CS489:
Applied Software Development

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

Lesson 10

Introduction to Containers and Containerization using Docker

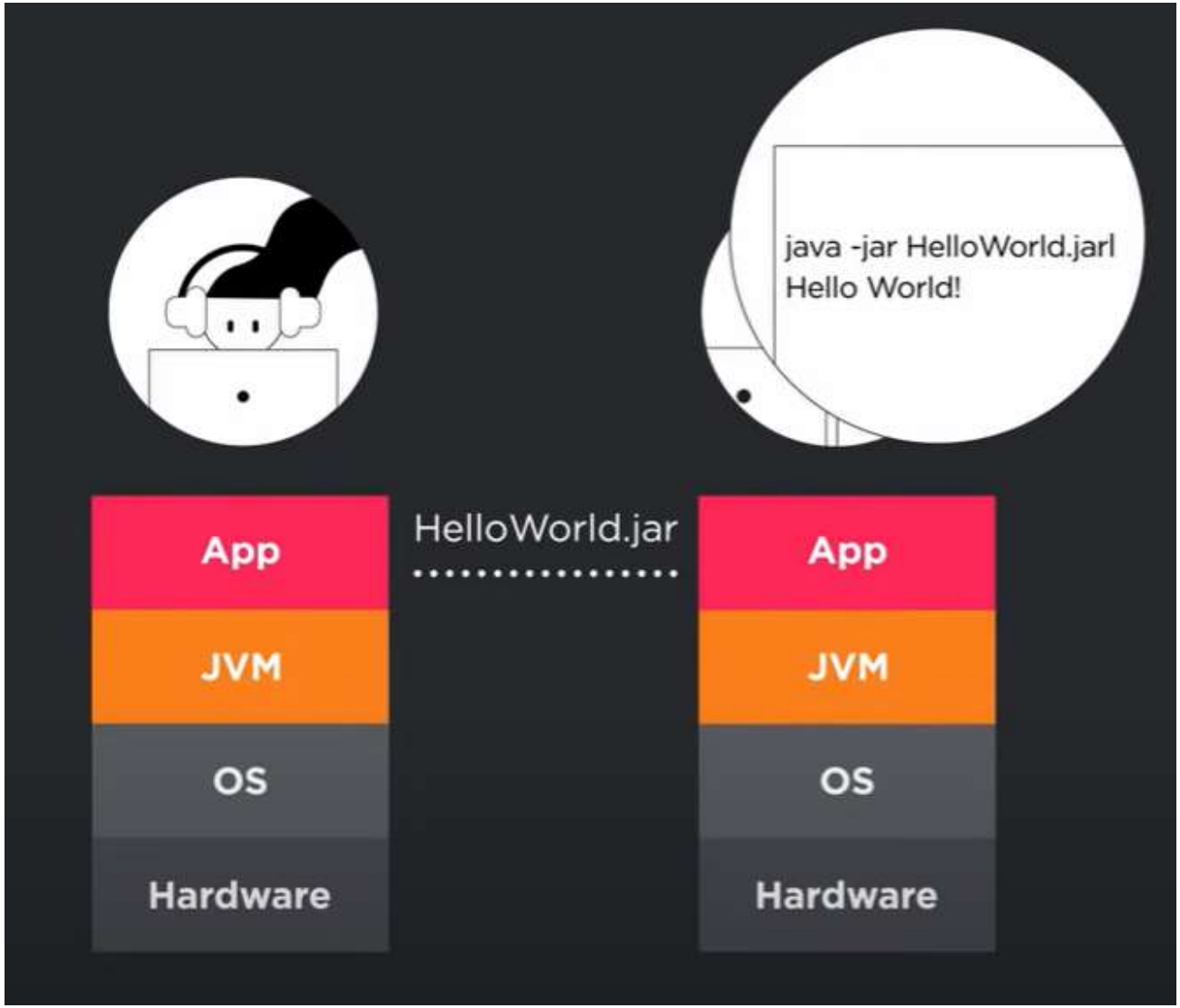
Wholeness

- 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.*

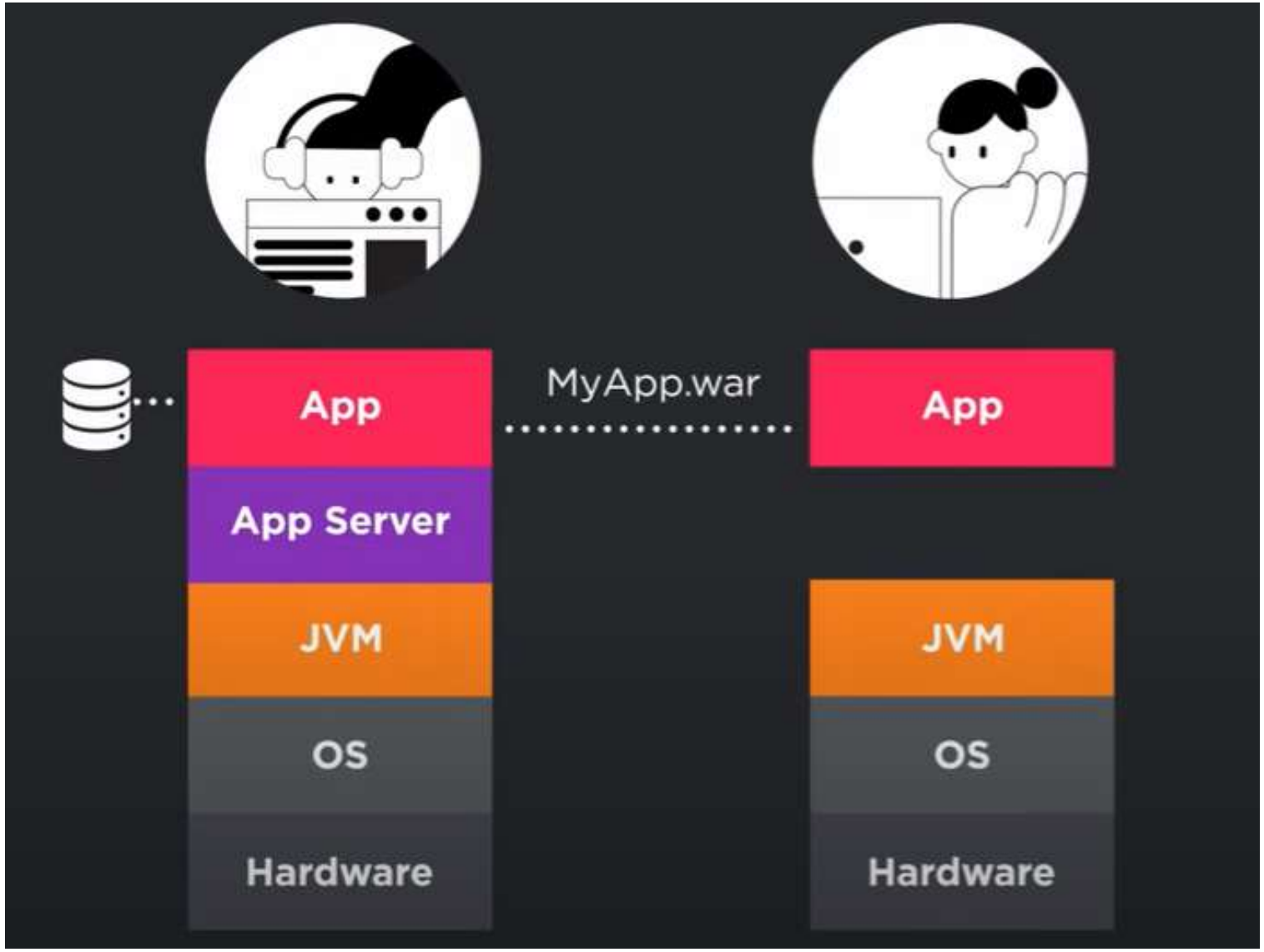
Why we need Containers

- 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)

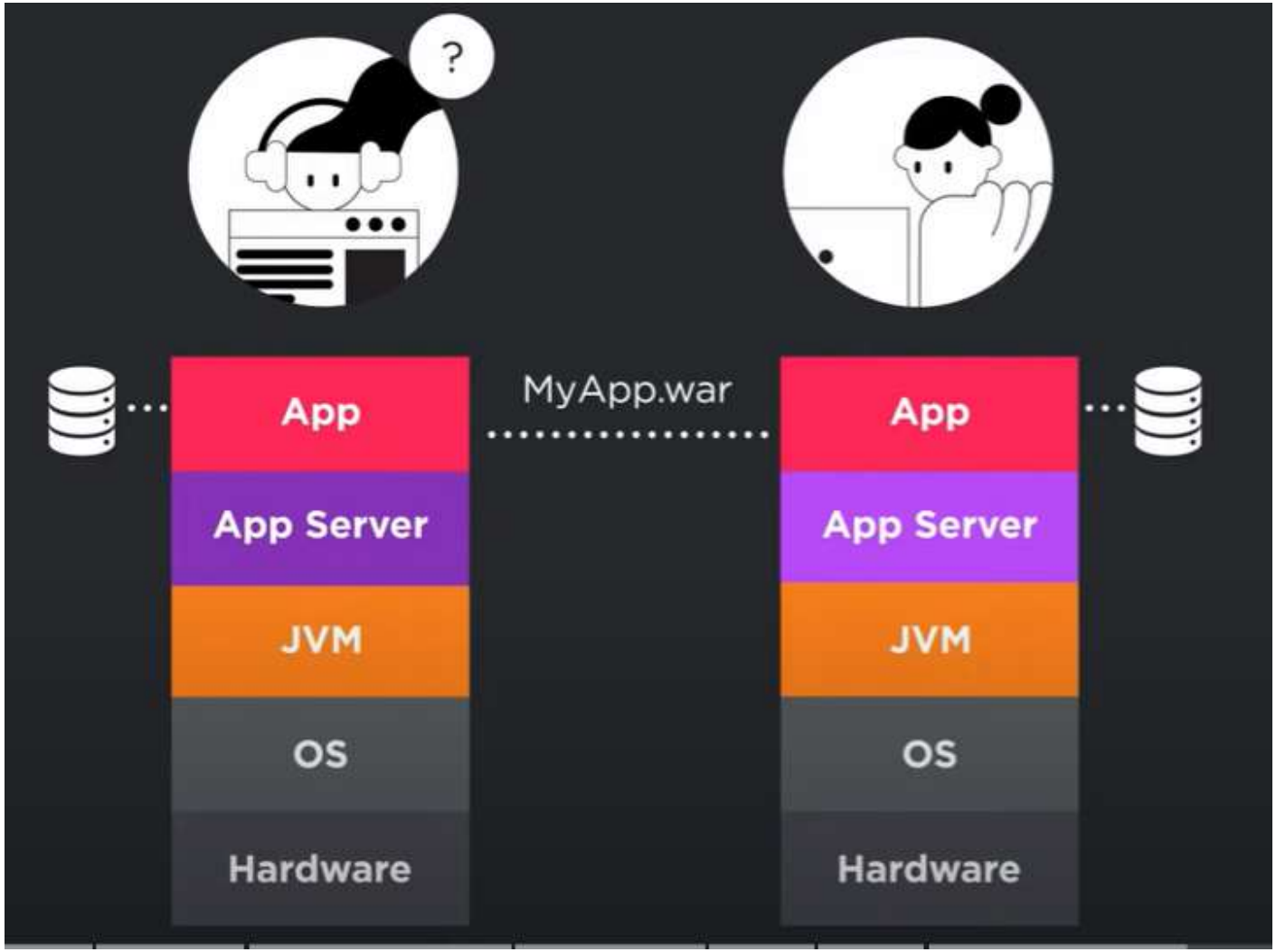
How to deploy/run app across Env



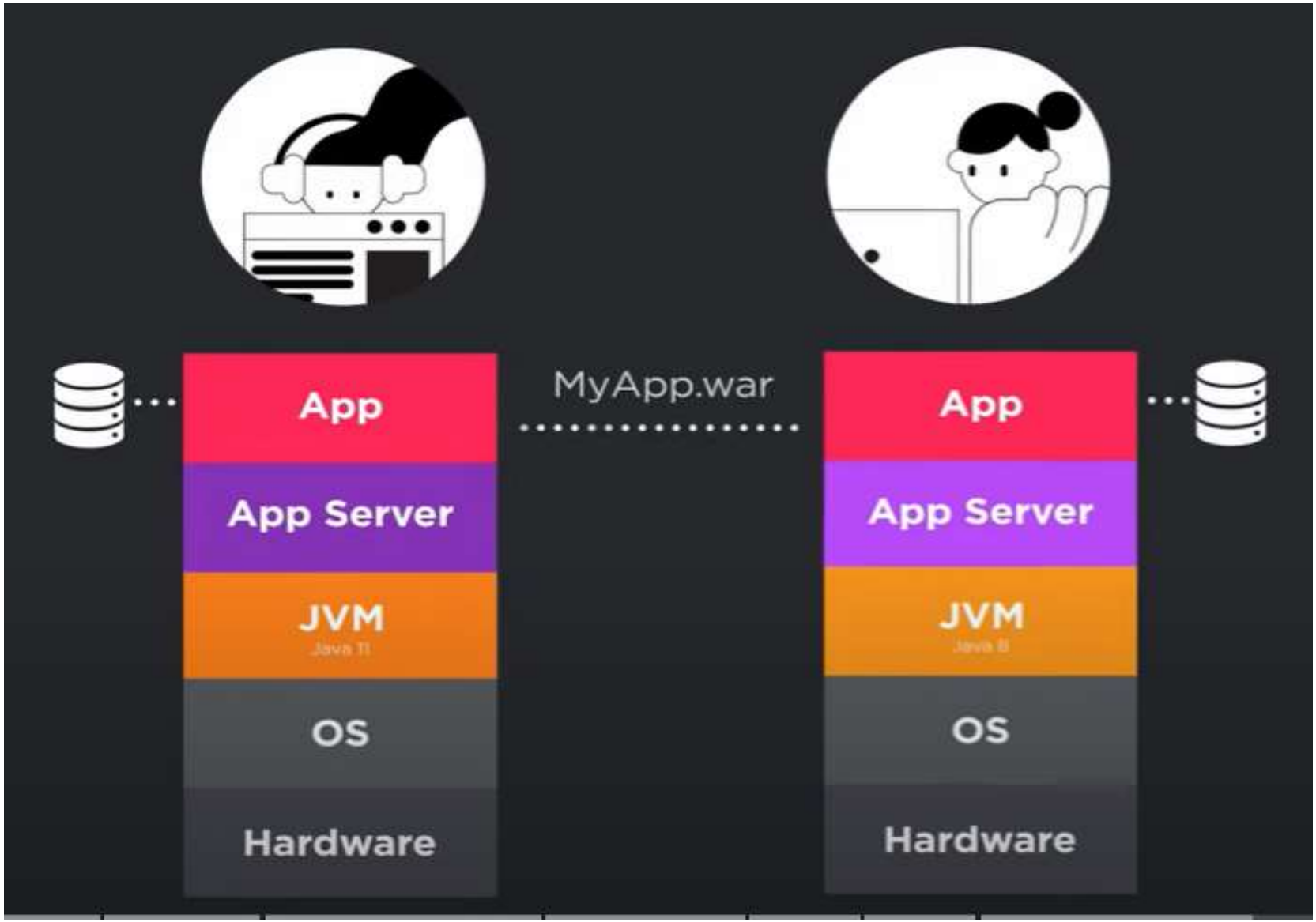
Deploy & run more complex App



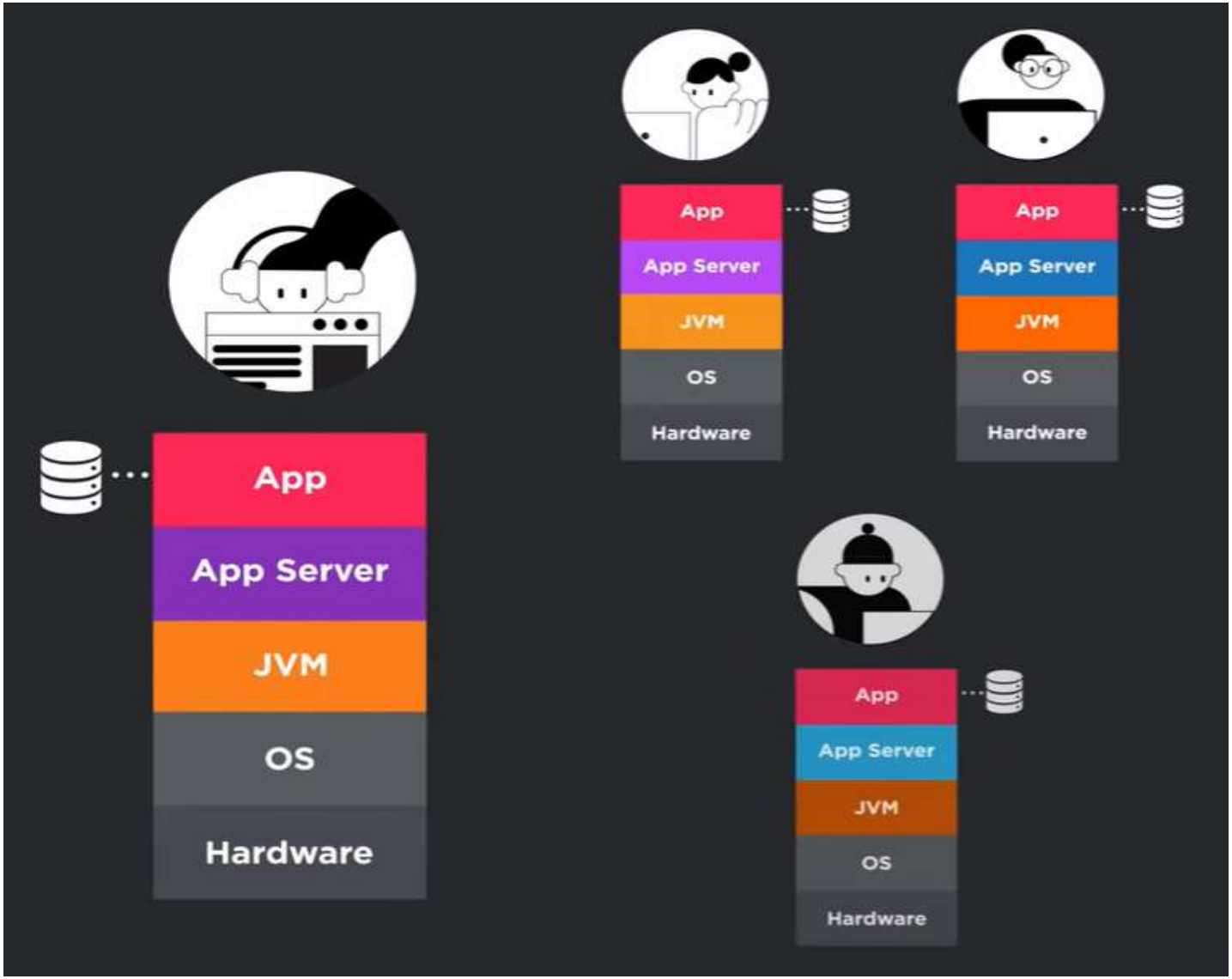
Deploy & run more complex app



Difference in the runtimes

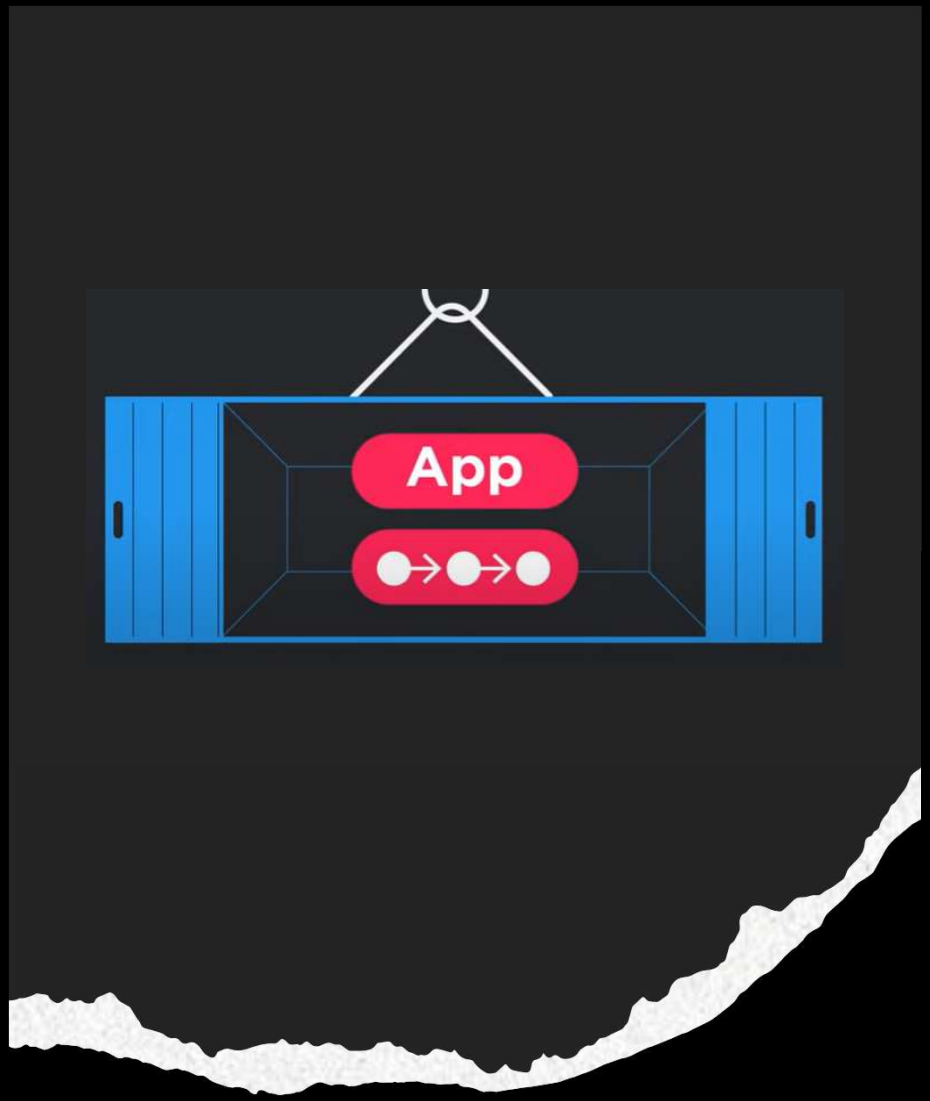


Need to deploy & run more instances



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

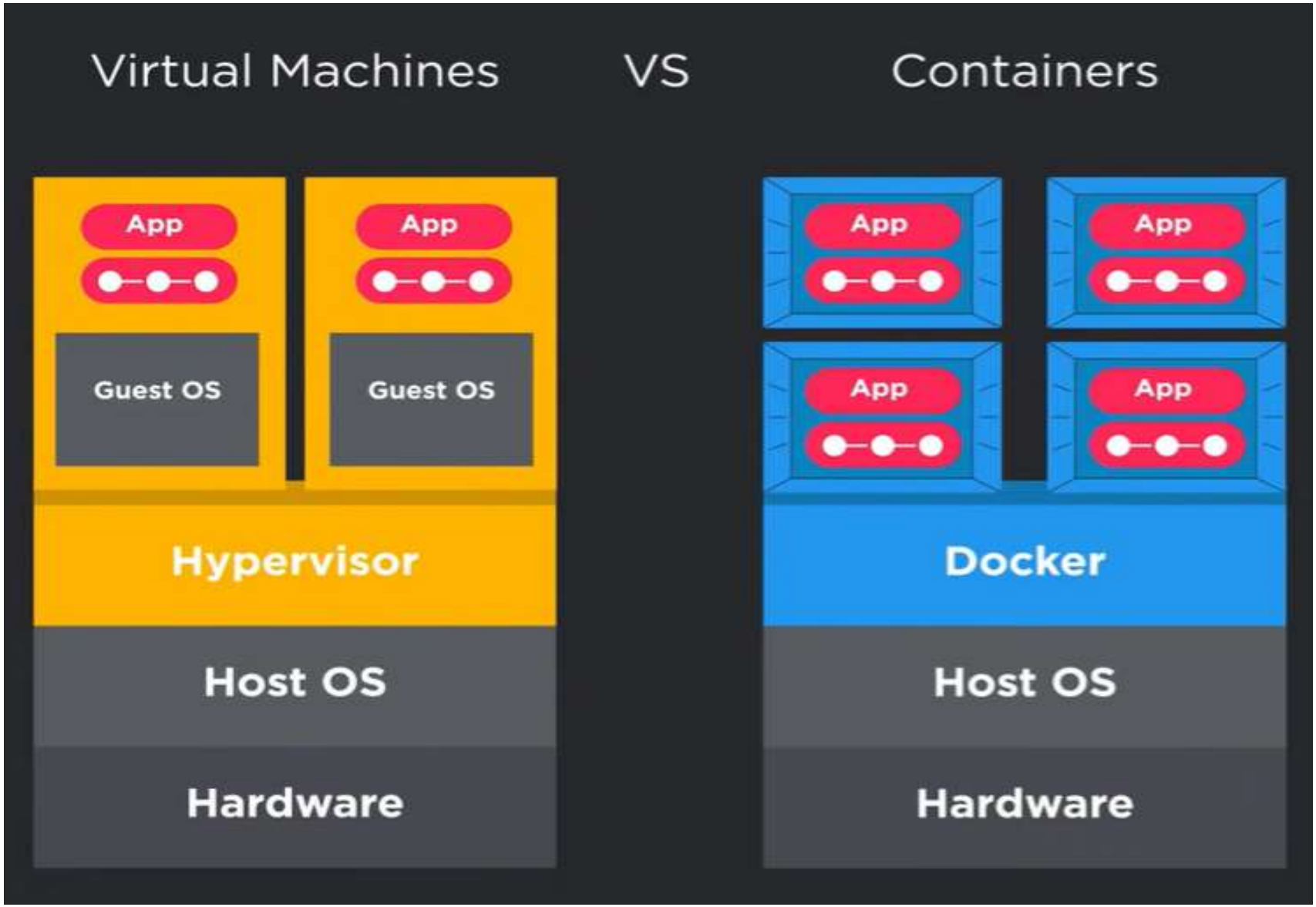
(Containers versus Virtual Machines)

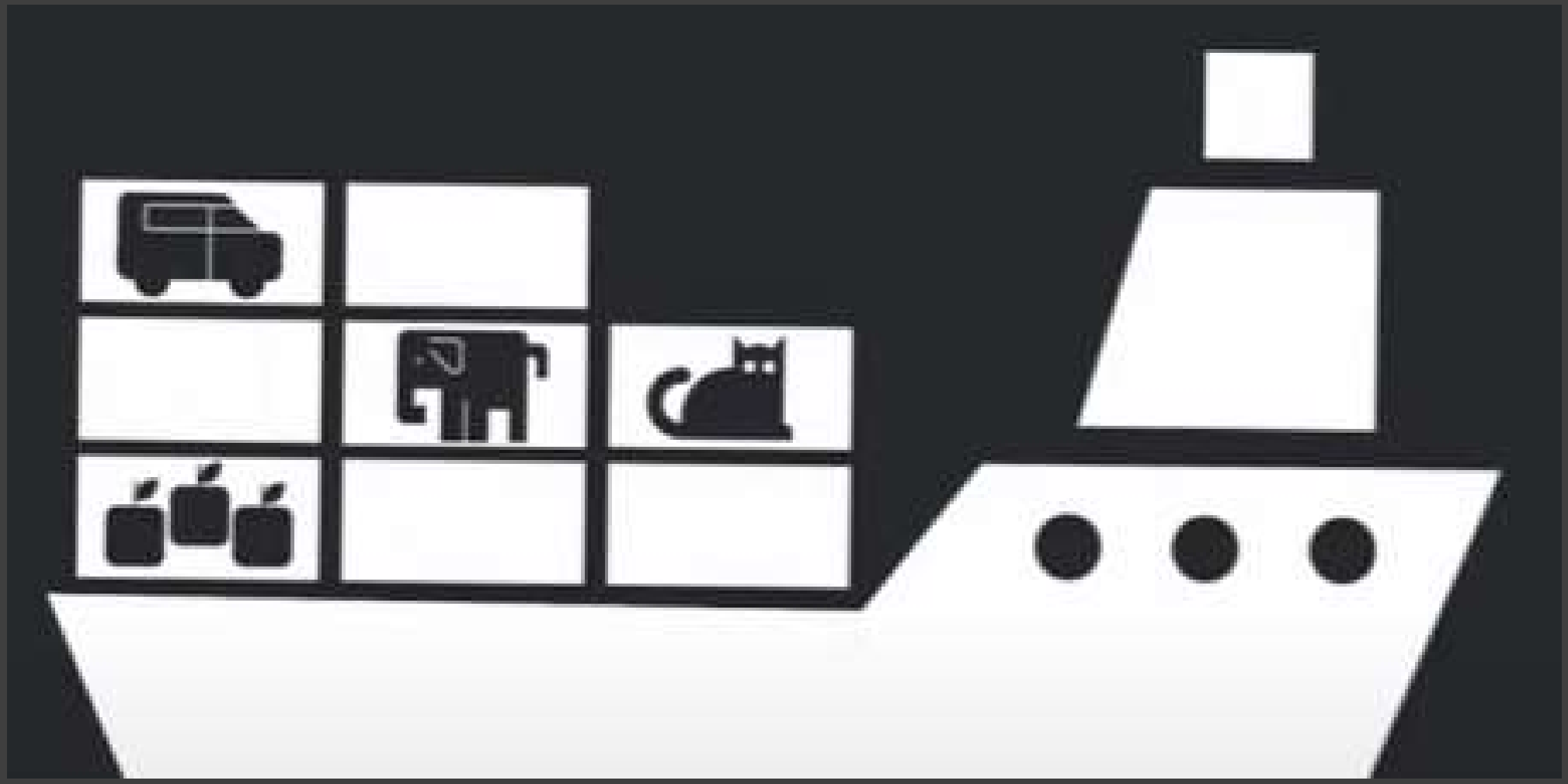
In contrast to Virtualization, Containers are more portable and more resource-efficient than virtual machines (VMs).

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



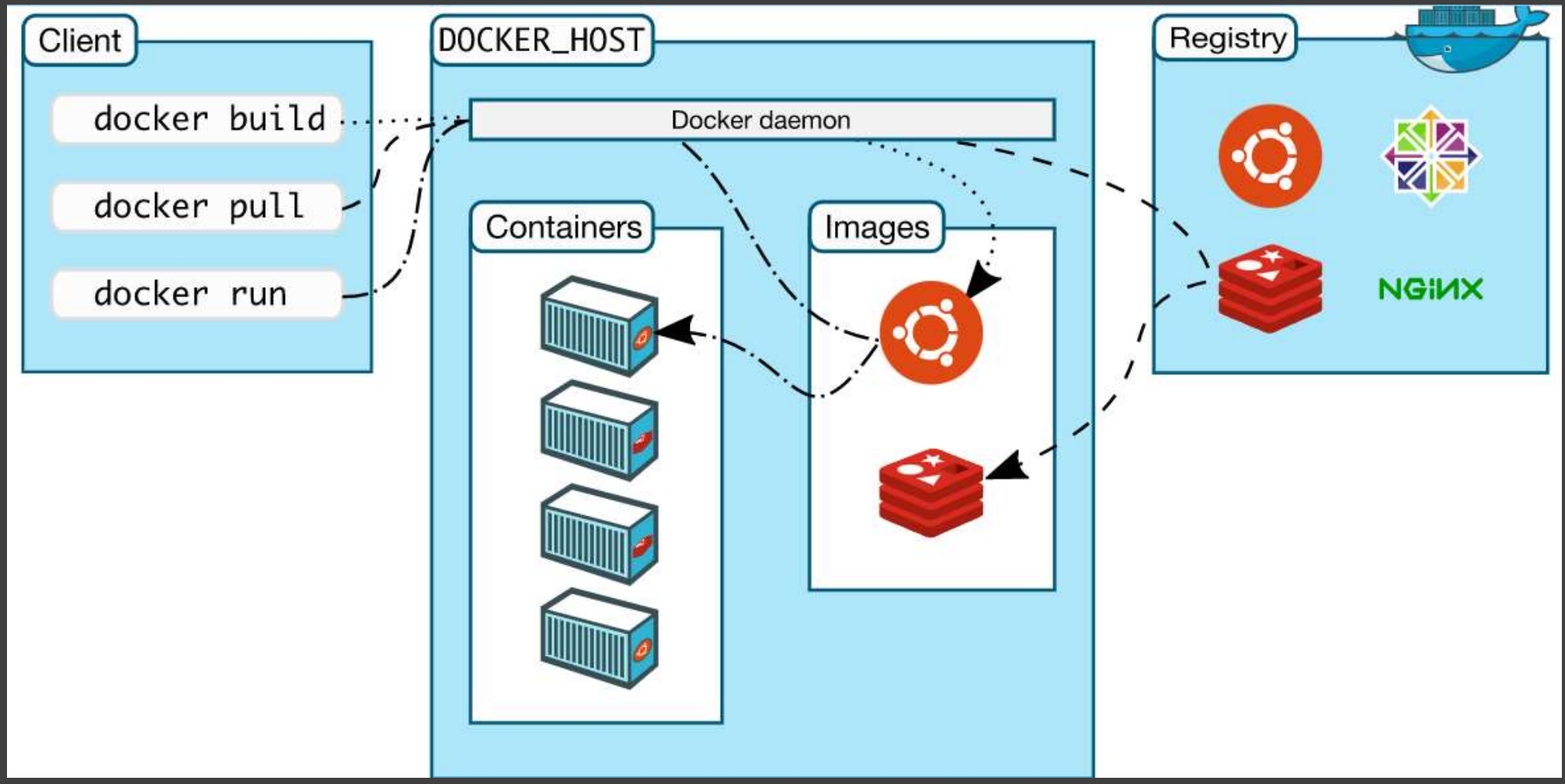


Origin of the name, “Container”

