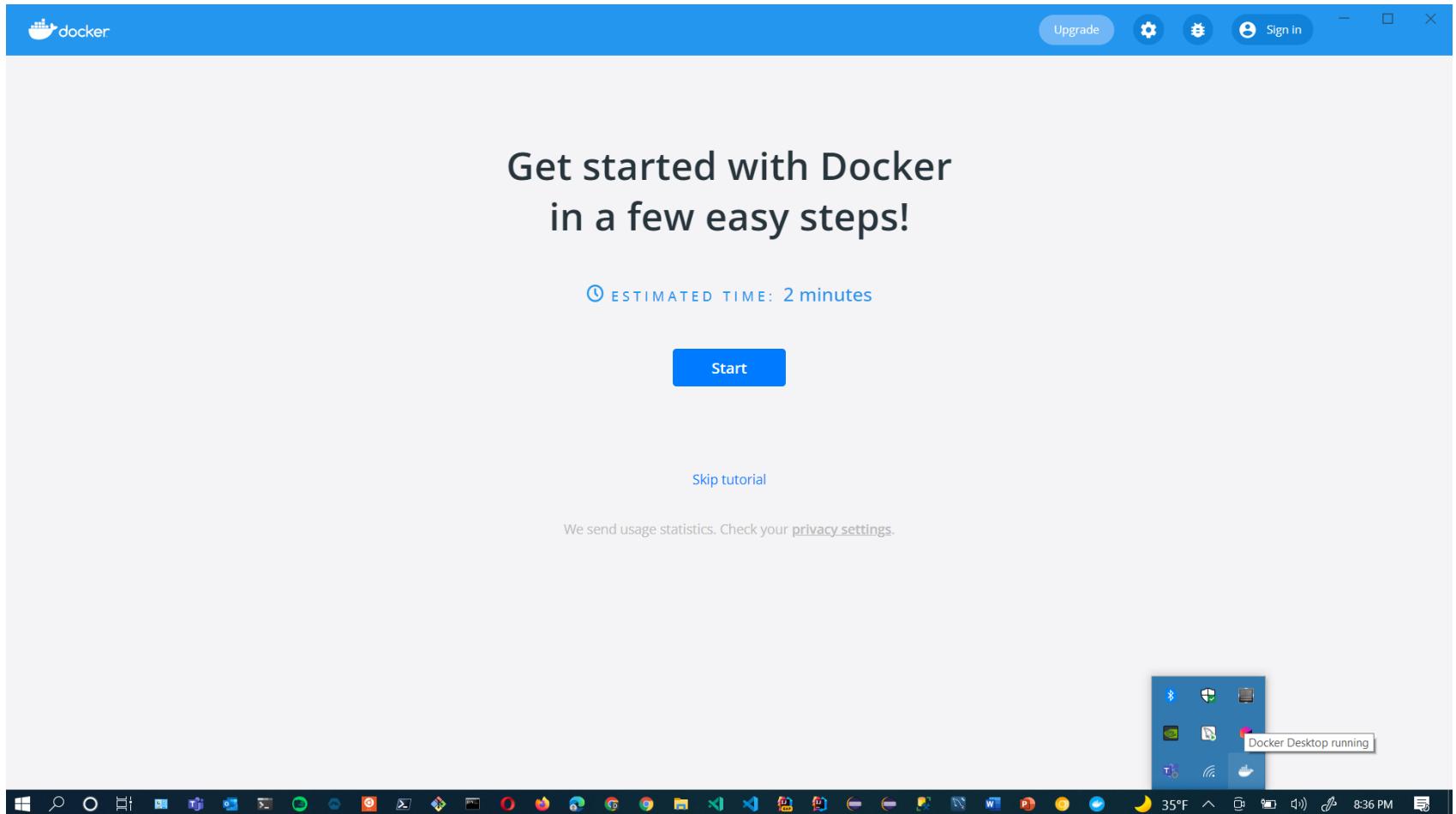
[Decline and Close Application](#)

[Accept](#)

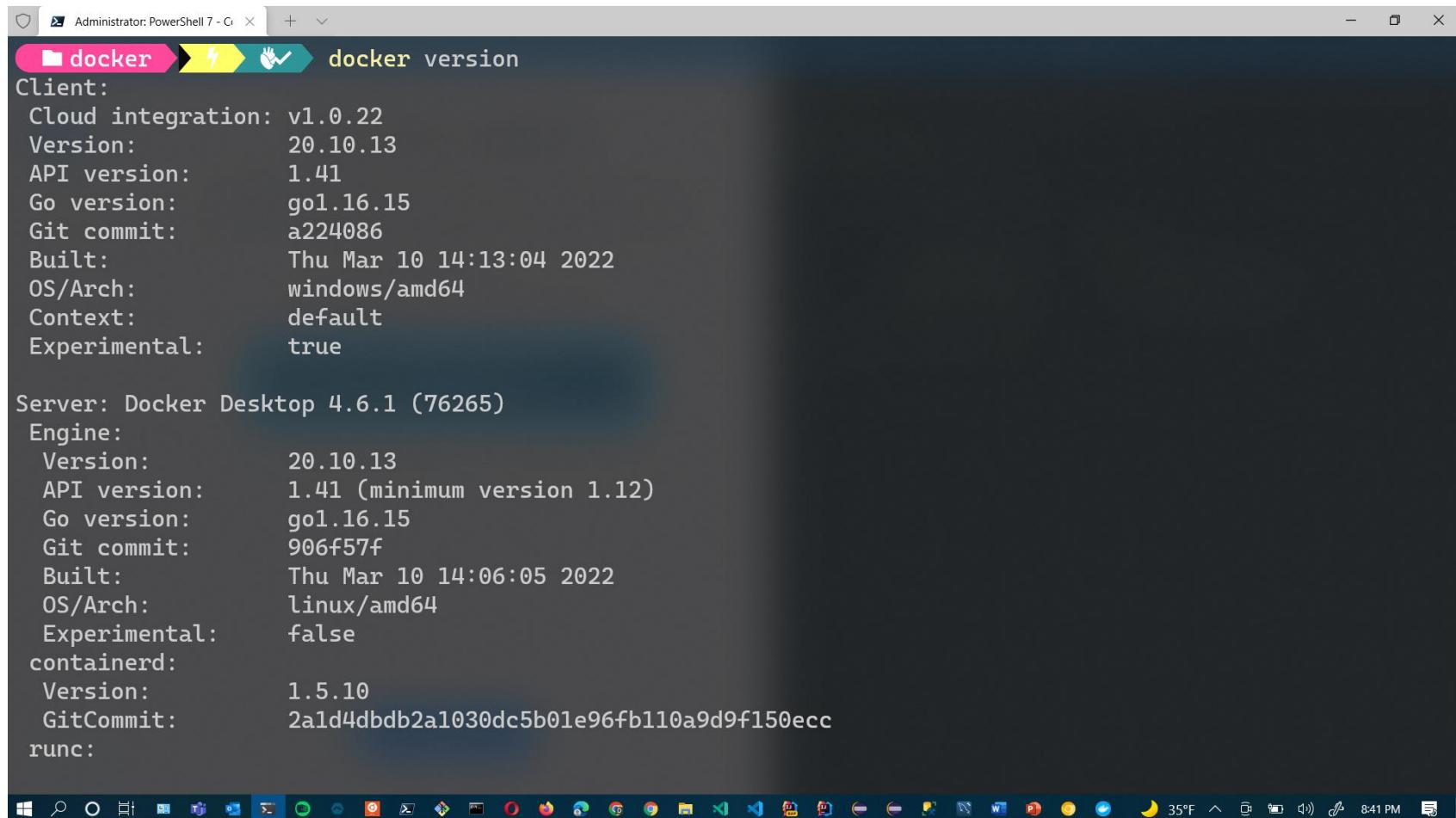
Starting...



Docker Desktop running



Verify running & version



A screenshot of a Windows PowerShell window titled "Administrator: PowerShell 7 - C:\Users\...". The window displays the output of the "docker version" command. The output is divided into Client and Server sections. The Client section shows details like Cloud integration version (v1.0.22), Version (20.10.13), API version (1.41), Go version (g01.16.15), Git commit (a224086), Built (Thu Mar 10 14:13:04 2022), OS/Arch (windows/amd64), Context (default), and Experimental (true). The Server section shows Docker Desktop version (4.6.1 (76265)), Engine details (Version 20.10.13, API version 1.41, Go version g01.16.15, Git commit 906f57f, Built Thu Mar 10 14:06:05 2022, OS/Arch linux/amd64, Experimental false), containerd (Version 1.5.10, GitCommit 2a1d4dbdb2a1030dc5b01e96fb110a9d9f150ecc), and runc.

```
Administrator: PowerShell 7 - C:\Users\...
docker version
Client:
Cloud integration: v1.0.22
Version: 20.10.13
API version: 1.41
Go version: g01.16.15
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: g01.16.15
  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’

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
```

Next Step

Skip tutorial

Getting Started

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  
3cede86c7d499: 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>
```

Next Step

Skip tutorial



1 Clone

First, clone a repository

2 Build

3 Run

4 Share

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.

[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

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

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

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

Directory: D:\oak\MyLearning\docker\repo

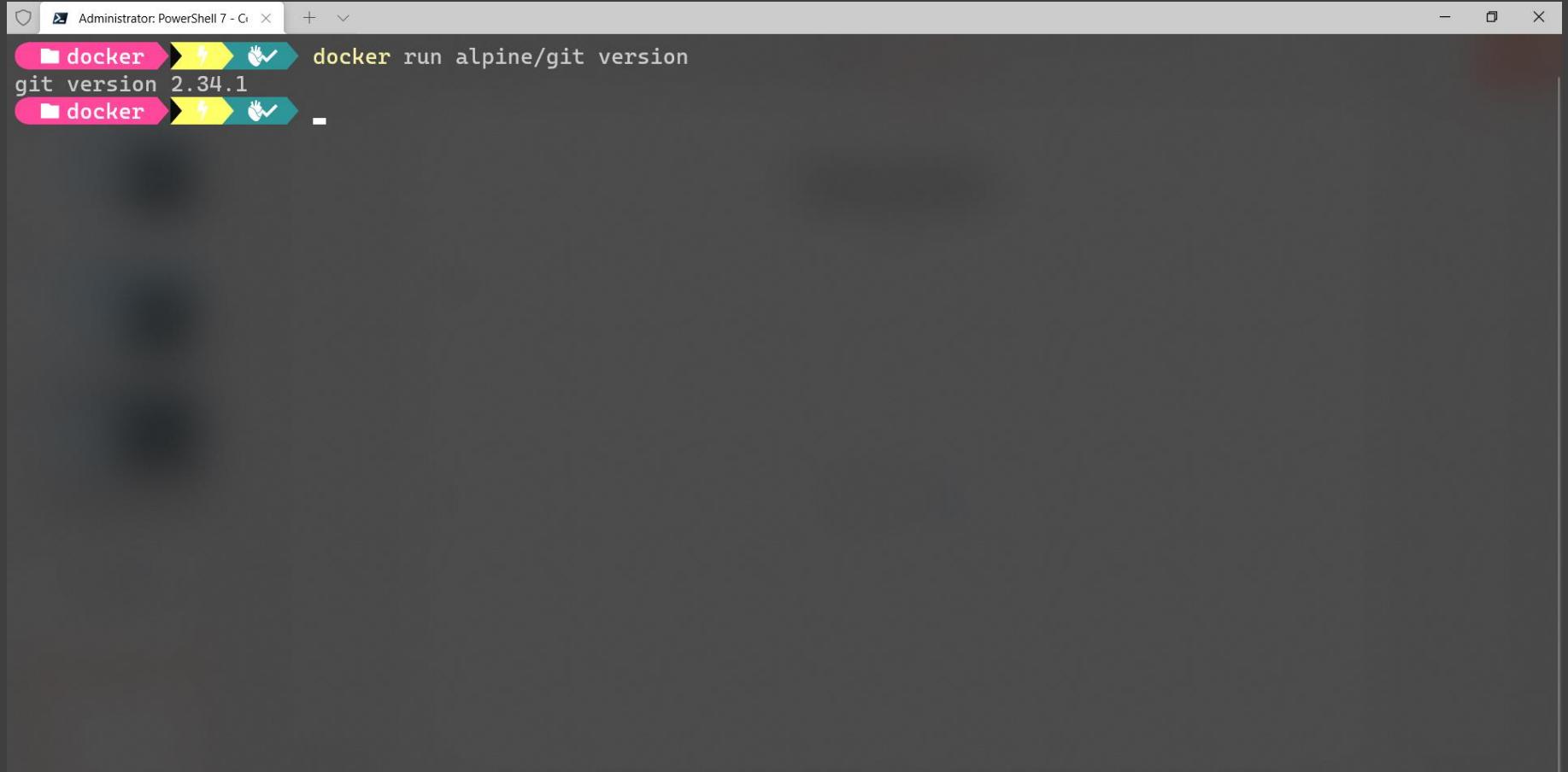
Mode                LastWriteTime         Length Name
----                -----        ----- 
d----       7/3/2020      6:29 PM          0 getting-started
d----      9/17/2019     3:40 PM          0 helloworld-express-nodejs
d----      9/17/2019     3:40 PM          0 nodejs-docker-webstorm

PS D:\oak\MyLearning\docker\repo> 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\repo> docker images --all
REPOSITORY          TAG           IMAGE ID      CREATED             SIZE
alpine/git          latest        b337a04161f7   6 days ago    38.2MB
PS D:\oak\MyLearning\docker\repo> 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   repo
```

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 command-line entries. The first entry is "docker run alpine/git version", which outputs "git version 2.34.1". The second entry is "docker", which has a trailing dash indicating it's still running or waiting for input.

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



The screenshot shows a code editor with XML configuration for a Maven plugin. The code is as follows:

```
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 code is syntax-highlighted, with the mainClass element highlighted in blue. A yellow selection bar is visible under the manifest and mainClass elements. The code editor has a vertical scrollbar on the right.

Demo

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

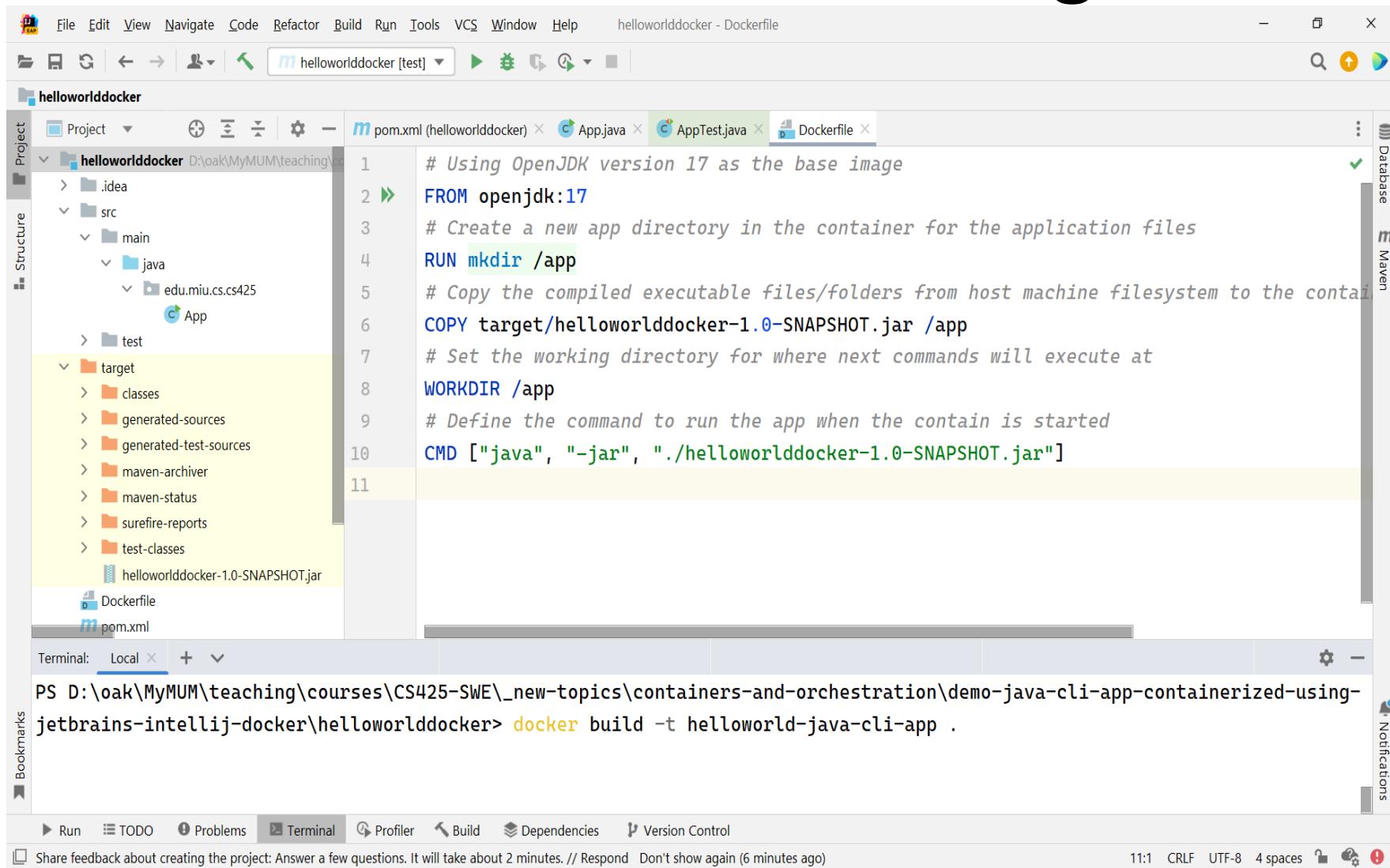
Demo

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:** helloworlddocker - Dockerfile.
- Toolbars:** Standard toolbar with icons for file operations.
- Project Structure:** Shows the project tree under 'helloworlddocker'. It includes a .idea folder, a src directory with main and test sub-directories, and a target directory containing classes, generated-sources, generated-test-sources, maven-archiver, maven-status, surefire-reports, test-classes, and a helloworlddocker-1.0-SNAPSHOT.jar file. The Dockerfile and pom.xml files are also listed in the project tree.
- Code Editor:** The Dockerfile tab is active, displaying the following content:

```
1 # Using OpenJDK version 17 as the base image
2 FROM openjdk:17
3 # Create a new app directory in the container for the application files
4 RUN mkdir /app
5 # Copy the compiled executable files/folders from host machine filesystem to the container
6 COPY target/helloworlddocker-1.0-SNAPSHOT.jar /app
7 # Set the working directory for where next commands will execute at
8 WORKDIR /app
9 # Define the command to run the app when the container is started
10 CMD ["java", "-jar", "./helloworlddocker-1.0-SNAPSHOT.jar"]
11
```
- Terminal:** Local terminal window showing the command: `Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them`.
- Bottom Navigation:** Run, TODO, Problems, Terminal, Profiler, Build, Dependencies, Version Control.
- Status Bar:** Share feedback about creating the project: Answer a few questions. It will take about 2 minutes. // Respond Don't show again (5 minutes ago).
- Right Sidebar:** Database and Maven tabs.

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..."), "test", and "target" (with "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, displaying the following 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 `pom.xml` file in the Project structure is also visible, indicating a Maven project. The terminal at the bottom shows the command `java -jar ./target/helloworlddocker-1.0-SNAPSHOT.jar` being run from the local terminal.

Build another docker image

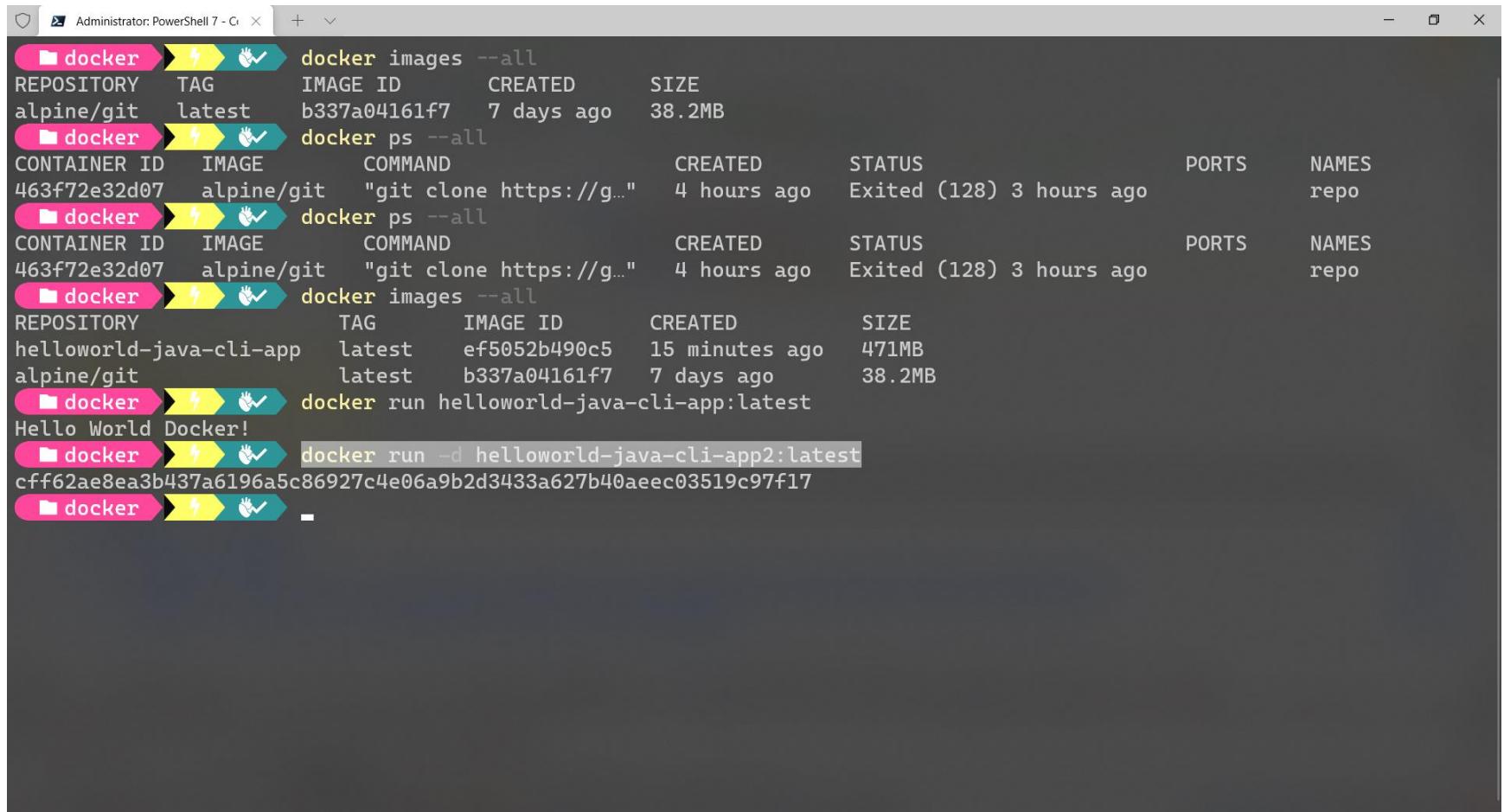
The screenshot shows the IntelliJ IDEA interface with a Java project named "helloworlddocker". The code editor displays the `App.java` file, which contains the following code:

```
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 `pom.xml` file in the project structure is also visible, indicating a Maven project. 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-app2 .
```

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



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

> 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

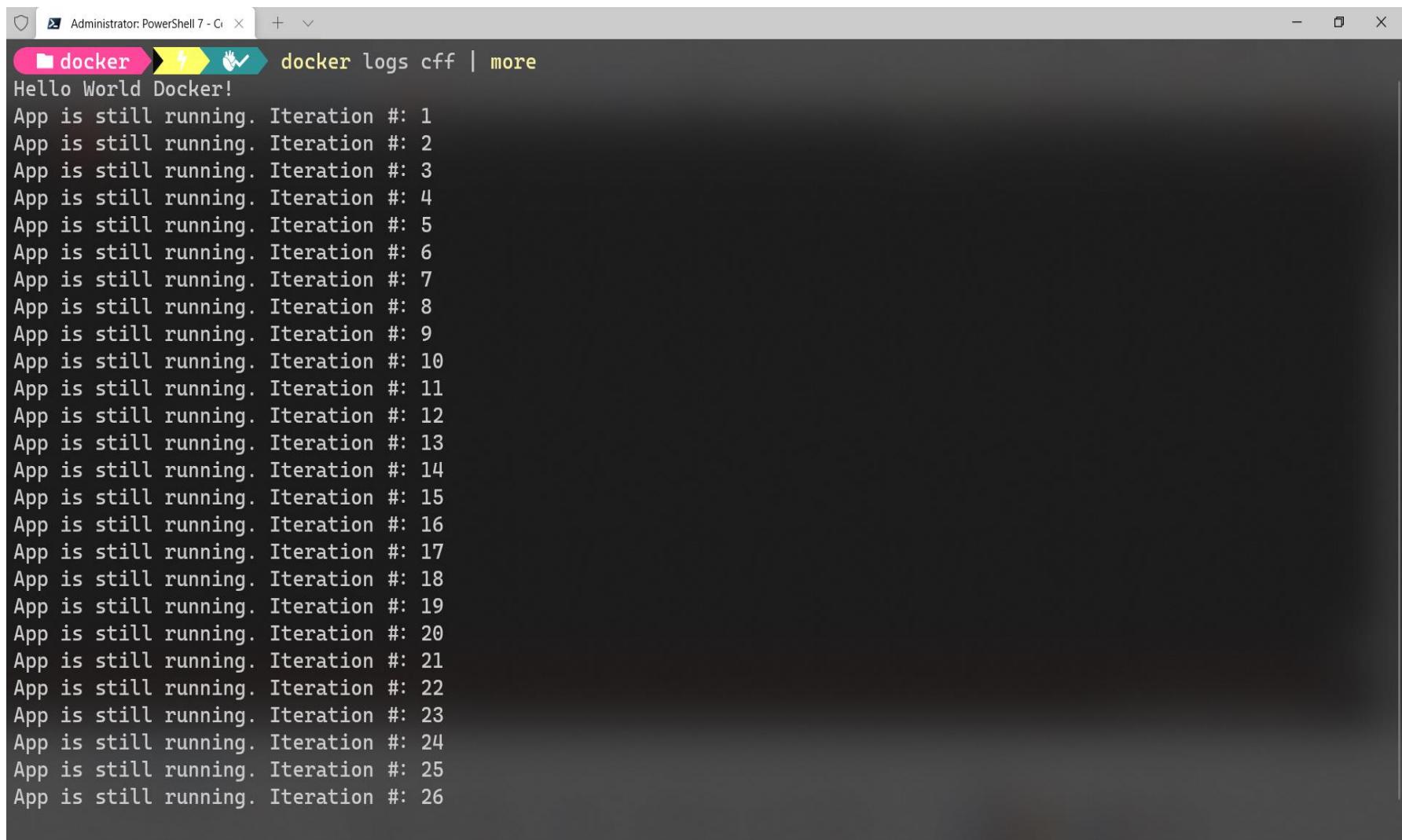
> docker ps --all
CONTAINER ID      IMAGE      COMMAND      CREATED      STATUS      PORTS      NAMES
463f72e32d07    alpine/git  "git clone https://g..."  4 hours ago  Exited (128)  3 hours ago
repo

> 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

> docker run helloworld-java-cli-app:latest
Hello World Docker!
> docker run -d helloworld-java-cli-app2:latest
cff62ae8ea3b437a6196a5c86927c4e06a9b2d3433a627b40aeeec03519c97f17

> docker
```

Display the container's log

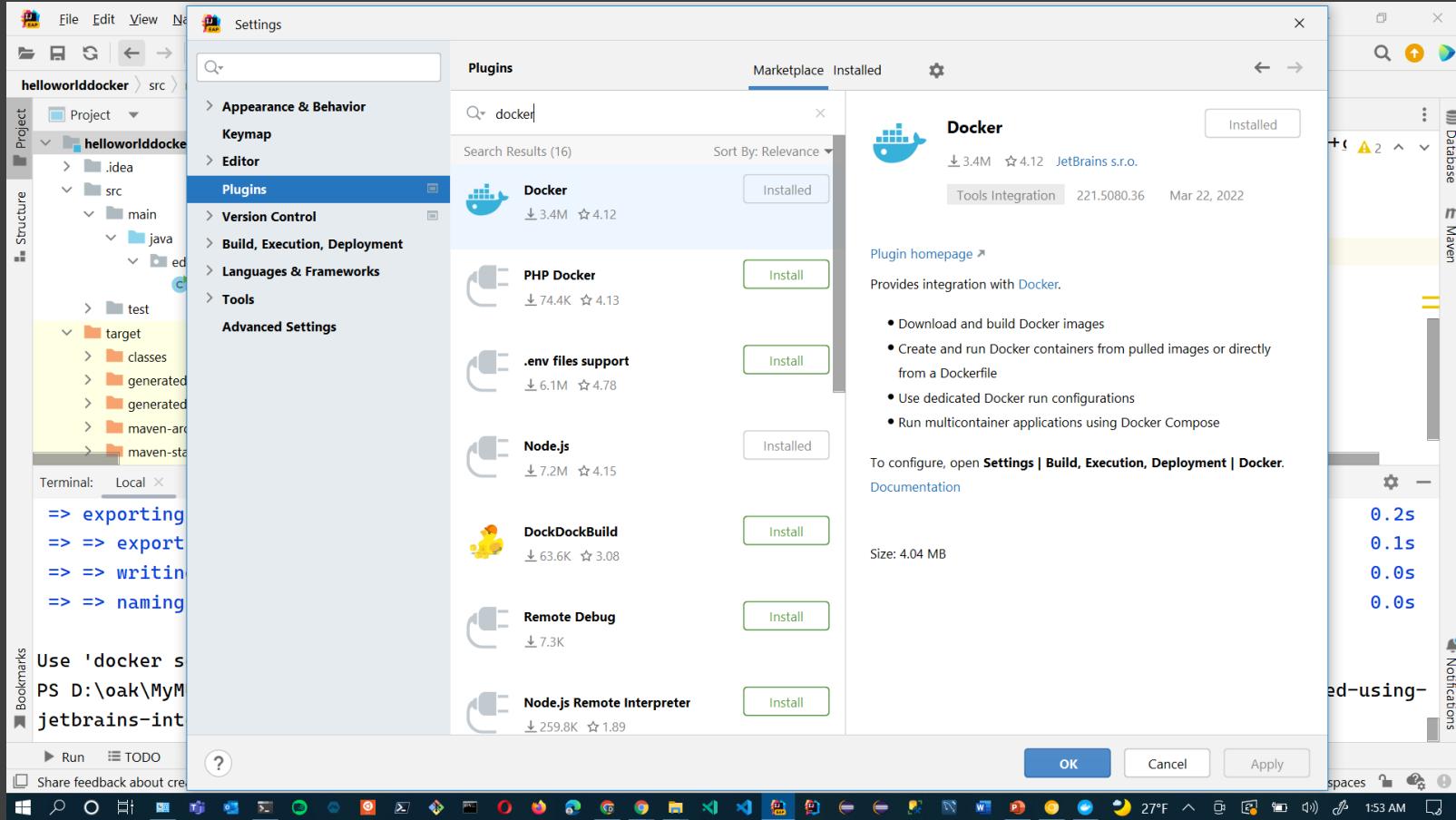


A screenshot of a Windows PowerShell window titled "Administrator: PowerShell 7 - C:\Users\...". The command entered is "docker logs cff | more". The output shows the application "Hello World Docker!" printing "App is still running. Iteration #: 1" through "App is still running. Iteration #: 26".

```
Administrator: PowerShell 7 - C:\Users\...\Desktop> 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
```

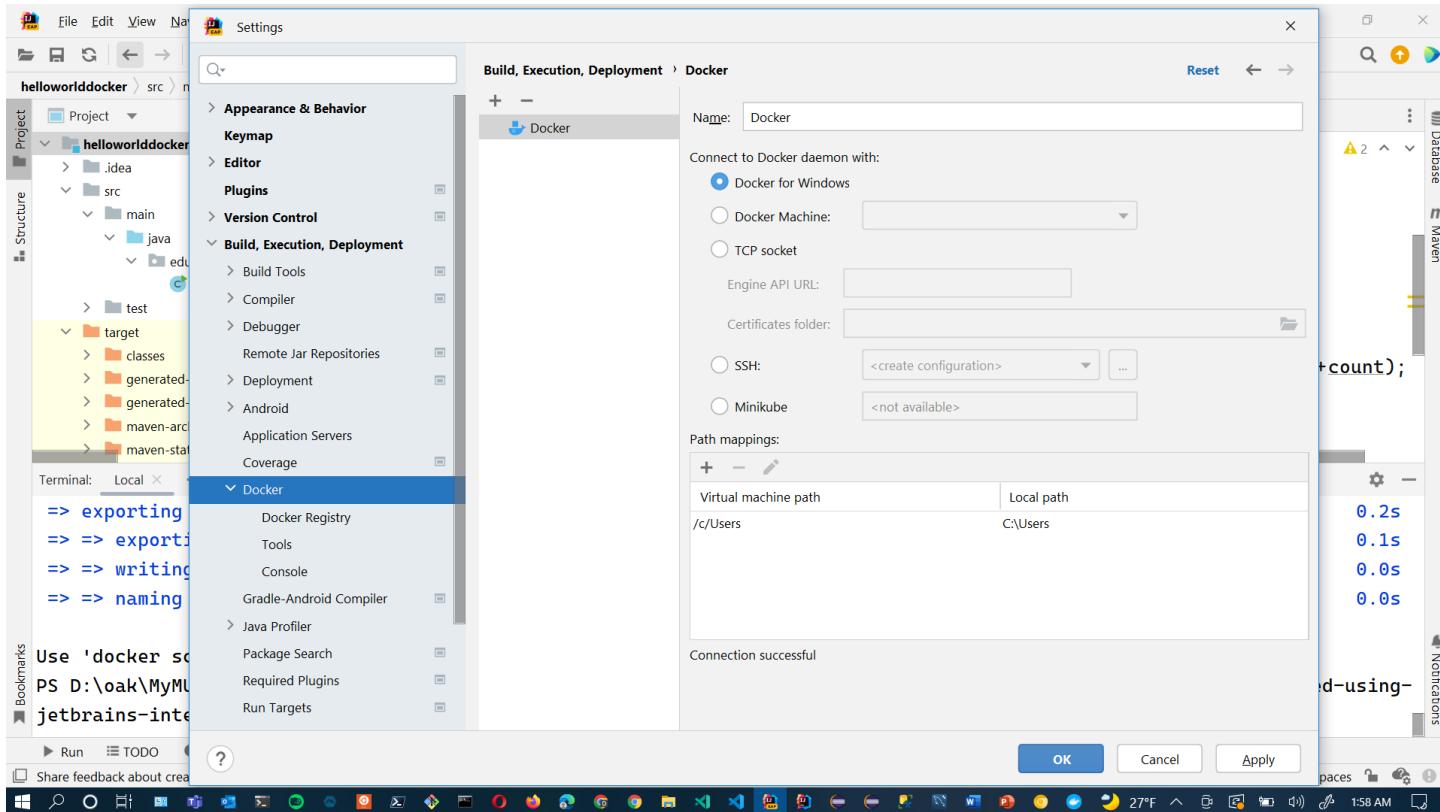
Check the status and stop the running container

```
Administrator: PowerShell 7 - C:\ + ▾ - ▾ ×
docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
mes
cff62ae8ea3b helloworld-java-cli-app2:latest "java -jar ./helloworld.jar" 9 minutes ago Up 9 minutes
fe
stive_hellman
docker container stop cff
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 ./helloworld.jar" 10 minutes ago Exited (143) 13 seconds ago
o festive_hellman
f55eadfbfa8 helloworld-java-cli-app:latest "java -jar ./helloworld.jar" 41 minutes ago Exited (0) 41 minutes ago
elastic_zhukovsky
463f72e32d07 alpine/git
repo
git clone https://github.com/docker/hello-world.git
-
```



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 a Java project named "helloworlddocker". The code editor displays the `App.java` file, which contains the following code:

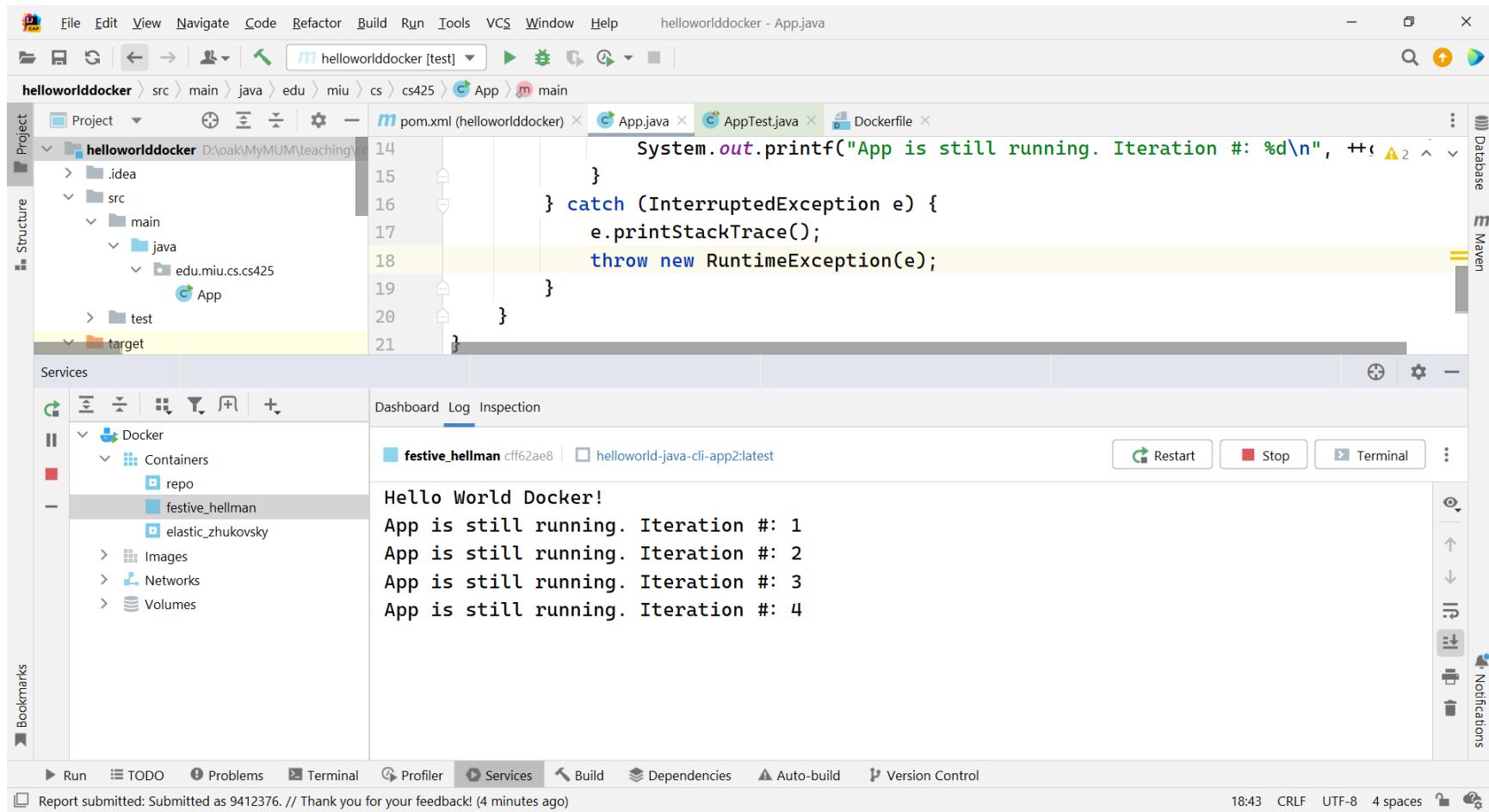
```
public class App {  
    public static void main( String[] args ) {  
        System.out.println( "Hello World Docker!" );  
        int count = 0;  
        try {  
            while(true) {  
                Thread.sleep( millis: 2000 );  
                System.out.printf("App is still running.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

The "Services" tool window is open, showing a list of services. The "Docker" service is selected, indicated by a blue background. A tooltip says "Double-click on the server node to connect".

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



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

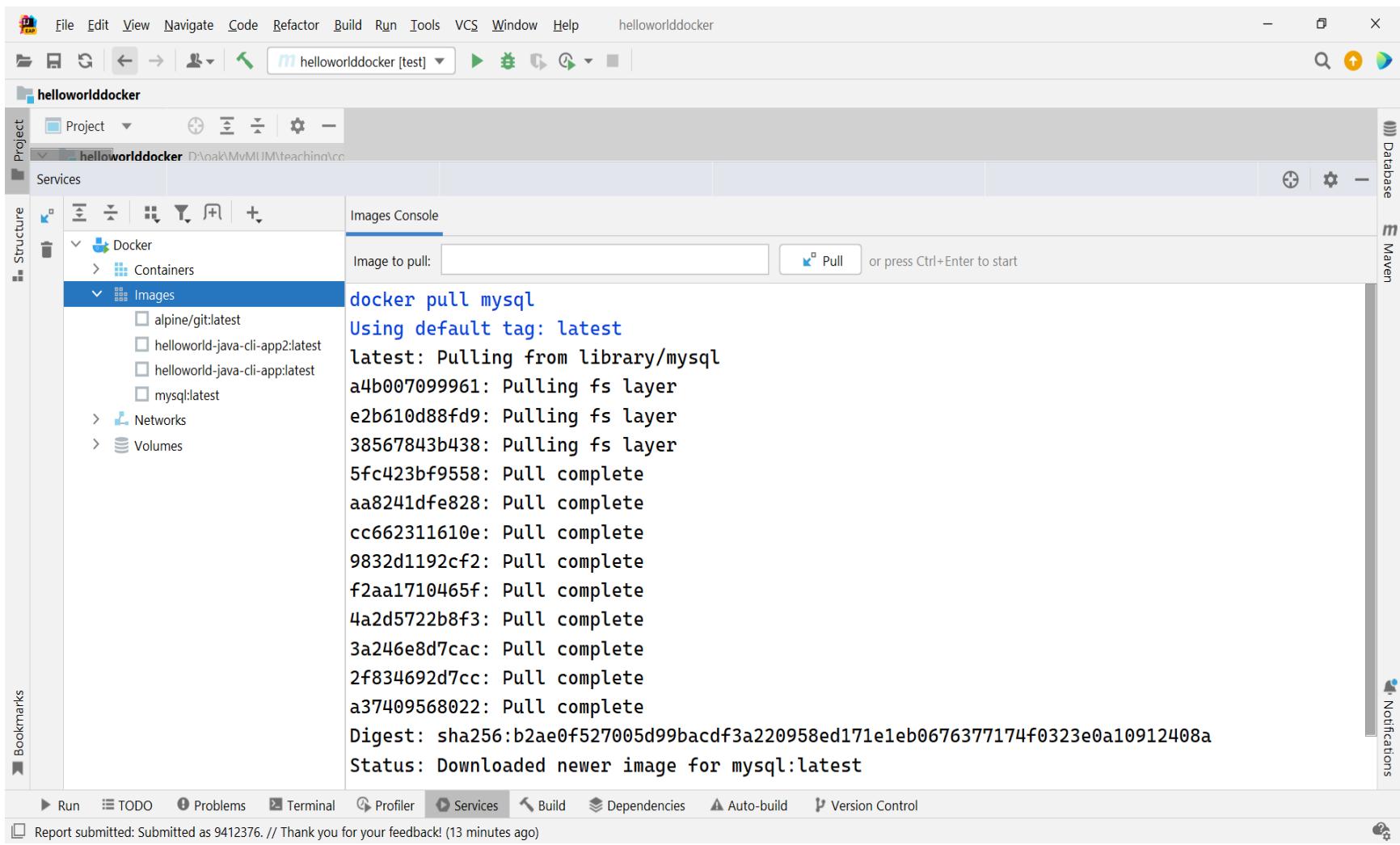
```
System.out.printf("App is still running. Iteration #: %d\n", ++iteration);
}
} catch (InterruptedException e) {
    e.printStackTrace();
    throw new RuntimeException(e);
}
```

The "Services" tool window is open, showing the "Docker" section. Under "Images", the "Images Console" tab is selected, displaying a list of Docker images. A search bar shows "Image to pull: m" and a dropdown menu lists various images, with "mysql" selected.

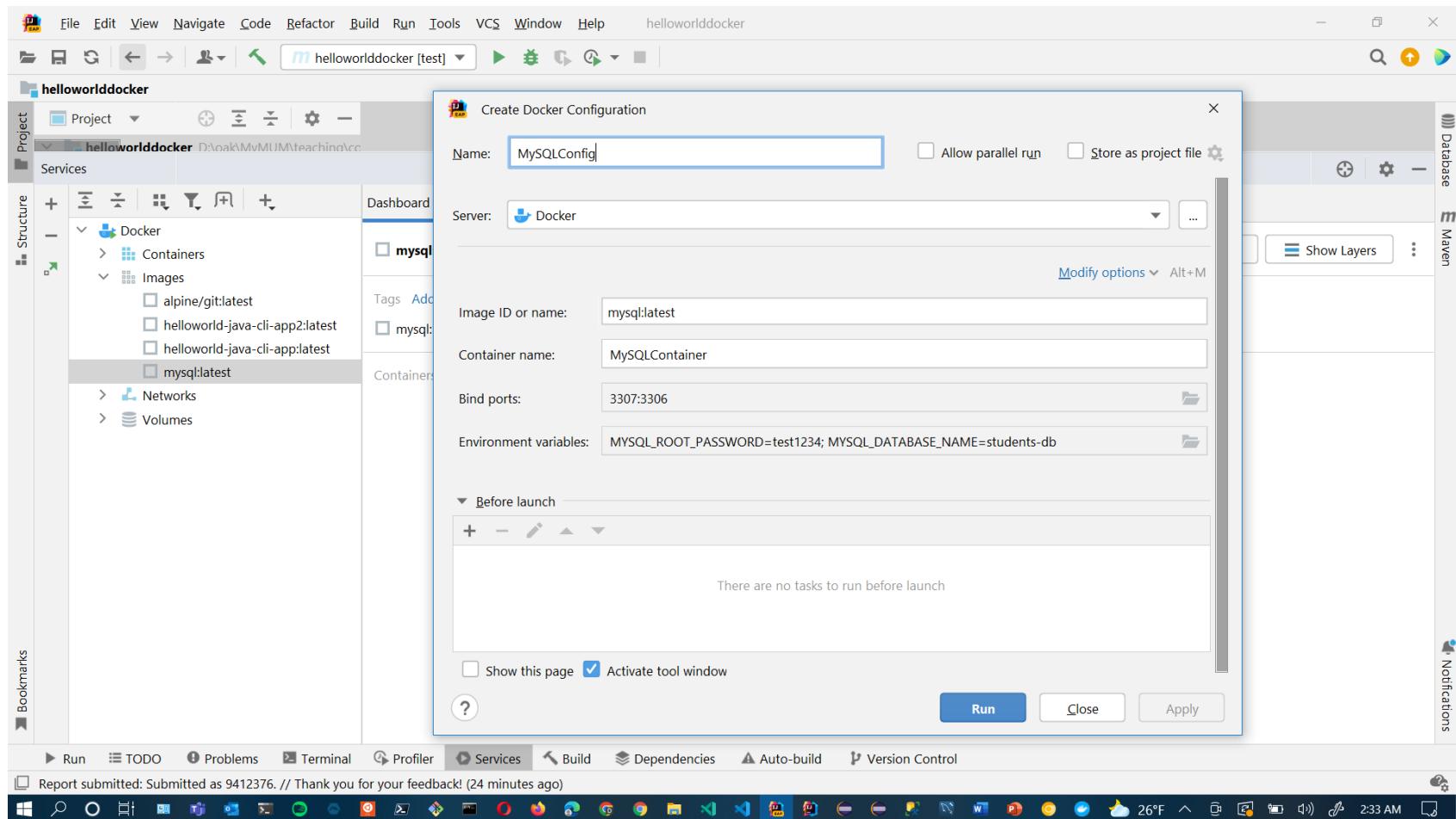
Working with Docker in IntelliJ IDEA

- 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.

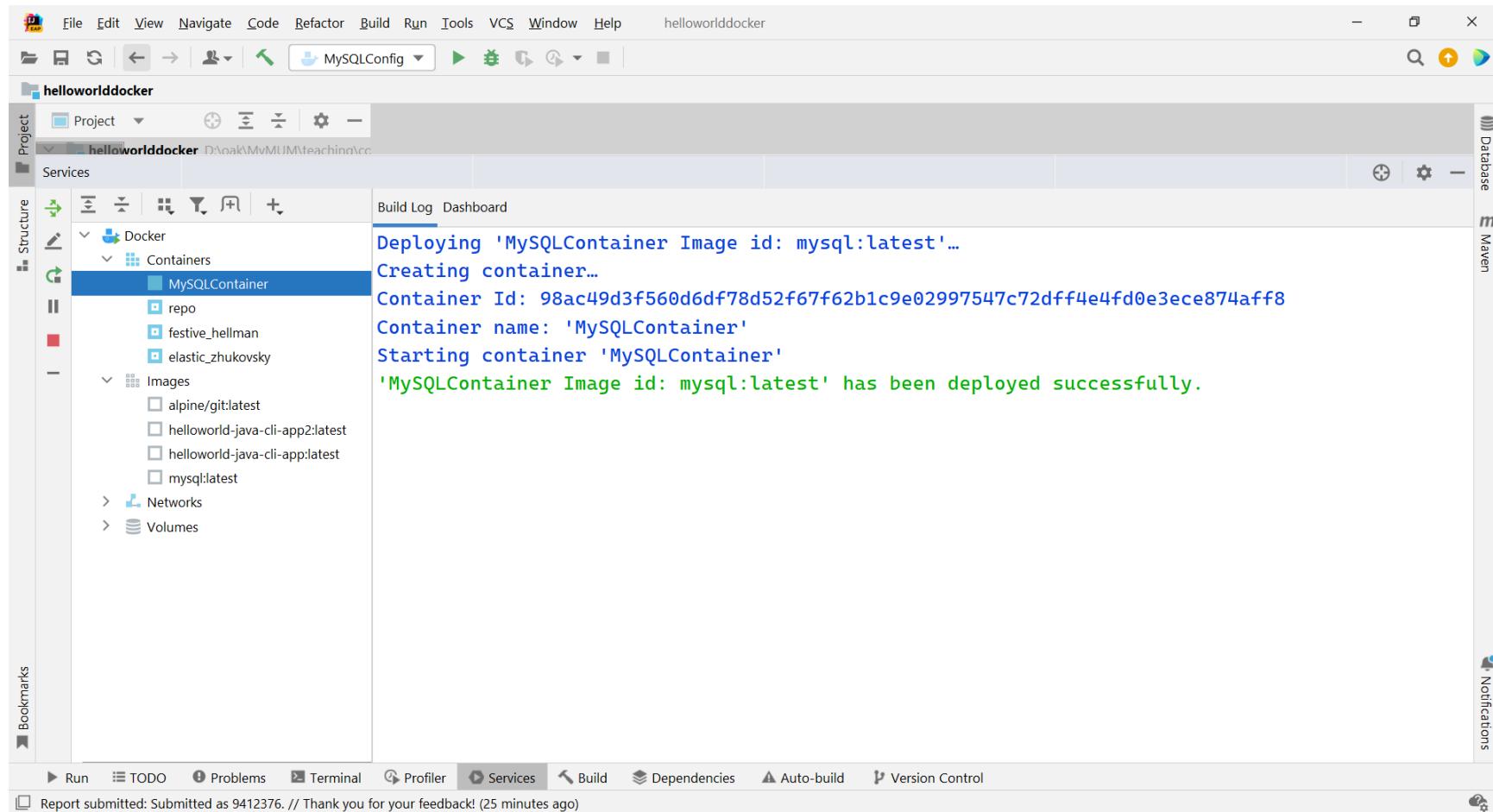
Setting-up a MySQL database container



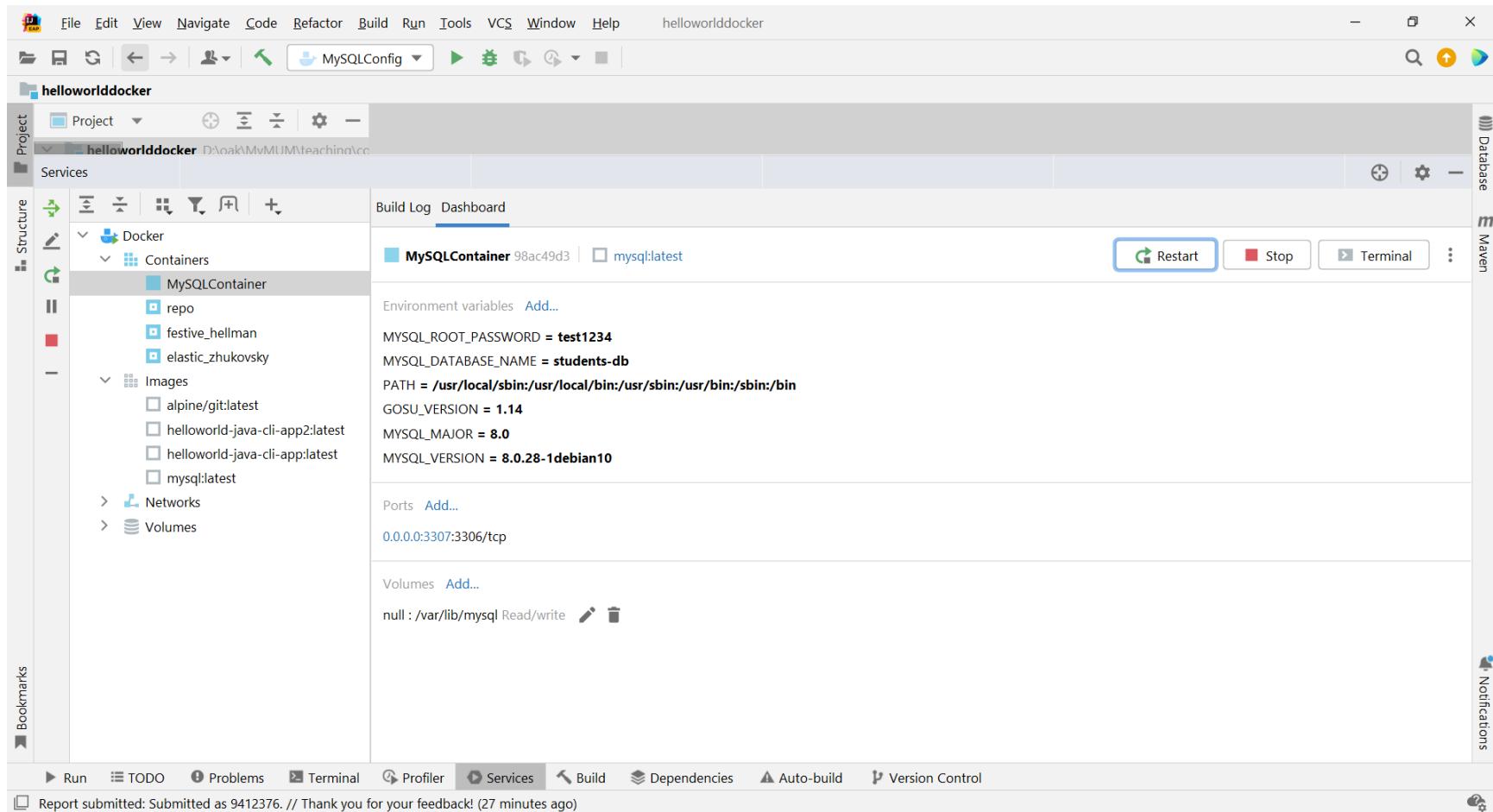
Create a MySQL DB container



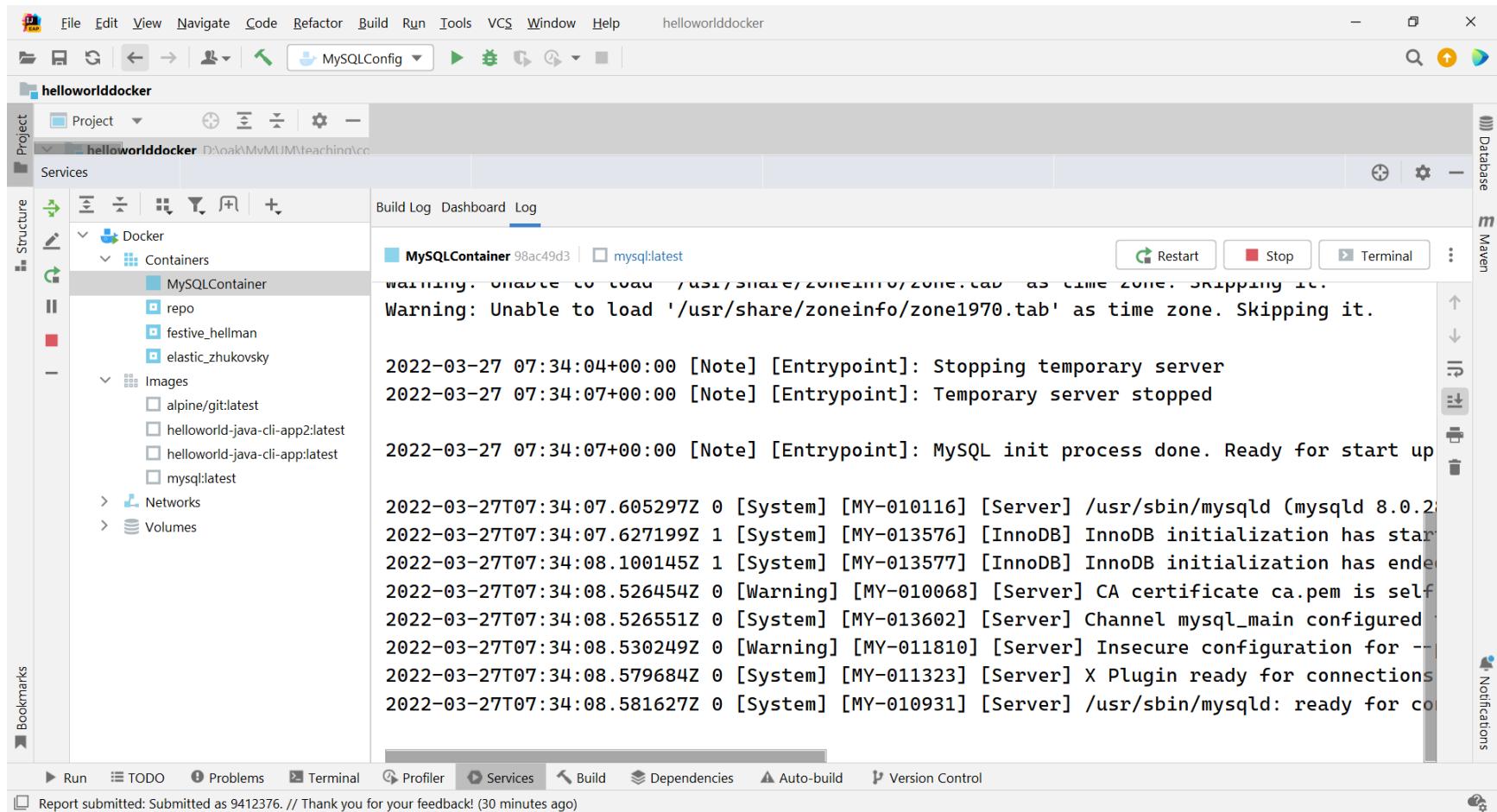
MySQL Database container created



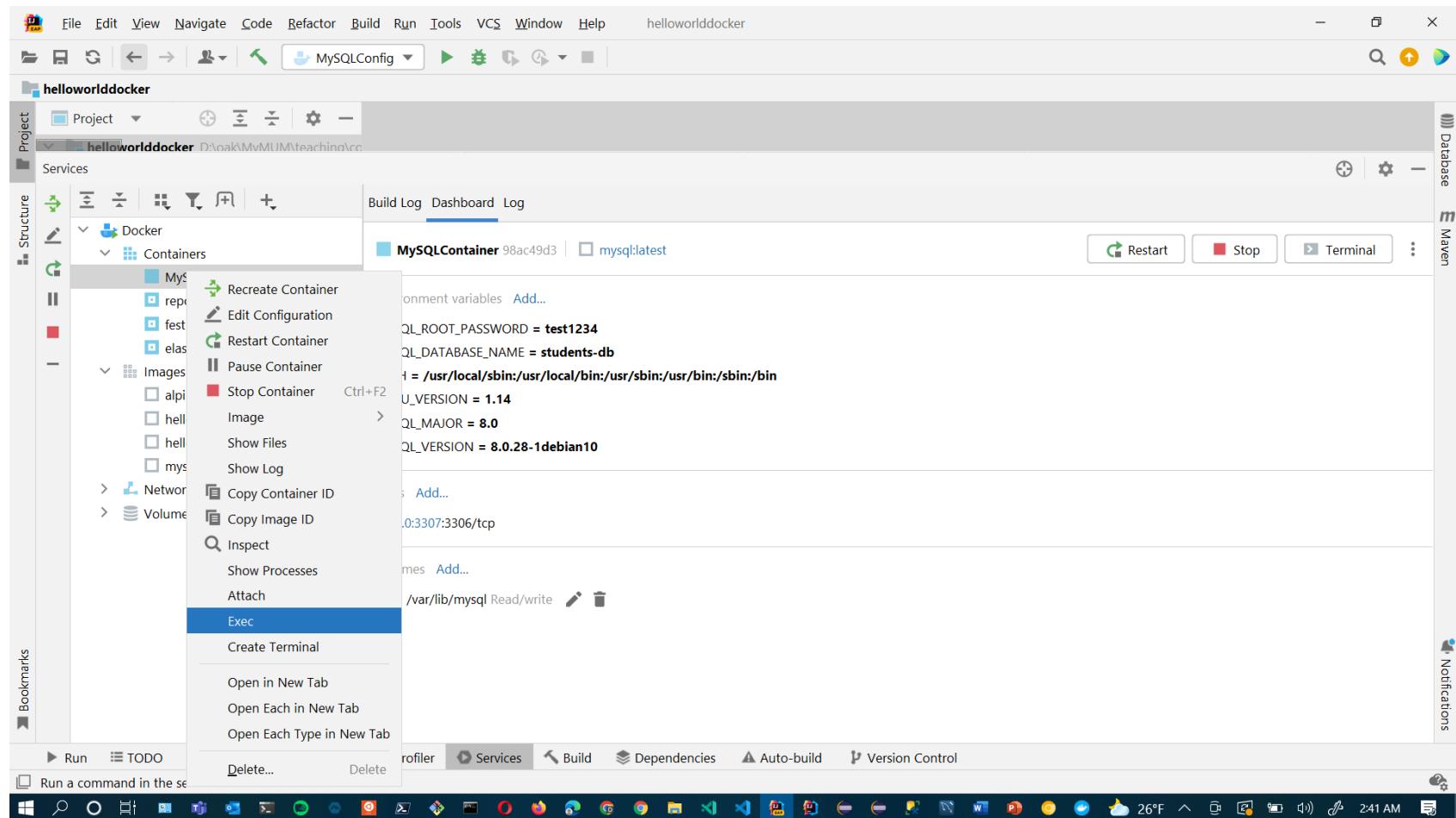
MySQL Database container running



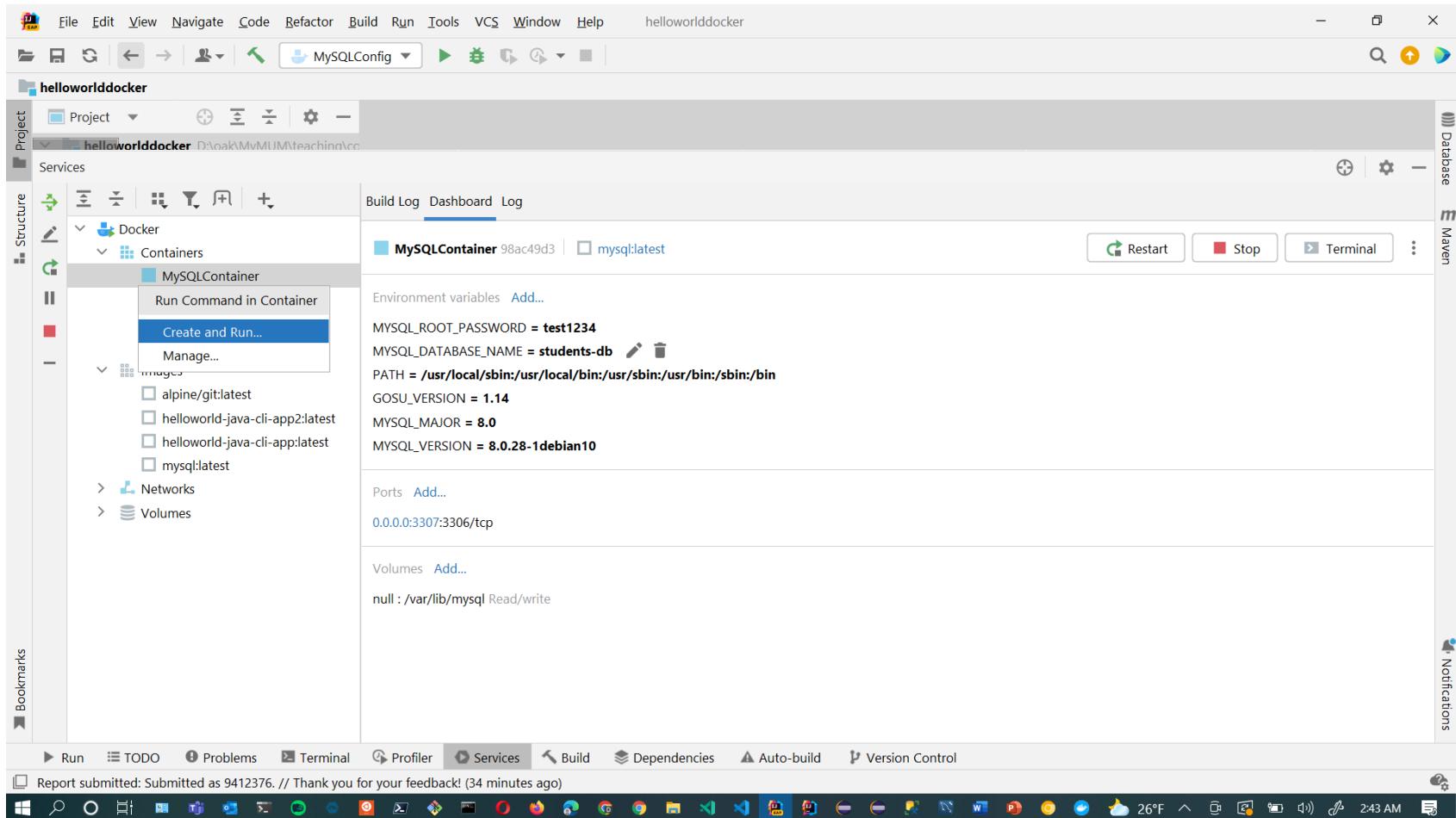
View the log



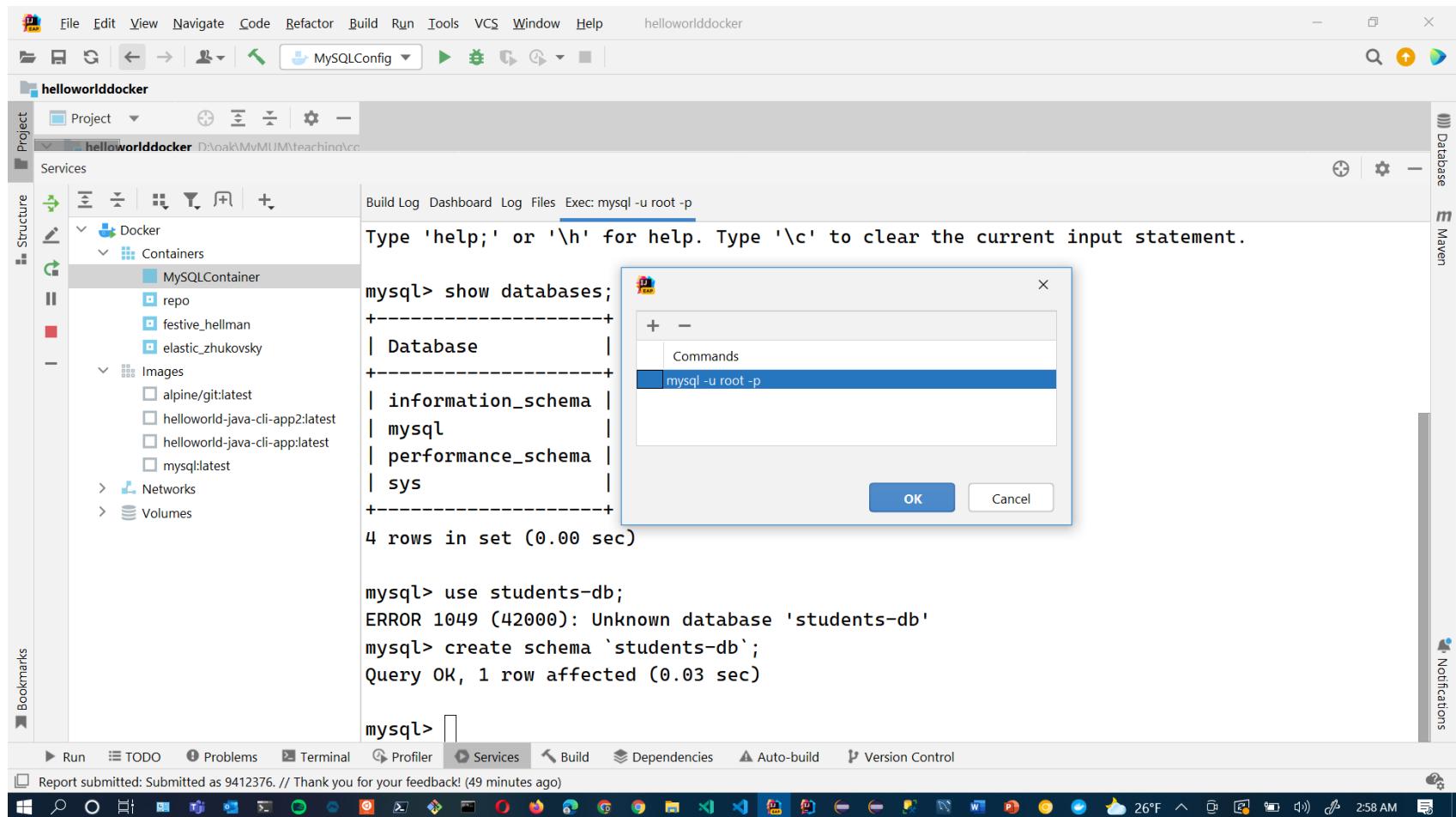
Connect with MySQL shell or client



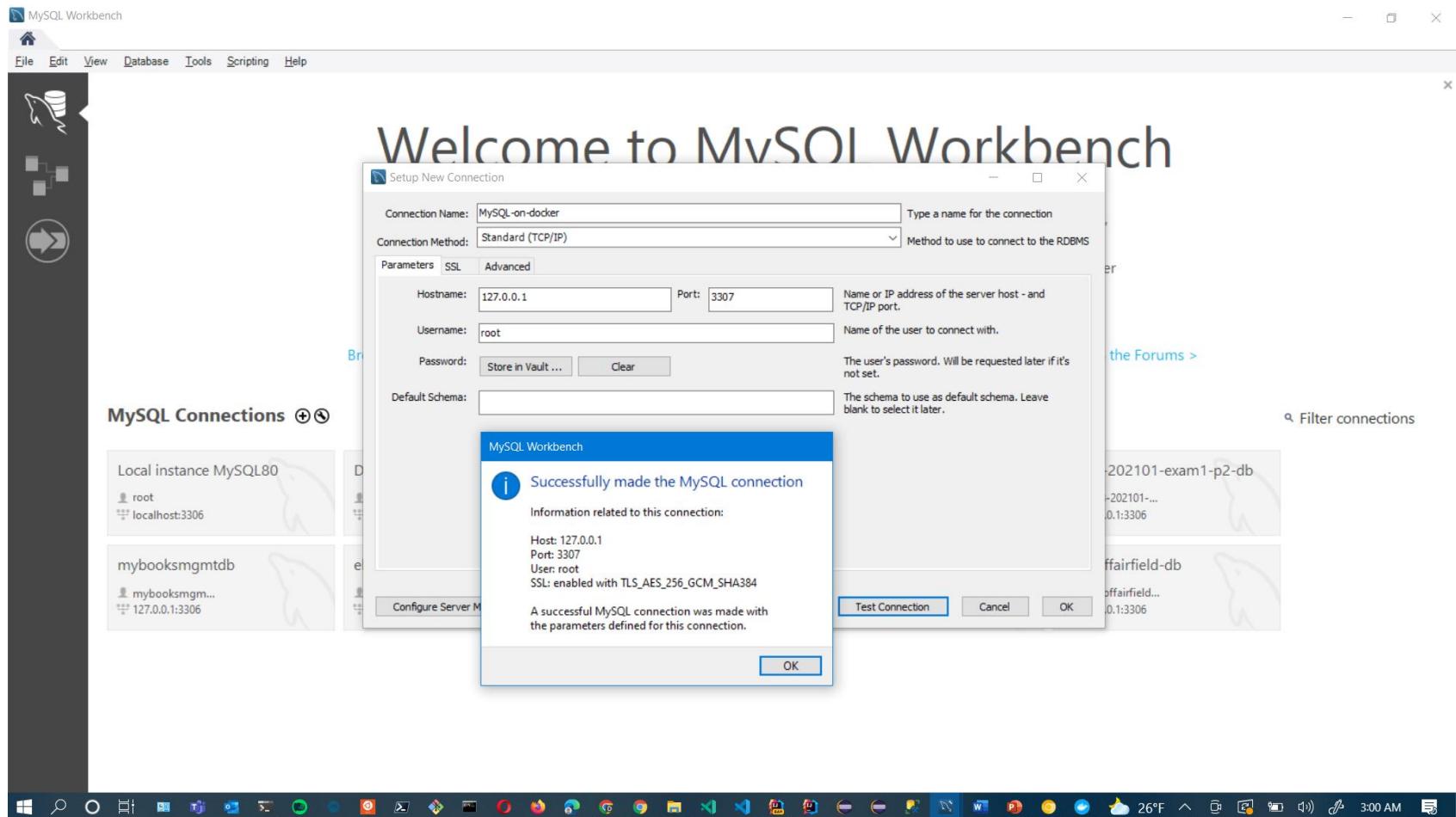
Connect with MySQL shell or client



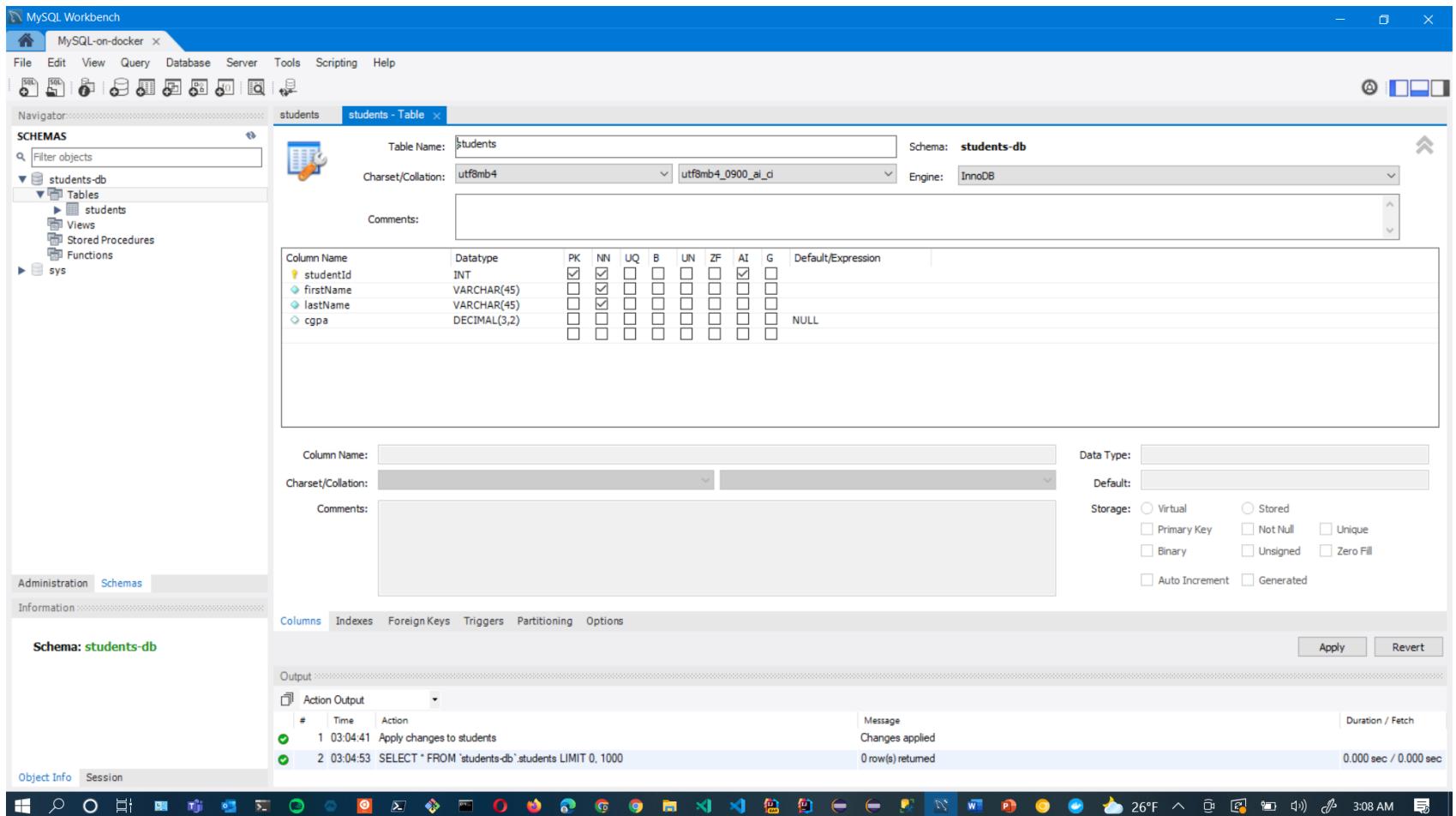
Connect and create a new database



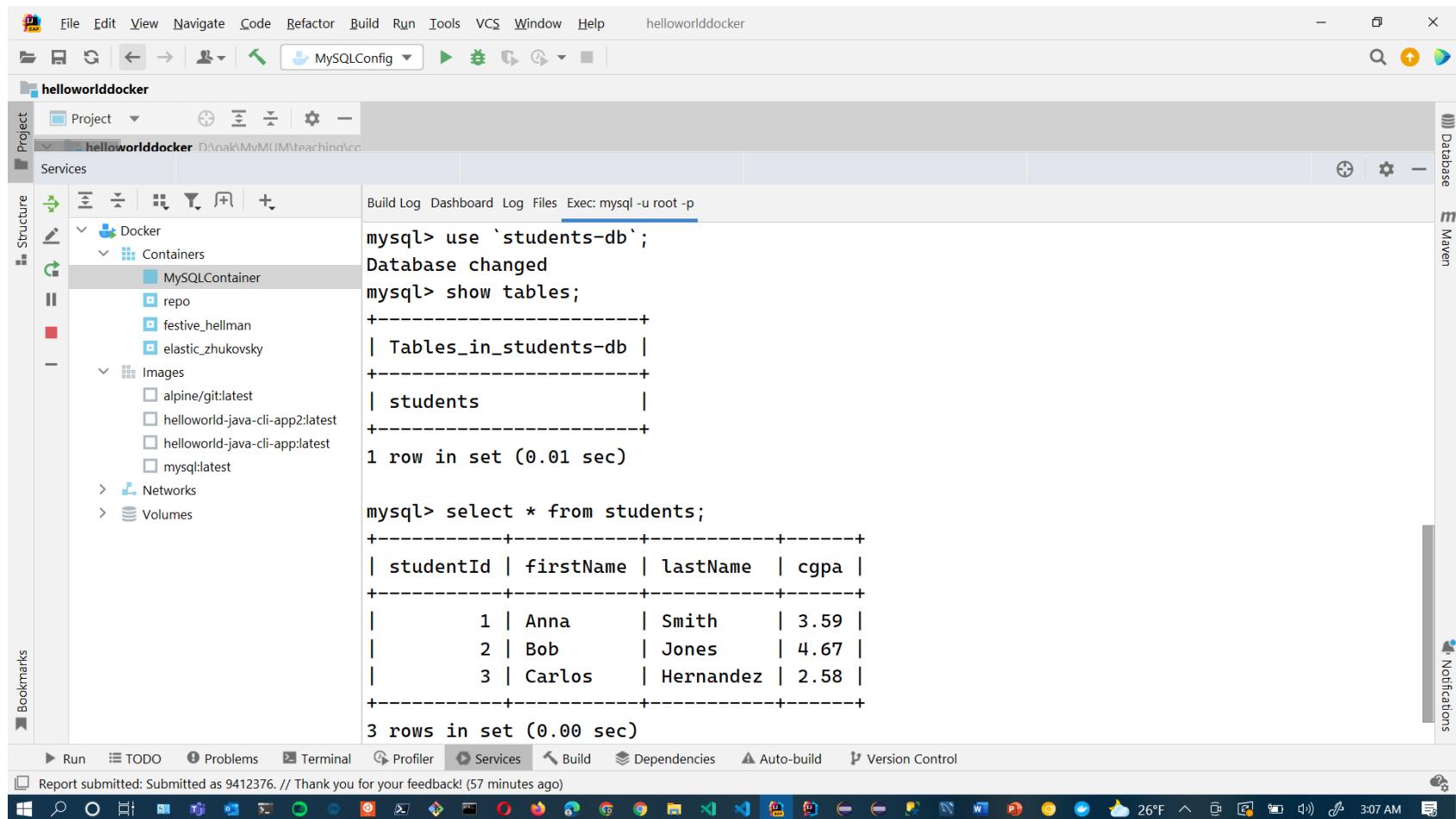
Connecting via MySQL Workbench from the host



Using MySQL Workbench



Working with mysql client in IntelliJ



The screenshot shows the IntelliJ IDEA interface with the "MySQLConfig" tool window open. The "Services" tab is selected, displaying Docker services. A MySQLContainer is selected, and the "Exec" tab shows a terminal session running MySQL commands:

```
mysql> use `students-db`;
Database changed
mysql> show tables;
+-----+
| Tables_in_students-db |
+-----+
| students |
+-----+
1 row in set (0.01 sec)

mysql> select * from students;
+-----+-----+-----+-----+
| studentId | firstName | lastName | cgpa |
+-----+-----+-----+-----+
| 1 | Anna | Smith | 3.59 |
| 2 | Bob | Jones | 4.67 |
| 3 | Carlos | Hernandez | 2.58 |
+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

The IntelliJ status bar at the bottom indicates: Report submitted: Submitted as 9412376. // Thank you for your feedback! (57 minutes ago)

Containerizing a Spring Boot Application

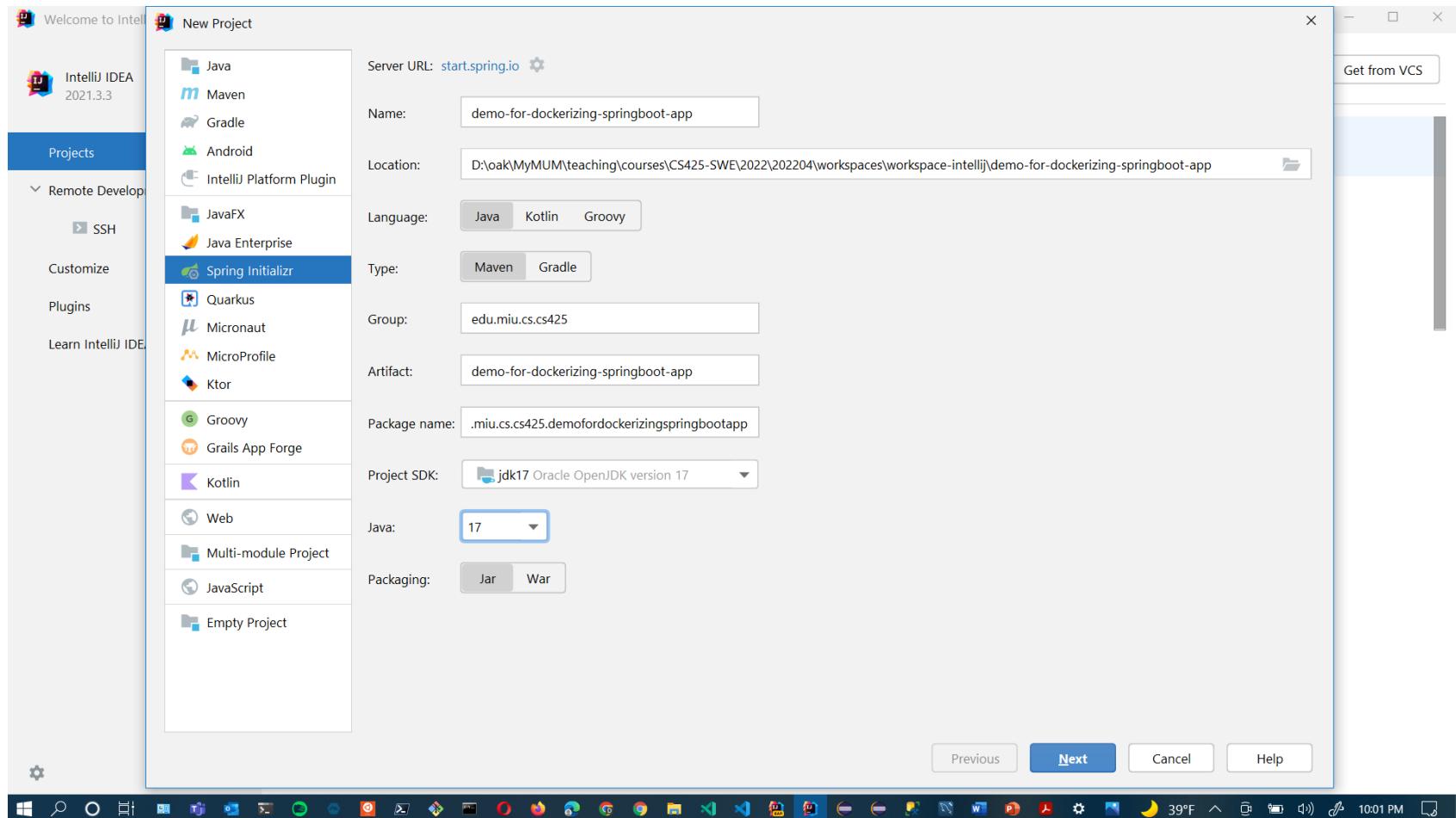
Best Practices for building a Docker Image of your Spring Boot Application

Containerize Spring Boot App with Docker

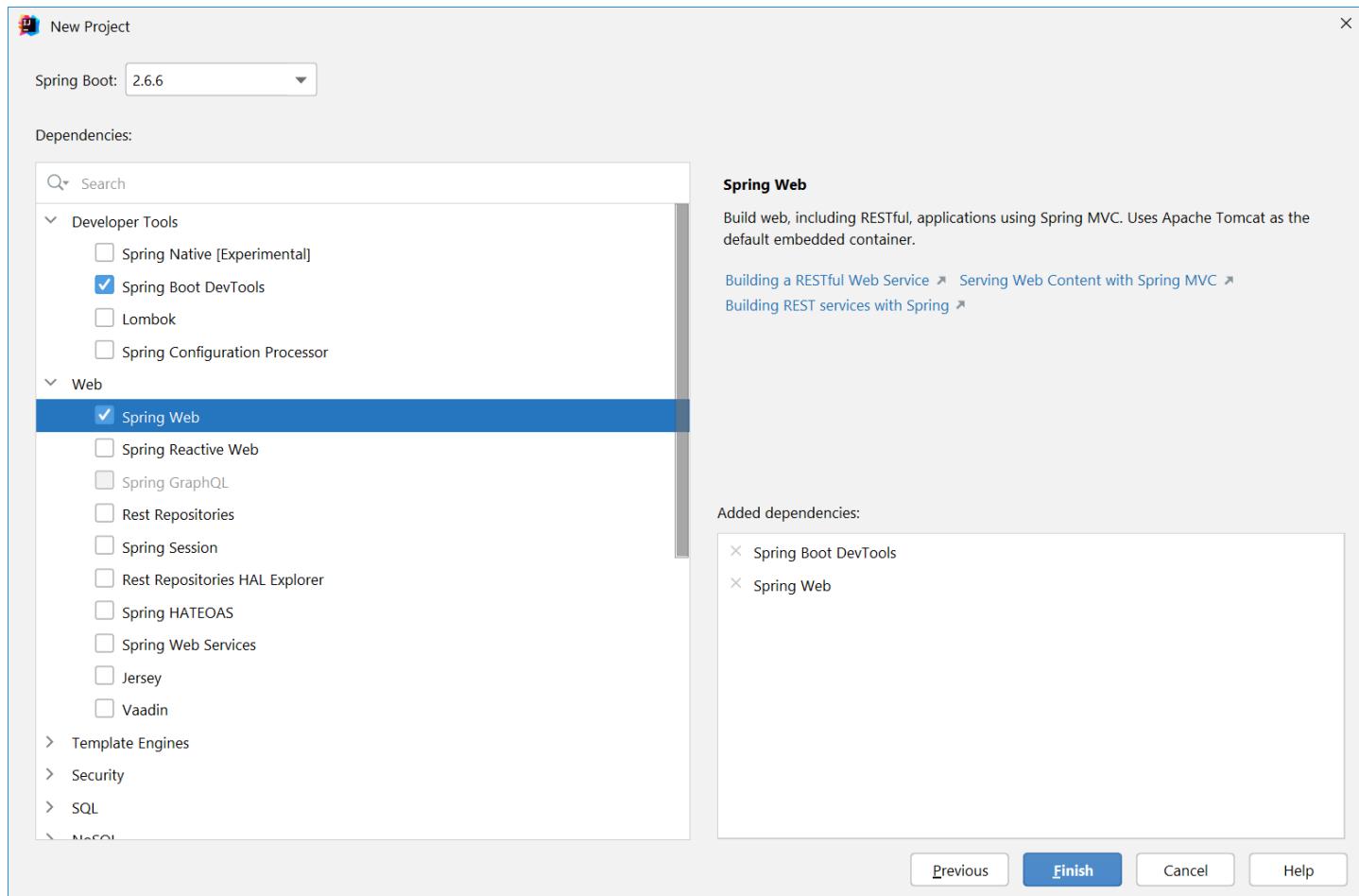
- Docker Image creation functionality was introduced in Spring Boot version 2.3.
- The main dependency required for this is Spring-Boot-DevTools



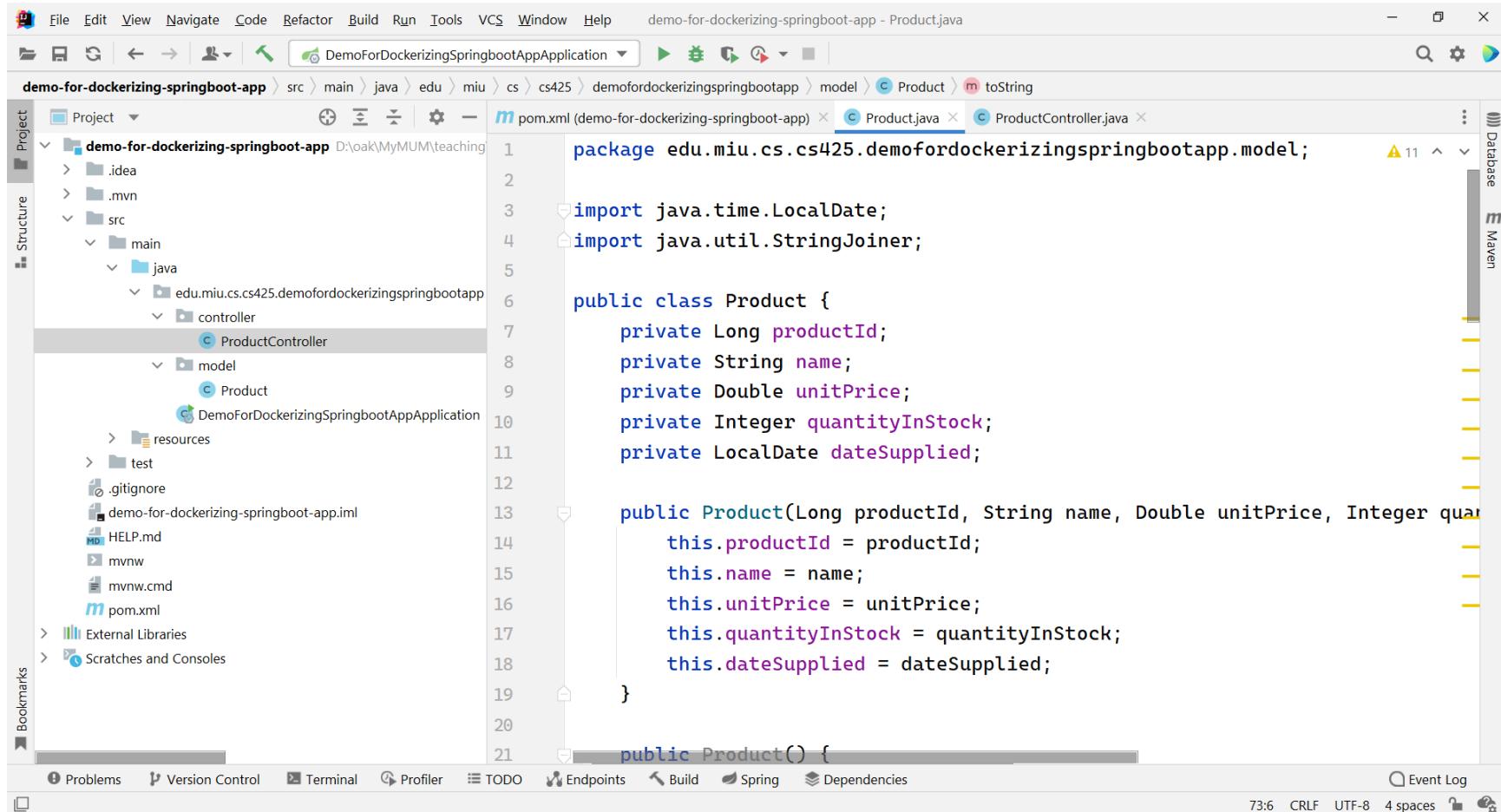
Create a new Spring Initializer Project in IntelliJ IDEA



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 code defines a `Product` class with attributes for `productId`, `name`, `unitPrice`, `quantityInStock`, and `dateSupplied`. It includes a constructor and a parameterless constructor.