- Derived from the Shipping industry
- Shares quite similar characteristics

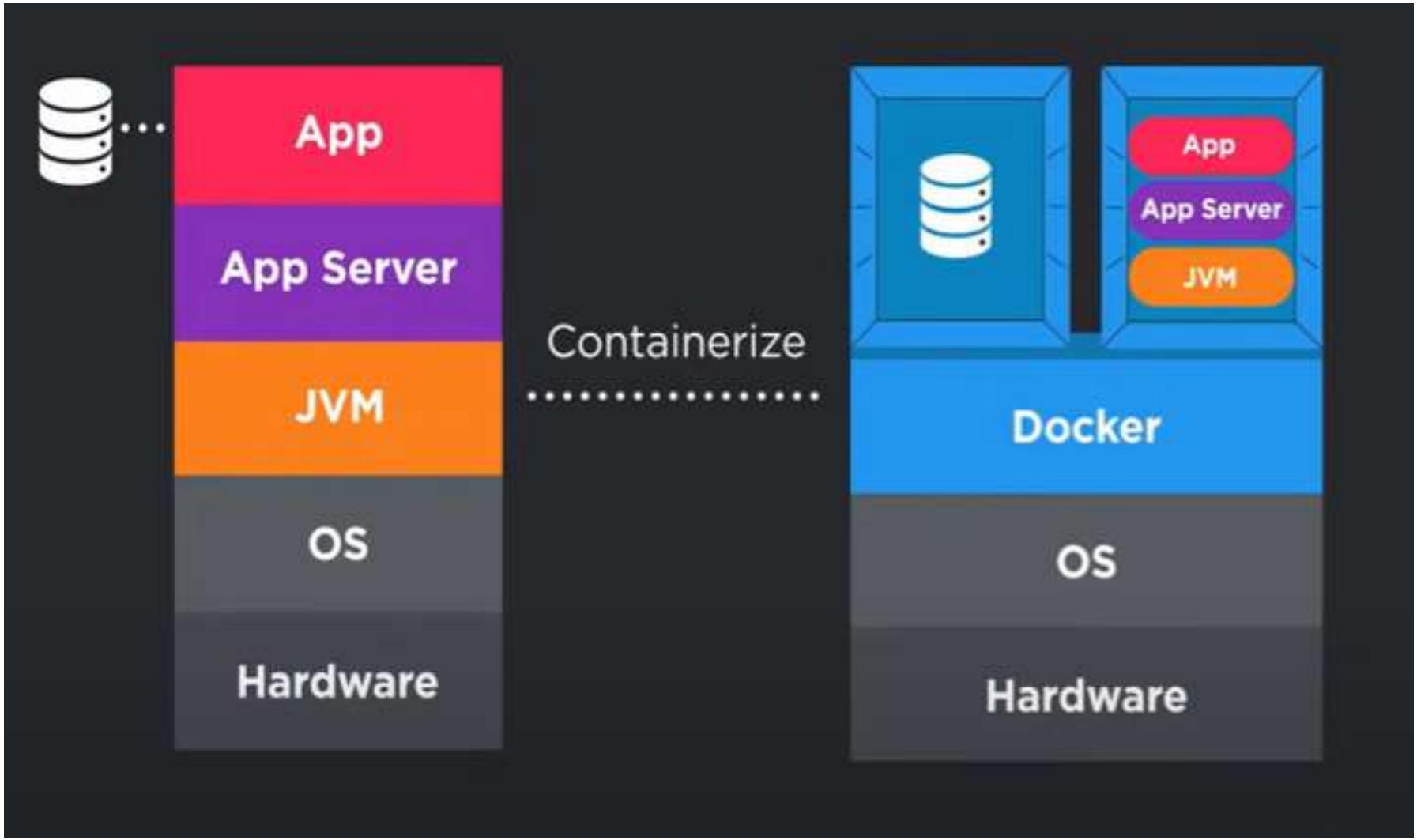
Container technologies

- 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)

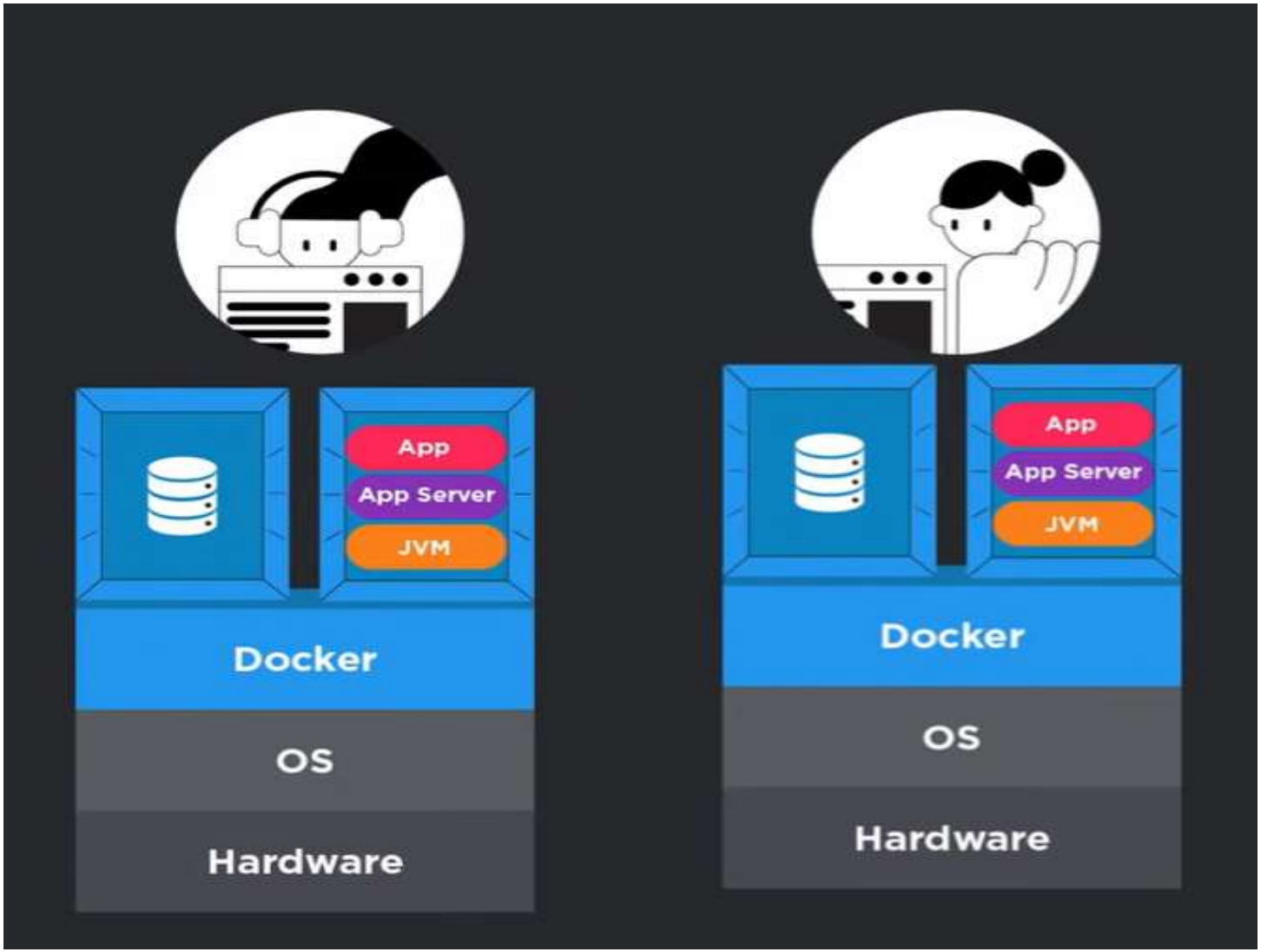


Architecture of Docker

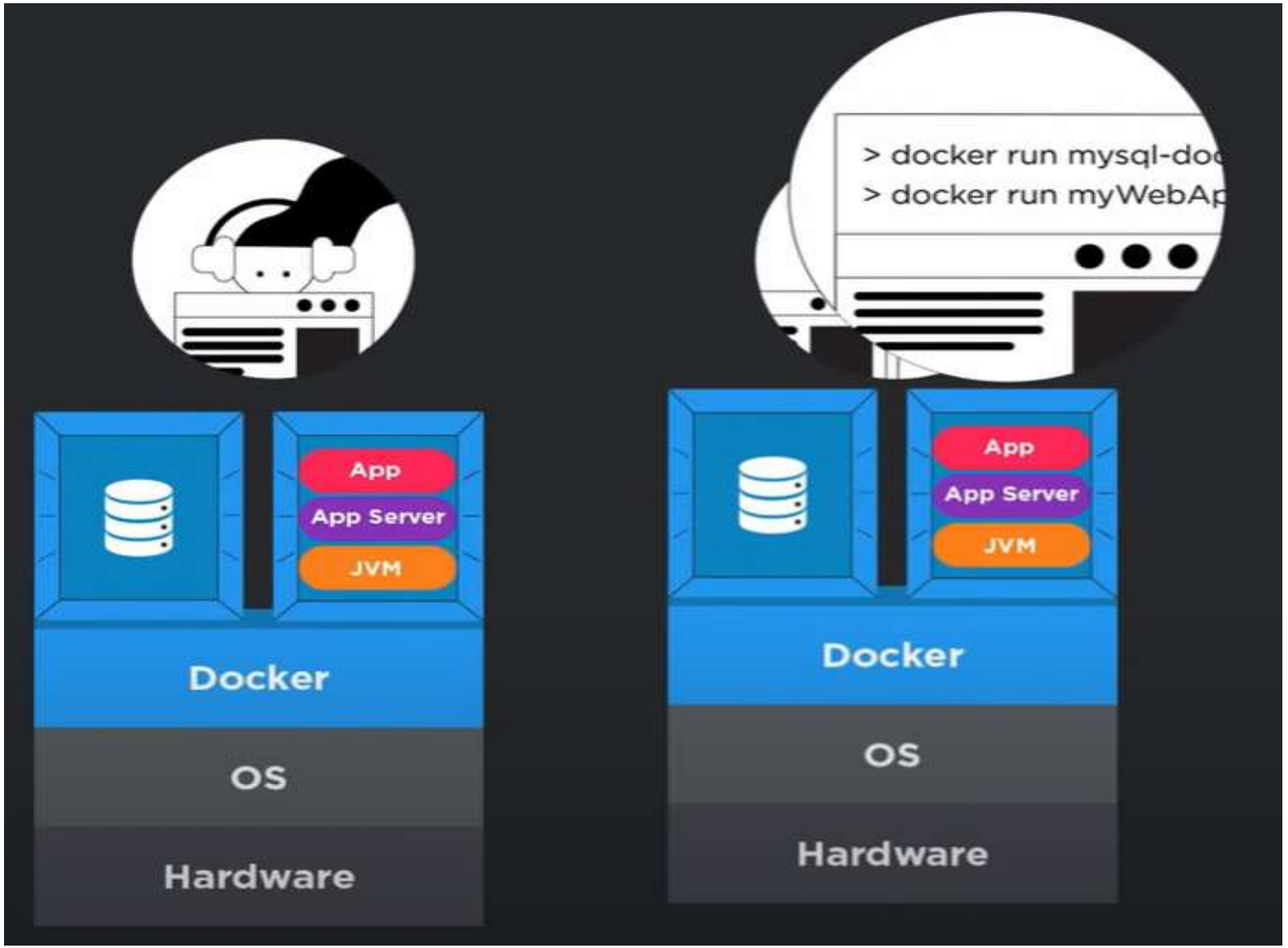
Containerize the app with docker



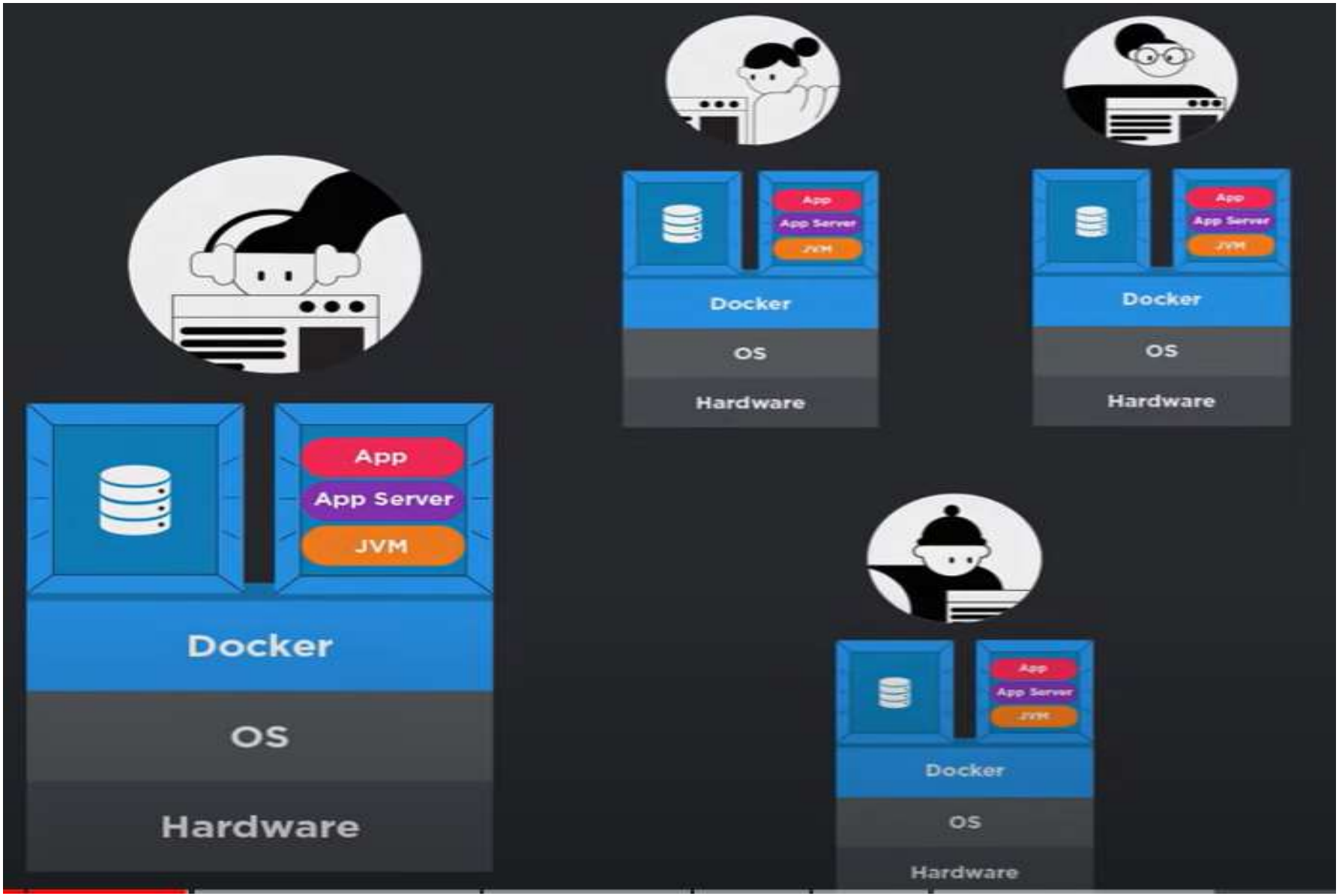
Replicate the configuration



Execute with docker run command

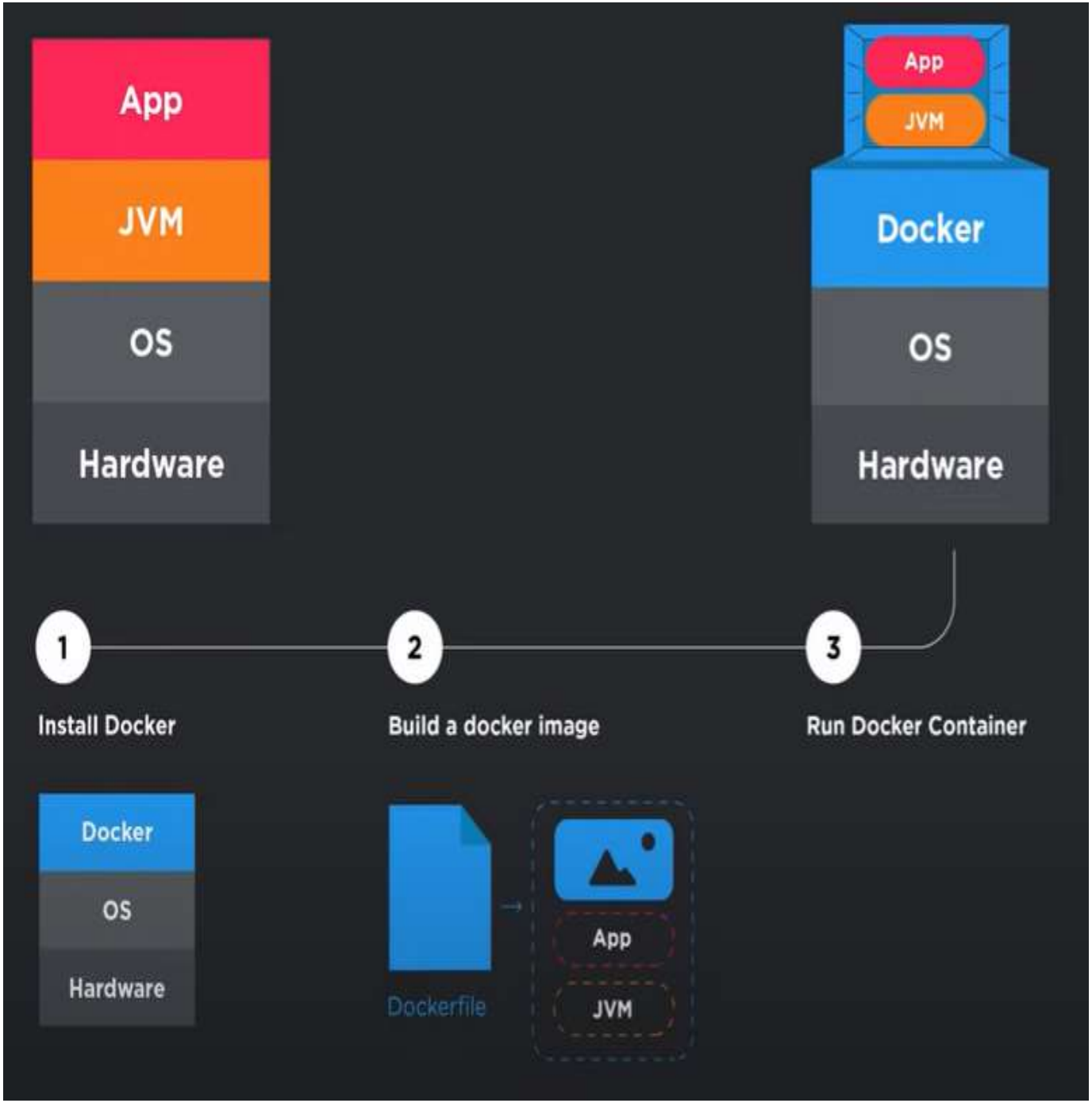


Containerized app – scales-out better

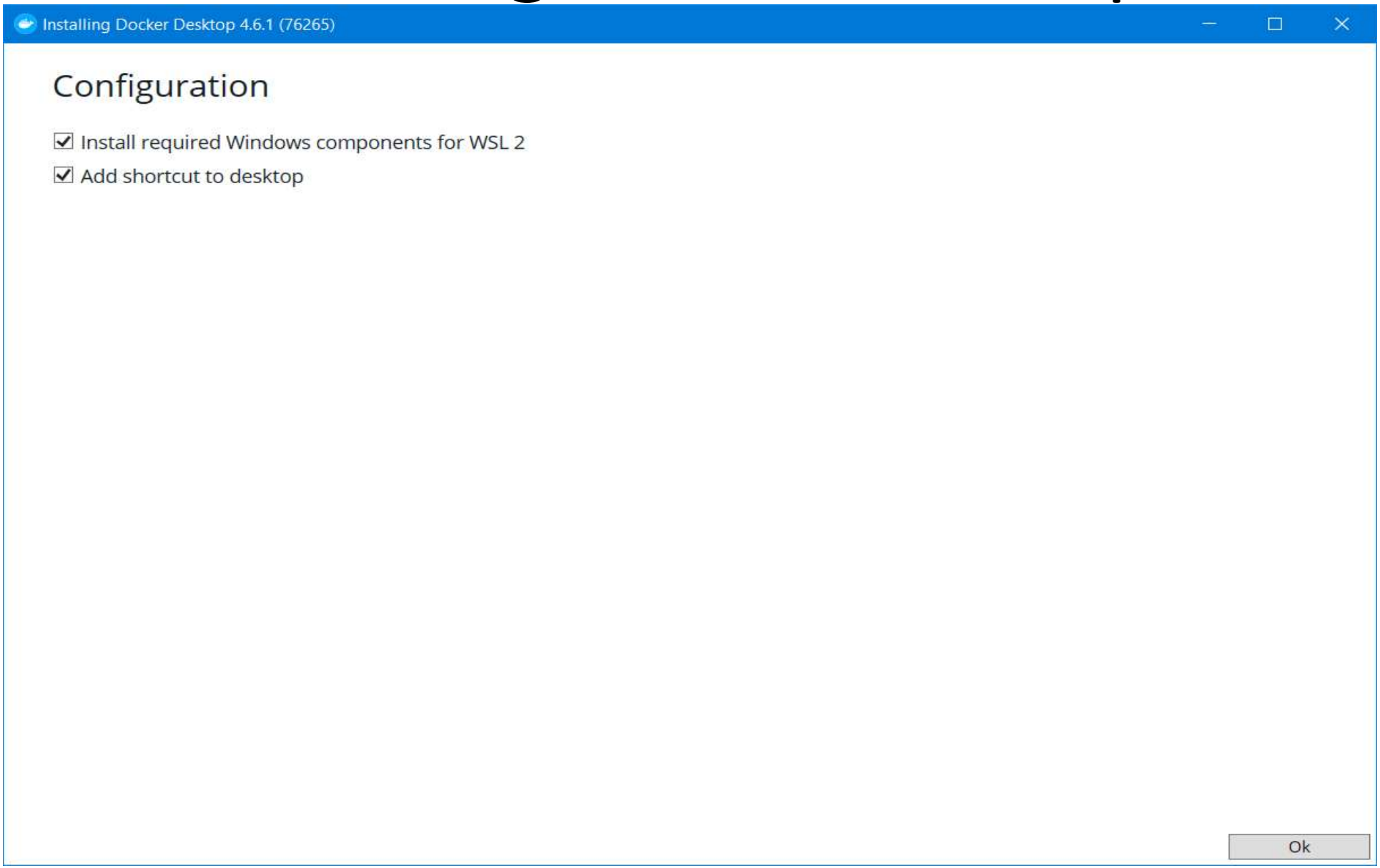


How to containerize software using Docker

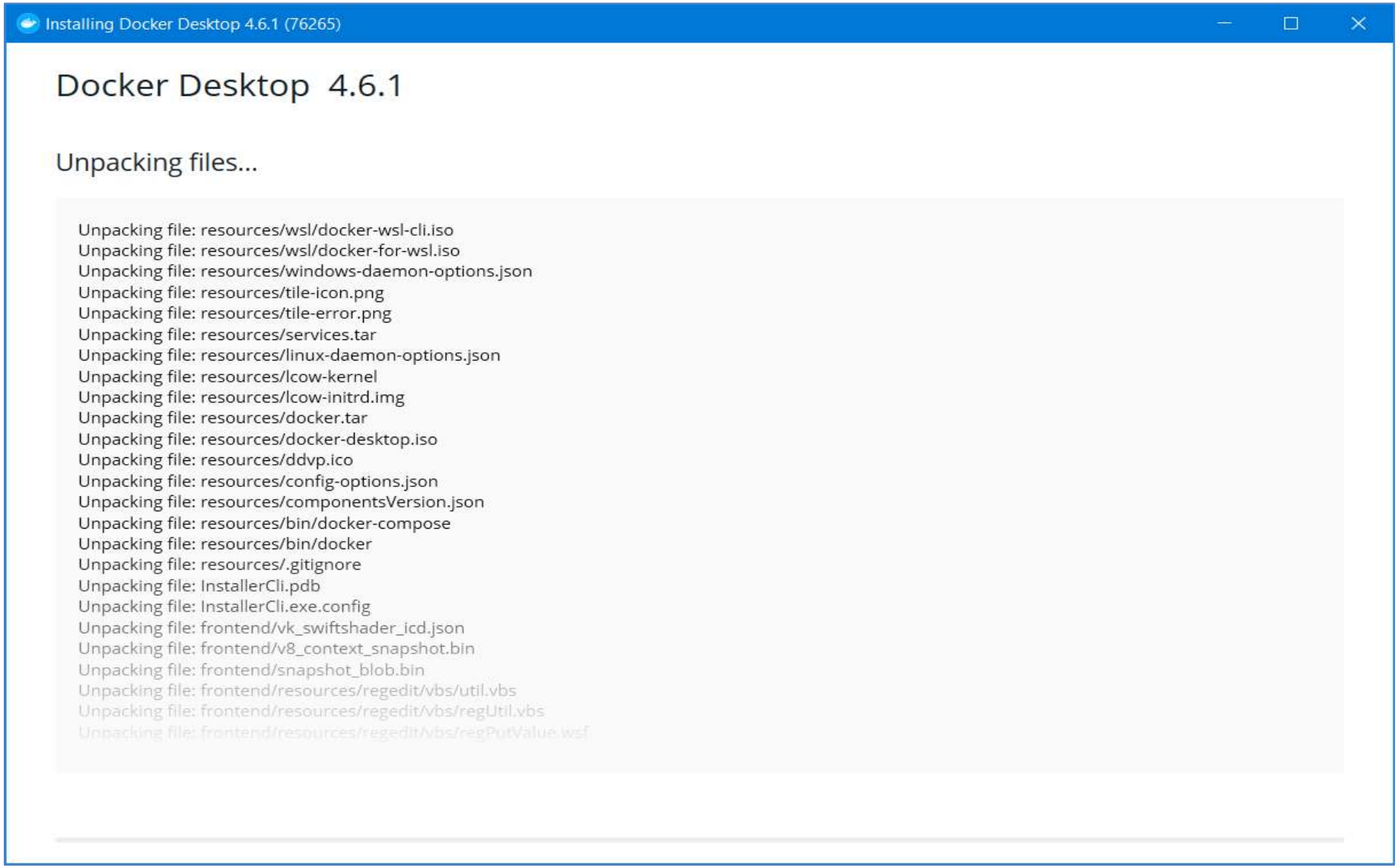
(https://www.docker.com/get-started/)



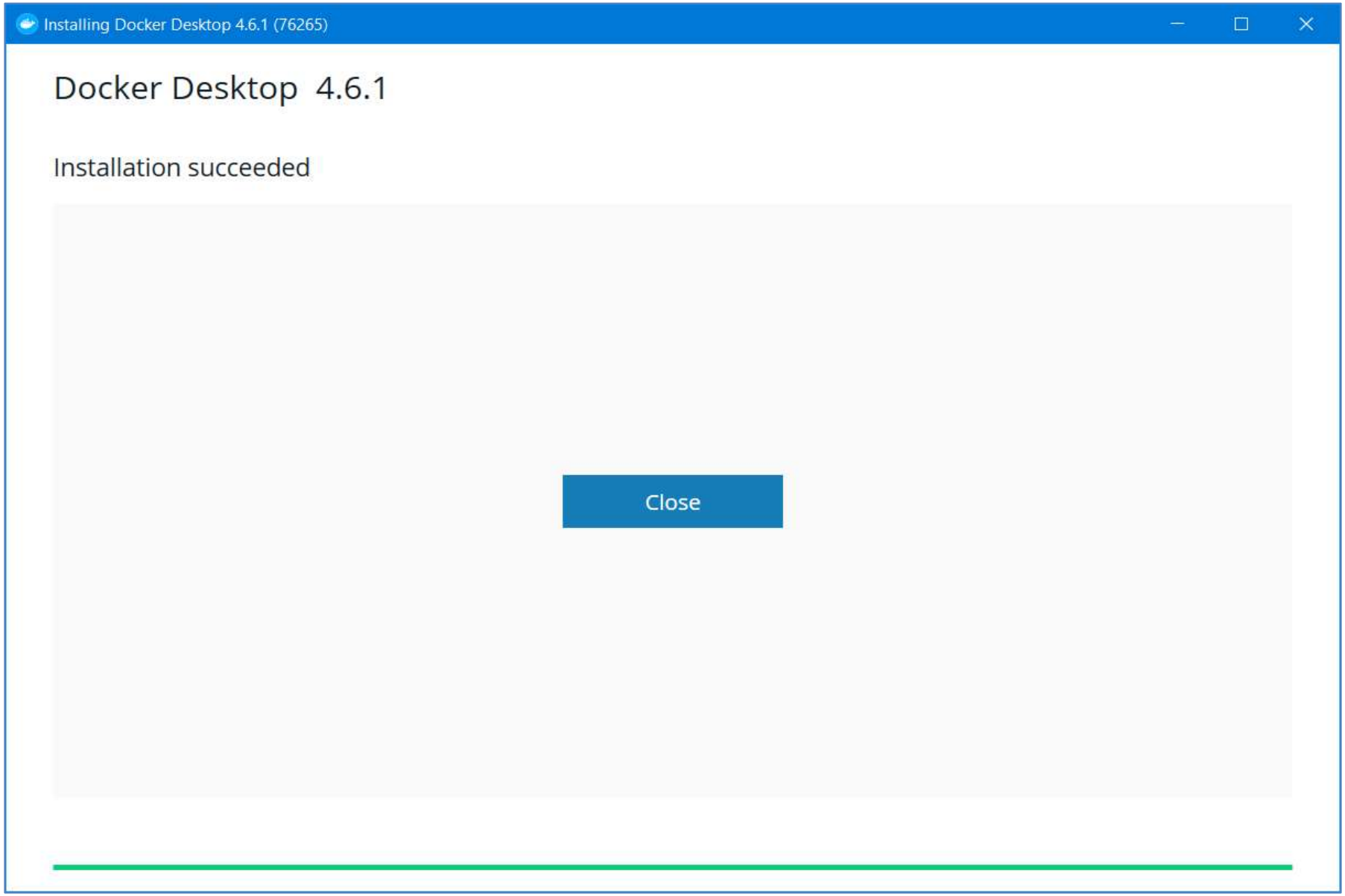
Installing Docker desktop



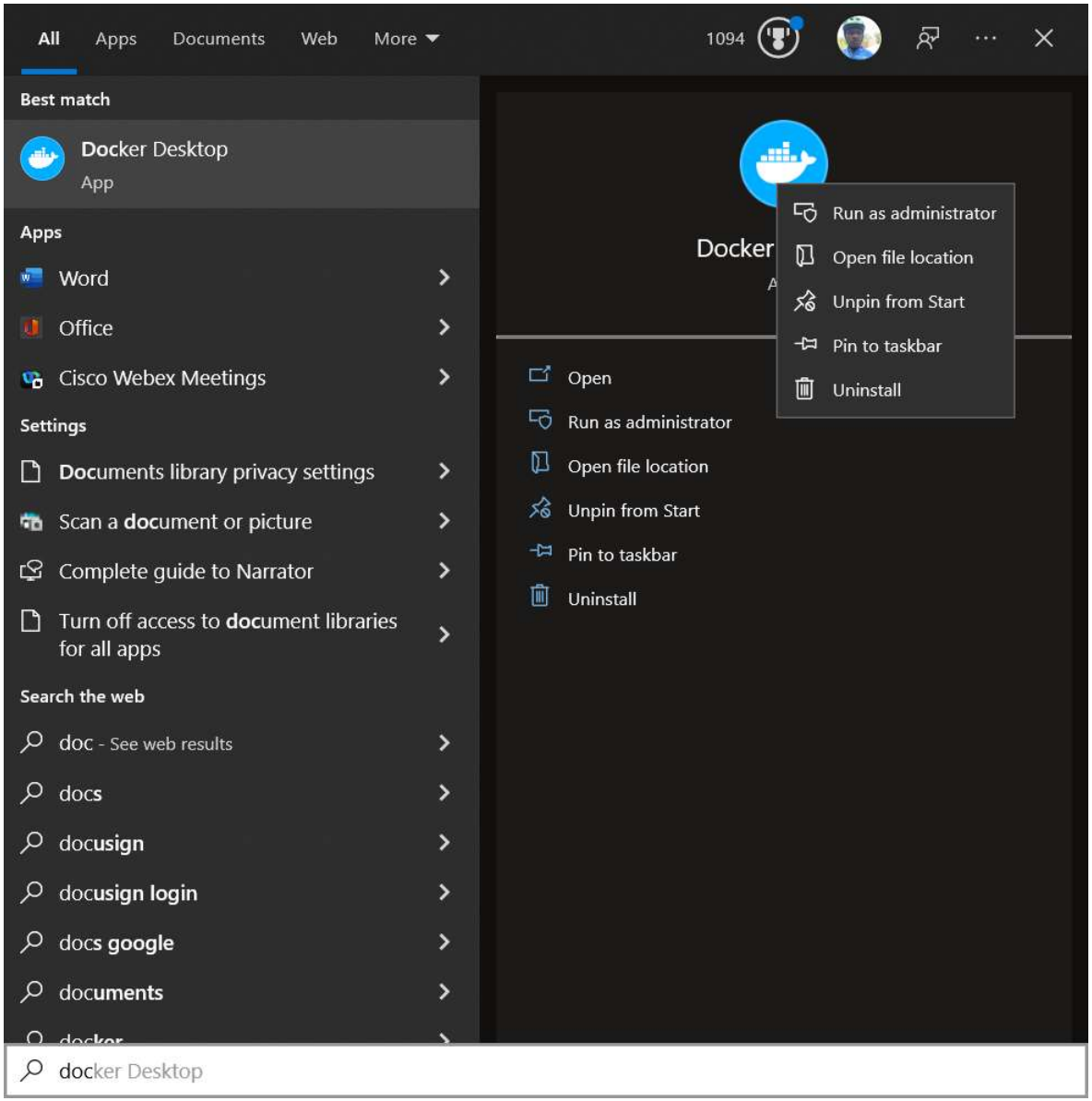
Installing Docker desktop



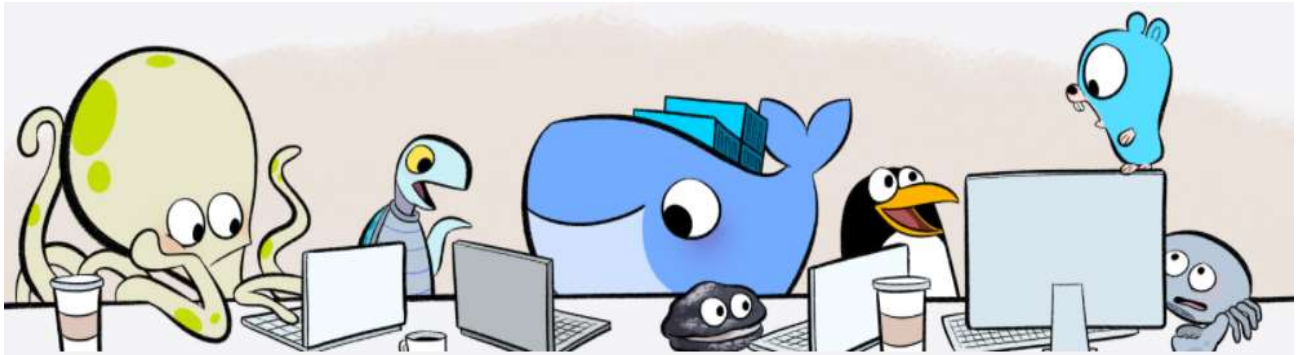
Docker desktop Installation complete



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:

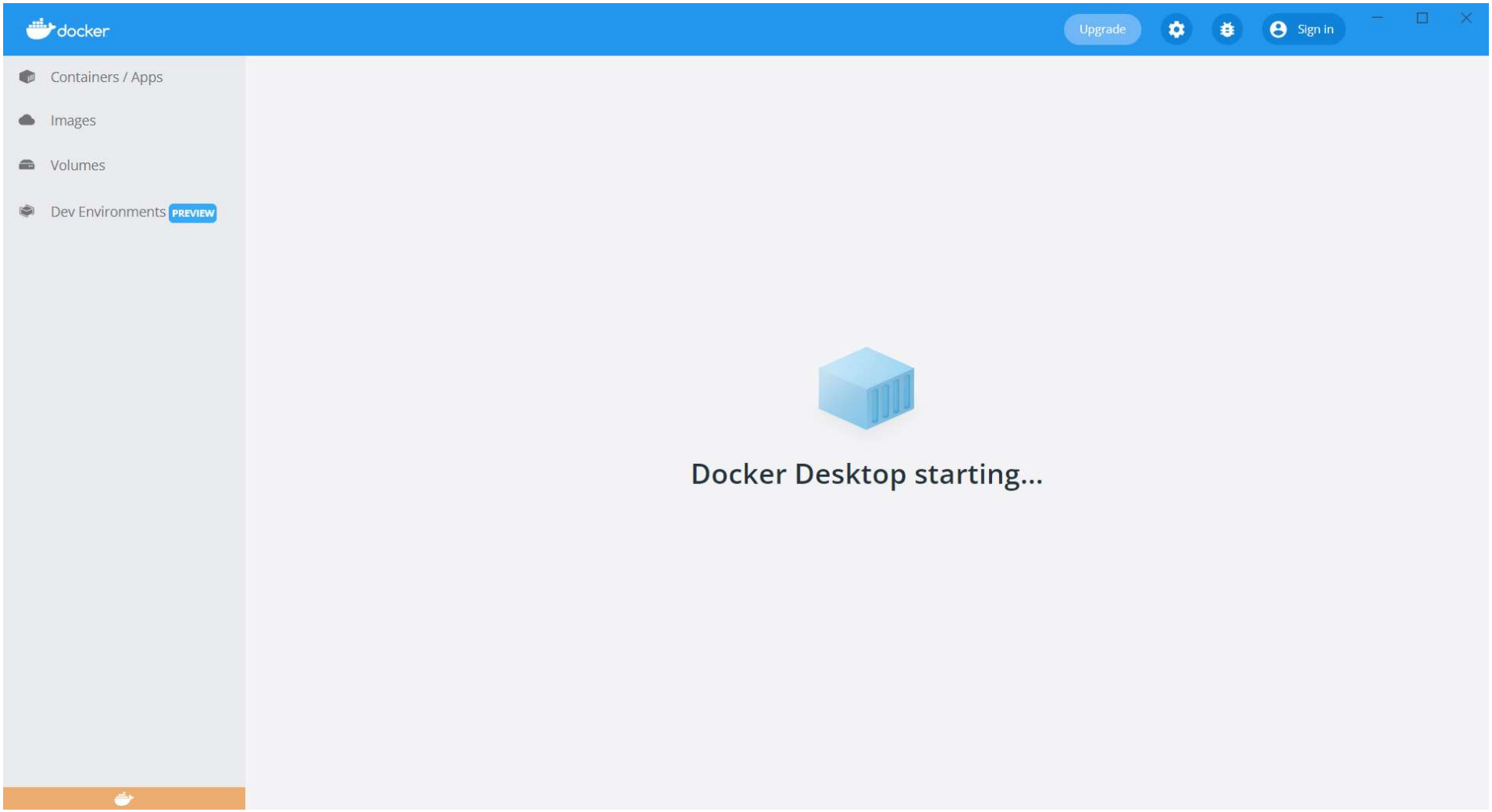
- Our Docker Subscription Service Agreement include a change to the terms of use for Docker Desktop.
  - It **remains free** for small businesses (fewer than 250 employees AND less than $10 million in annual revenue), personal use, education, and non-commercial open source projects.
  - It requires a 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 ☐

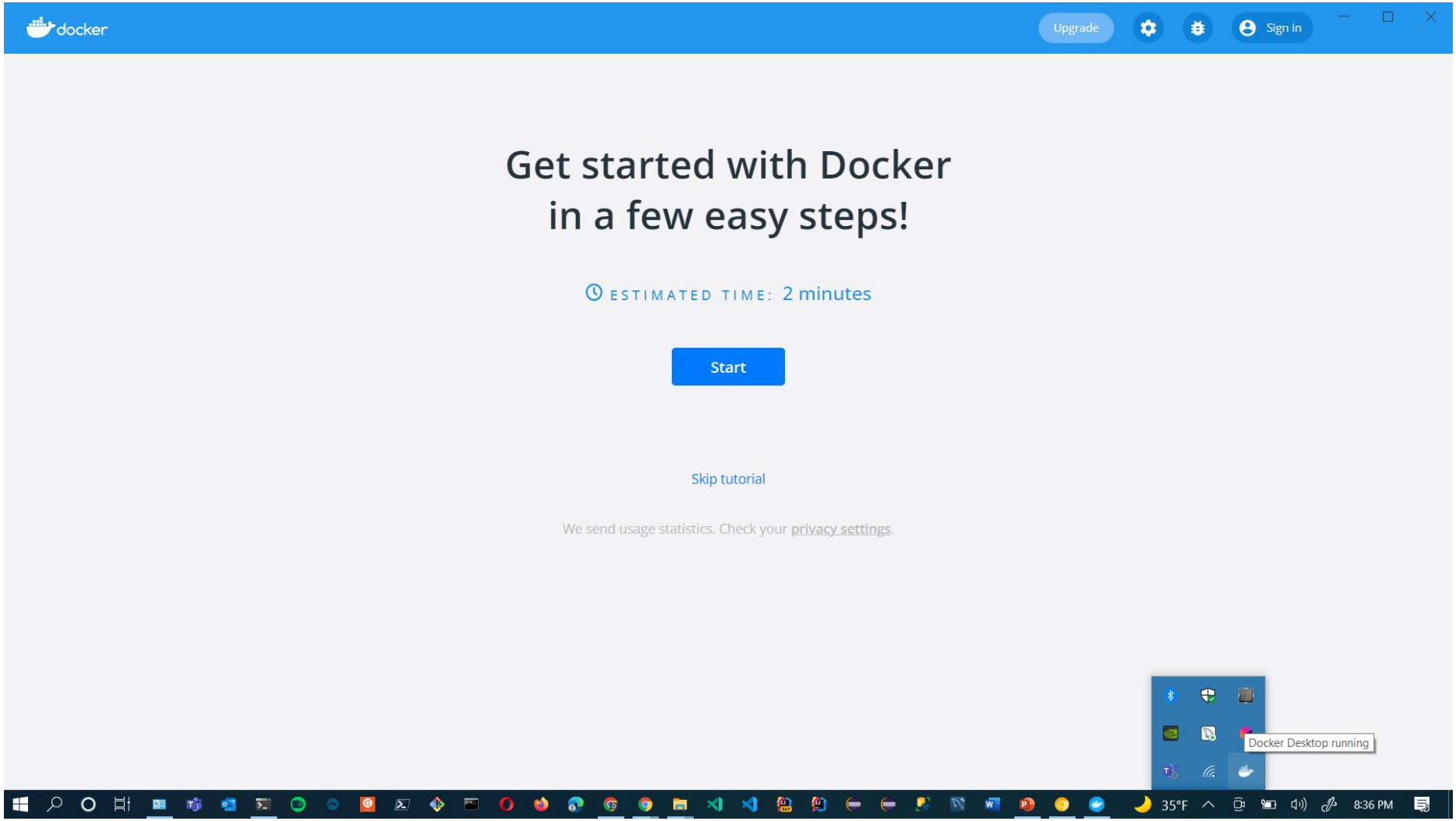
View Full Terms

Decline and Close Application | Accept

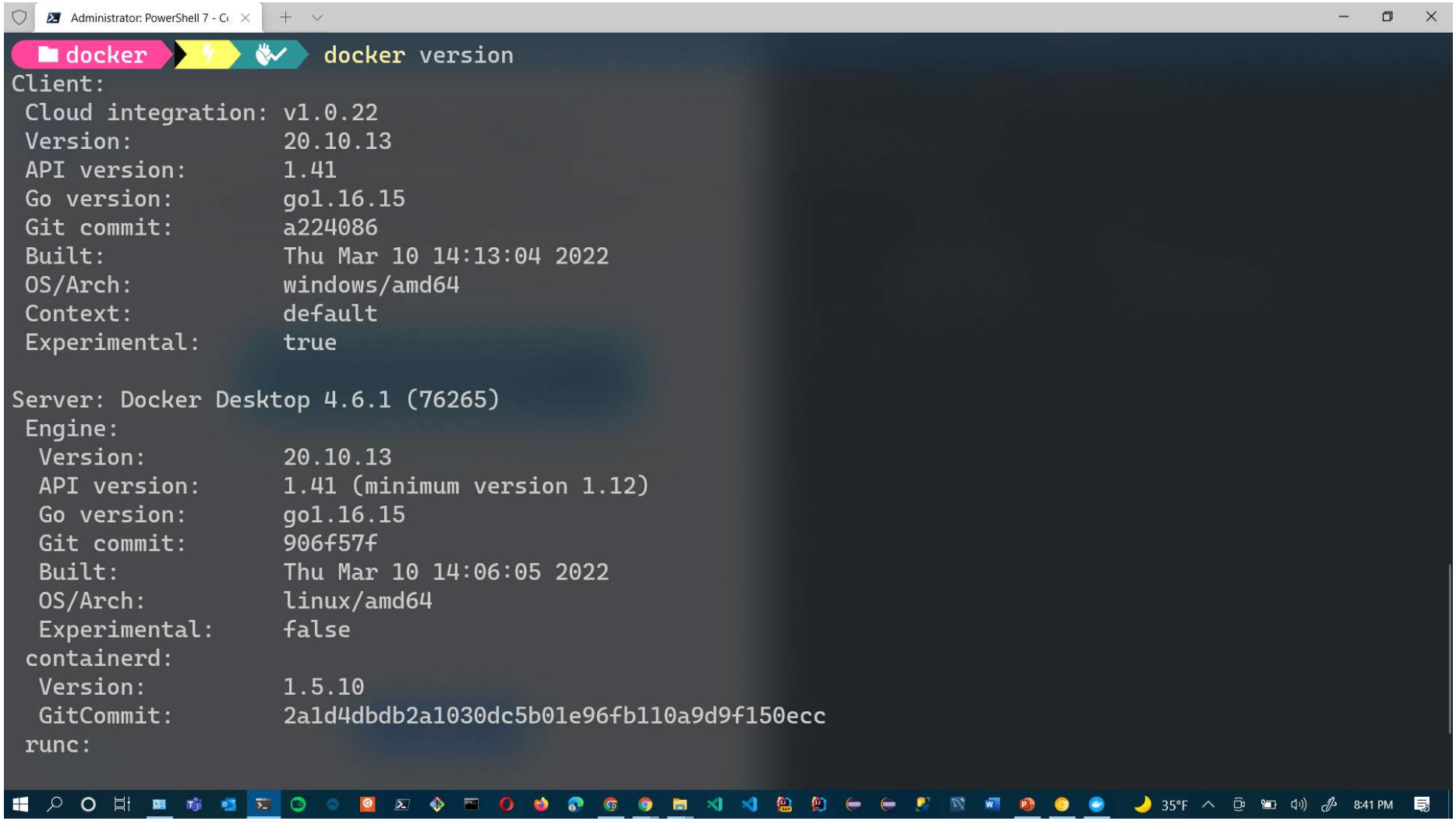
Starting…


docker
Upgrade
Sign in
Containers / Apps
Images
Volumes
Dev Environments
PREVIEW
Docker Desktop starting...

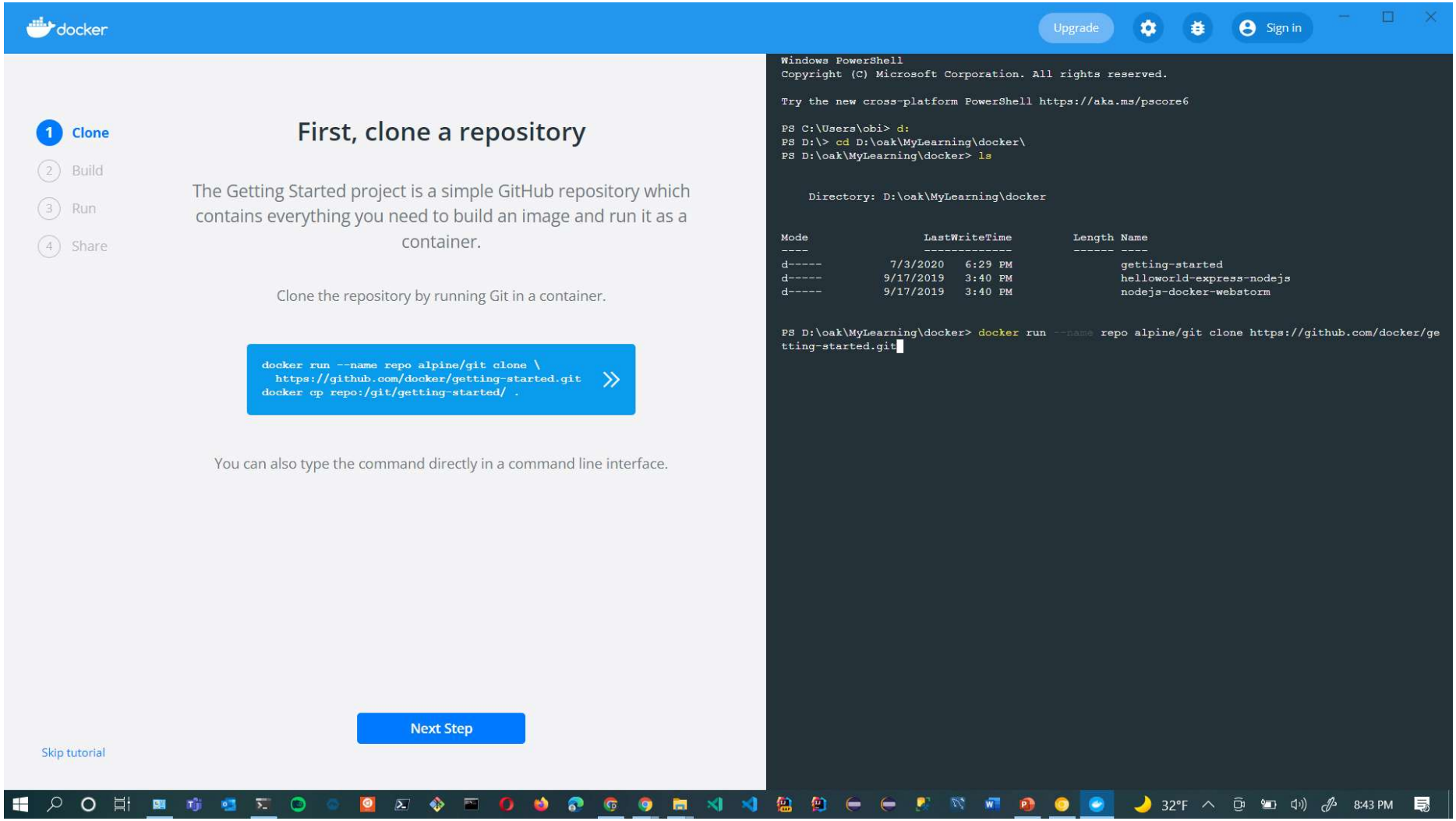
Docker Desktop running



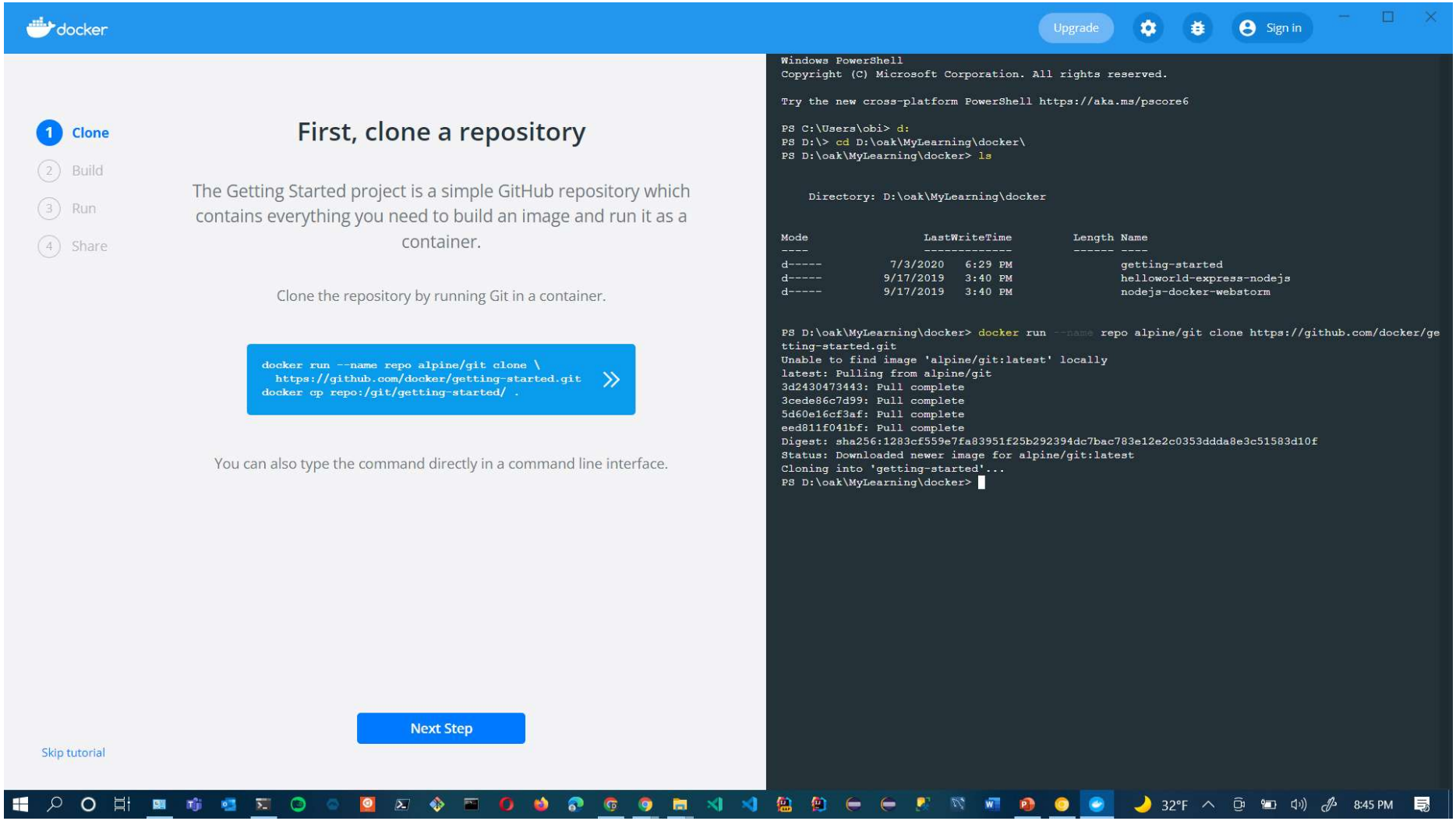
Verify running & version

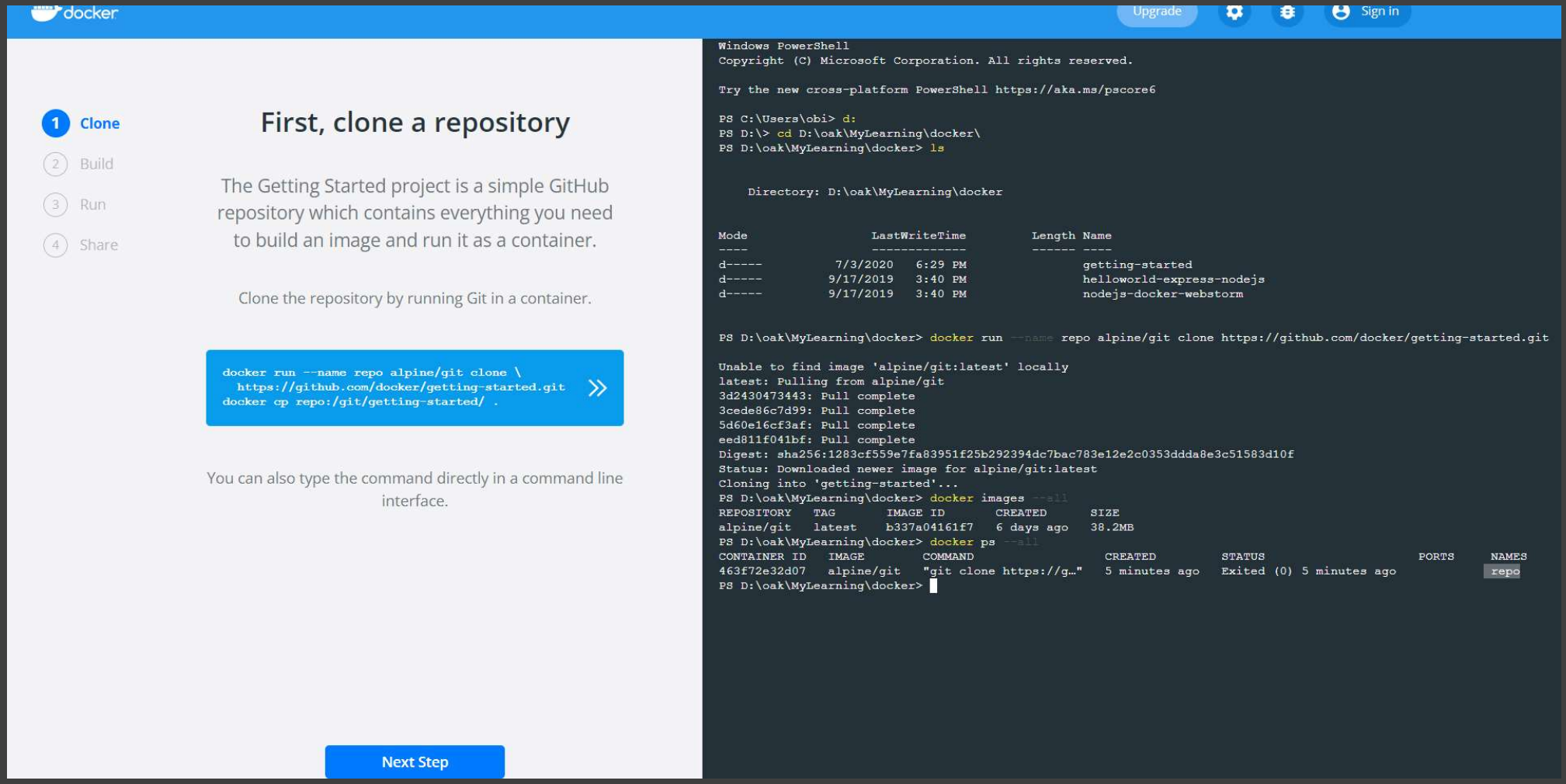


Practice with the ‘GettingStarted’



Getting Started



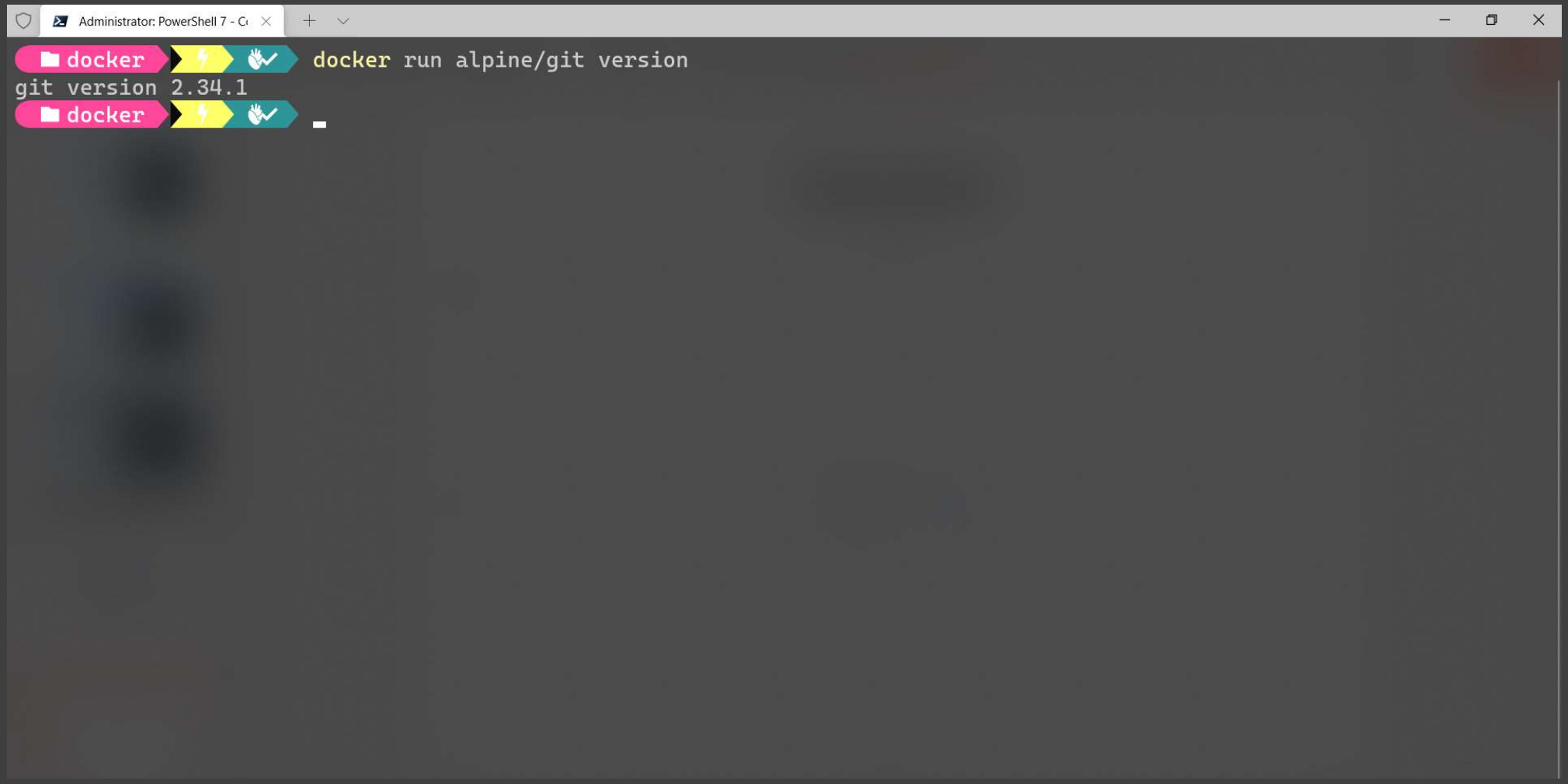


Practicing with the ‘Getting Started’

- 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’



Solution

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

Demo

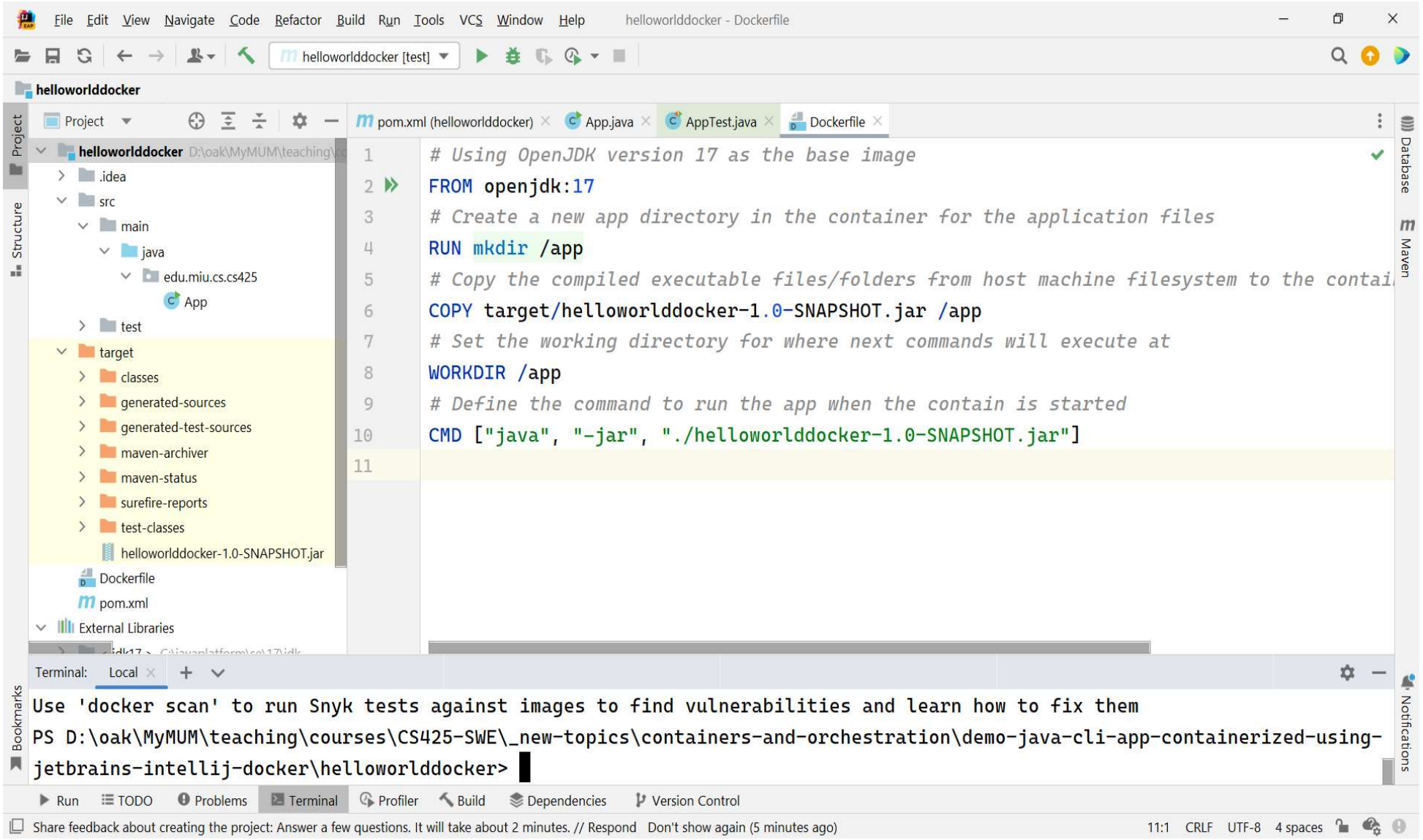
- Create a simple Java CLI Application using Apache Maven quickstart archetype
- Add config in maven-jar-plugin to specify the mainClass

```
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>
```

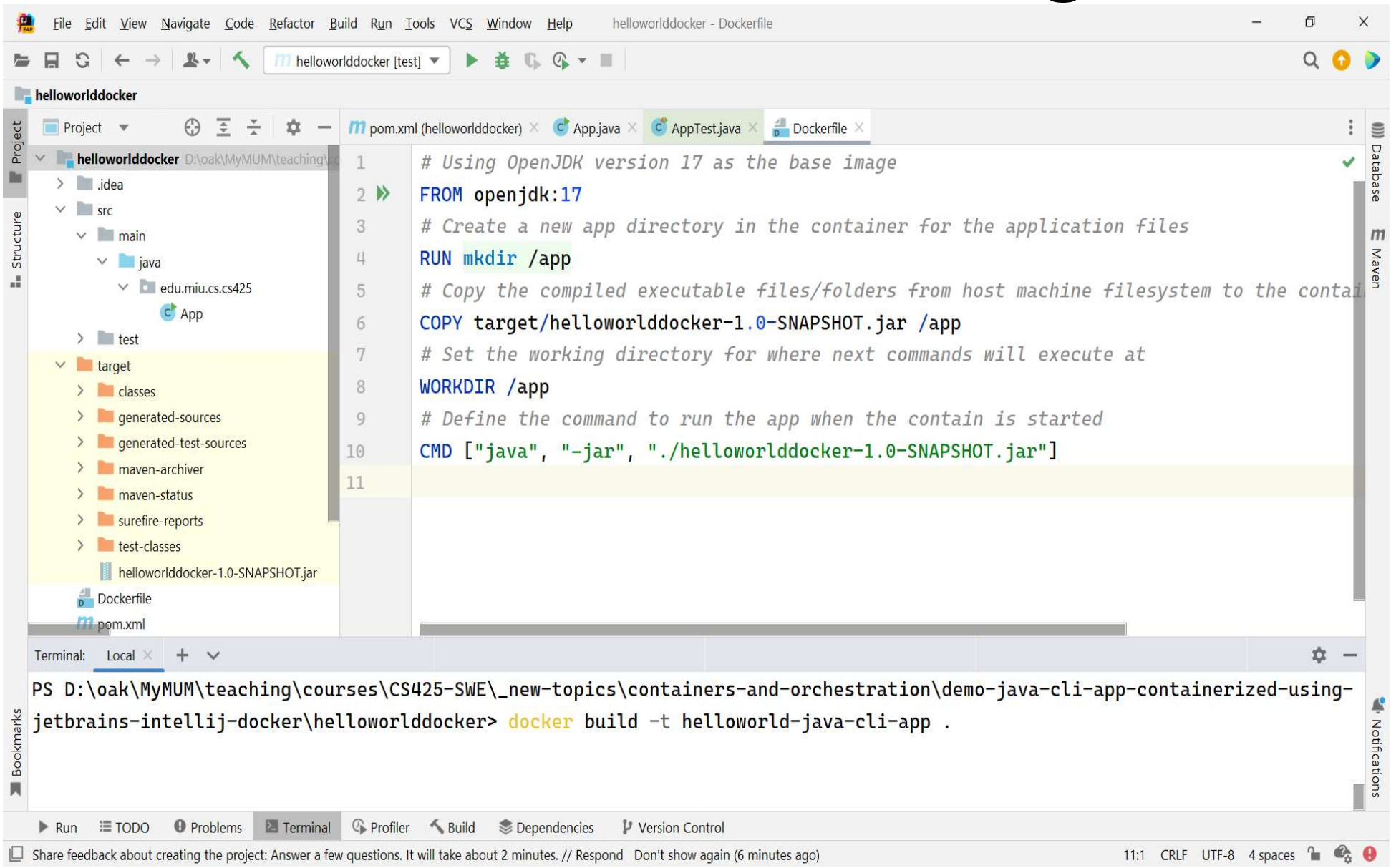
Demo

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

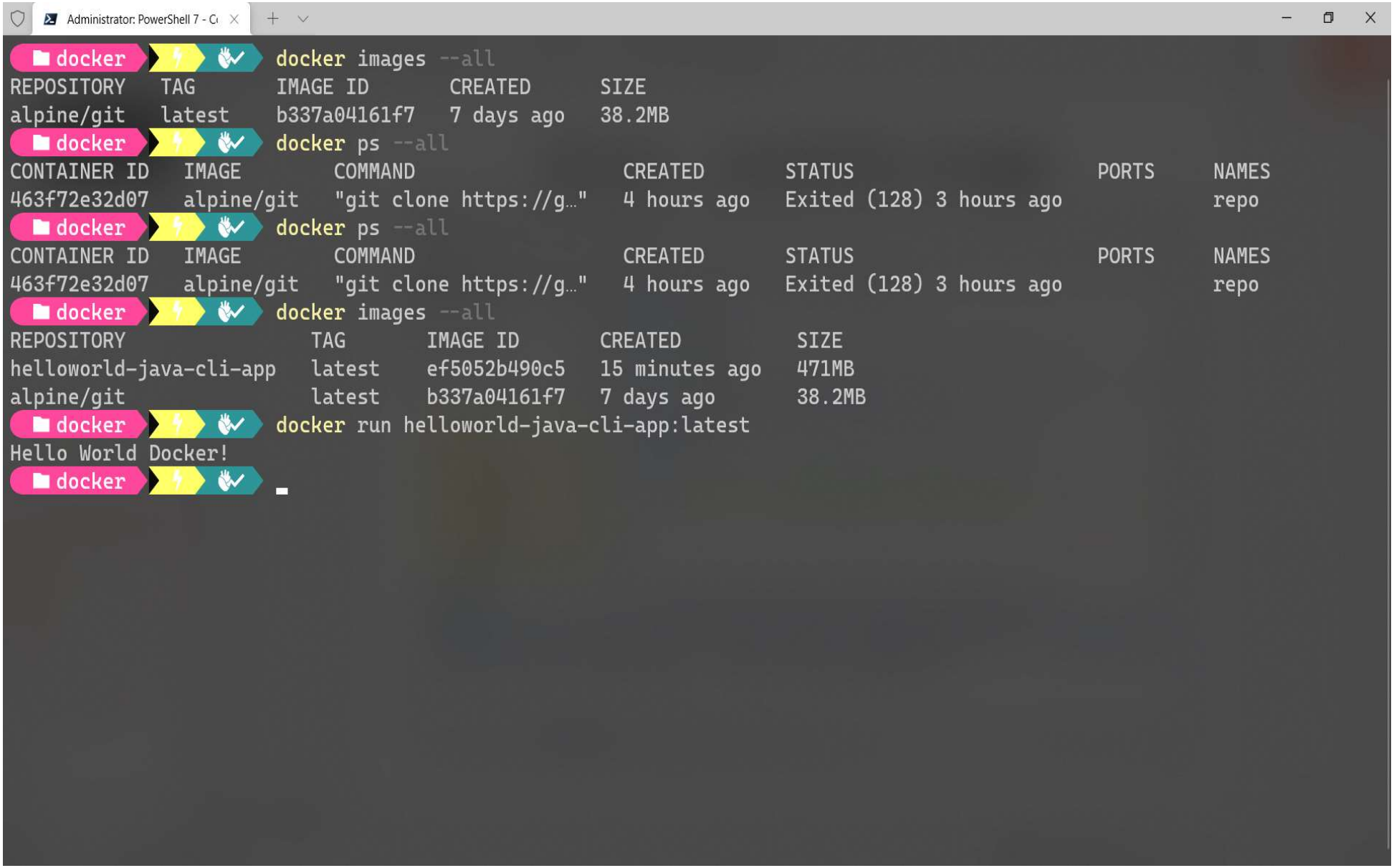
Demo


File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
helloworlddocker - Dockerfile
helloworlddocker [test]
helloworlddocker
Project
pom.xml (helloworlddocker)
App.java
AppTest.java
Dockerfile
helloworlddocker D:\oak\MyMUM\teaching\
.idea
src
main
java
edu.miu.cs.cs425
App
test
target
classes
generated-sources
generated-test-sources
maven-archiver
maven-status
surefire-reports
test-classes
helloworlddocker-1.0-SNAPSHOT.jar
Dockerfile
pom.xml
External Libraries
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 contai
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 contain is started
10 CMD ["java", "-jar", "./helloworlddocker-1.0-SNAPSHOT.jar"]
11
Terminal: Local
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>
Run TODO Problems Terminal Profiler Build Dependencies Version Control
Share feedback about creating the project: Answer a few questions. It will take about 2 minutes. // Respond Don't show again (5 minutes ago)
11:1 CRLF UTF-8 4 spaces
Database
Maven
Notifications
Bookmarks
Structure

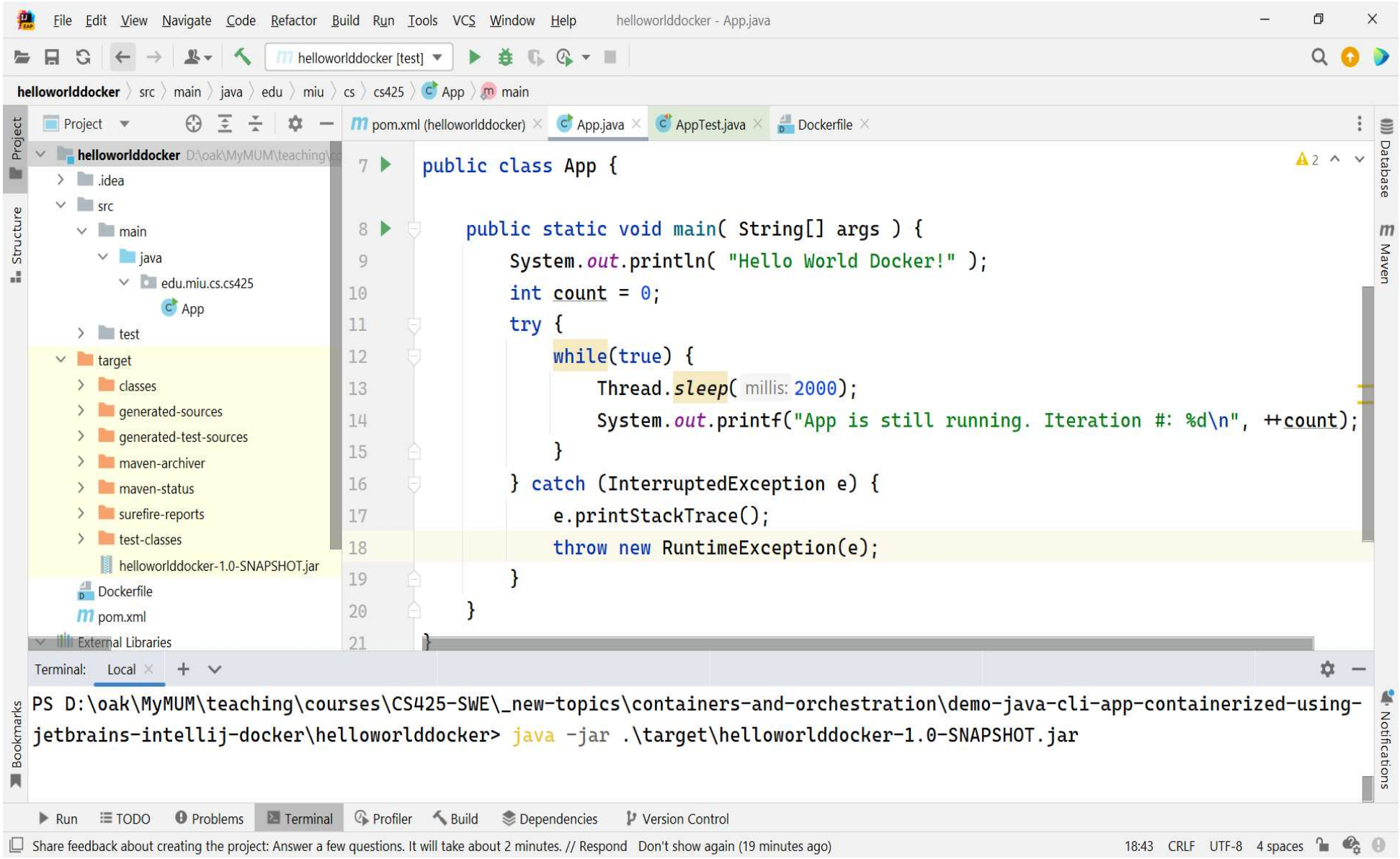
Demo – build the image



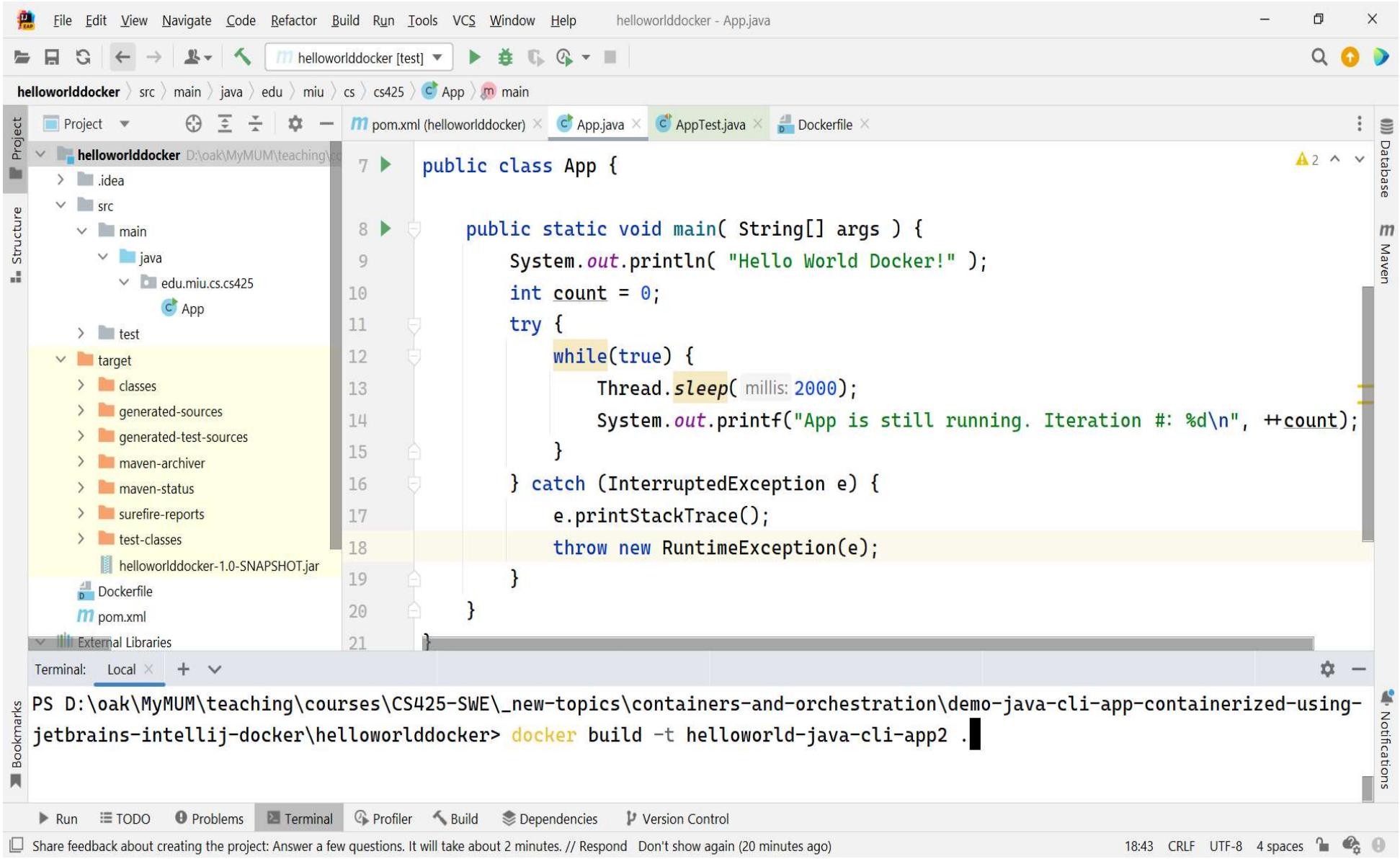
Run the container



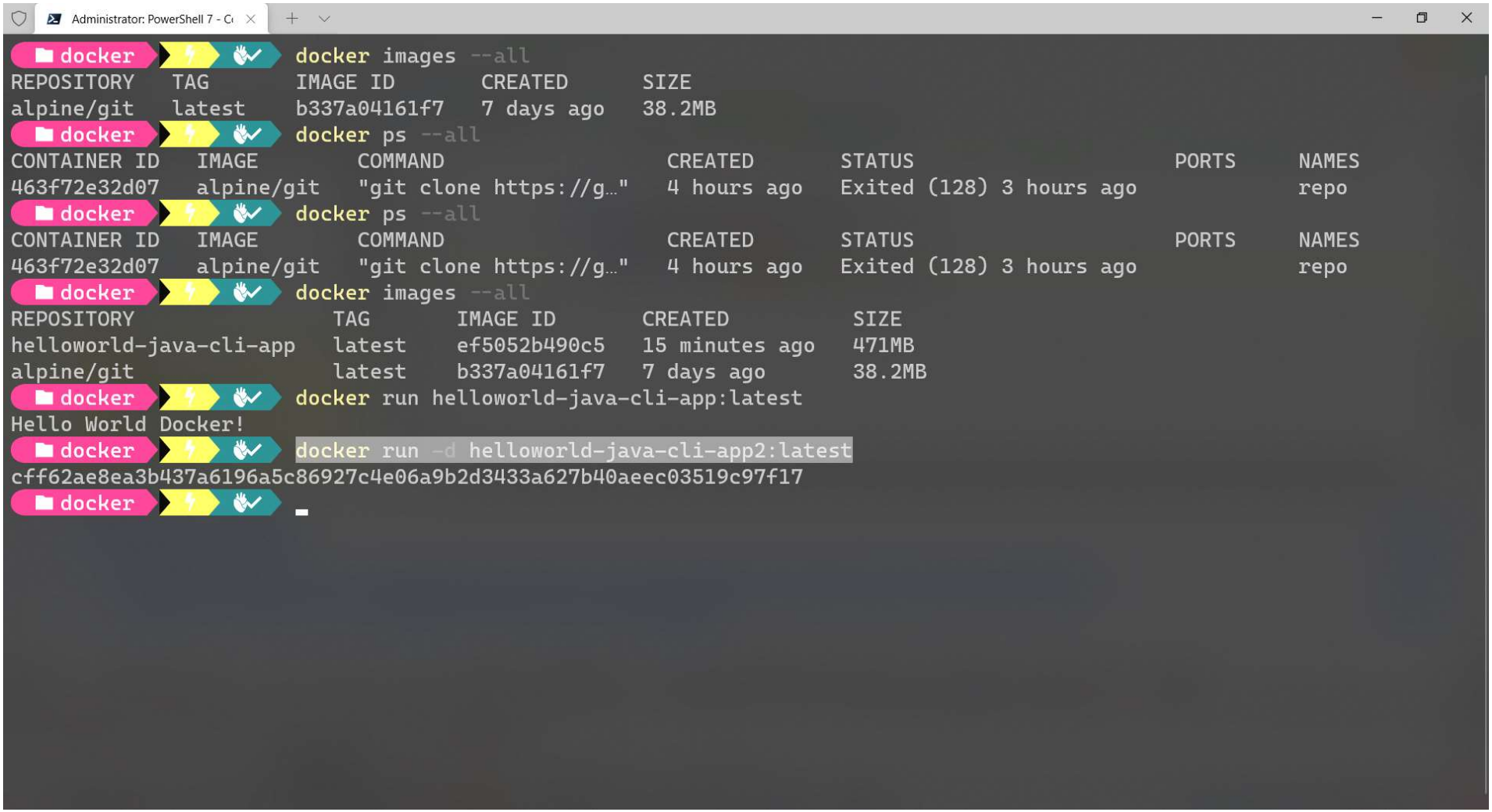
Simulate a running server-like app



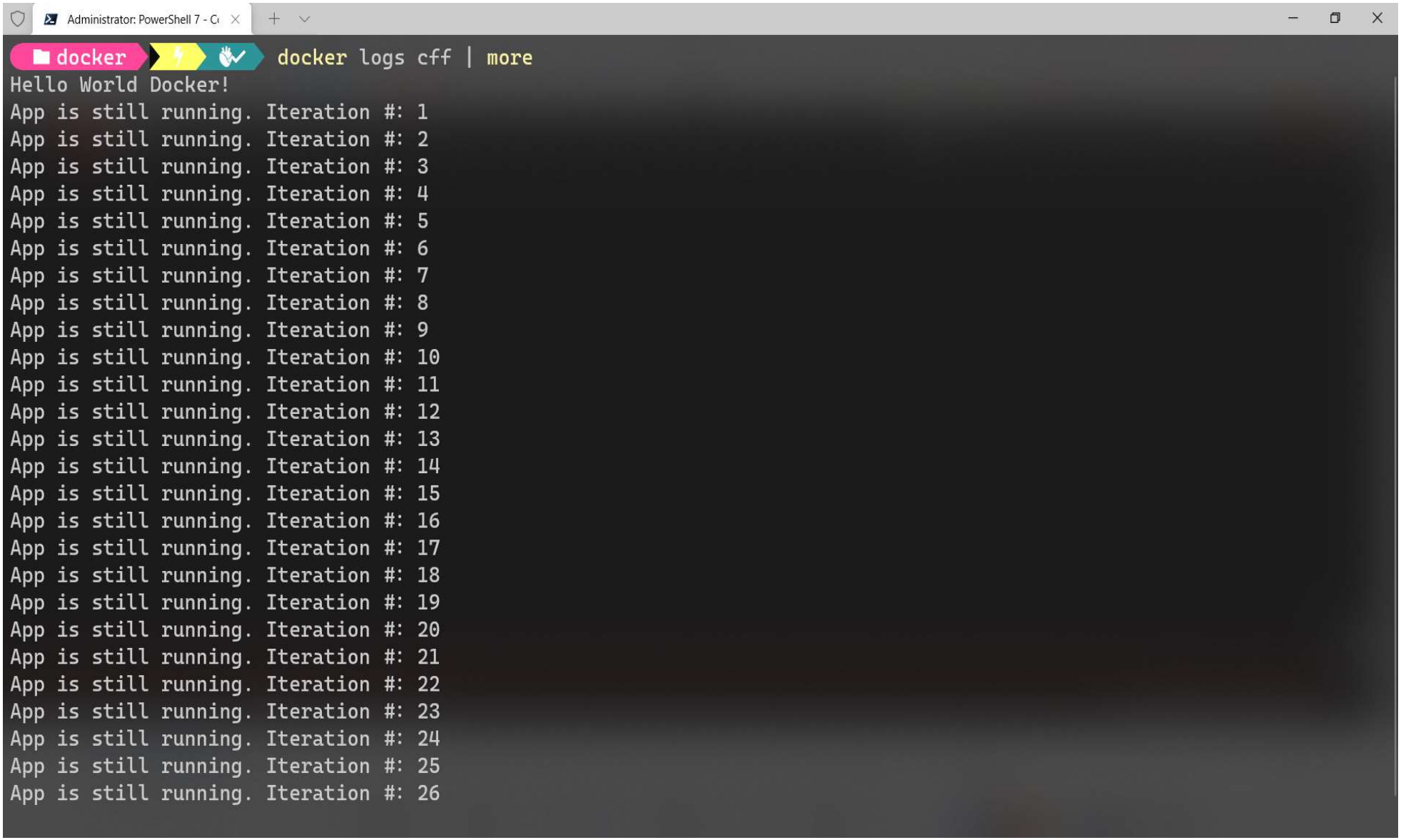
Build another docker image



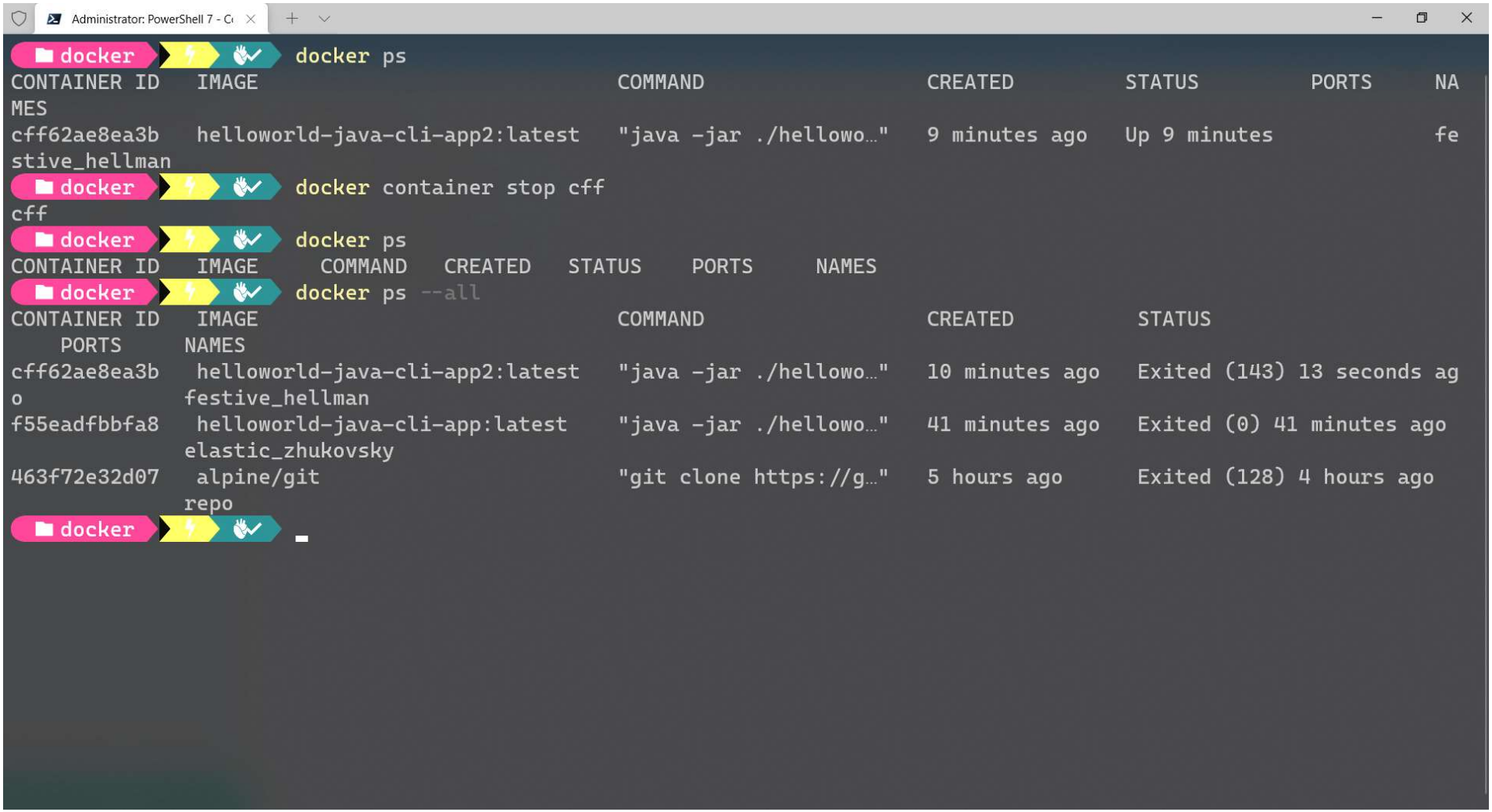
Execute/start/run the container (in detached mode)

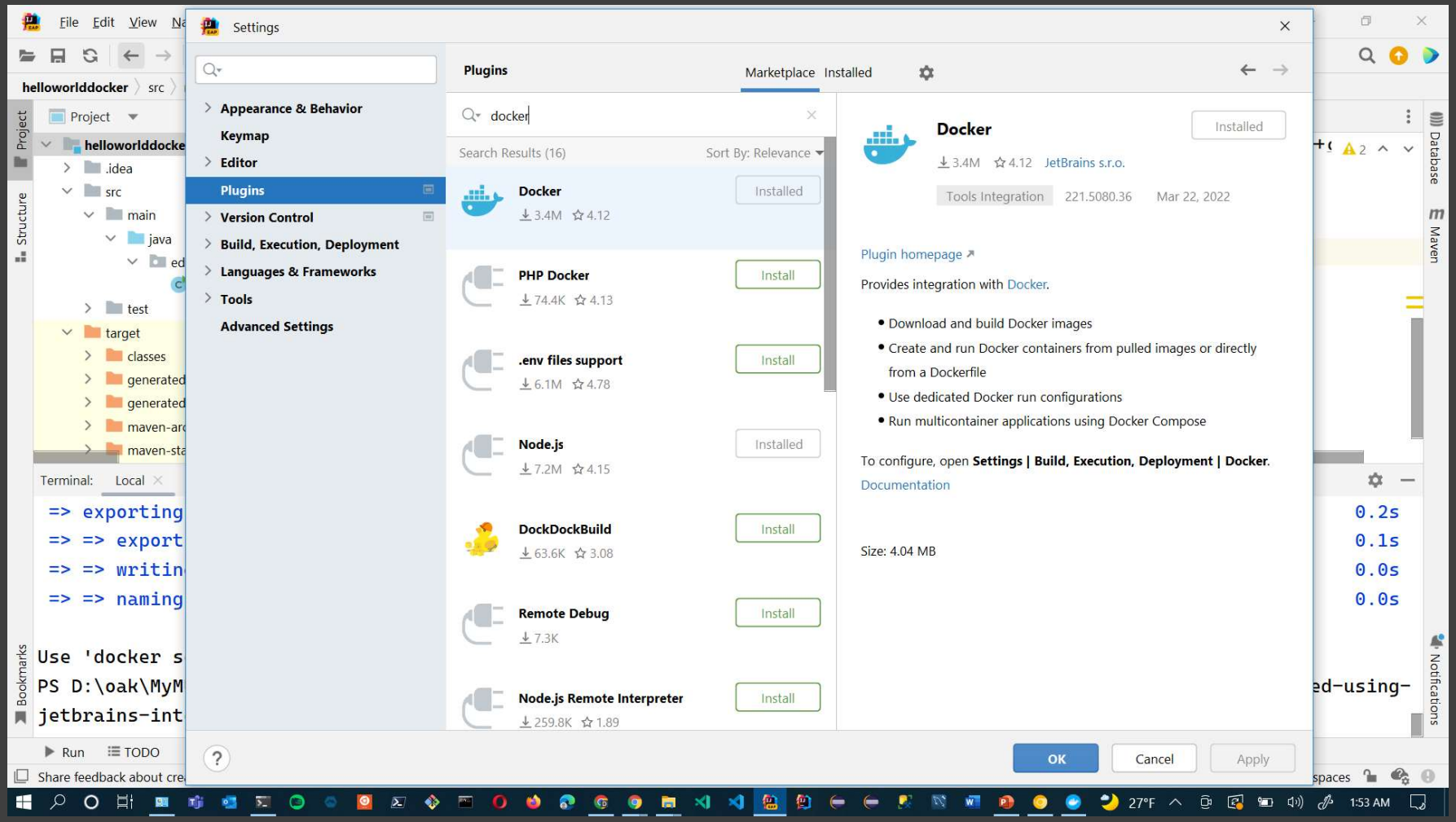


Display the container’s log



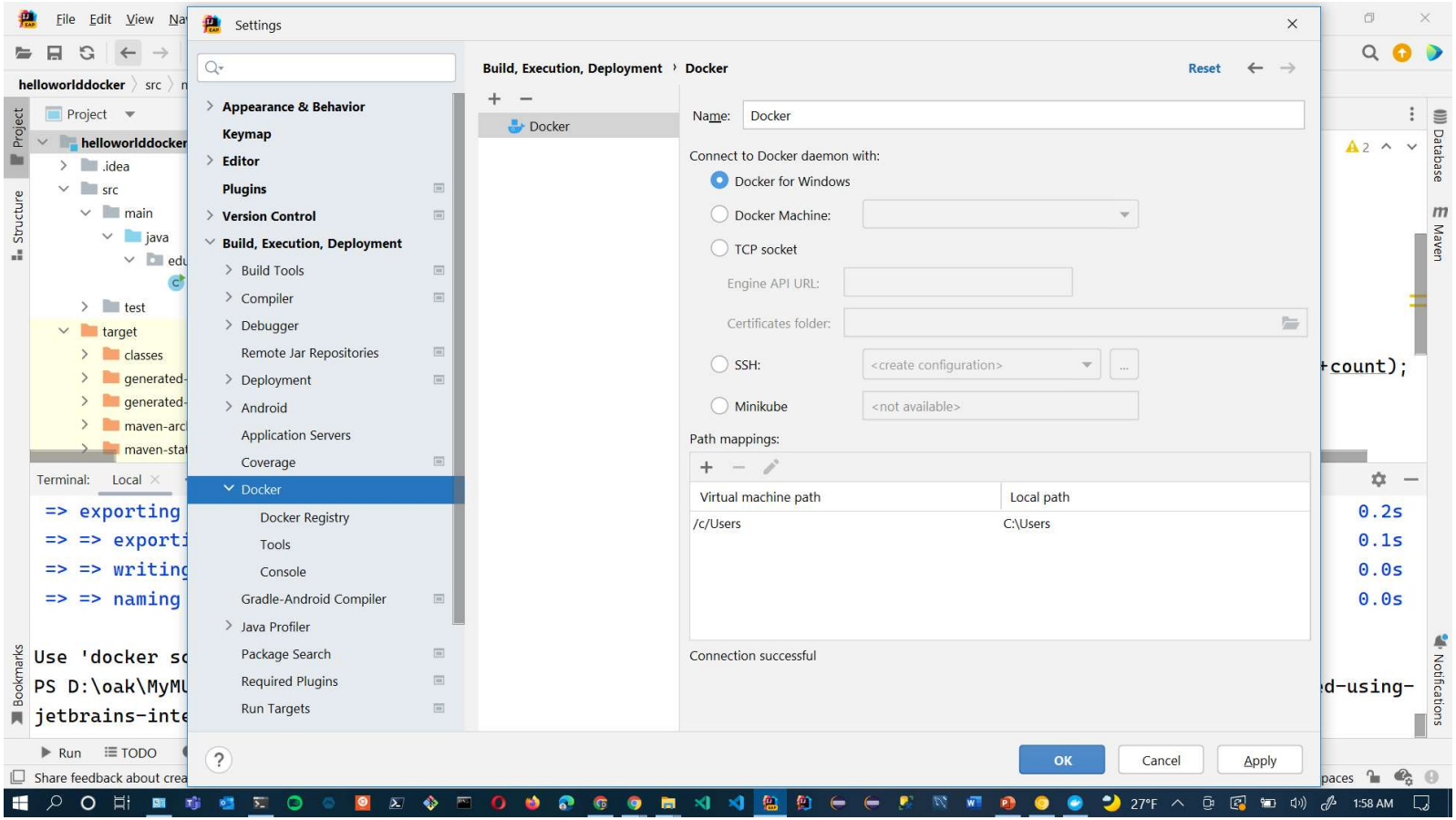
Check the status and stop the running container





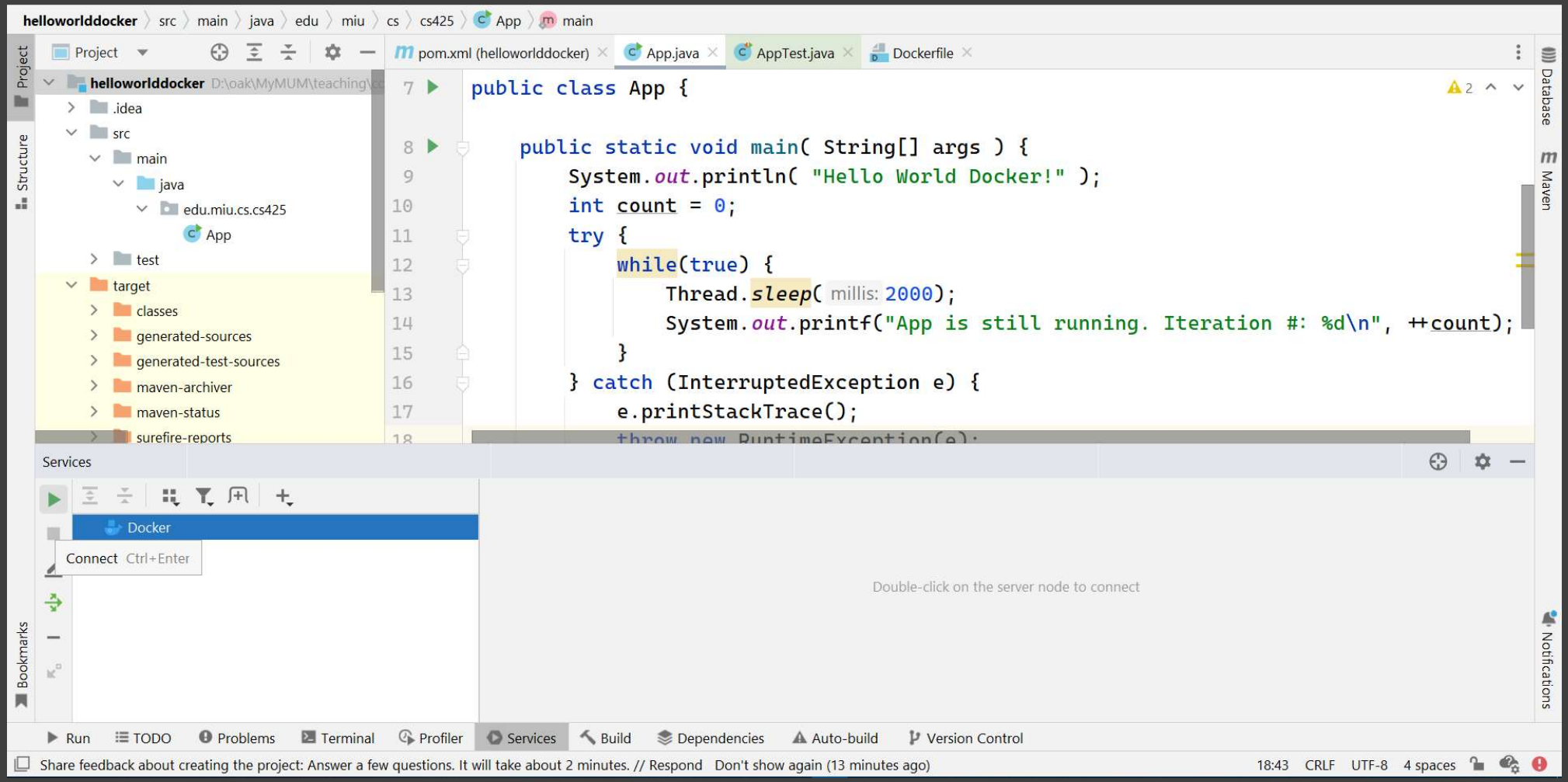
Operating Docker with IntelliJ IDEA

- 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

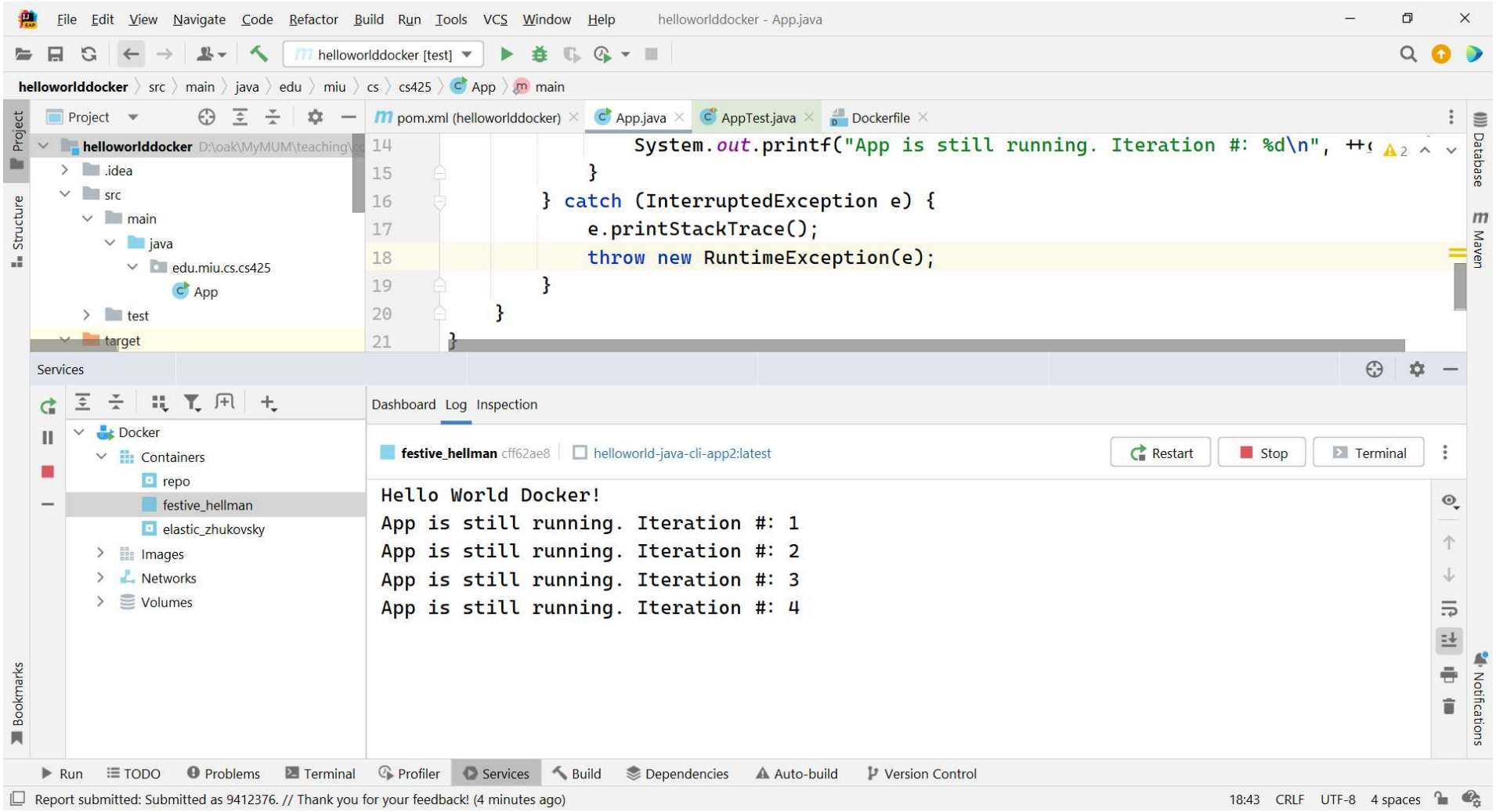
- Open Settings dialog, go to Build, Exec, Deployment; Select “Docker” and click the + button
- IntelliJ IDEA auto-detects a running Docker for your OS and connects to the docker daemon successfully
- Click “Ok” and Docker appears in the Services tab