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

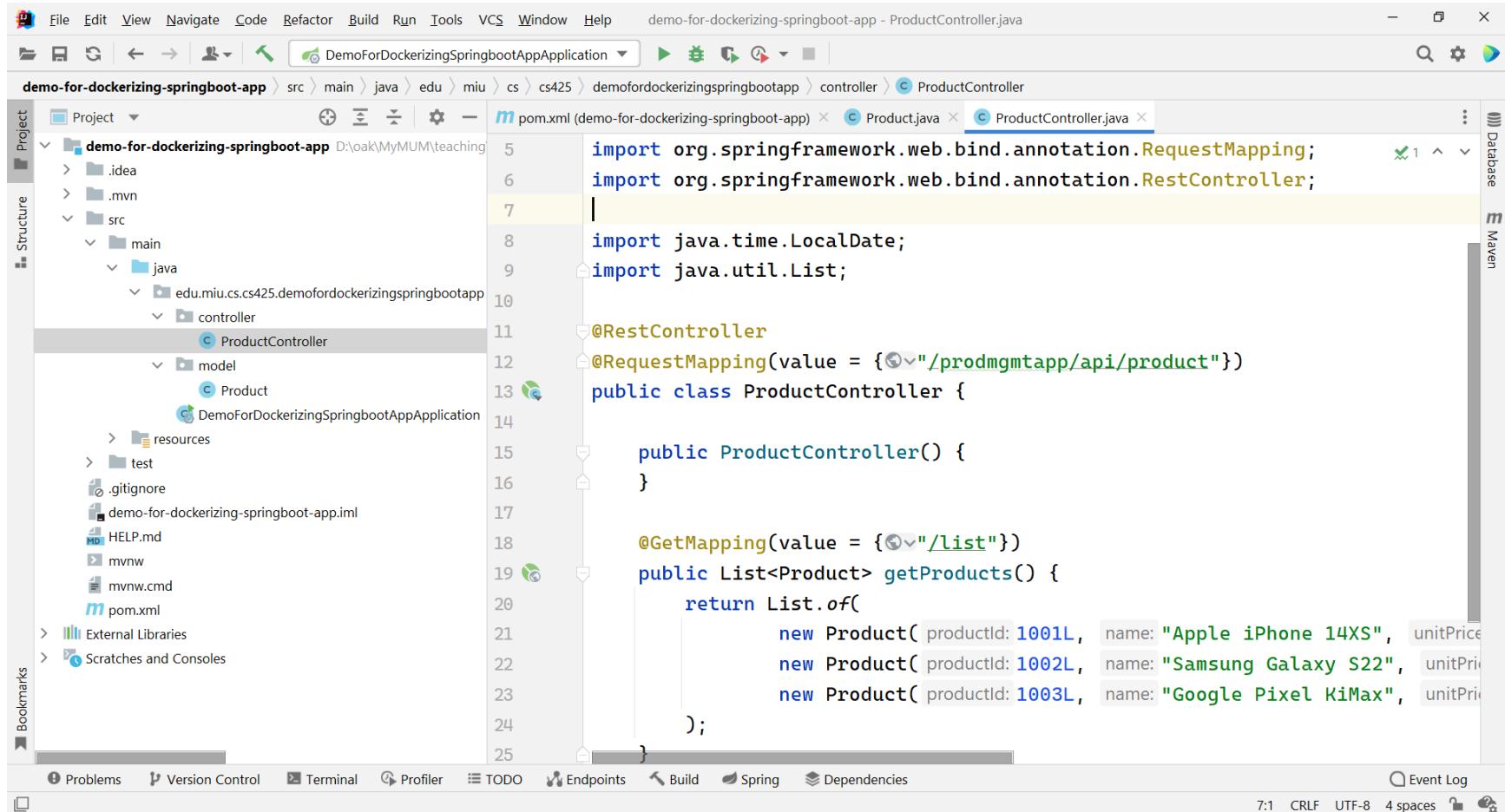
import java.time.LocalDate;
import java.util.StringJoiner;

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() {
    }
}
```

Code the RestController class



The screenshot shows the IntelliJ IDEA 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 Java development toolbar with icons for file operations, run, debug, and build.
- Project Structure:** Shows the project structure under "demo-for-dockerizing-springboot-app". The "src/main/java/edu/miu/cs/cs425/demofordockerizingspringbootapp/controller/ProductController.java" file is selected.
- Code Editor:** Displays the code for the ProductController class. The code includes imports for RequestMapping, RestController, LocalDate, and List, along with annotations for @RestController and @RequestMapping, and a method getProducts() that returns a list of products.
- Maven Integration:** Maven-related tabs and icons are visible on the right side of the editor.
- Bottom Navigation:** Problems, Version Control, Terminal, Profiler, TODO, Endpoints, Build, Spring, Dependencies, and Event Log tabs.
- Status Bar:** Shows file encoding (7:1 CRLF), character set (UTF-8), and code style (4 spaces).

```
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 = {"${}/list"})
    public List<Product> getProducts() {
        return List.of(
            new Product( productId: 1001L, name: "Apple iPhone 14XS", unitPrice: 1000.0 ),
            new Product( productId: 1002L, name: "Samsung Galaxy S22", unitPrice: 1200.0 ),
            new Product( productId: 1003L, name: "Google Pixel KiMax", unitPrice: 1500.0 )
        );
    }
}
```

Run the App

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 - DemoForDockerizingSpringbootAppApplication.java
- Toolbars:** Standard toolbar with icons for file operations.
- Project Structure:** Shows the project structure for "demo-for-dockerizing-springboot-app". The "src/main/java" package contains "edu.miu.cs.cs425.demofordockerizingspringbootapp" which has "controller" and "model" sub-packages, and "DemoForDockerizingSpringbootAppApplication".
- Code Editor:** Displays the code for `DemoForDockerizingSpringbootAppApplication.java`. The code imports `SpringApplication` and `SpringBootApplication`, and defines a main class with a main method that runs the application.
- Run Tab:** Set to "DemoForDockerizingSpringbootAppApplication".
- Console Tab:** Shows 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] DemoForDockerizingSpringbootAppApplication : Started DemoForDockerizingSpringbootAppApplication in 2.803 seconds (JVM running for 3.003)
```
- Bottom Navigation:** Run, Problems, Version Control, Terminal, Profiler, TODO, Endpoints, Build, Spring, Dependencies.
- Status Bar:** Build completed successfully in 6 sec, 99 ms (a minute ago), 5:1 LF, UTF-8, 4 spaces, Event Log.

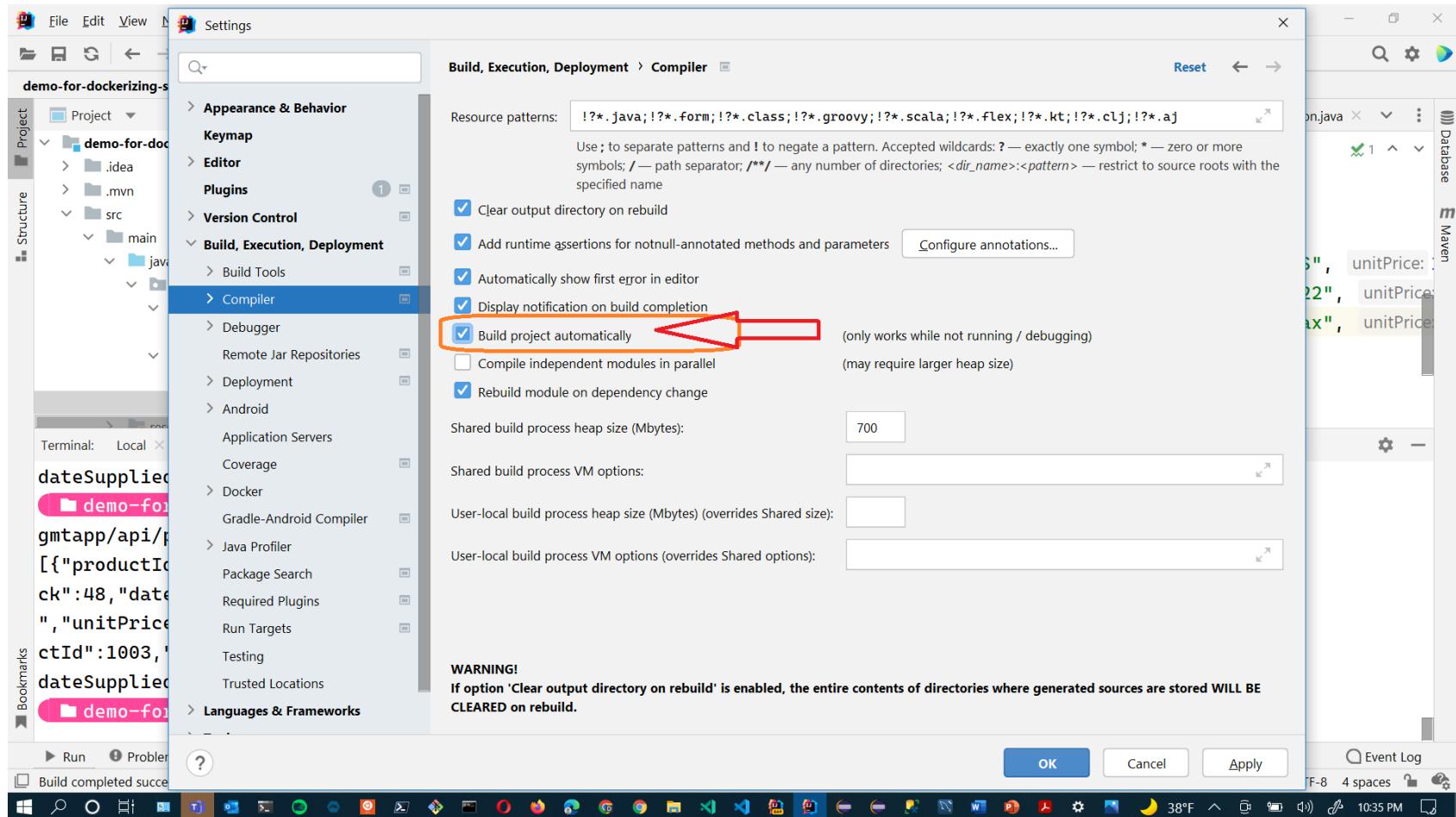
Using curl or browser run HTTP GET url

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

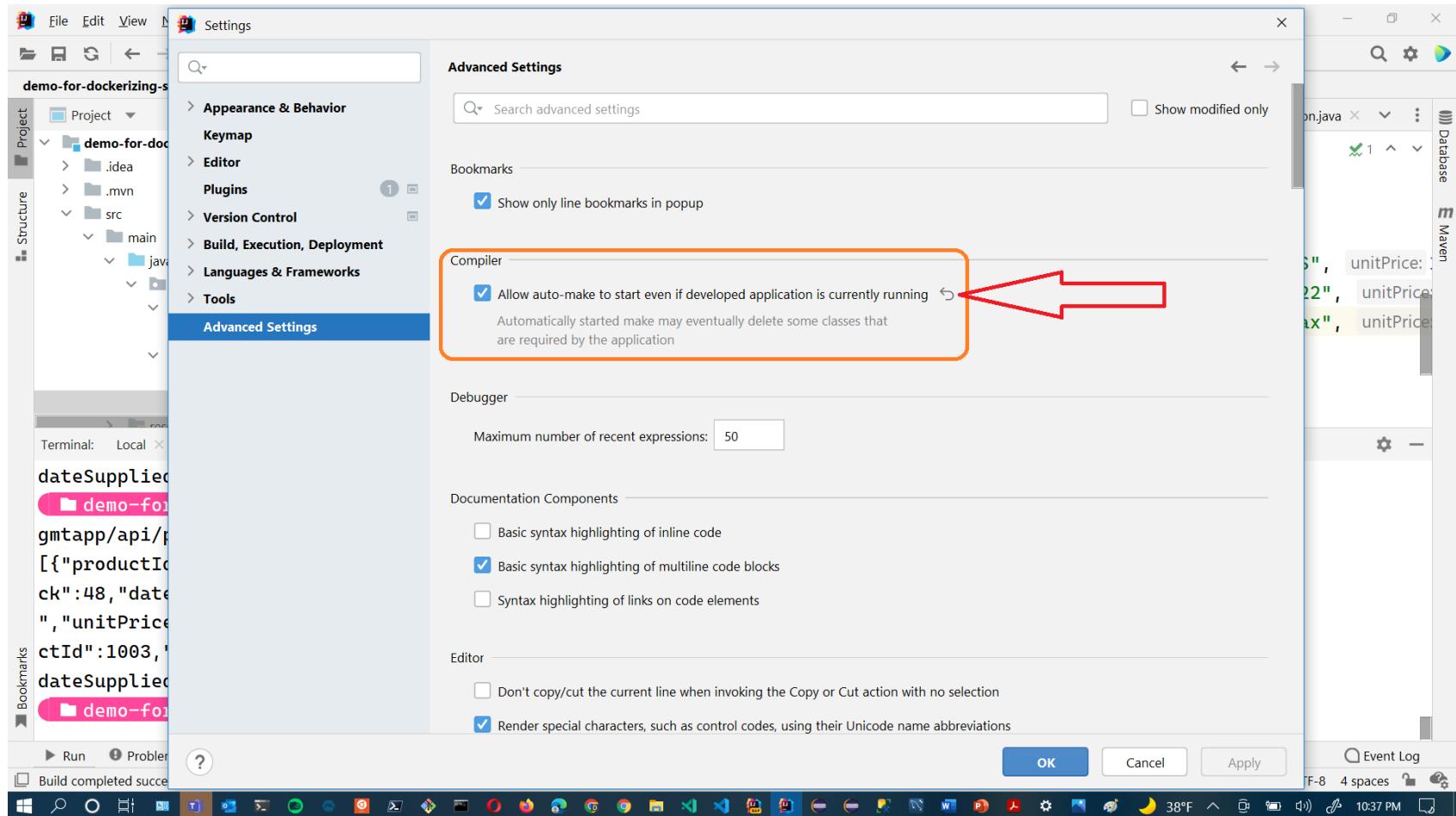
- Project Structure:** The project is named "demo-for-dockerizing-springboot-app". The code editor displays the file "ProductController.java".
- Code Editor:** The code for the ProductController is shown, including annotations for REST controllers and endpoints.
- Terminal:** The terminal window shows two curl commands being run:
 - curl http://localhost:8080/prodmgmtapp/api/product
 - curl http://localhost:8080/prodmgmtapp/api/list
- Output:** The terminal shows the JSON response for the "/list" endpoint, listing three products:

```
[{"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 Run, Problems, Version Control, Terminal, Profiler, TODO, Endpoints, Build, Spring, Dependencies, and Event Log.
- Status Bar:** The status bar at the bottom indicates a successful build: "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 the IntelliJ IDEA interface with the following details:

- Project Structure:** The project is named "demo-for-dockerizing-springboot-app". The code editor shows the file "ProductController.java" with the following code snippet:

```
return List.of(
    new Product( productId: 1001L, name: "Apple iPhone 14XS", unitPrice: 1850.95),
    new Product( productId: 1002L, name: "Samsung Galaxy S22", unitPrice: 1700.99),
    new Product( productId: 1009L, name: "Google Pixel KiMax", unitPrice: 1350.5)
);
```

- Terminal:** The terminal window shows three consecutive curl commands being run against the local host at port 8080, each returning a JSON list of products.

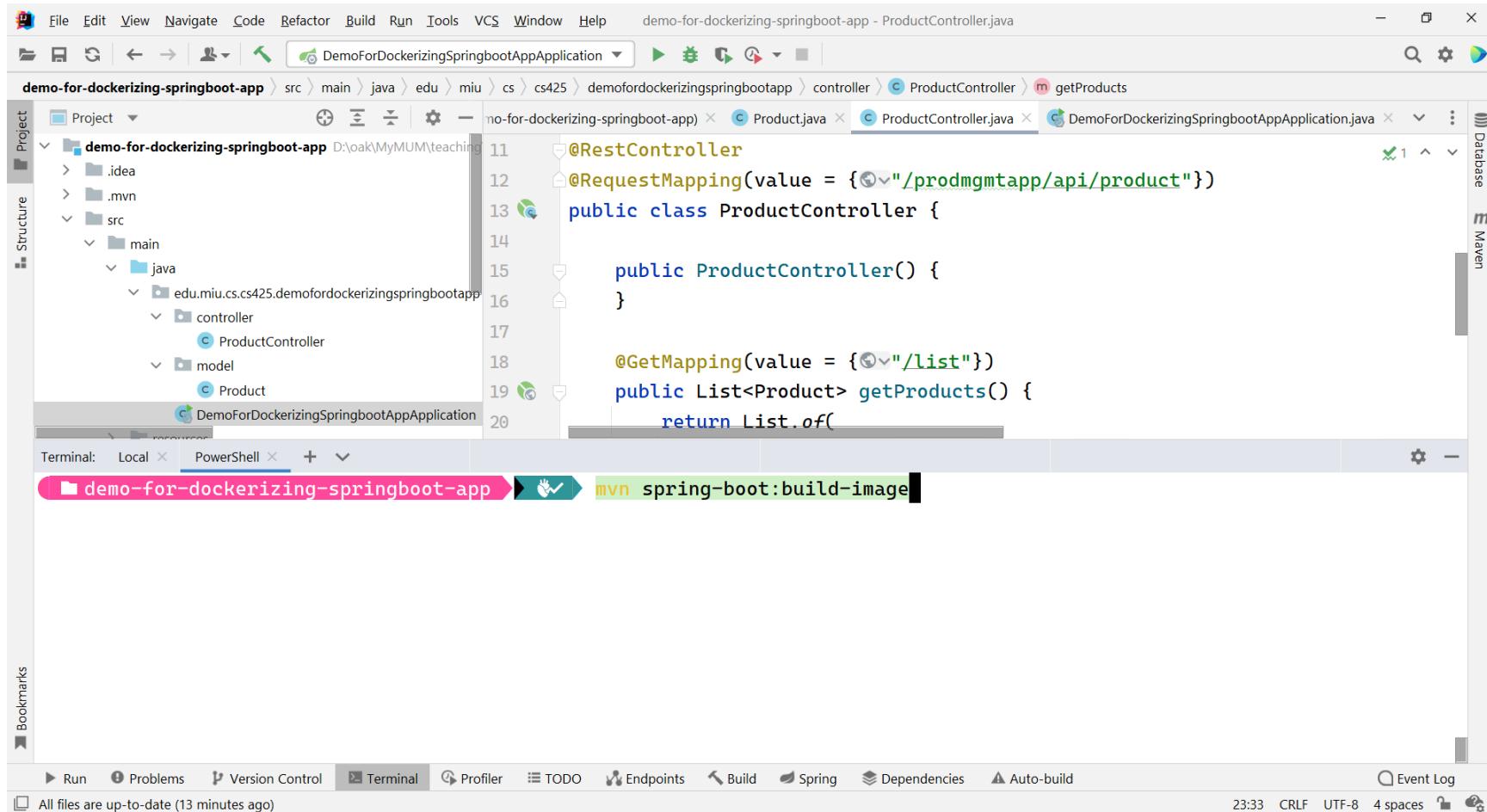
```
curl http://localhost:8080/prodmgmtapp/api/product/list
[{"productId":1001,"name":"Apple iPhone 14XS","unitPrice":1850.95,"quantityInStock":48,"dateSupplied":"2022-01-15"}, {"productId":1002,"name":"Samsung Galaxy S22","unitPrice":1700.99,"quantityInStock":115,"dateSupplied":"2021-10-27"}, {"productId":1003,"name":"Google 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"}]
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 Status Bar:** Shows "All files are up-to-date (2 minutes ago)" and other system information like time and encoding.

How to Build Docker Image of the Spring Boot Application

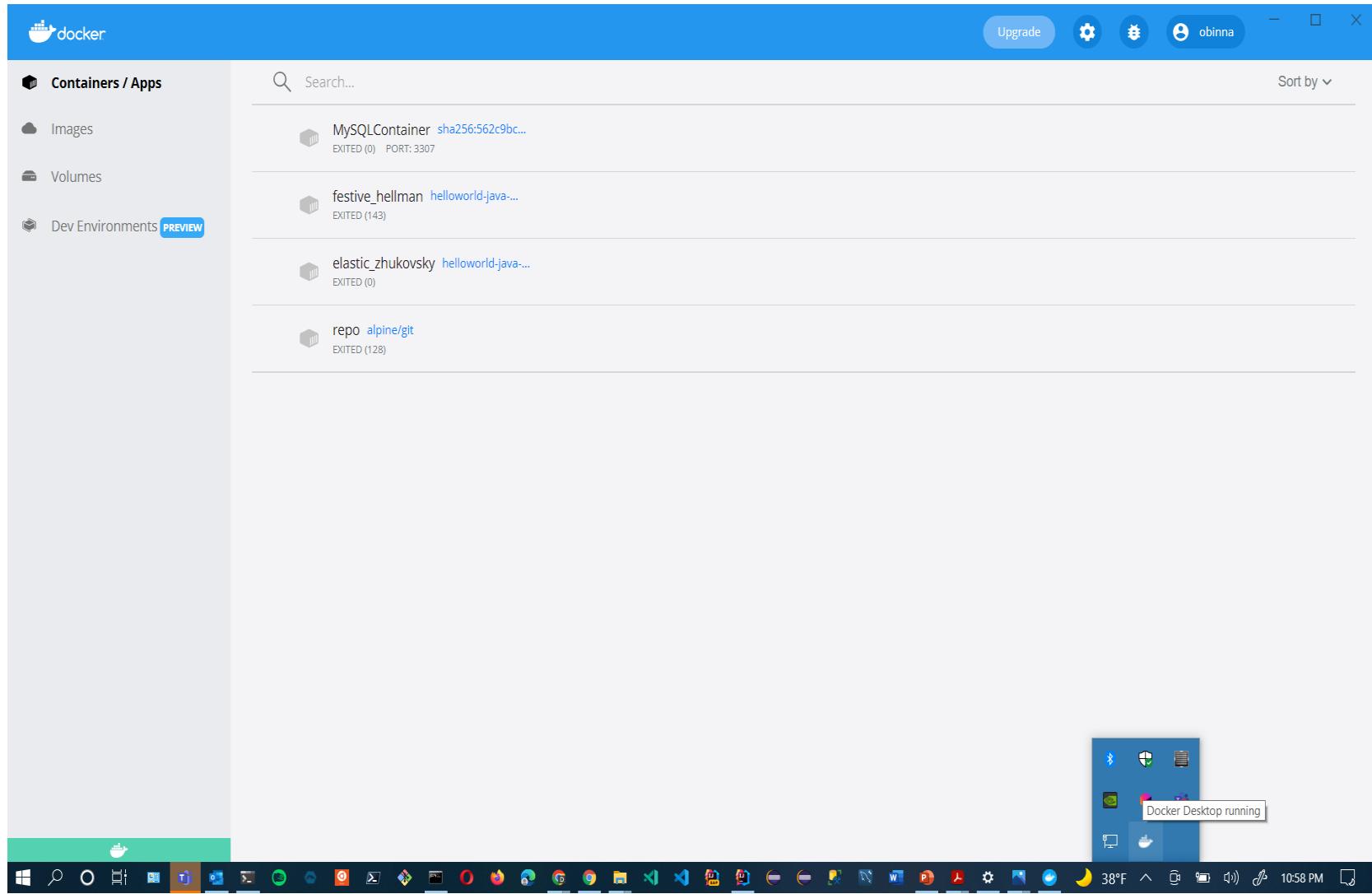
- Beginning with Spring Boot version 2.3, the `spring-boot-maven-plugin` can create an OCI (e.g. docker) image from a `.jar` or `.war` package file using CNCF's Cloud Native Buildpacks
- Images can be built by using the `mvn build-image` goal
- Open terminal and run:
 > `mvn spring-boot:build-image`

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 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. "controller" contains "ProductController.java".
- Code Editor:** The code for "ProductController.java" is displayed. It defines a REST controller with a single endpoint: `@GetMapping(value = {"/list"}) public List<Product> getProducts() { return List.of(); }`.
- Terminal:** The terminal window shows the output of a Maven build command:

```
[INFO] [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%
```
- Bottom Bar:** The bottom bar includes tabs for Run, Problems, Version Control, Terminal, Profiler, TODO, Endpoints, Build, Spring, Dependencies, Auto-build, and Event Log. A status message at the bottom left says "All files are up-to-date (21 minutes ago)".

Successfully Built Image

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.
- Project Bar:** demo-for-dockerizing-springboot-app - ProductController.java
- Toolbars:** Standard, Version Control, Profiler, TODO, Endpoints, Build, Spring, Dependencies, Auto-build.
- Left Sidebar:** Project (demo-for-dockerizing-springboot-app), Structure, Bookmarks.
- Central Area:** Code editor showing `ProductController.java` with the following code:

```
11  @RestController
12  @RequestMapping(value = {"${prodmgmtapp}/api/product"})
13  public class ProductController {
14
15      public ProductController() {
16
17
18      @GetMapping(value = {"${prodmgmtapp}/list"})
19      public List<Product> getProducts() {
20          return List.of()
```
- Bottom Terminal:** Local (PowerShell) terminal showing build logs:

```
[INFO] Successfully built image 'docker.io/library/demo-for-dockerizing-springboot-app:0.0.1-SNAPSHOT'
[INFO]
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] -----
[INFO] Total time:  01:37 min
[INFO] Finished at: 2022-04-01T23:01:53-05:00
[INFO]
```
- Status Bar:** All files are up-to-date (23 minutes ago), 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 a 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 = {"${prodmgmtapp}/api/product"})
13  public class ProductController {
14
15      public ProductController() {
16
17
18      @GetMapping(value = {"${prodmgmtapp}/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 command being run:

```
demo-for-dockerizing-springboot-app docker run --tty --publish 8080:8080 demo-for-dockerizing-springboot-app:0.0.1-SNAPSHOT
```

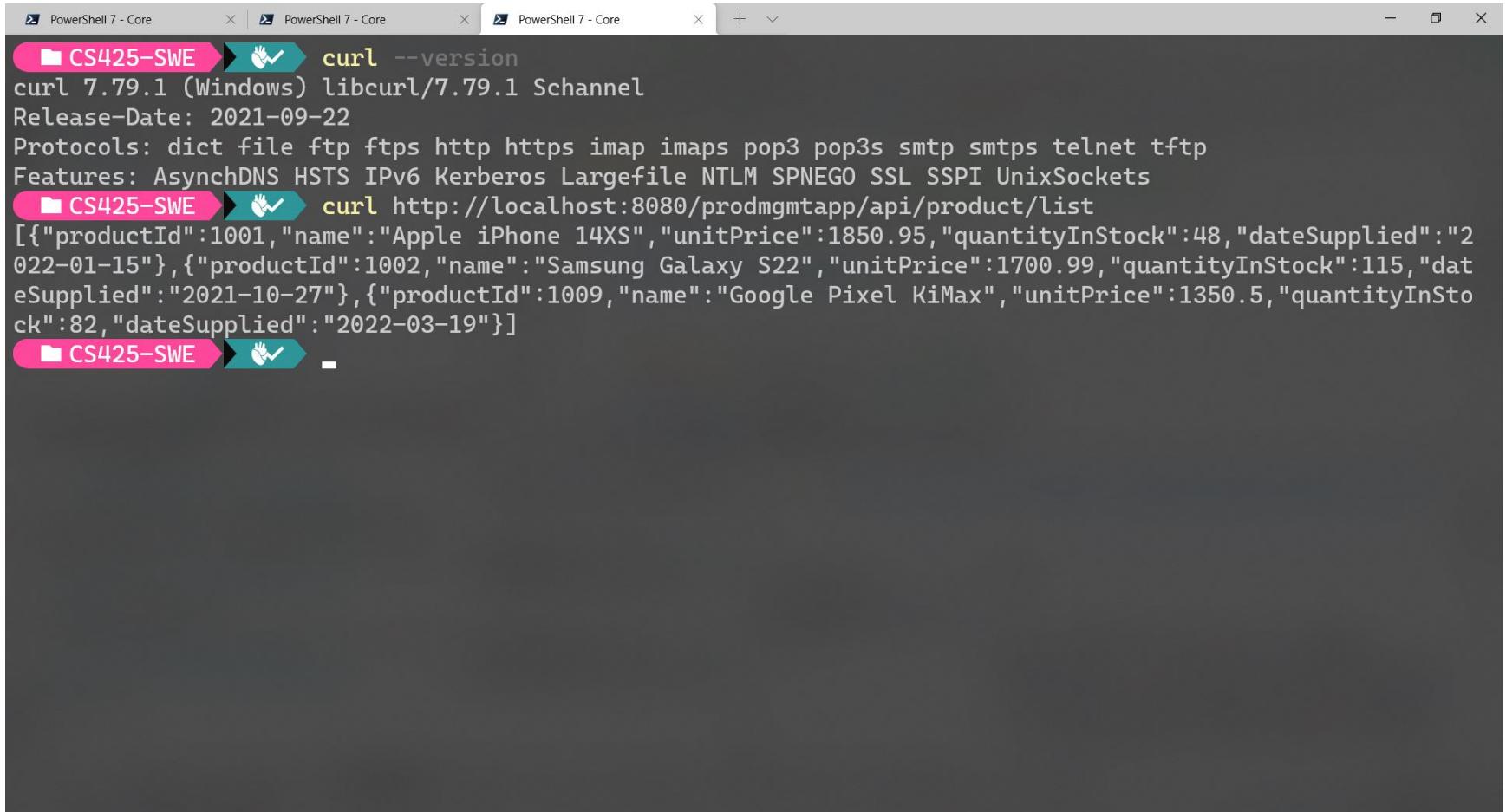
Test the container endpoint

The screenshot shows the IntelliJ IDEA IDE interface. The top navigation bar includes File, Edit, View, Navigate, Code, Refactor, Build, Run, Tools, VCS, Window, Help, and demo-for-dockerizing-springboot-app - ProductController.java. The main window displays the Project structure on the left, showing the demo-for-dockerizing-springboot-app module with its subfolders (.idea, .mvn, src) and main package (edu.miu.cs.cs425.demofordockerizingspringbootapp). The src/main/java directory contains controller, model, and DemoForDockerizingSpringbootAppApplication.java files. The controller package contains ProductController.java, which is currently selected. The code in ProductController.java defines a ProductController class with a constructor and a getProducts() method annotated with @GetMapping(value = "/list"). The right side of the screen shows the code editor with lines 11 through 20 of the ProductController.java code. Below the code editor is a Terminal window titled 'PowerShell' showing the output of a curl command. The curl command is: curl http://localhost:8080/prodmgmtapp/api/product/list. The response from the server is as follows:

```
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  
StatusCode : 200  
StatusDescription :  
Content : [{"productId":1001,"name":"Apple iPhone 14XS","unitPrice":1850.95,"quantityInStock":48,"dateSupplied":"2022-01-15"}, {"productId":1002,"name":"Samsung
```

The bottom of the interface shows various toolbars and status bars, including Run, Problems, Version Control, Terminal, Profiler, TODO, Endpoints, Build, Spring, Dependencies, Auto-build, Event Log, and a status bar indicating 'All files are up-to-date (40 minutes ago)'.

Test the Container Endpoint



The screenshot shows three separate PowerShell windows side-by-side, all titled "PowerShell 7 - Core". Each window has a pink header bar with the text "CS425-SWE" and a teal arrow icon.

- Window 1:** Contains the command `curl --version`. The output is:

```
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
```
- Window 2:** Contains the command `curl http://localhost:8080/prodmgmtapp/api/product/list`. The output is:

```
[{"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"}]
```
- Window 3:** Contains a single character: `-`.

In Split Screen

The screenshot shows a terminal window split vertically. The left pane displays Docker logs for a Spring Boot application, while the right pane shows PowerShell help output.

Left Pane (Docker Logs):

```
PowerShell 7 - Core
x + 
CS425-SWE ➔ 🐳 docker run --tty --publish 8080:8080 demo-for-dockerizing-springboot-app:0.0.1 -SNAPSHOT
Setting Active Processor Count to 4
Calculating JVM memory based on 24828540K available memory
Calculated JVM Memory Configuration: -XX:MaxDirectMemorySize=10M -Xmx24241905K -XX:MaxMetaspaceSize=74634K -XX:ReservedCodeCacheSize=240M -Xss1M (Total Memory: 24828540K, Thread Count: 250, Loaded Class Count: 10763, Headroom: 0%)
Enabling Java Native Memory Tracking
Adding 128 container CA certificates to JVM trusts store
Spring Cloud Bindings Enabled
Picked up JAVA_TOOL_OPTIONS: -Djava.security.properties=/layers/paketo-buildpacks_bellsoft-liberica/java-security-properties/java-security.properties -XX:+ExitOnOutOfMemoryError -XX:ActiveProcessorCount=4 -XX:MaxDirectMemorySize=10M -Xmx24241905K -XX:MaxMetaspaceSize=74634K -XX:ReservedCodeCacheSize=240M -Xss1M -XX:+UnlockDiagnosticVMOptions -XX:NativeMemoryTracking=summary -XX:+PrintNMTStatistics -Dorg.springframework.cloud.bindings.boot.enable=true
```

Right Pane (PowerShell Help):

```
PowerShell 7.2.2
Copyright (c) Microsoft Corporation.

https://aka.ms/powershell
Type 'help' to get help.

Loading personal and system profiles took 712ms.
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"}]
```