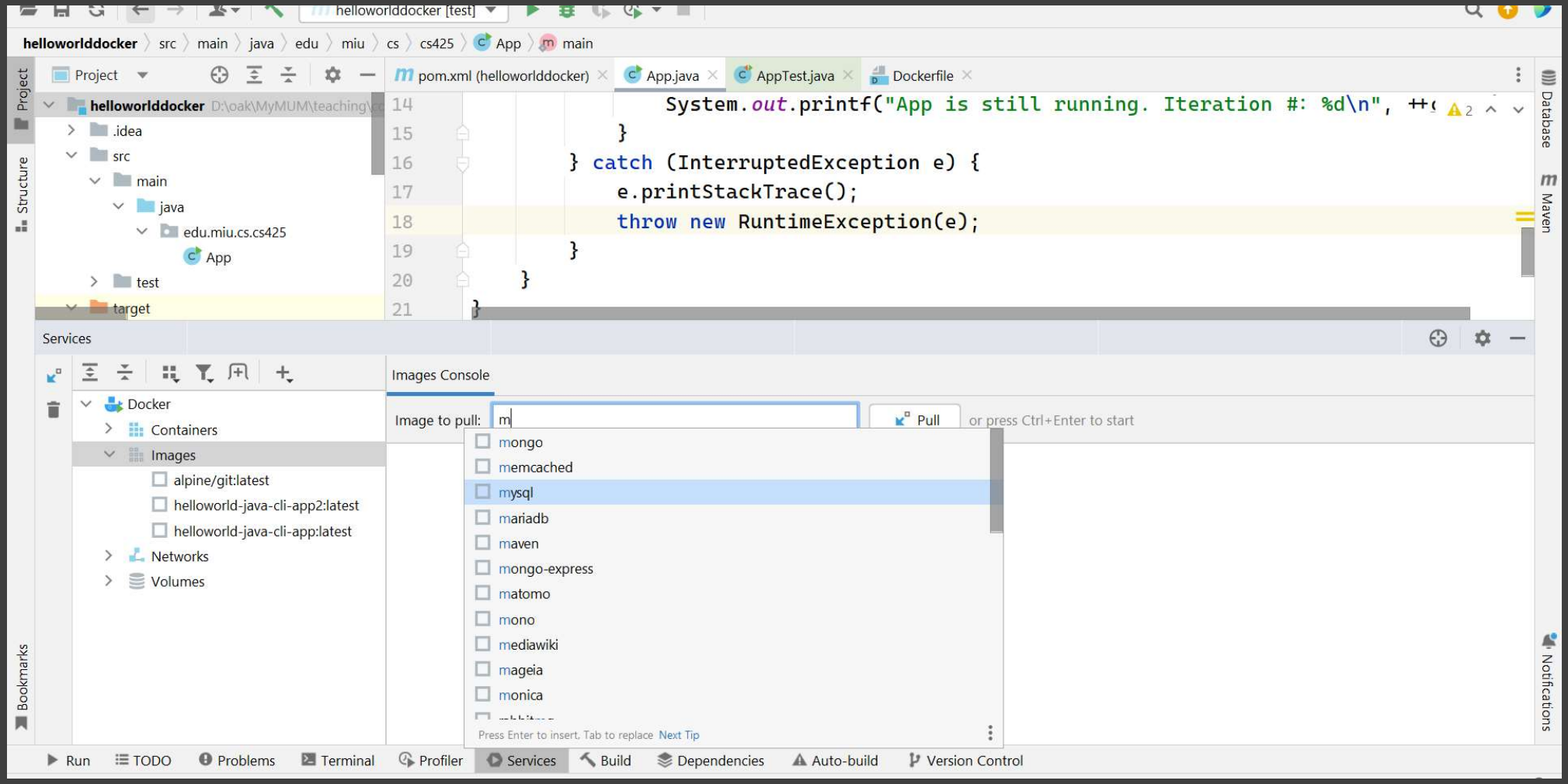


Operating Docker in IntelliJ

- 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

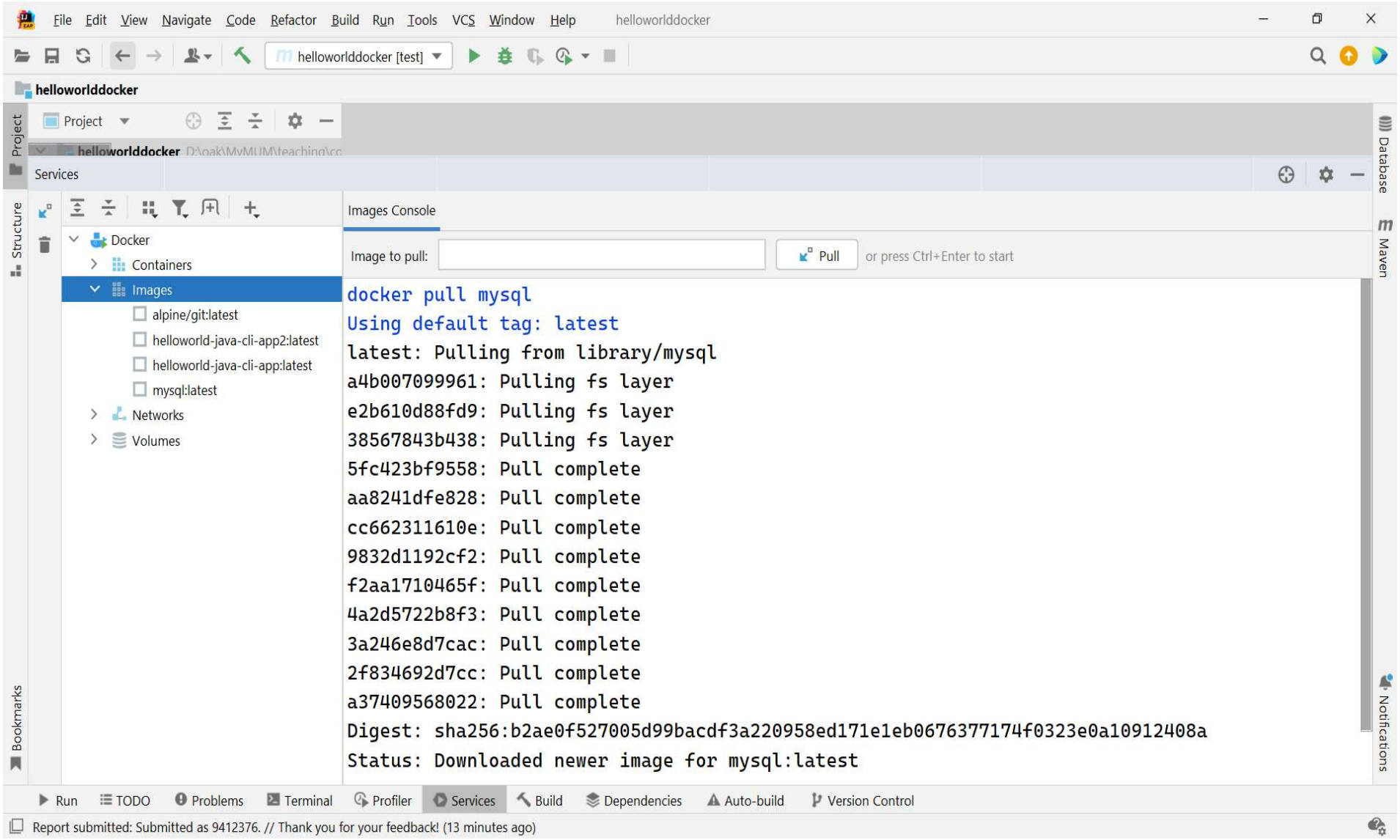




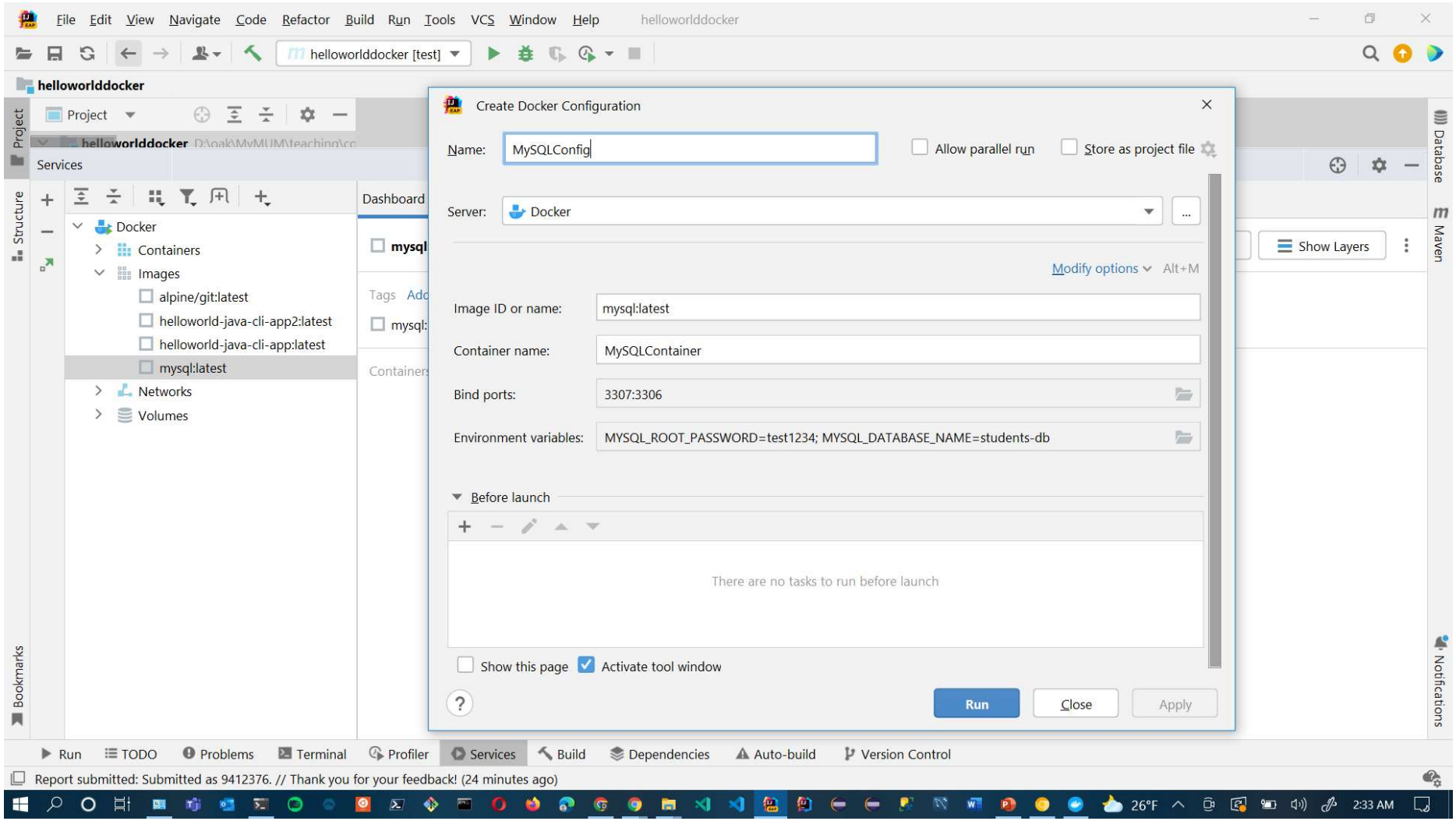
Working with Docker in IntelliJ IDEA

- 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.

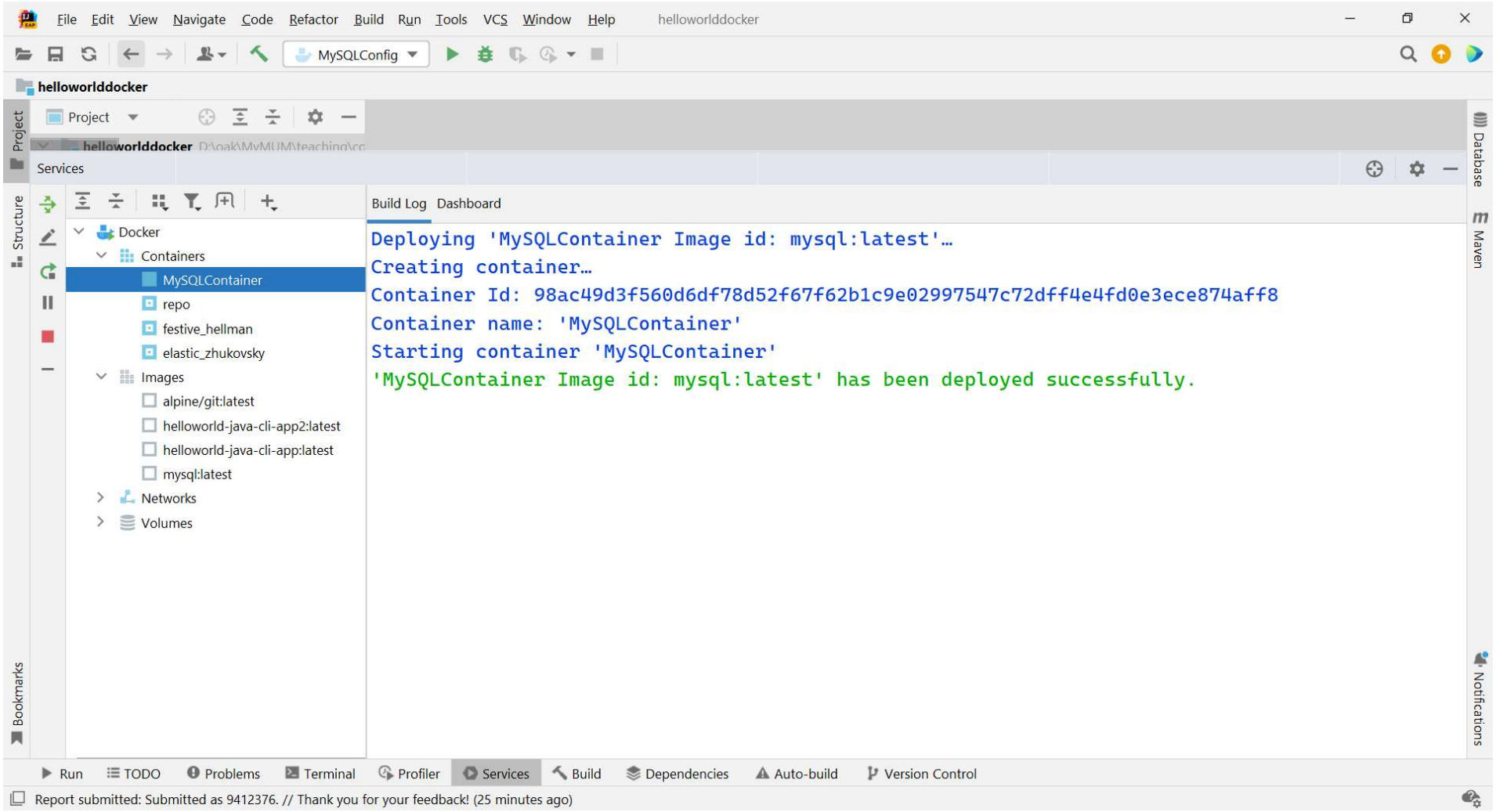
Setting-up a MySQL database container



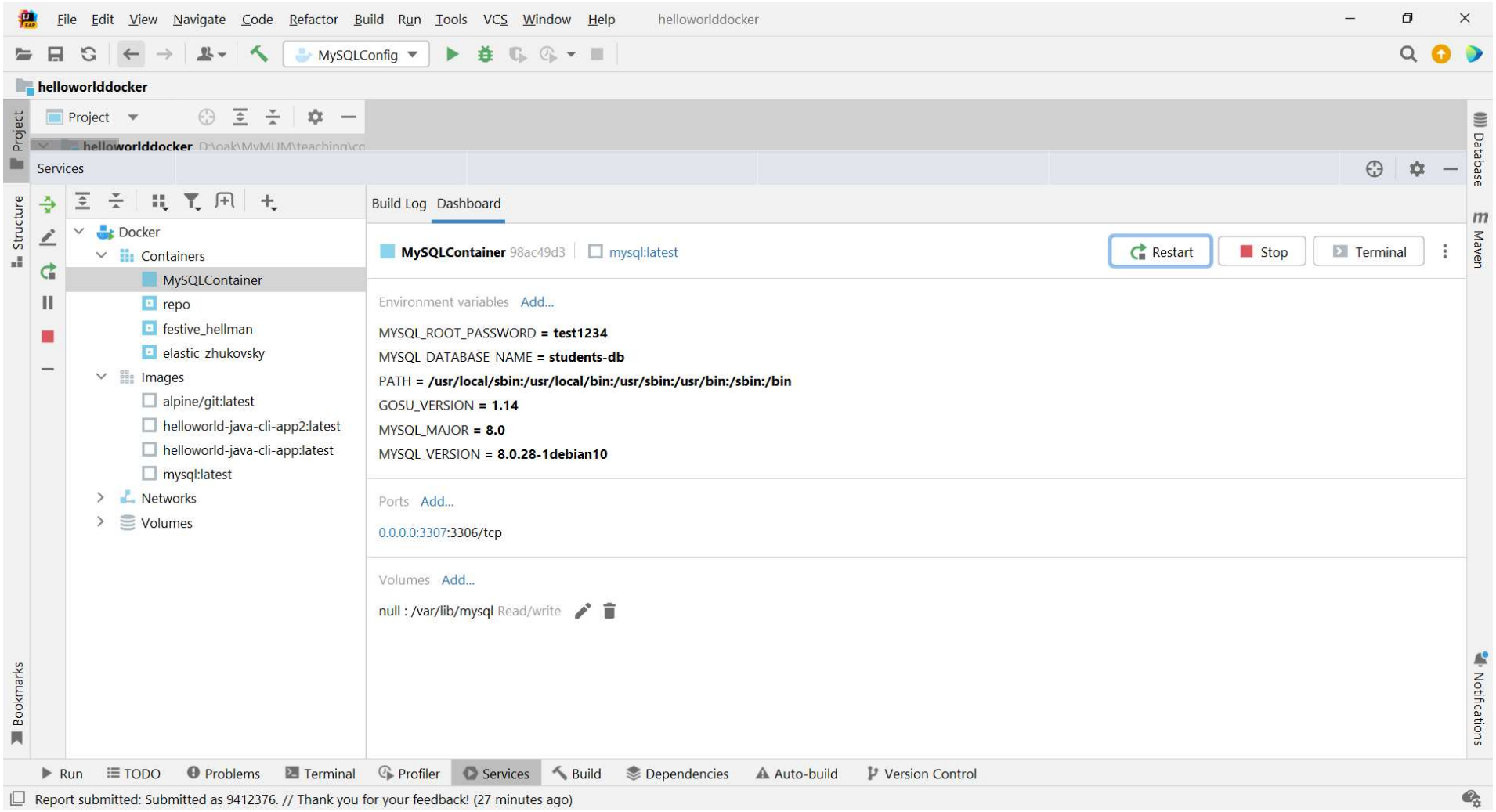
Create a MySQL DB container



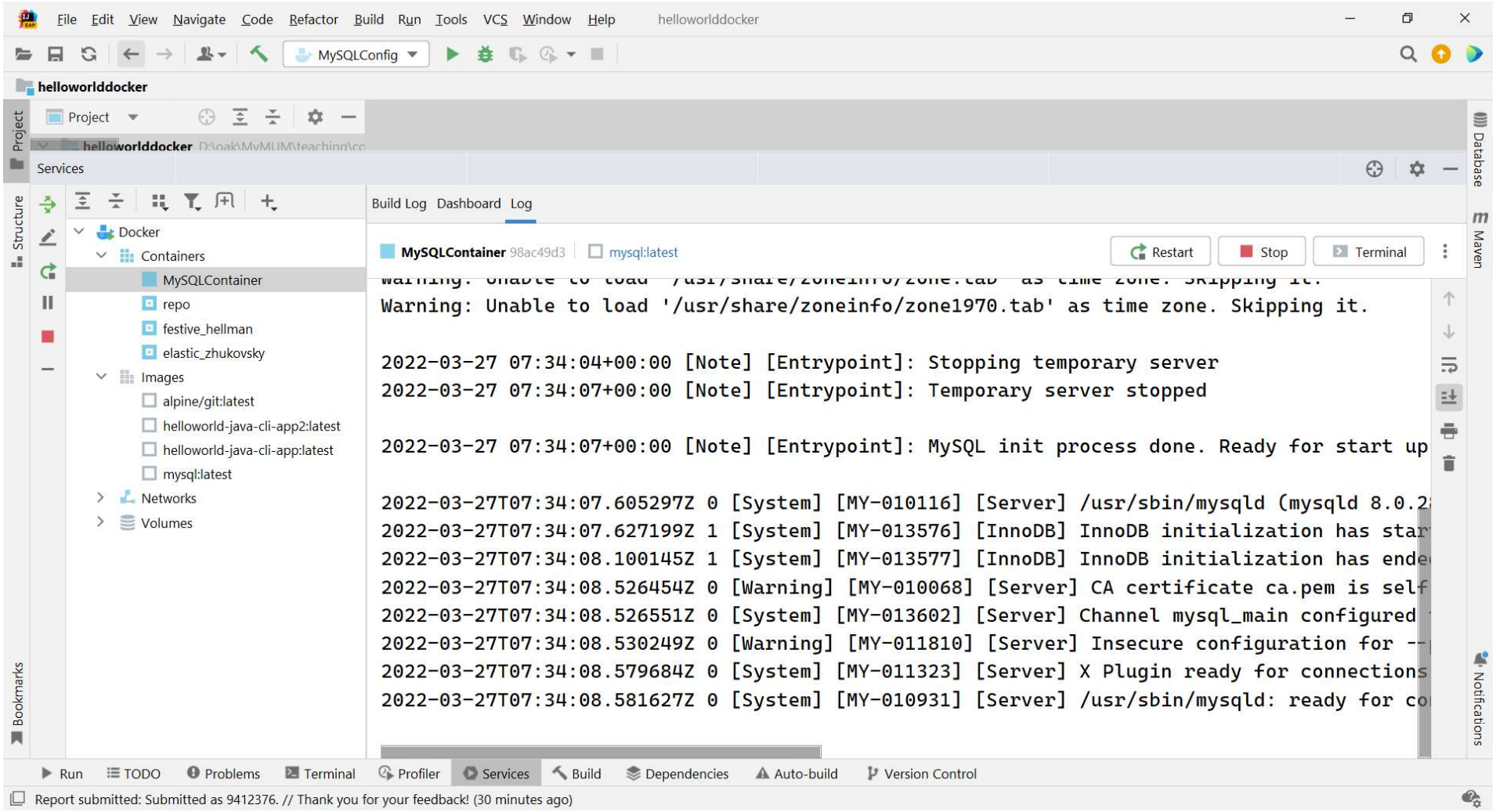
MySQL Database container created



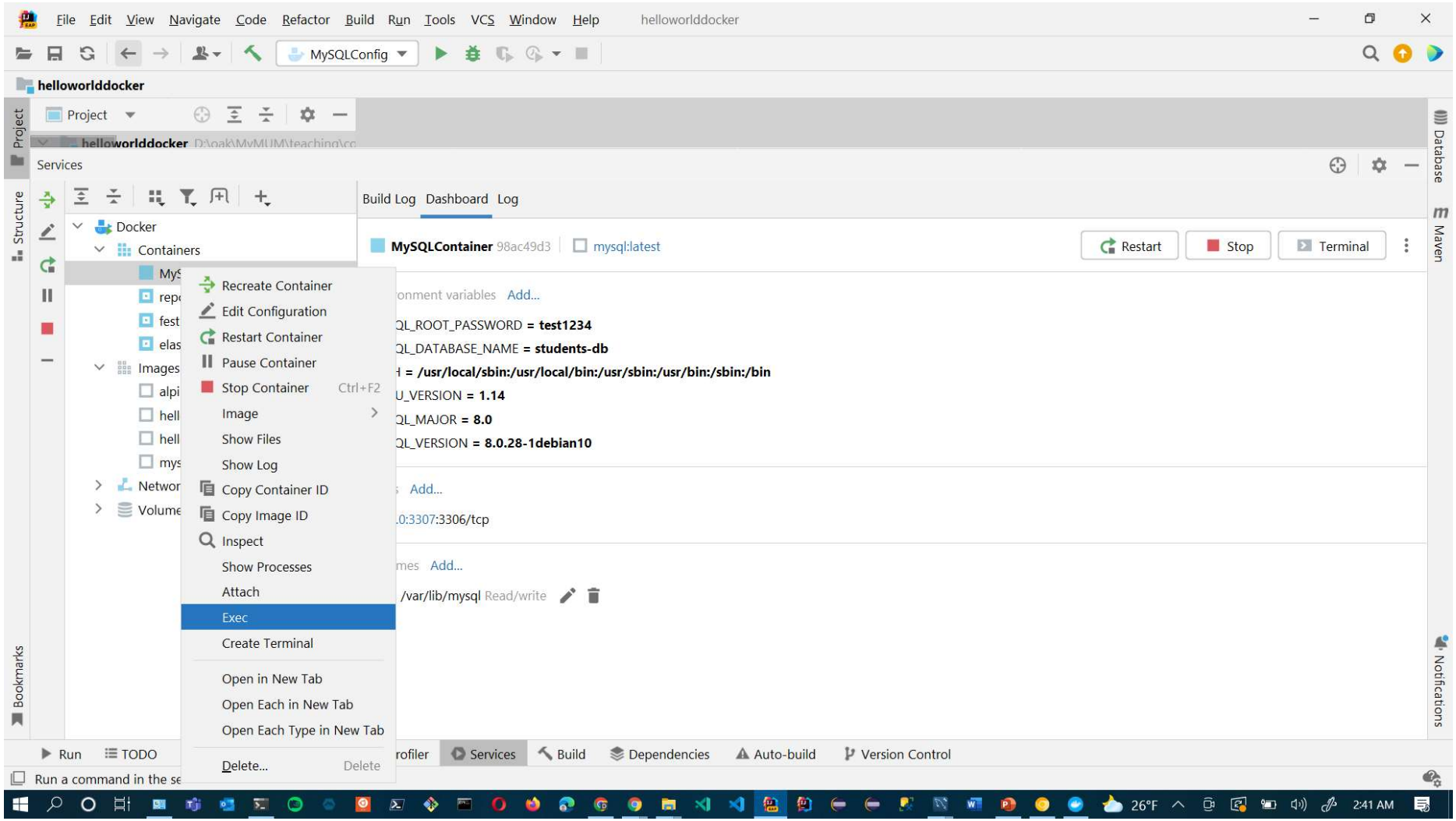
MySQL Database container running



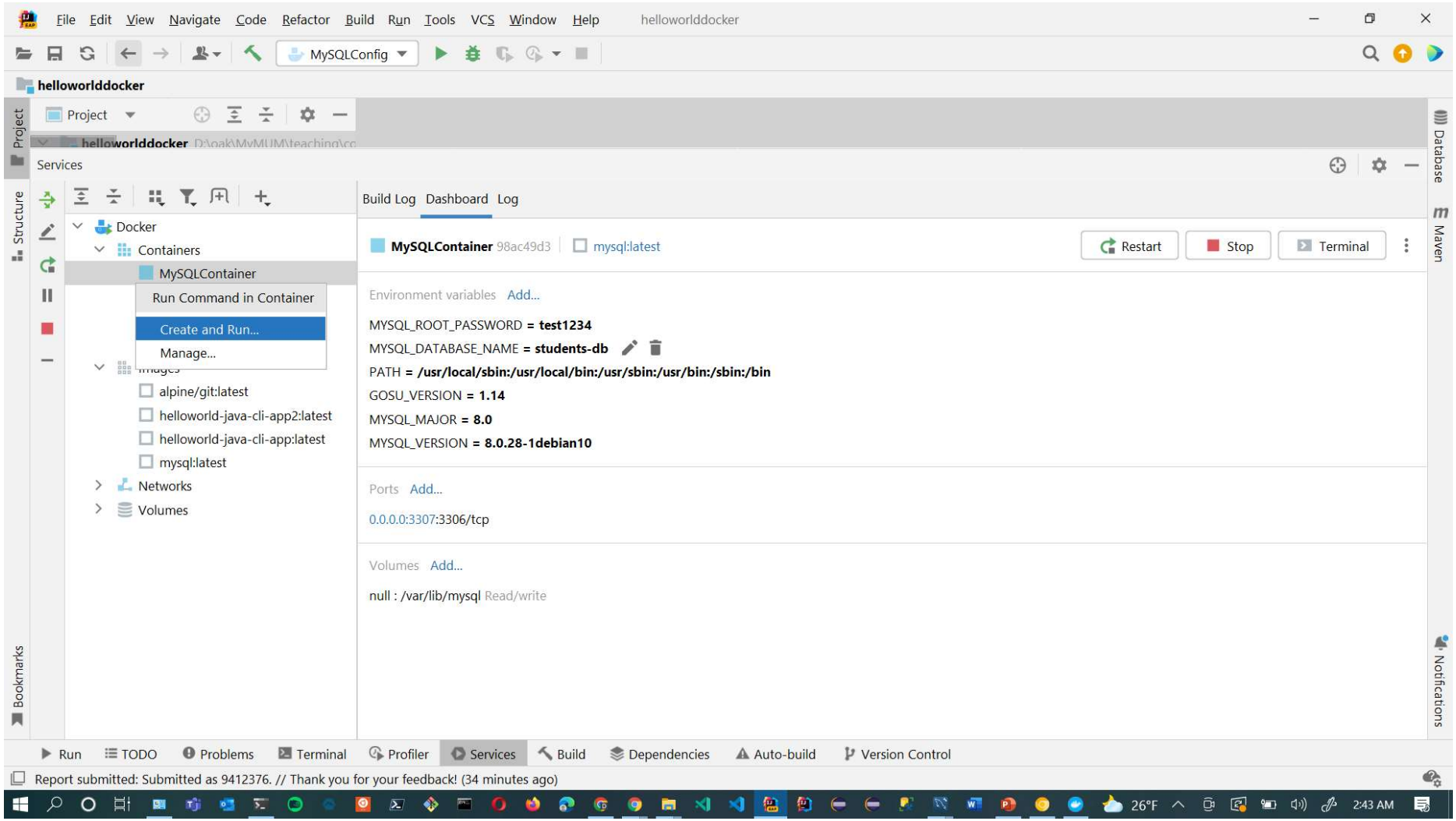
View the log


File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
helloworlddocker
MySQLConfig
helloworlddocker
Project
Services
Build Log Dashboard Log
Docker
Containers
MySQLContainer
repo
festive_hellman
elastic_zhukovsky
Images
alpine/git:latest
helloworld-java-cli-app2:latest
helloworld-java-cli-app:latest
mysql:latest
Networks
Volumes
MySQLContainer 98ac49d3
mysql:latest
Restart
Stop
Terminal
Warning: Unable to load '/usr/share/zoneinfo/zone1970.tab' as time zone. Skipping it.
2022-03-27 07:34:04+00:00 [Note] [Entrypoint]: Stopping temporary server
2022-03-27 07:34:07+00:00 [Note] [Entrypoint]: Temporary server stopped
2022-03-27 07:34:07+00:00 [Note] [Entrypoint]: MySQL init process done. Ready for start up
2022-03-27T07:34:07.605297Z 0 [System] [MY-010116] [Server] /usr/sbin/mysqld (mysqld 8.0.2
2022-03-27T07:34:07.627199Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has star
2022-03-27T07:34:08.100145Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization has ende
2022-03-27T07:34:08.526454Z 0 [Warning] [MY-010068] [Server] CA certificate ca.pem is self
2022-03-27T07:34:08.526551Z 0 [System] [MY-013602] [Server] Channel mysql_main configured
2022-03-27T07:34:08.530249Z 0 [Warning] [MY-011810] [Server] Insecure configuration for --
2022-03-27T07:34:08.579684Z 0 [System] [MY-011323] [Server] X Plugin ready for connections
2022-03-27T07:34:08.581627Z 0 [System] [MY-010931] [Server] /usr/sbin/mysqld: ready for co
Run TODO Problems Terminal Profiler Services Build Dependencies Auto-build Version Control
Report submitted: Submitted as 9412376. // Thank you for your feedback! (30 minutes ago)
Database
Maven
Notifications
Bookmarks
Structure

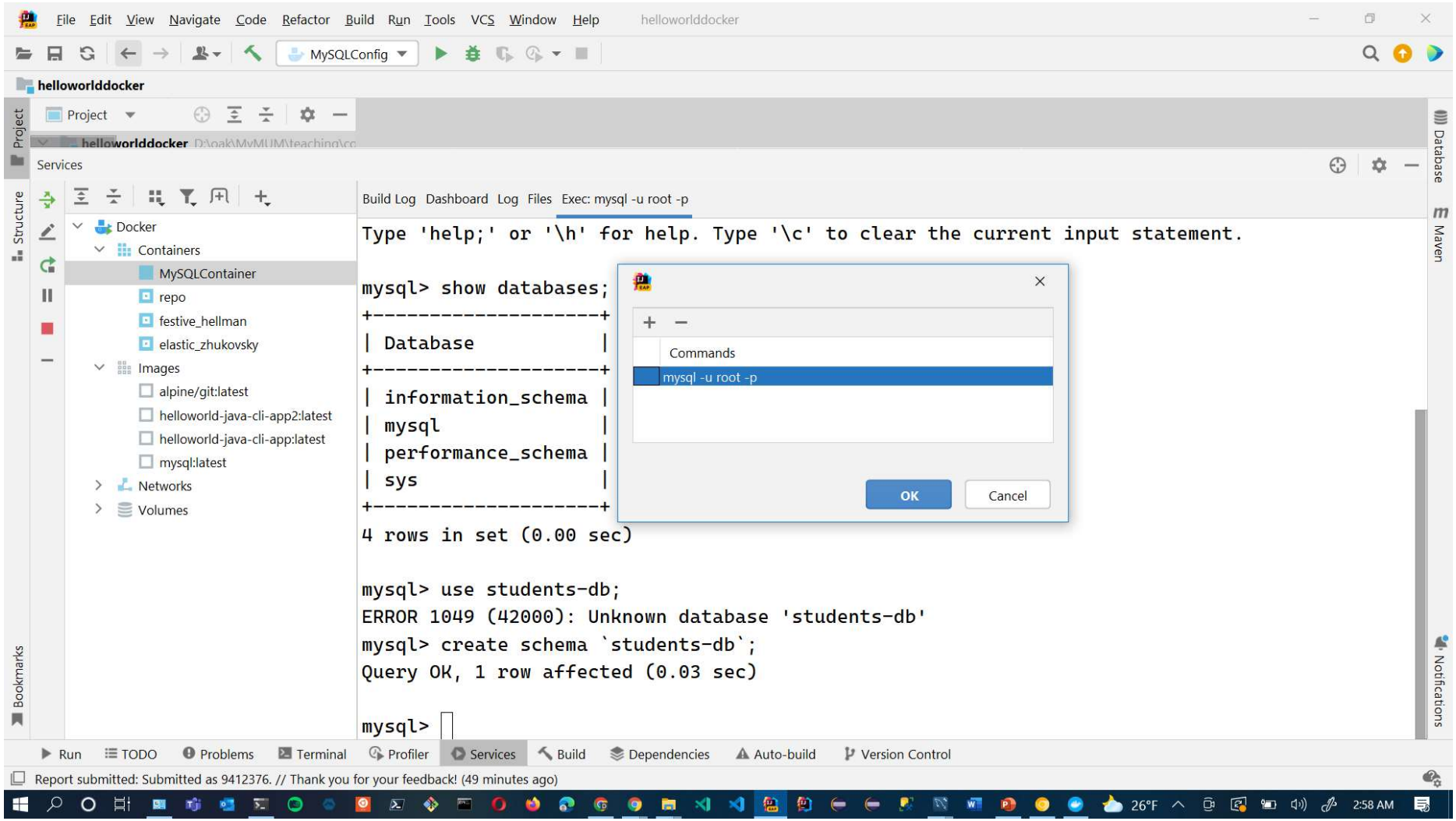
Connect with MySQL shell or client



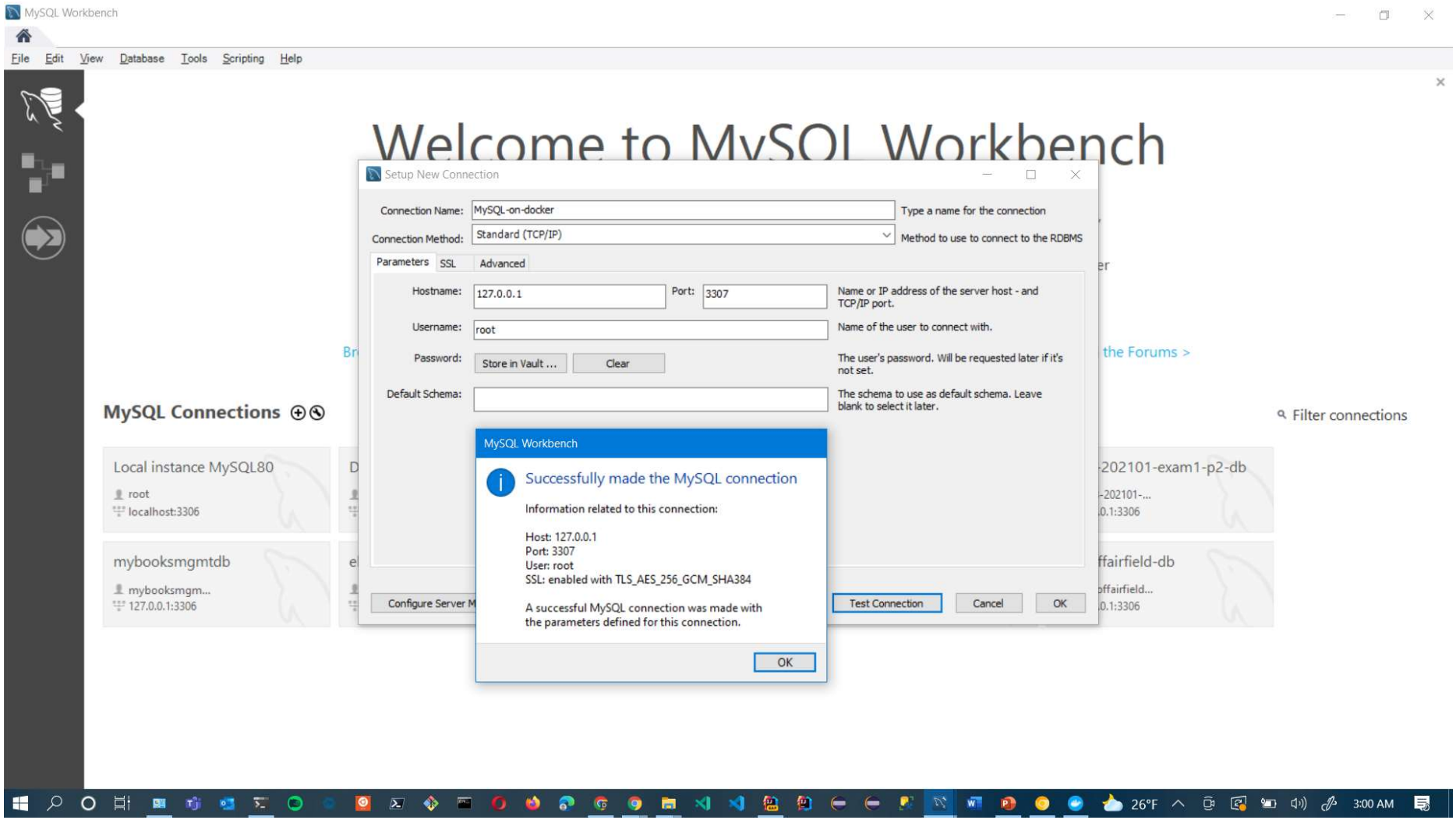
Connect with MySQL shell or client



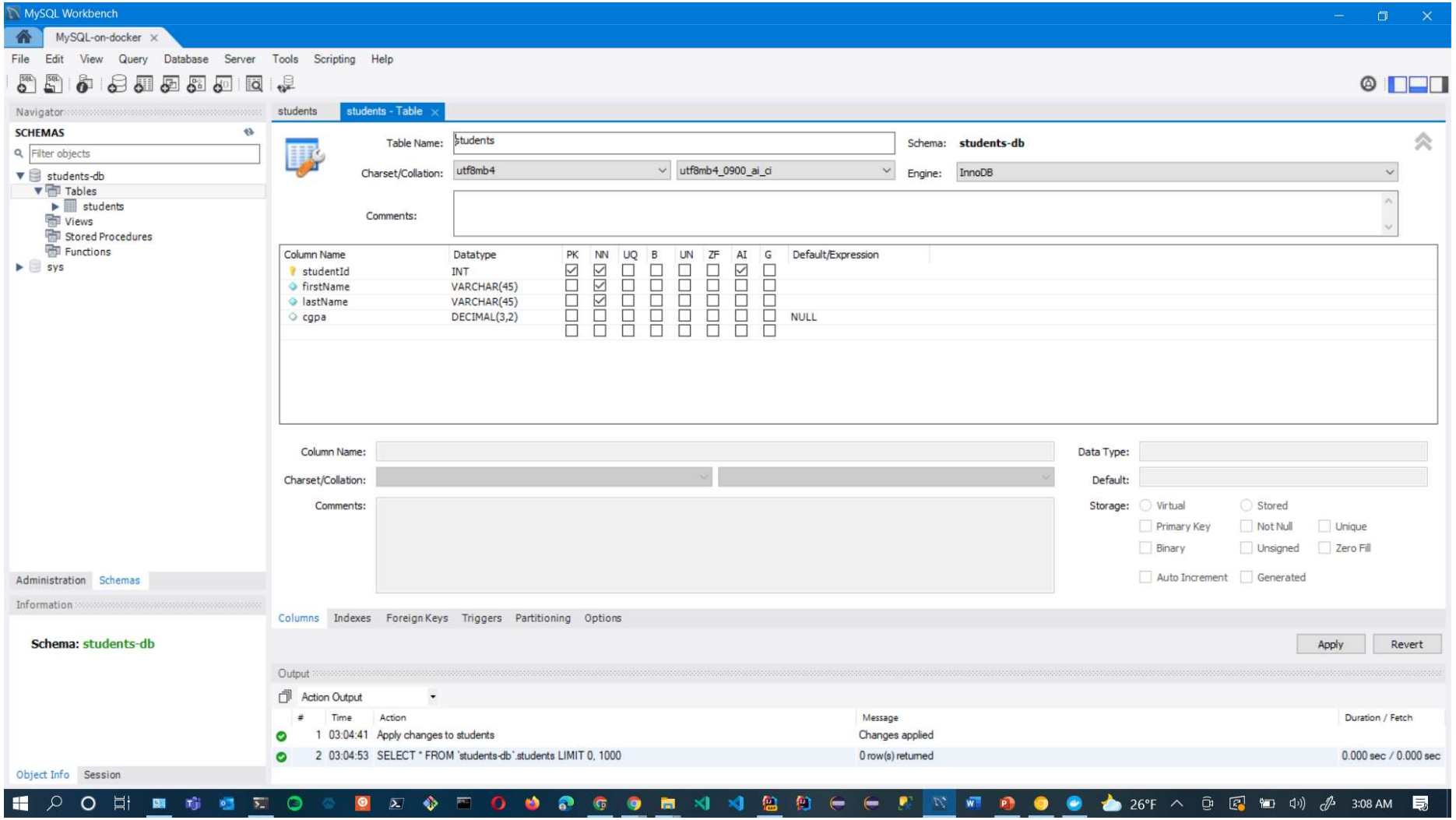
Connect and create a new database



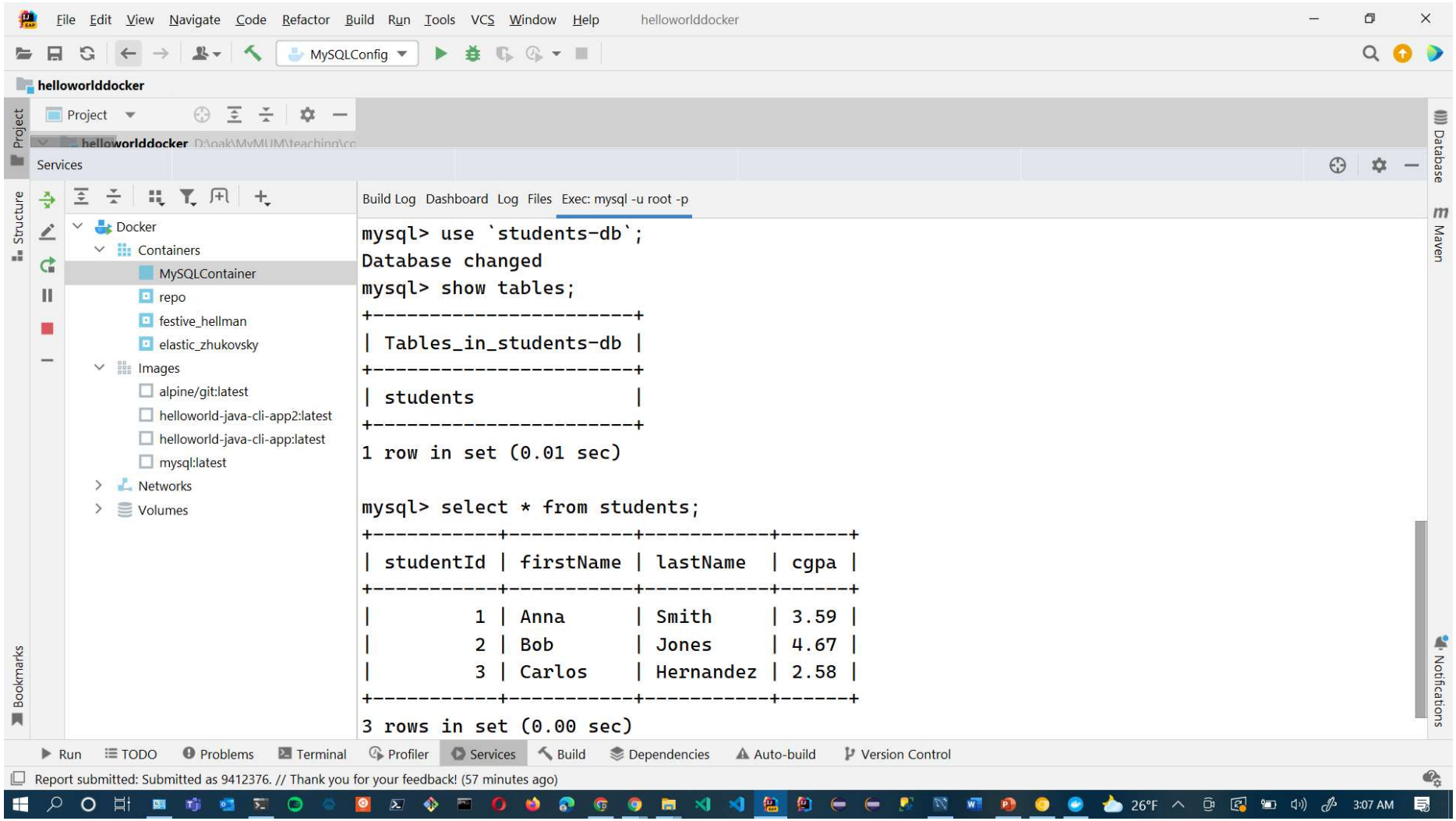
Connecting via MySQL Workbench from the host



Using MySQL Workbench



Working with mysql client in IntelliJ



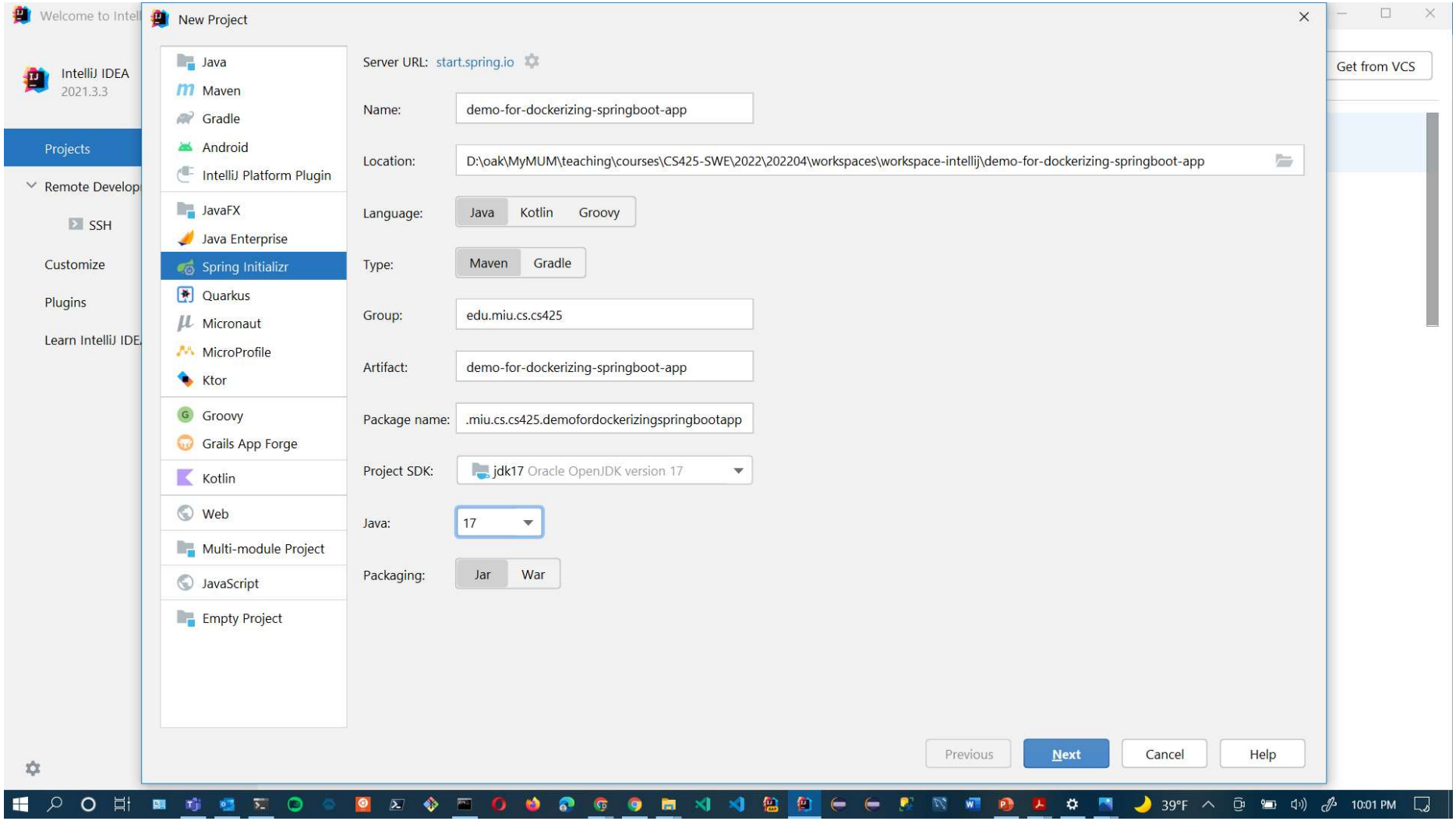
Containerizing a Spring Boot Application

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

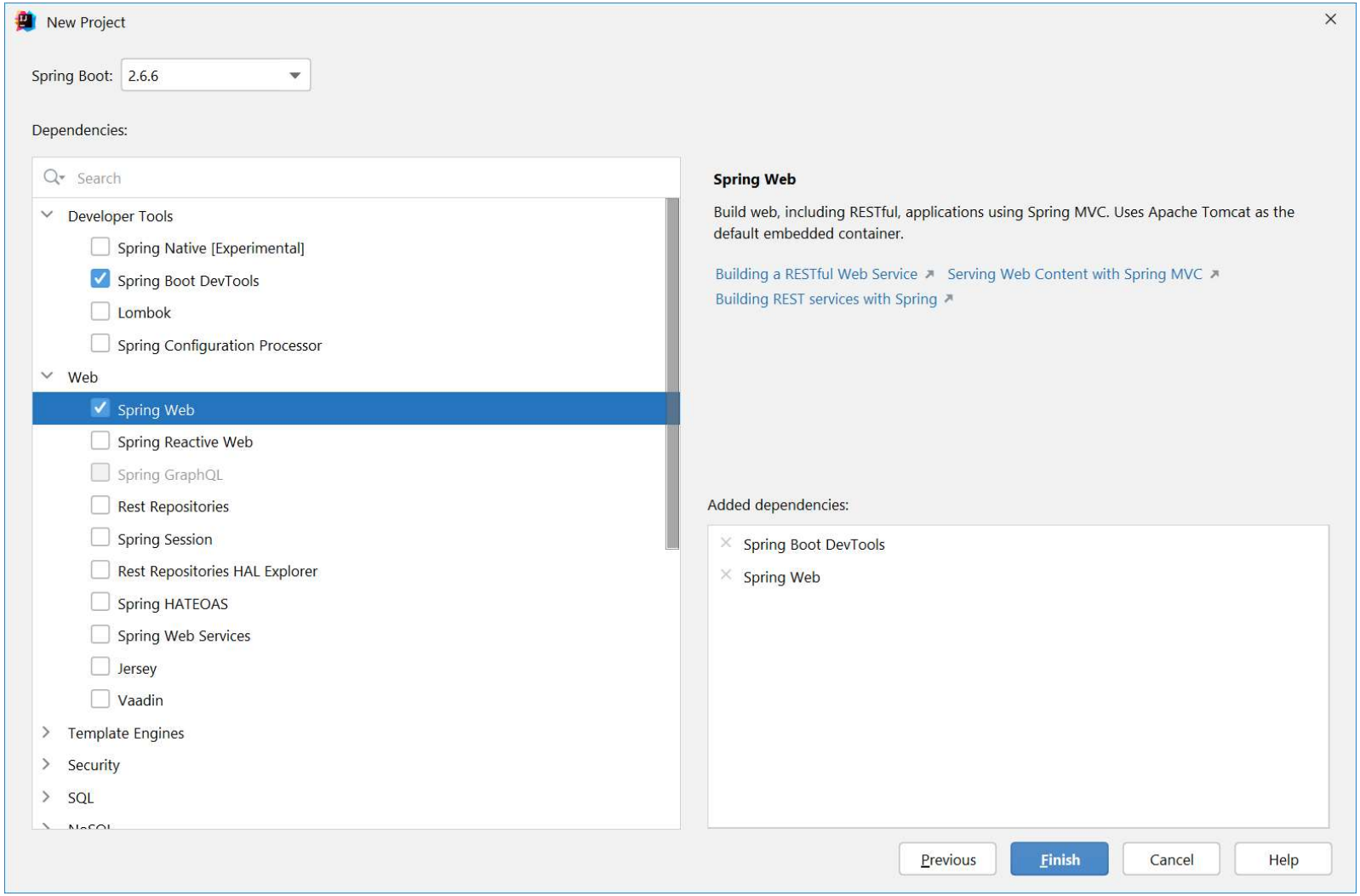
Containerize Spring Boot App with Docker

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

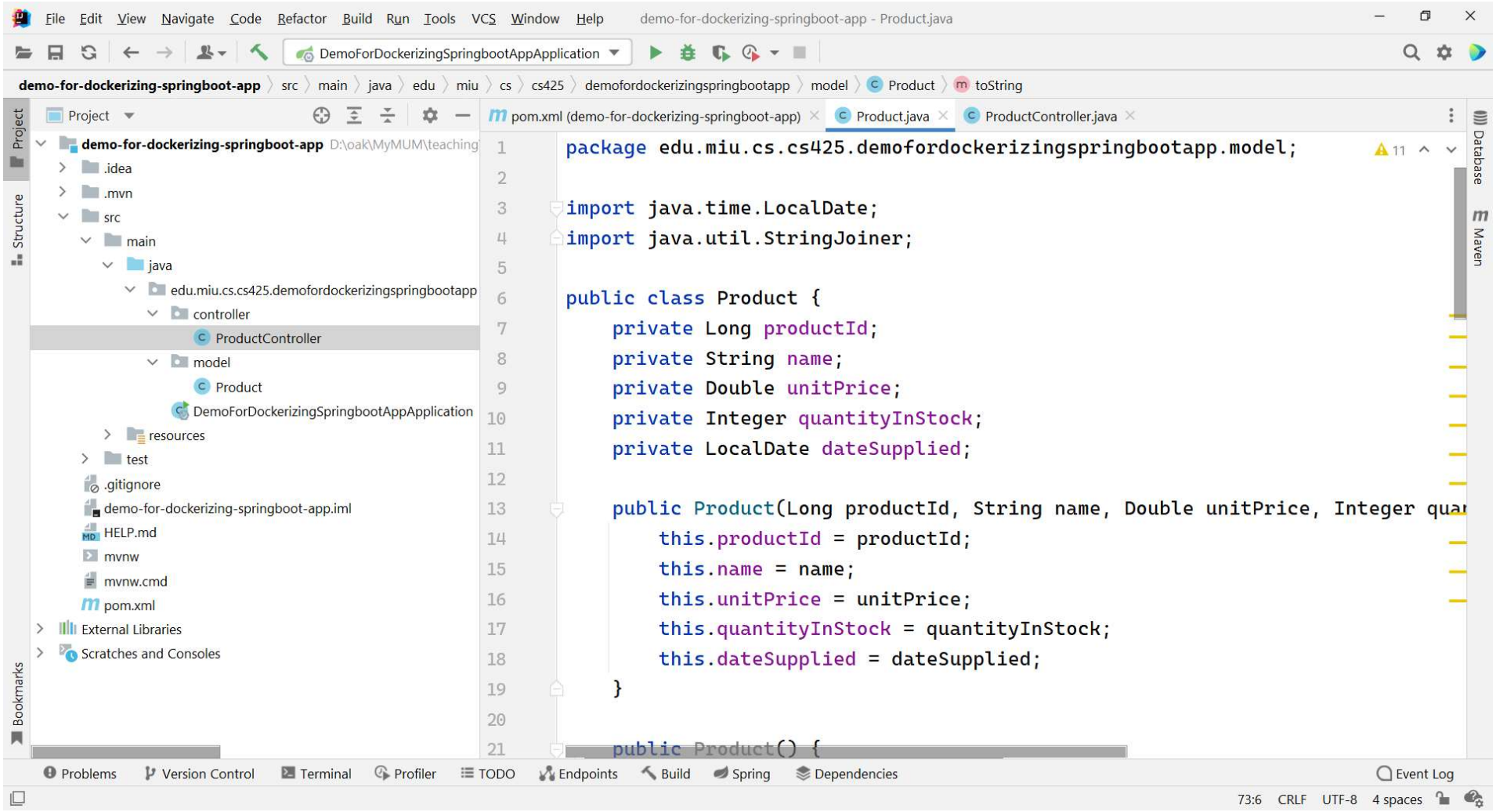
Create a new Spring Initializer Project in IntelliJ IDEA



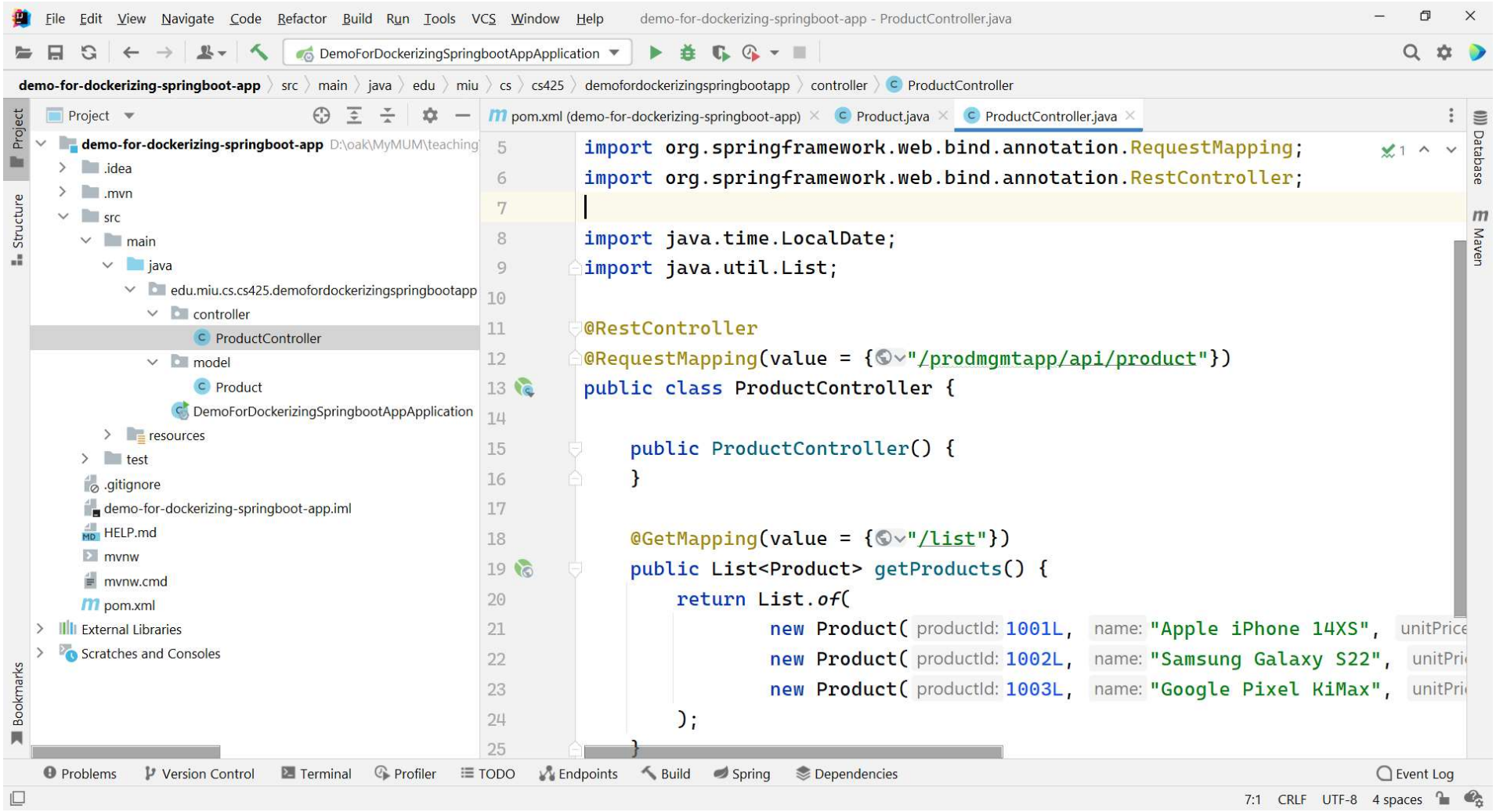
For the Dependencies: Spring Boot DevTools and Spring Web



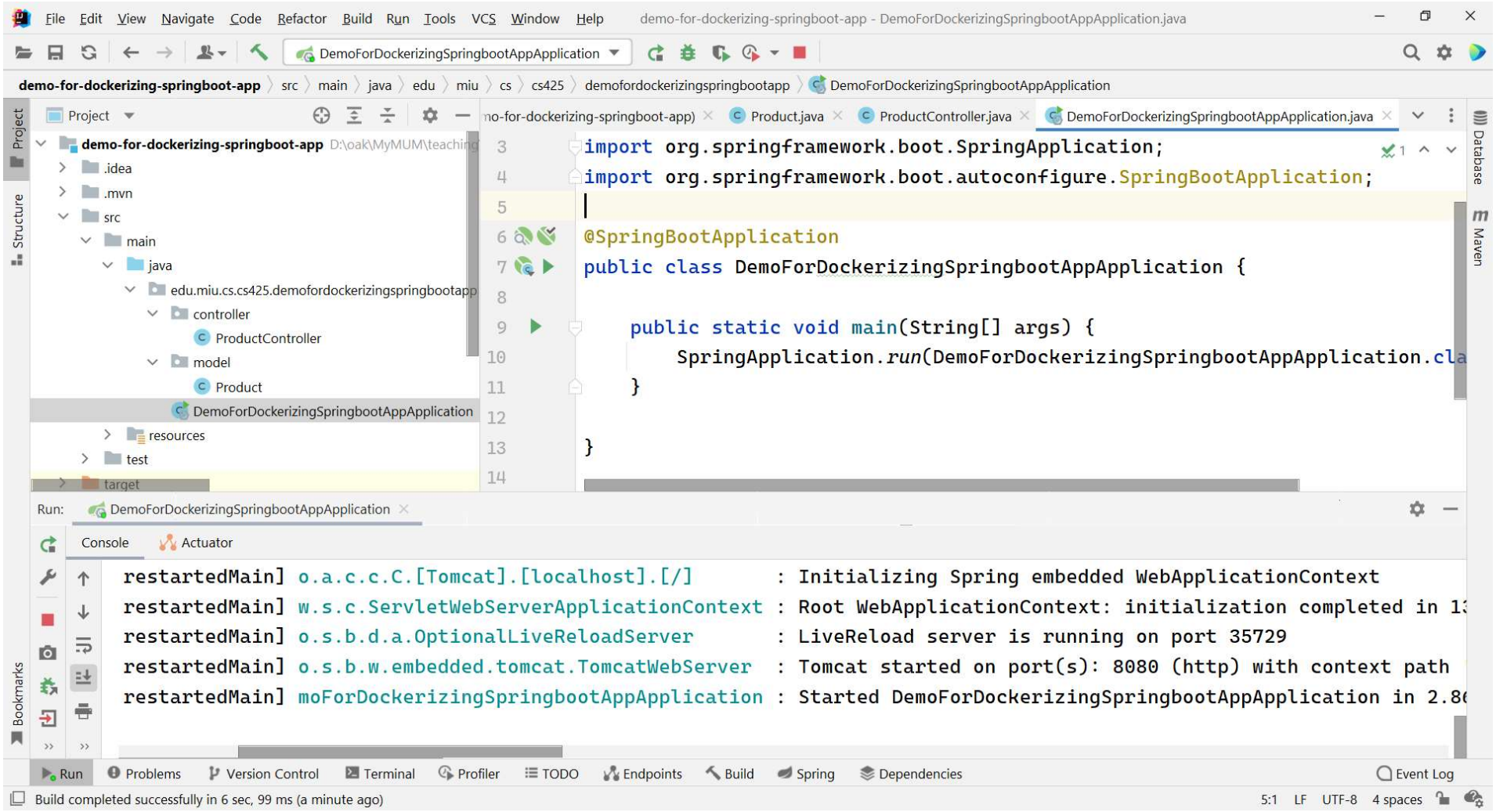
Code model class: product.java



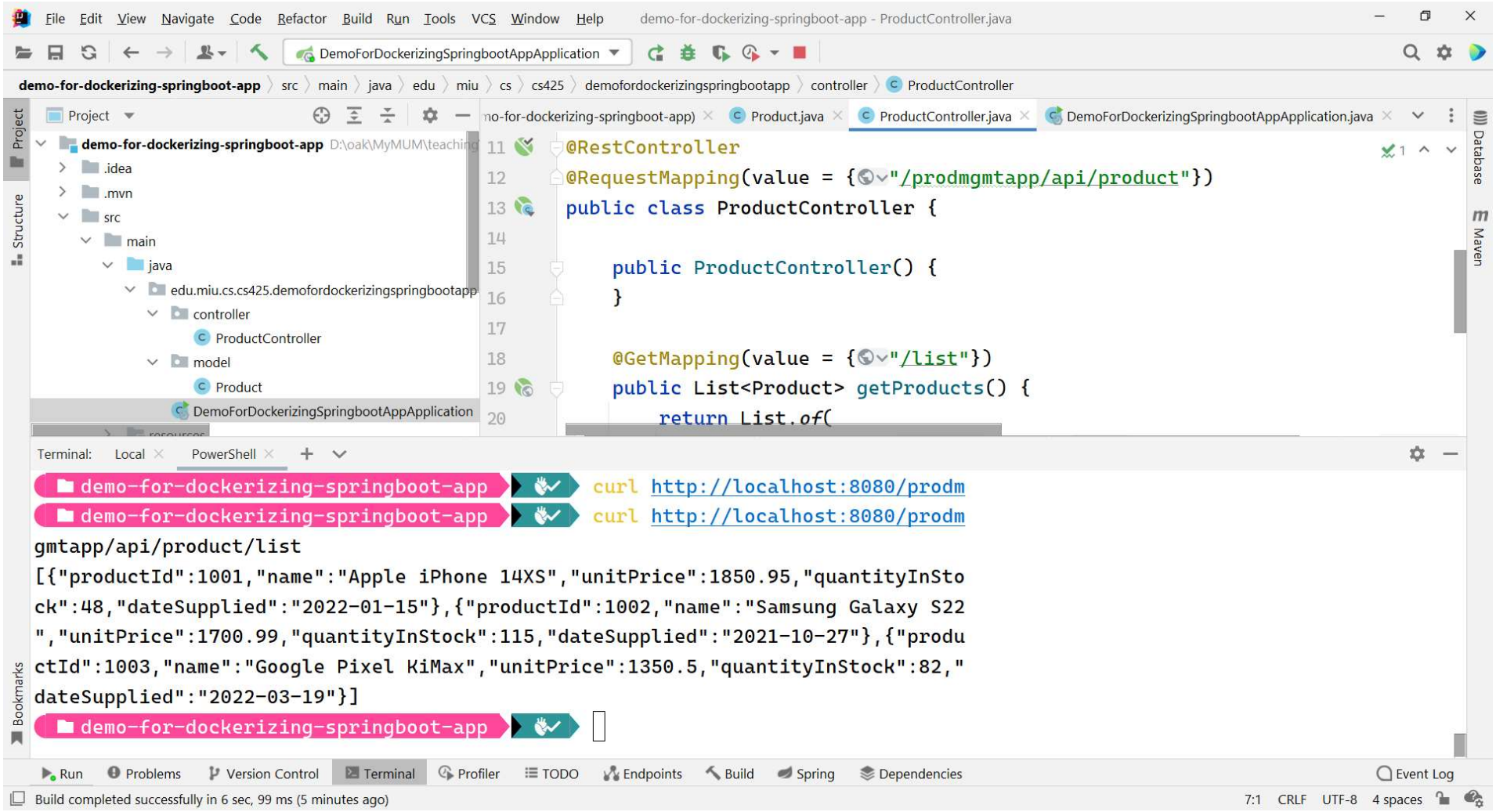
Code the RestController class



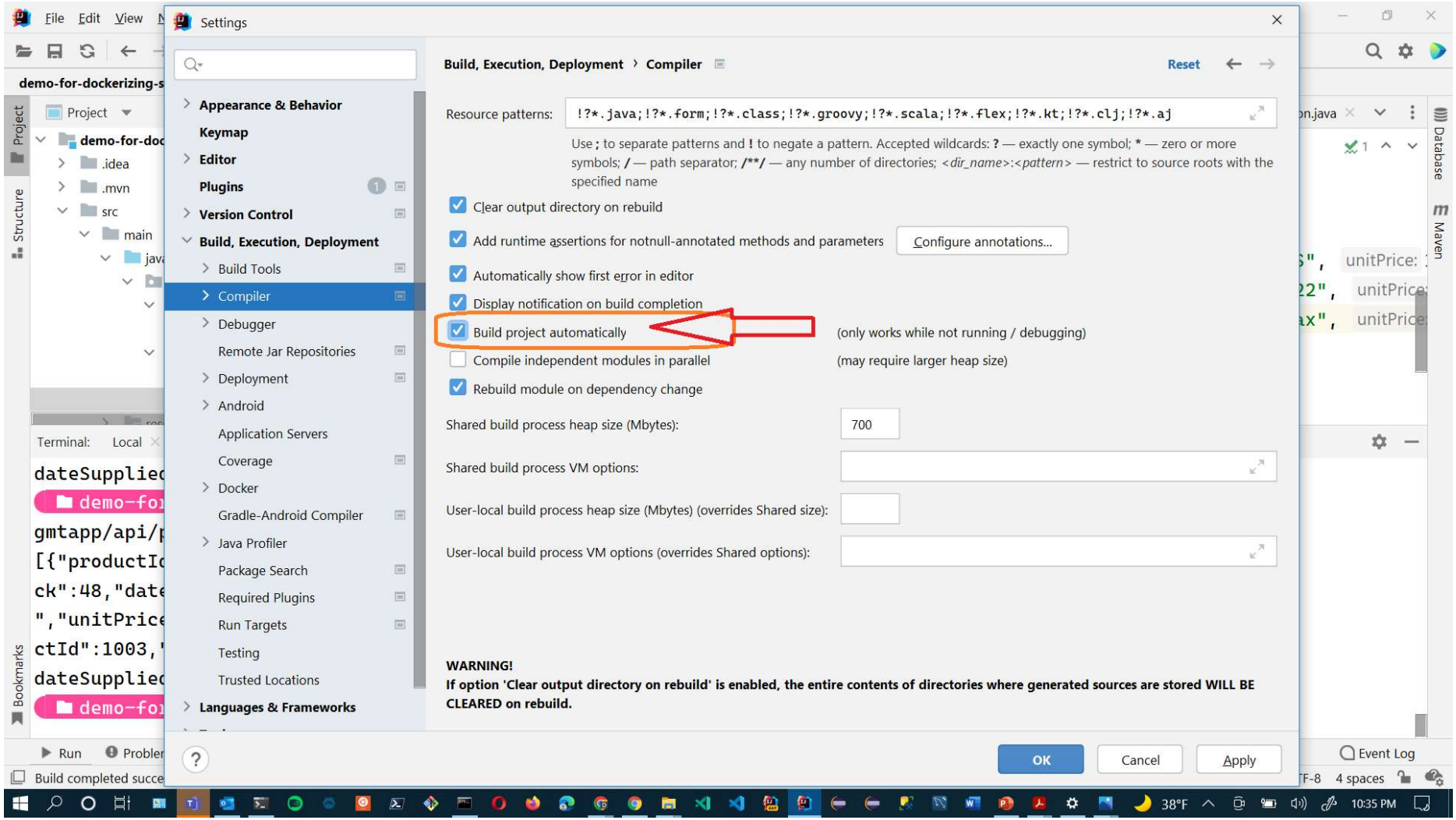
Run the App


File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
demo-for-dockerizing-springboot-app - DemoForDockerizingSpringbootAppApplication.java
DemoForDockerizingSpringbootAppApplication
demo-for-dockerizing-springboot-app src main java edu miu cs cs425 demofordockerizingspringbootapp DemoForDockerizingSpringbootAppApplication
Project
demo-for-dockerizing-springboot-app D:\oak\MyMUM\teaching
.idea
.mvn
src
main
java
edu.miu.cs.cs425.demofordockerizingspringbootapp
controller
ProductController
model
Product
DemoForDockerizingSpringbootAppApplication
resources
test
target
Product.java
ProductController.java
DemoForDockerizingSpringbootAppApplication.java
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class DemoForDockerizingSpringbootAppApplication {
8
9 public static void main(String[] args) {
10 SpringApplication.run(DemoForDockerizingSpringbootAppApplication.cla
11 }
12
13 }
14
Run: DemoForDockerizingSpringbootAppApplication
Console
Actuator
restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1
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 2.8
Run Problems Version Control Terminal Profiler TODO Endpoints Build Spring Dependencies Event Log
Build completed successfully in 6 sec, 99 ms (a minute ago)
5:1 LF UTF-8 4 spaces