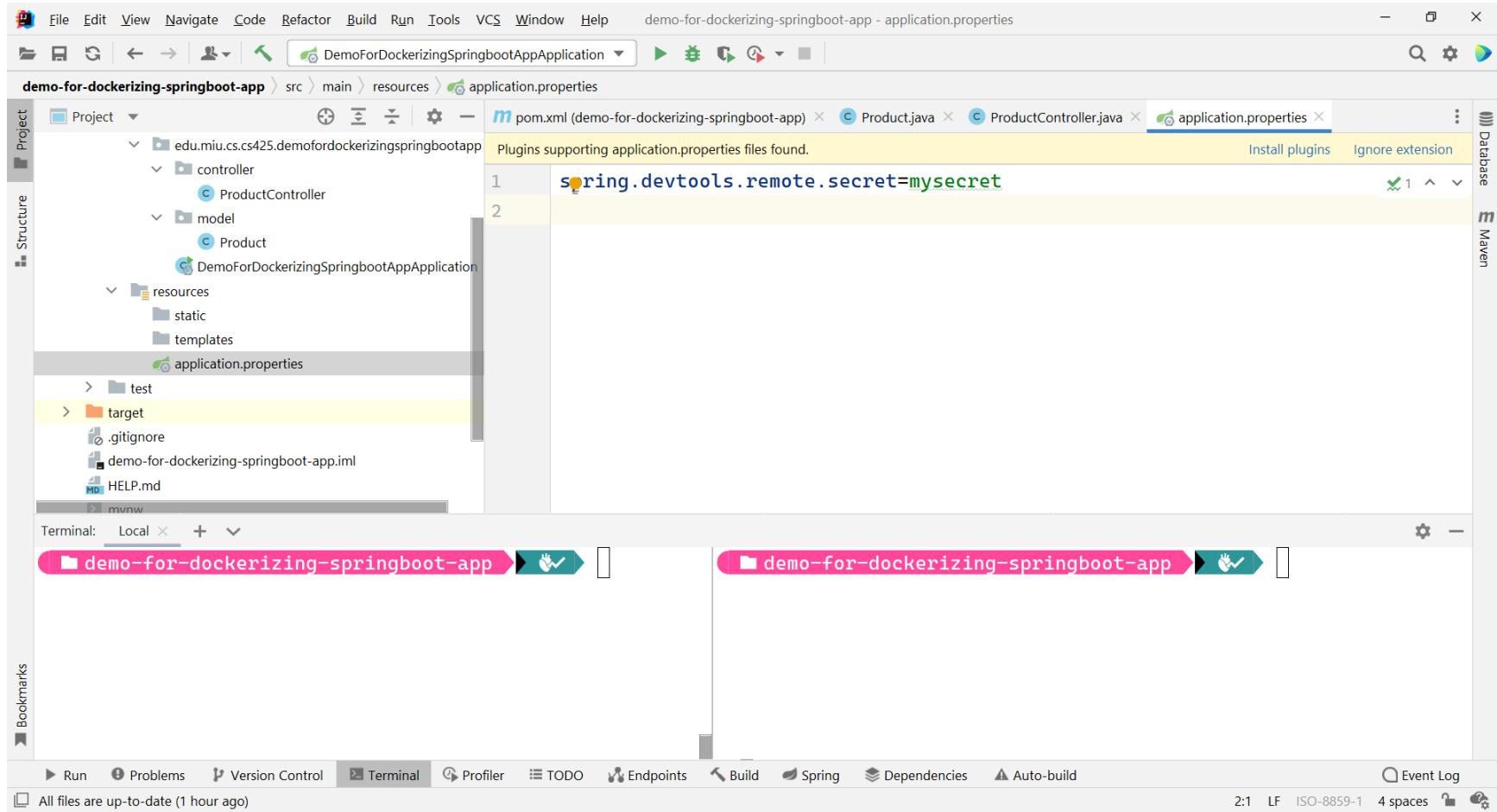
Next: Enable DevTools auto-reload in the container

- 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) × Product.java × 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 project 'demo-for-dockerizing-springboot-app' open. The left sidebar displays the project structure, including the 'controller' and 'model' packages under 'edu.miu.cs.cs425.demofordockerizingspringbootapp'. The 'resources' folder contains 'static' and 'templates' subfolders, along with 'application.properties'. The 'target' folder is highlighted in yellow. The bottom navigation bar includes tabs for Run, Problems, Version Control, Terminal, Profiler, TODO, Endpoints, Build, Spring, Dependencies, Auto-build, and Event Log.

```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <configuration>
                <excludeDevtools>false</excludeDevtools>
            </configuration>
        </plugin>
    </plugins>
</build>
```

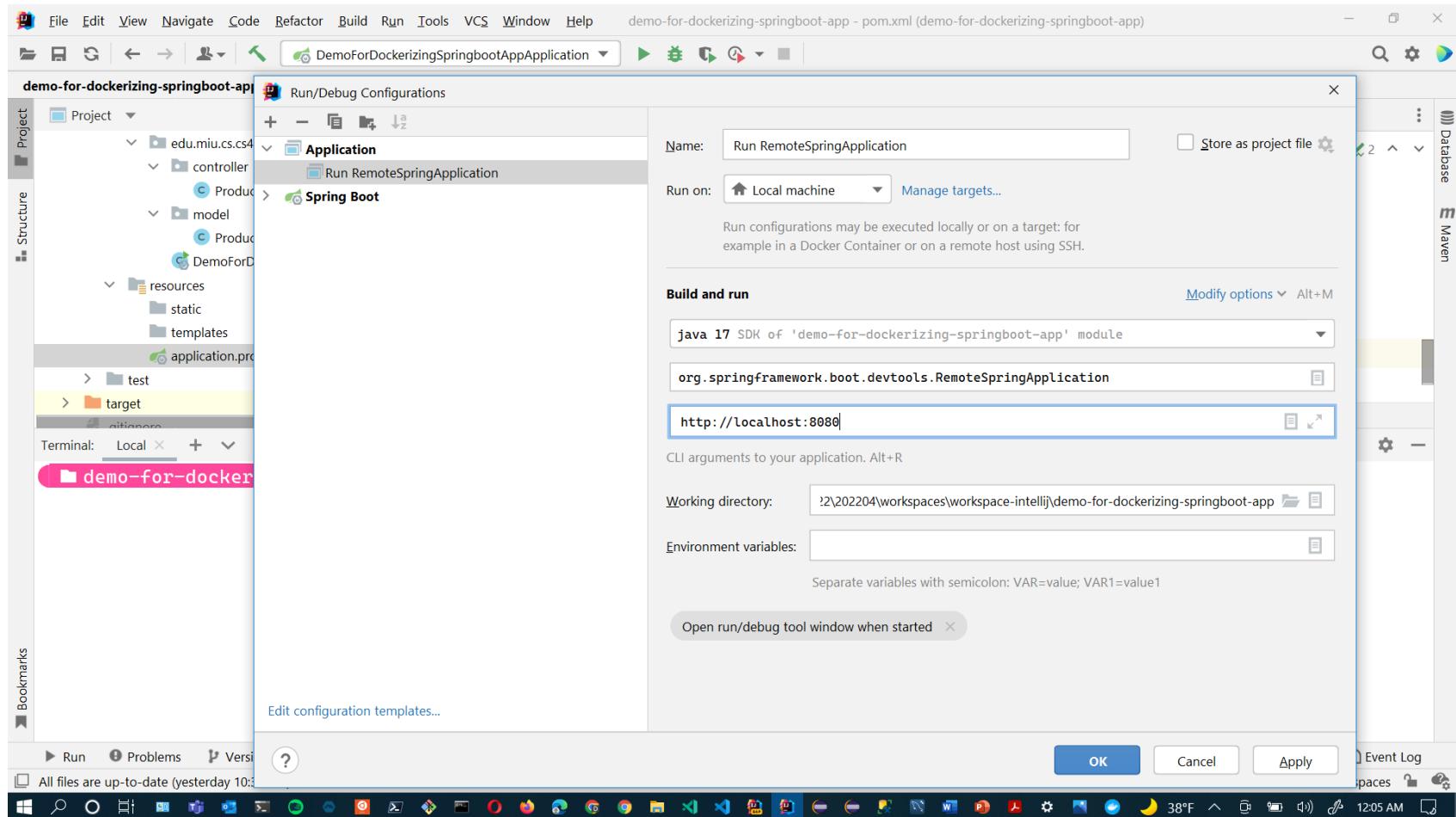
Stop the container, delete it and re-build image again

The screenshot shows a terminal window with two tabs. The left tab is titled 'PowerShell 7 - Core' and contains the command 'mv'. The right tab is titled 'demo-for-dockerizing-springboot-app' and contains the command 'spring-boot:build-image'. Below these tabs is a list of Docker containers from the command 'docker ps --all'. The list includes:

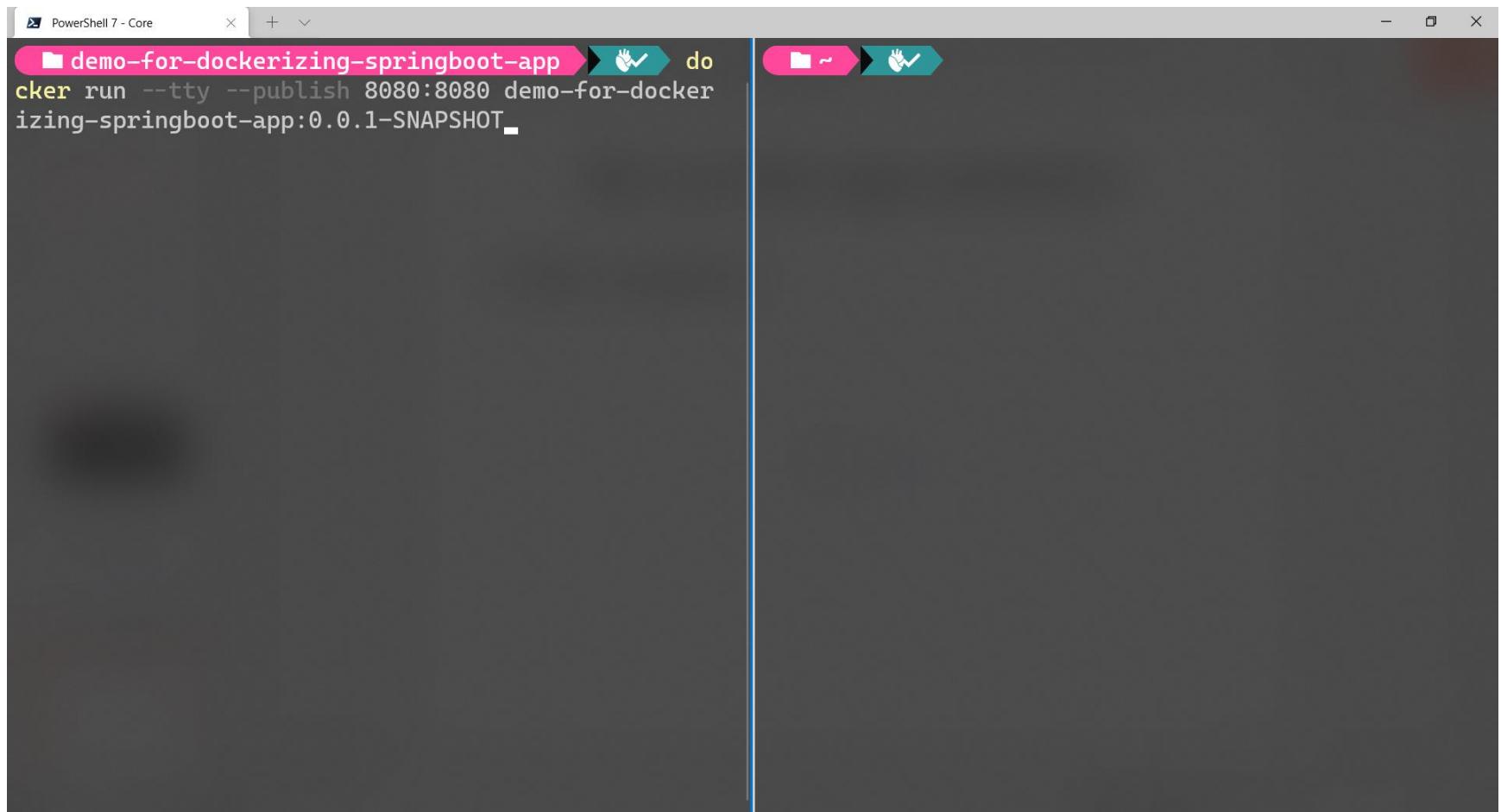
| CONTAINER ID | IMAGE | COMMAND | CREATED | STATUS |
|--------------|---------------------------------|------------------------|------------|-------------------------|
| | | | | |
| 98ac49d3f560 | 562c9bc24a08 | docker-entrypoint.s... | 5 days ago | Exited (0) 5 days ago |
| cff62ae8ea3b | helloworld-java-cli-app2:latest | java -jar ./hellowo... | 5 days ago | Exited (143) |
| f55eadfbfa8 | helloworld-java-cli-app:latest | java -jar ./hellowo... | 5 days ago | Exited (0) 5 days ago |
| 463f72e32d07 | alpine/git | git clone https://g... | 6 days ago | Exited (128) 6 days ago |

At the bottom of the list, there is a command '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:

```
docker run --tty --publish 8080:8080 demo-for-dockerizing-springboot-app:0.0.1-SNAPSHOT
```

The command uses the Docker CLI to run a container from the image "demo-for-dockerizing-springboot-app:0.0.1-SNAPSHOT". The "--tty" option allows the container to interact with the host's terminal, and the "--publish" option maps port 8080 on the host to port 8080 on the container.

Run the RemoteSpringApplication

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)
- Toolbar:** Run RemoteSpringApplication, Stop 'Run RemoteSpringApplication' Ctrl+F2, ProductController.java, application.properties.
- Project Tool Window:** Shows the project structure with packages .idea, .mvn, src (main.java, edu.miu.cs.cs425.demofordockerizingspringbootapp (controller, model, DemoForDockerizingSpringbootAppApplication), resources (static, templates)), and application.properties.
- Code Editor:** pom.xml (partial code shown):

```
</plugins>
</build>
</project>
```

- Run Tool Window:** Shows the run configuration "Run RemoteSpringApplication".
- Terminal:** Displays log output from the application start:2022-04-02 00:09:49.219 INFO 4612 --- [main] o.s.b.devtools.RemoteSpringApplication : Starting RemoteS
2022-04-02 00:09:49.224 INFO 4612 --- [main] o.s.b.devtools.RemoteSpringApplication : No active profil
2022-04-02 00:09:49.562 WARN 4612 --- [main] o.s.b.d.r.c.RemoteClientConfiguration : The connection t
2022-04-02 00:09:49.641 INFO 4612 --- [main] o.s.b.d.a.OptionalLiveReloadServer : LiveReload serve
2022-04-02 00:09:49.661 INFO 4612 --- [main] o.s.b.devtools.RemoteSpringApplication : Started RemoteSp
- Bottom Status Bar:** Stop process, Run, Problems, Version Control, Terminal, Profiler, TODO, Endpoints, Build, Spring, Dependencies, Auto-build, Event Log, 51:1, LF, UTF-8, 4 spaces, 38°F, 12:10 AM.

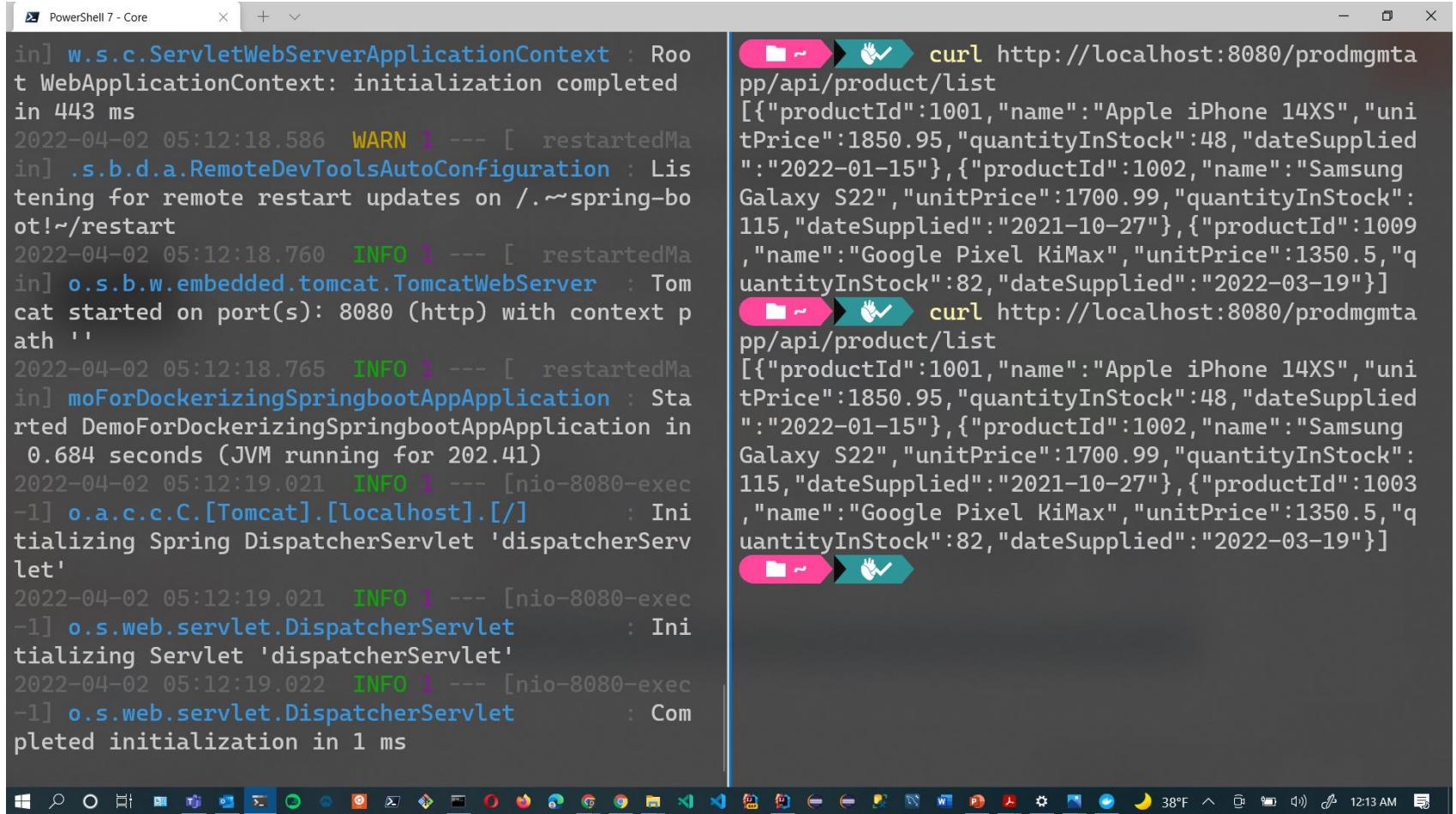
Modify the code and see it reload

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 - ProductController.java
- Toolbars:** Run RemoteSpringApplication, Database, Maven.
- Project Tool Window:** Shows the project structure with packages: .idea, .mvn, src (main, java, edu, miu, miu, cs, cs425, demofordockerizingspringbootapp, controller, ProductController, getProducts), pom.xml, Product.java, ProductController.java (selected), application.properties.
- Code Editor:** Displays the `ProductController.java` file:

```
@GetMapping(value = {"/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)
    );
}
```
- Run Tool Window:** Shows the run configuration: Run RemoteSpringApplication.
- Log Tool Window:** Shows application logs from 2022-04-02 00:09:49 to 00:12:20, indicating the application started, classes were uploaded, and a delayed reload trigger was set up.
- 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 (3 minutes ago), 24:11, CRLF, UTF-8, 4 spaces, 38°F, 12:12 AM.

Verify the output



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