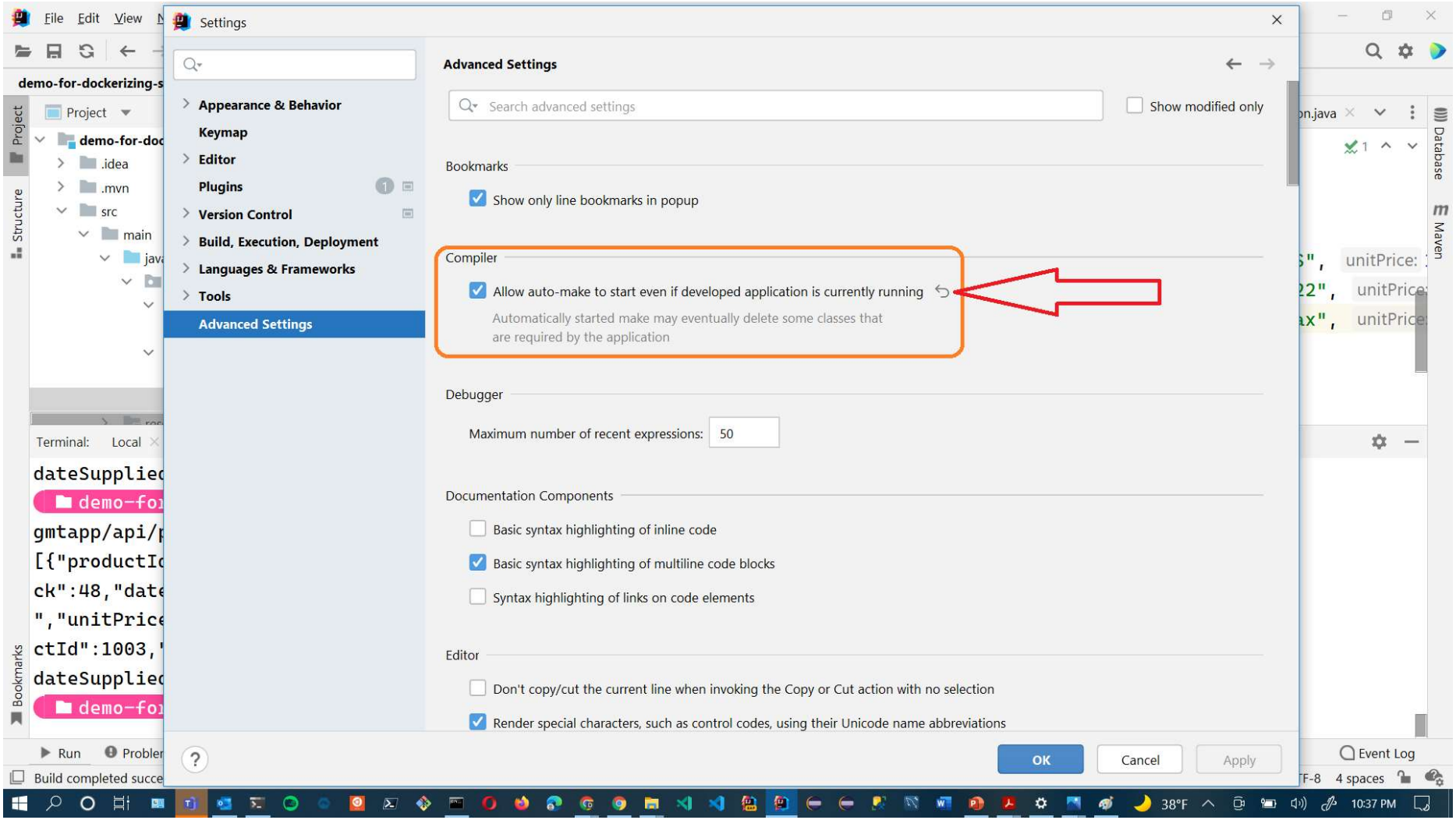
Using curl or browser run HTTP GET url



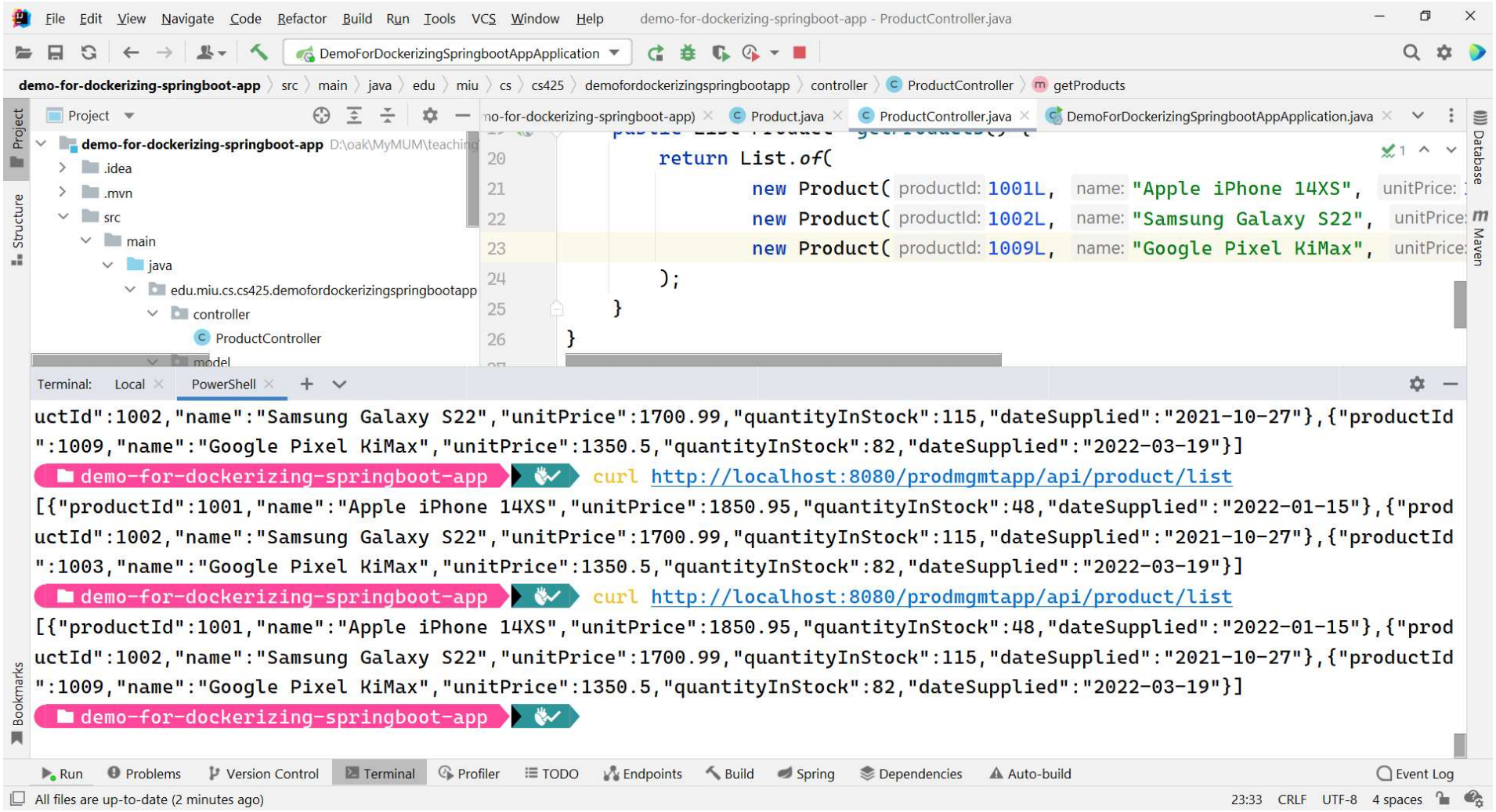
Open IntelliJ Settings and enable this:



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)

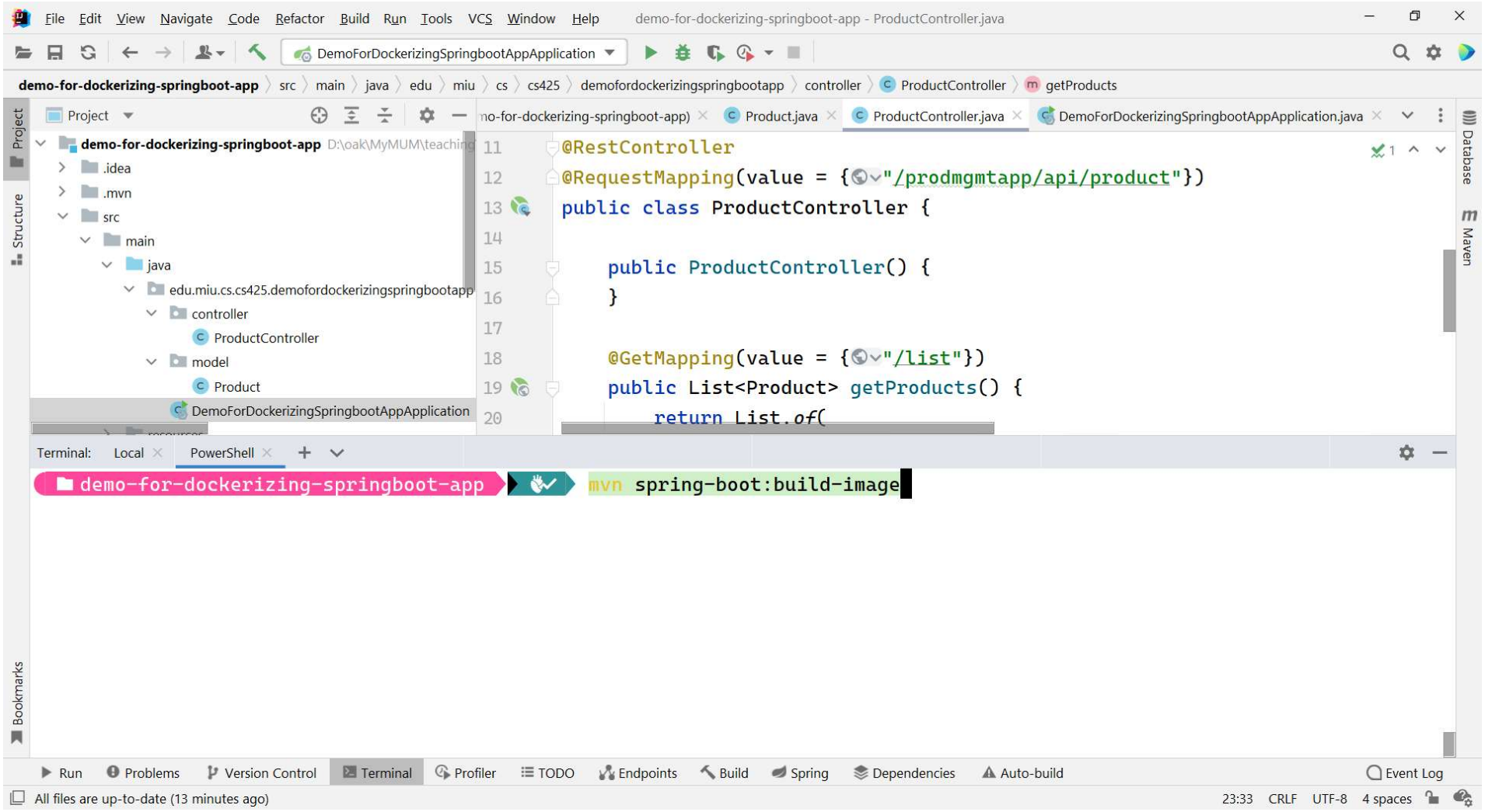


How to Build Docker Image of the Spring Boot Application

- 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:

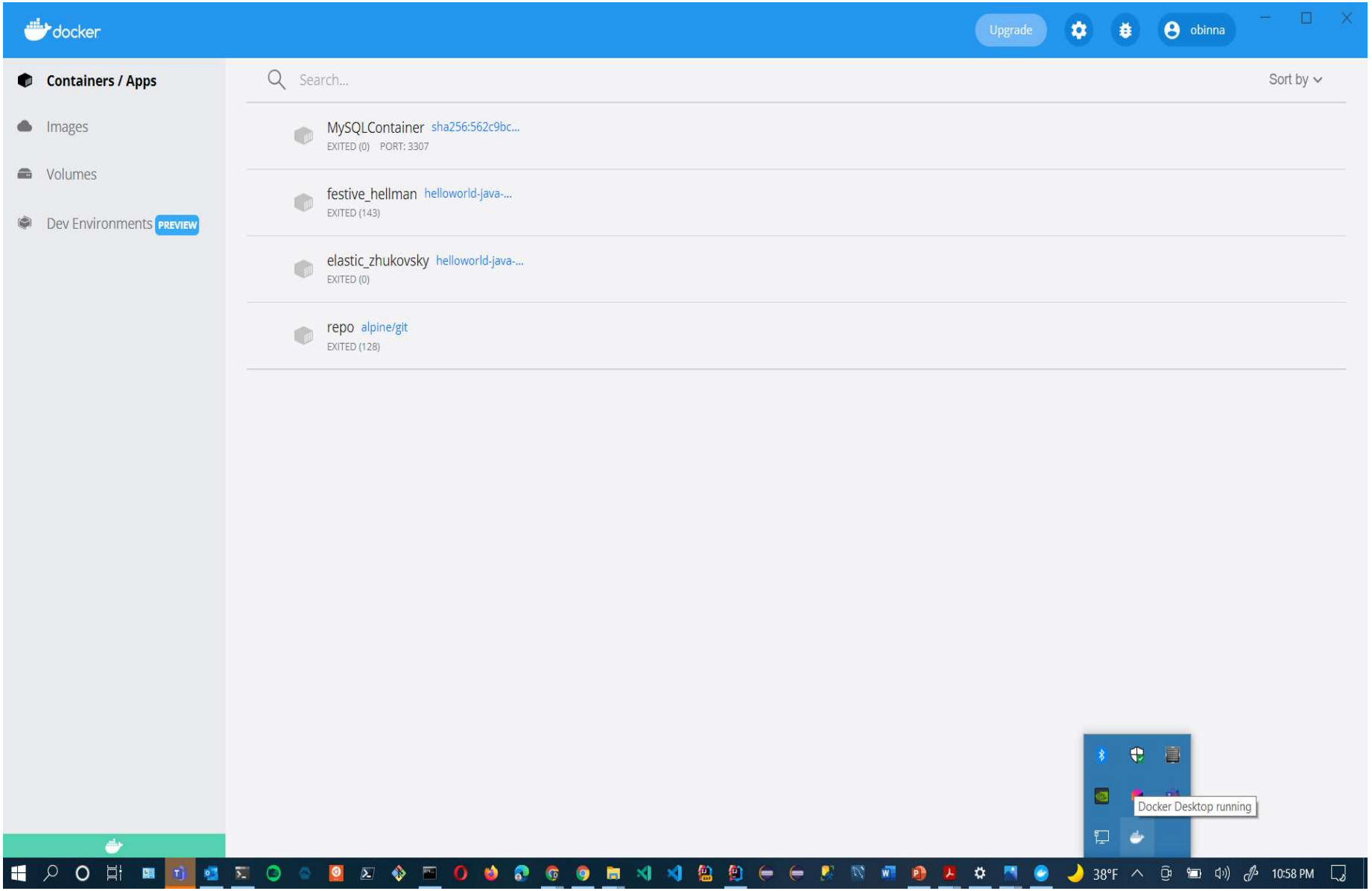
```
> mvn spring-boot:build-image
```

Build Image

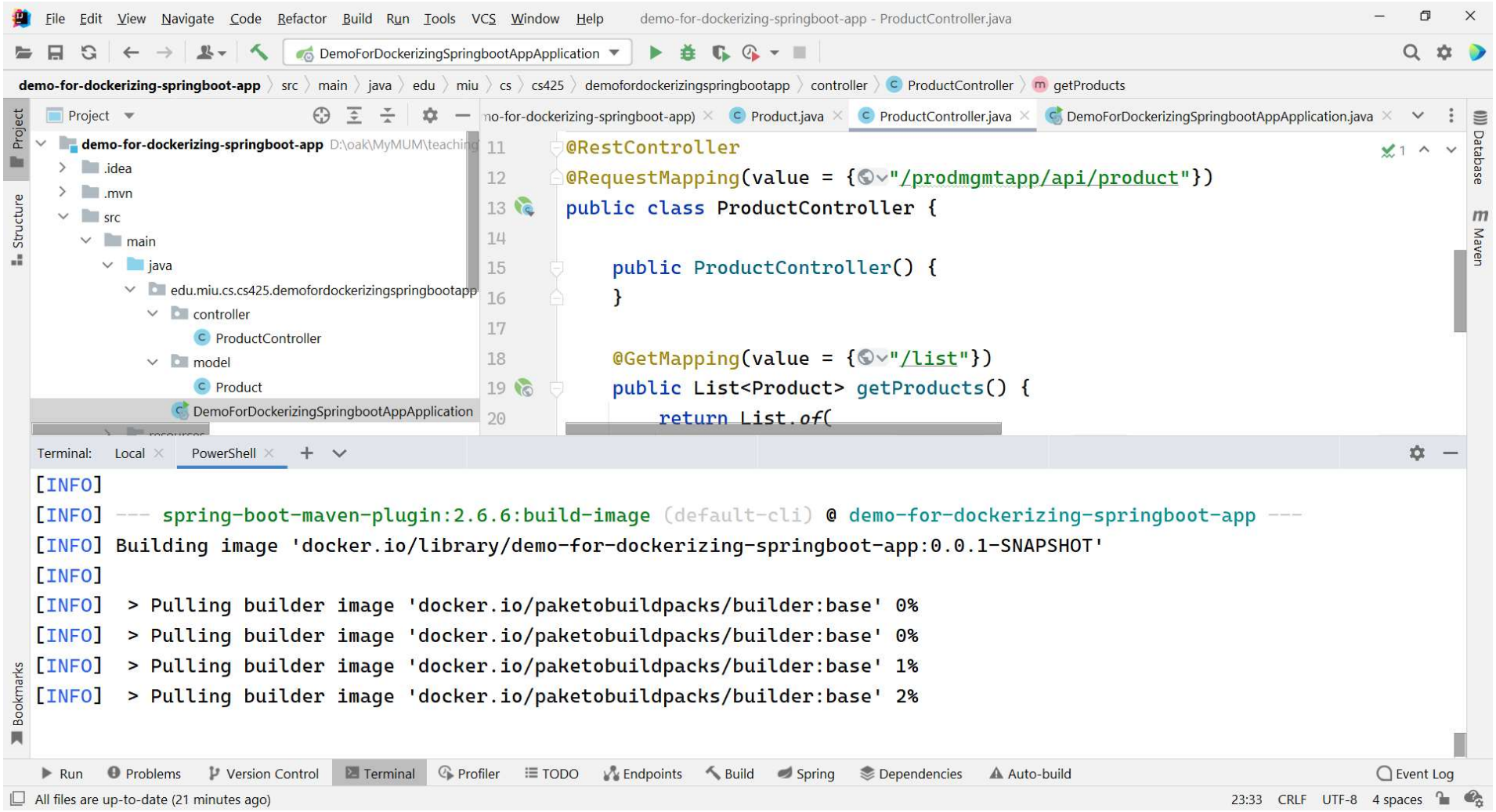


Note: Make sure docker daemon is running before executing build-image

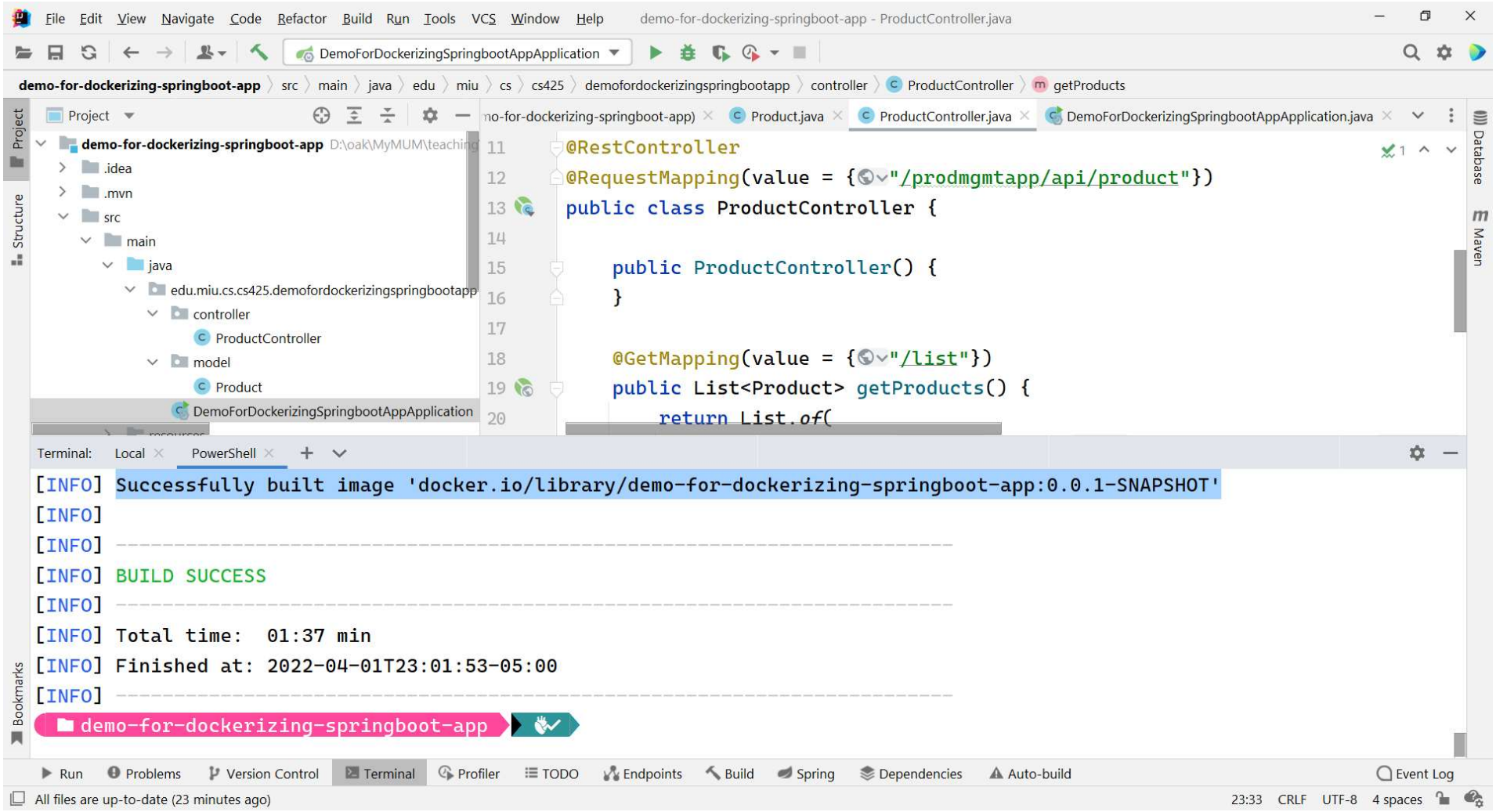
Docker is up and running



Building Docker Image



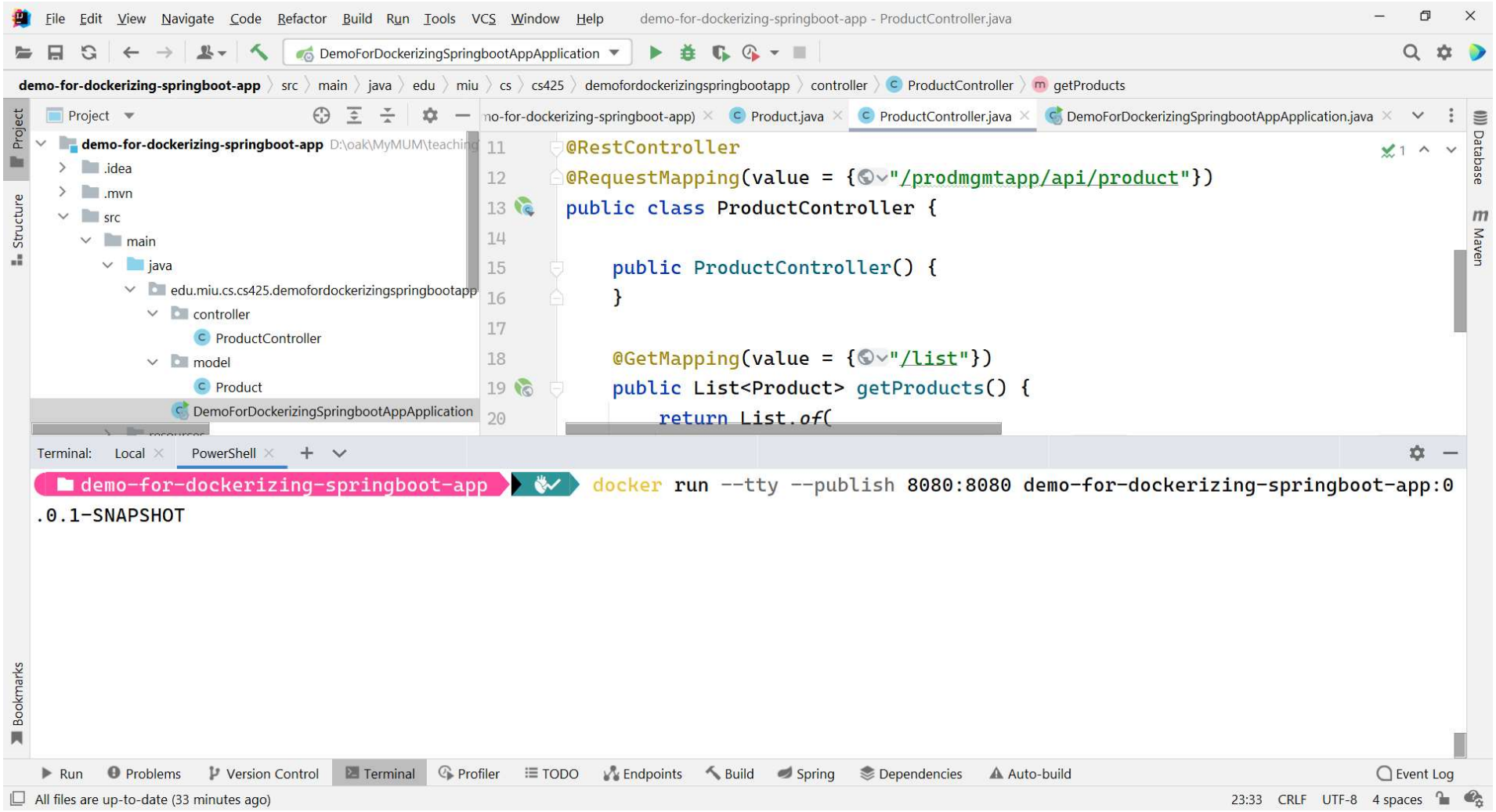
Successfully Built Image



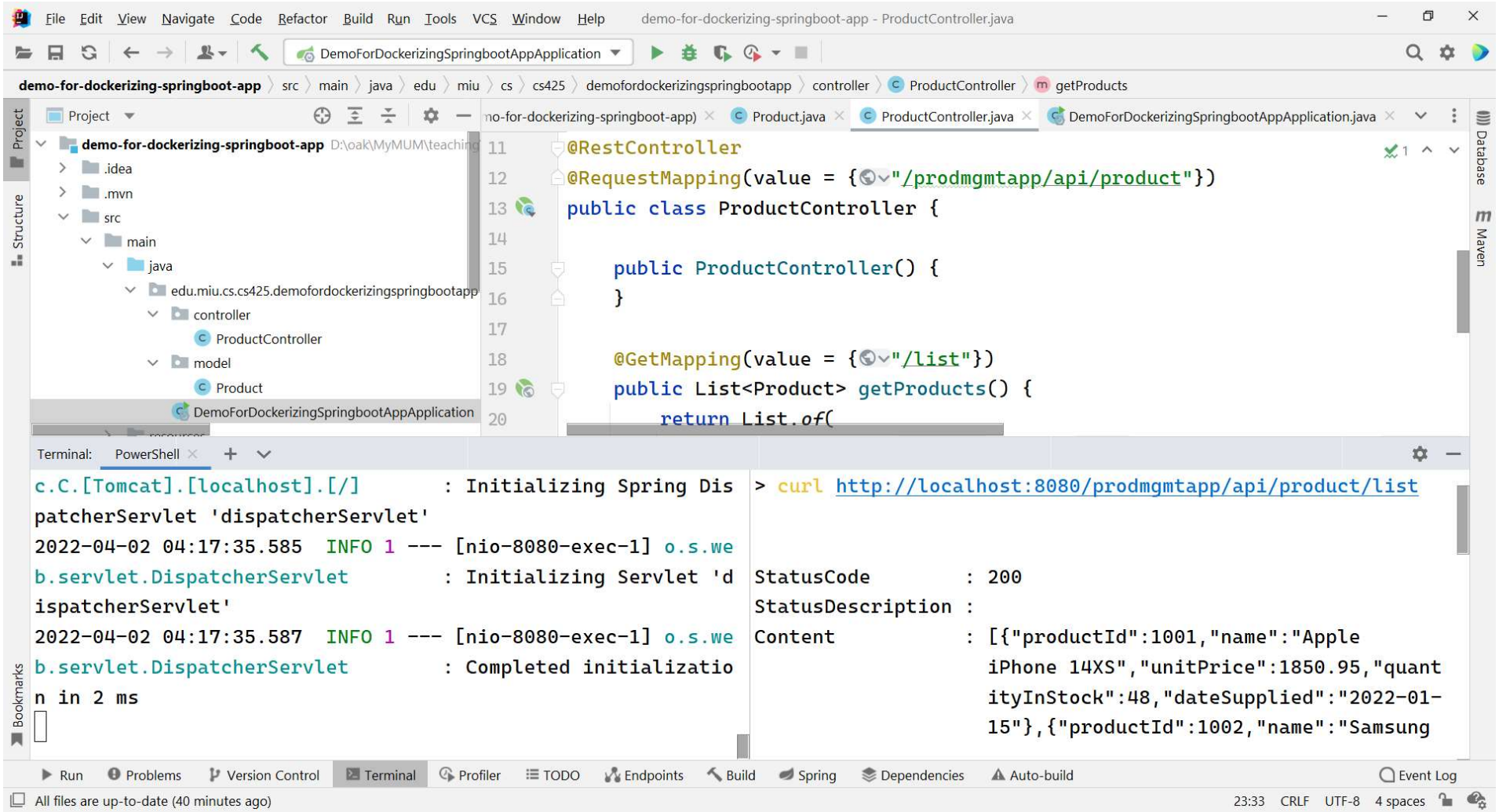
Start the Application in Docker container

- 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

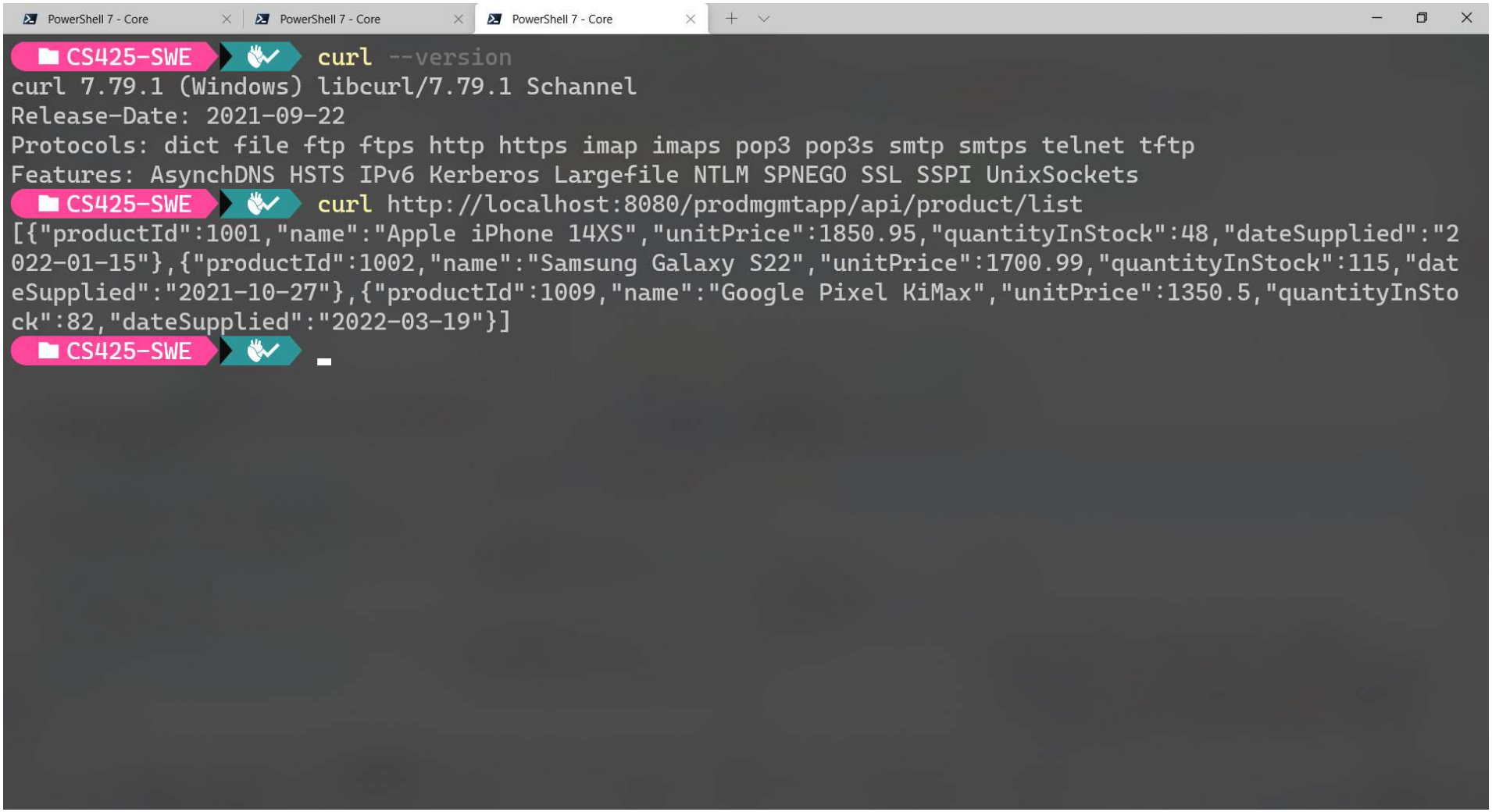
Creates and Run a container



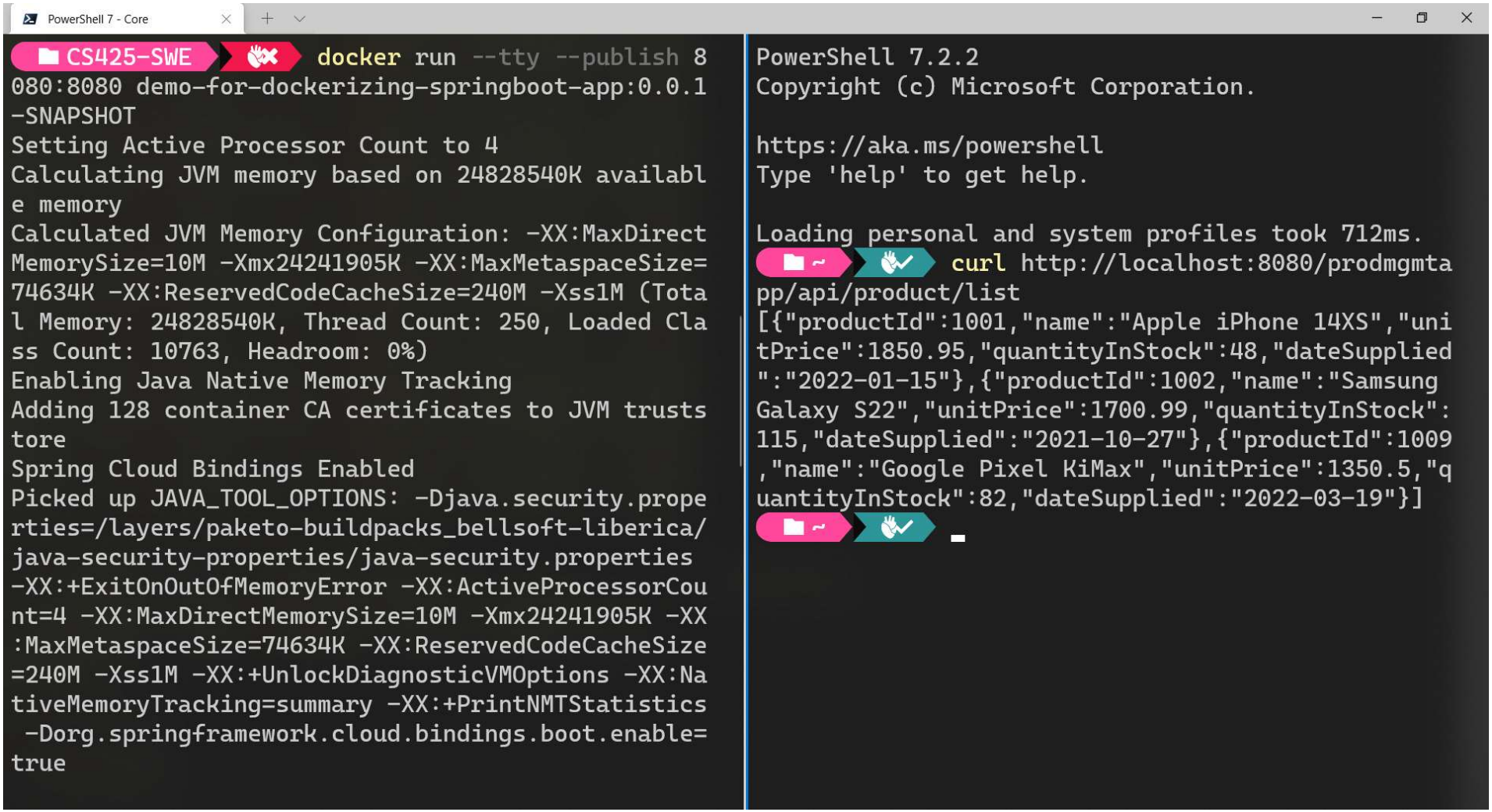
Test the container endpoint



Test the Container Endpoint



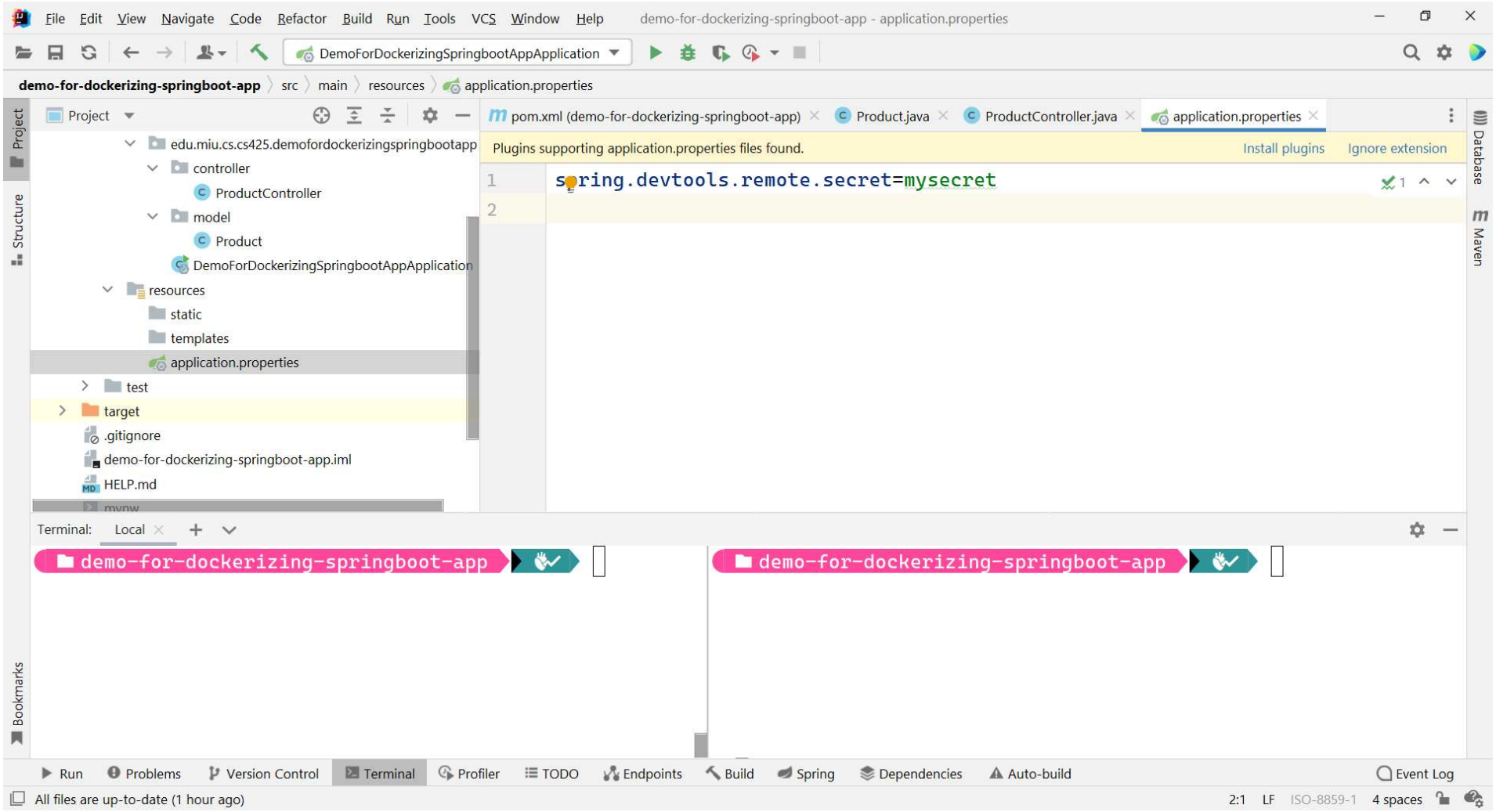
In Split Screen



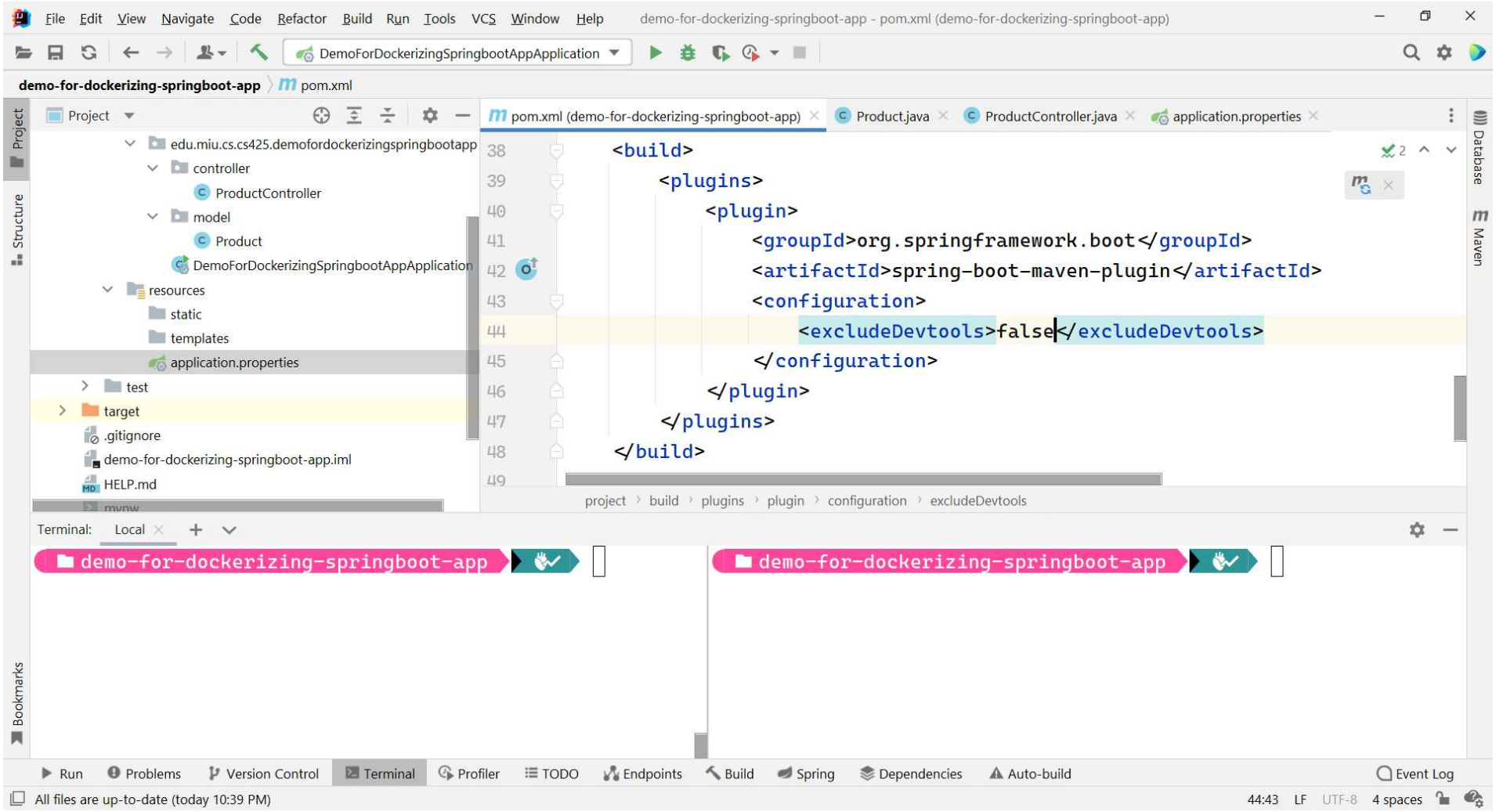
Next: Enable DevTools auto-reload in the container

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

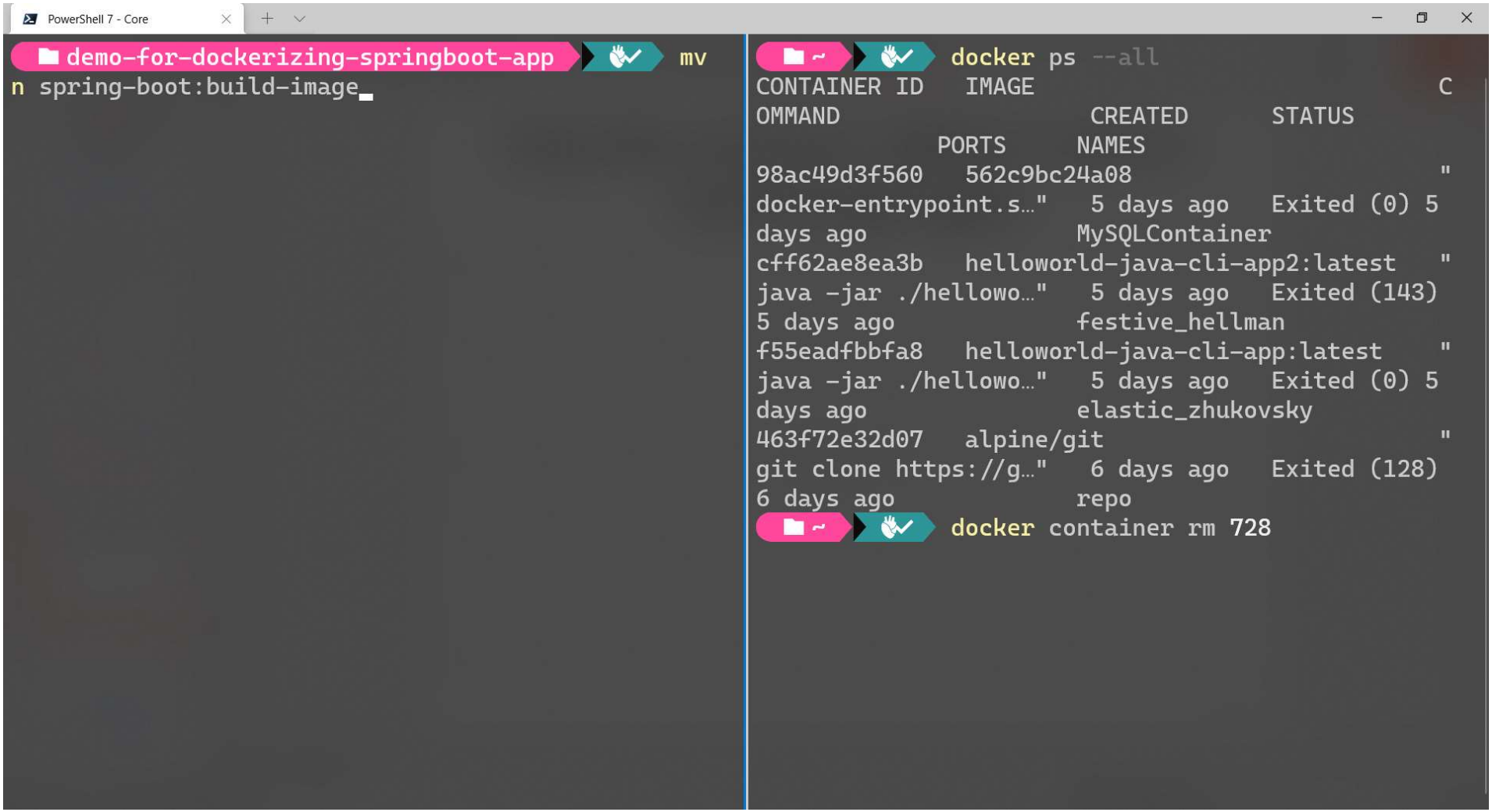
application.properties



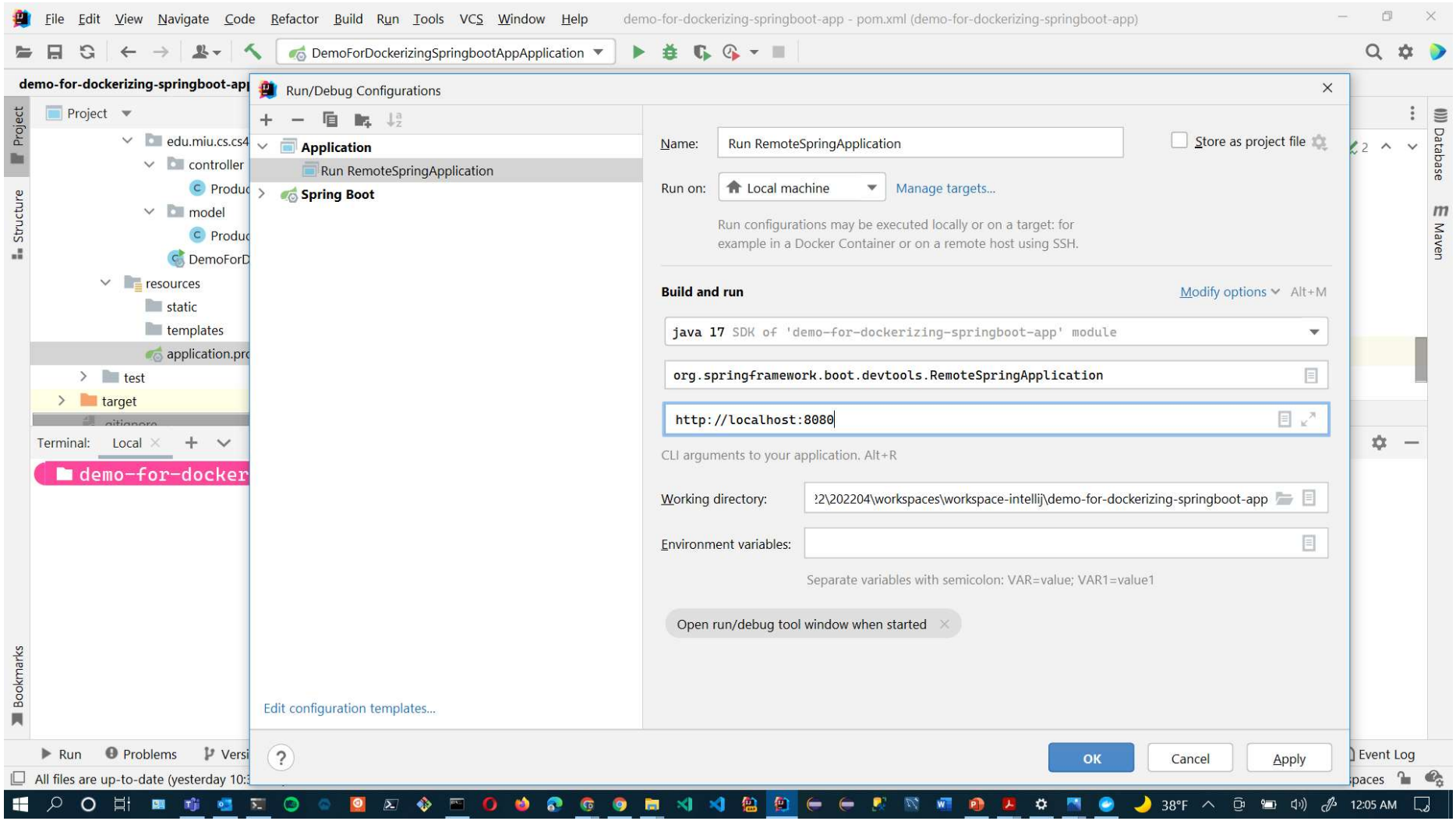
pom.xml


File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
demo-for-dockerizing-springboot-app - pom.xml (demo-for-dockerizing-springboot-app)
DemoForDockerizingSpringbootAppApplication
demo-for-dockerizing-springboot-app
pom.xml
Project
edu.miu.cs.cs425.demofordockerizingspringbootapp
controller
ProductController
model
Product
DemoForDockerizingSpringbootAppApplication
resources
static
templates
application.properties
test
target
.gitignore
demo-for-dockerizing-springboot-app.iml
HELP.md
mvnw
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>
49
project › build › plugins › plugin › configuration › excludeDevtools
Terminal: Local
demo-for-dockerizing-springboot-app
demo-for-dockerizing-springboot-app
Run Problems Version Control Terminal Profiler TODO Endpoints Build Spring Dependencies Auto-build Event Log
All files are up-to-date (today 10:39 PM)
44:43 LF UTF-8 4 spaces
Database
Maven
Bookmarks

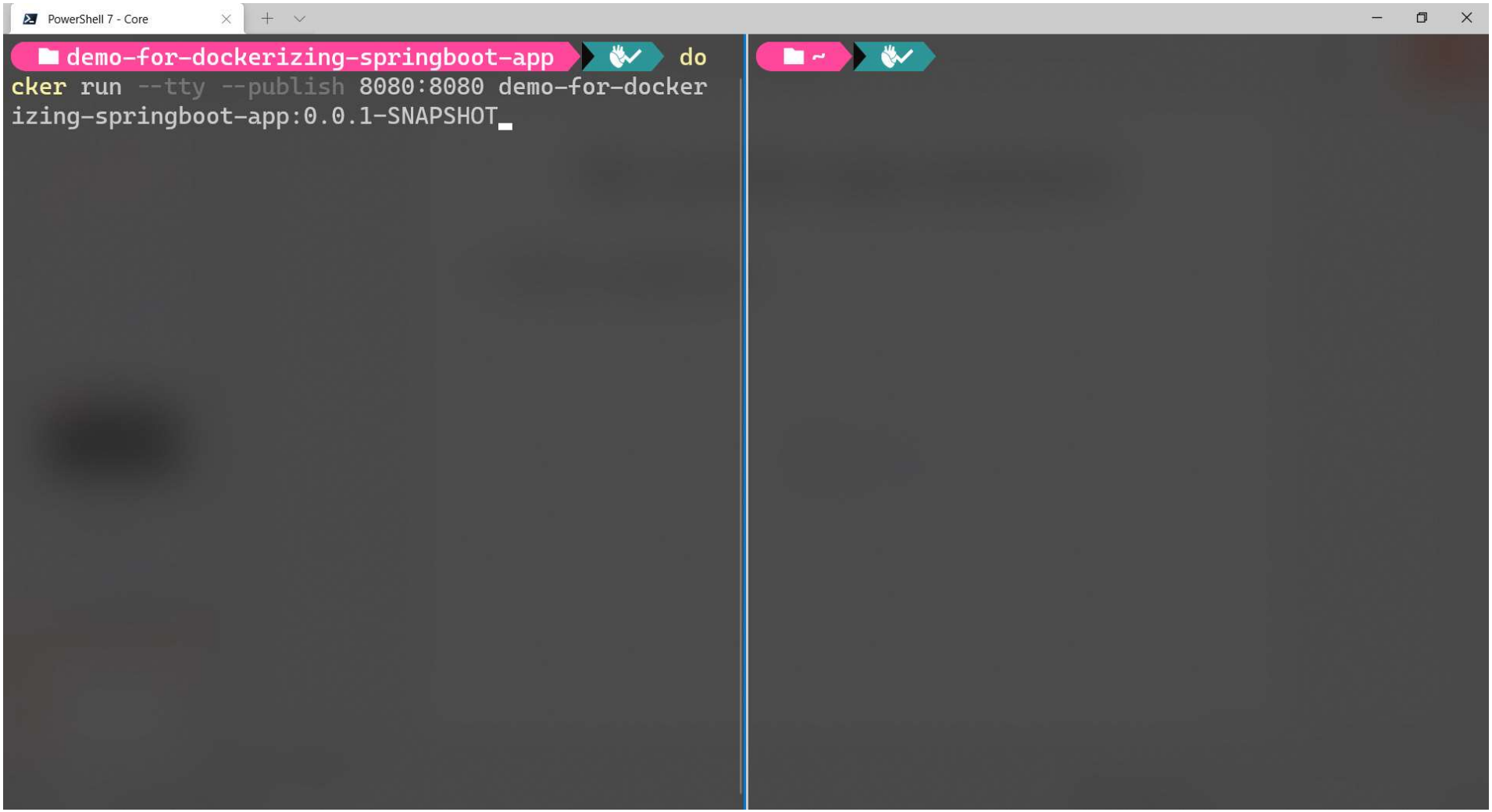
Stop the container, delete it and re-build image again



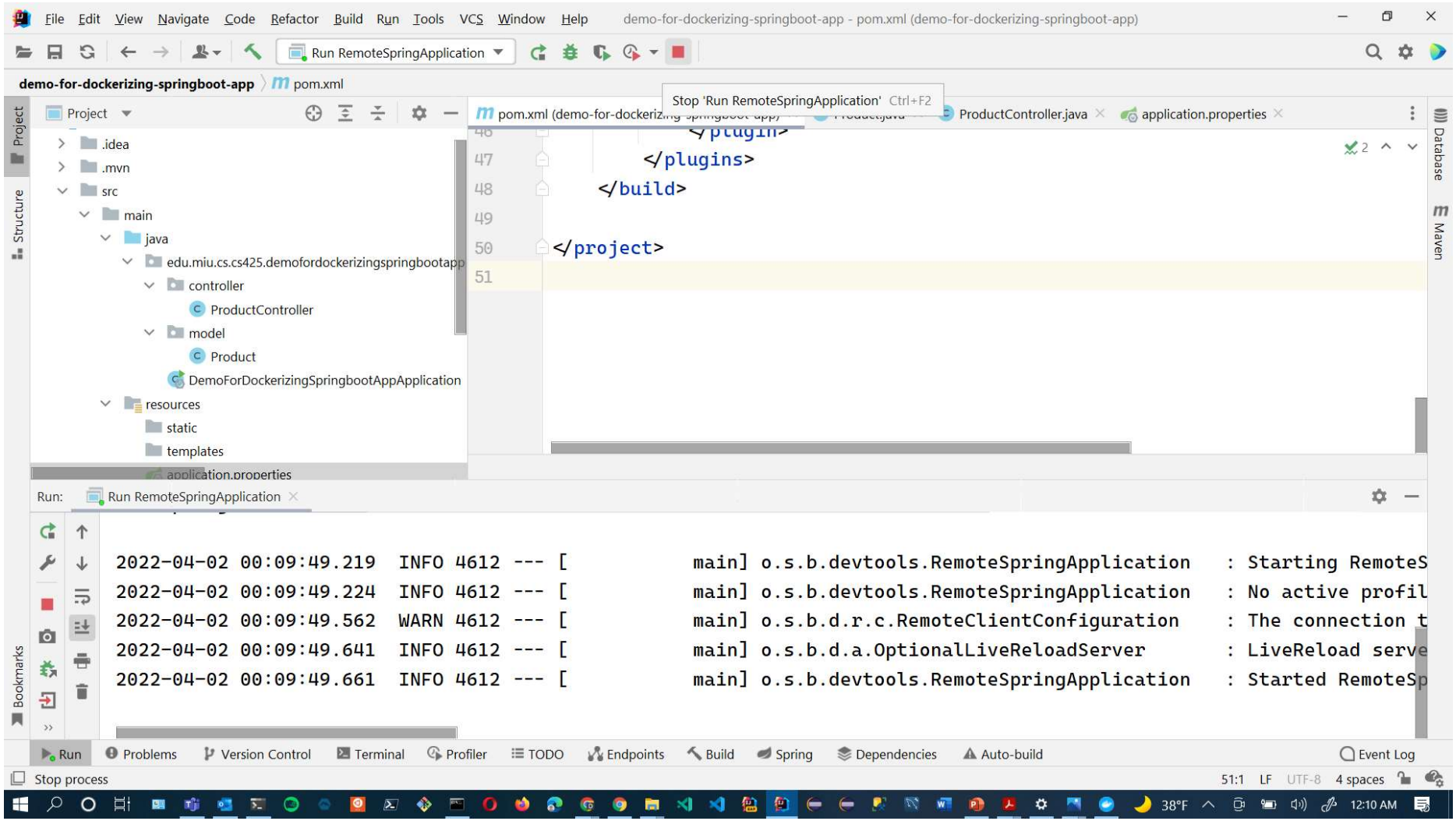
Create a Run Config for a RemoteSpringApplication



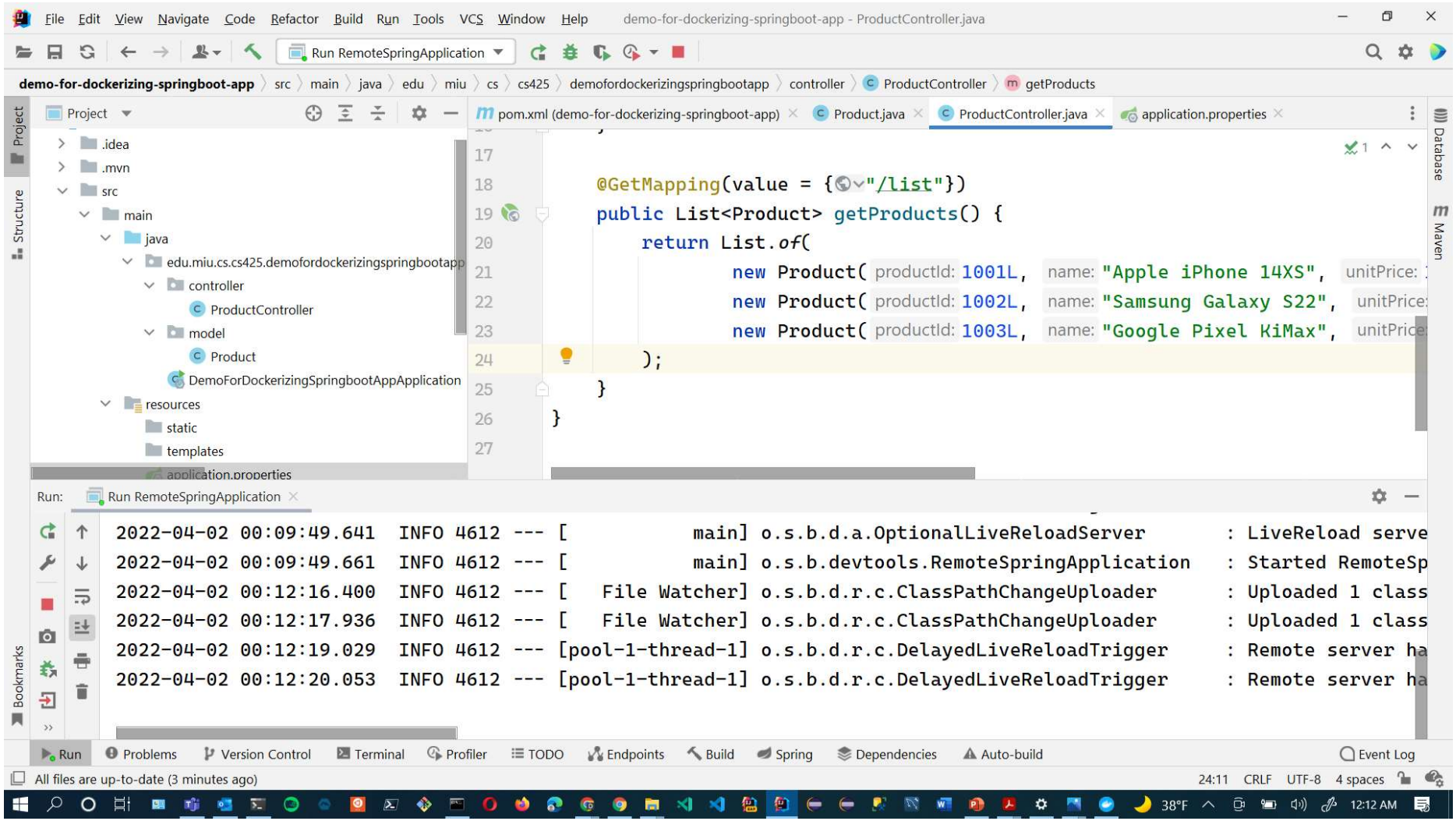
Re-run the app container



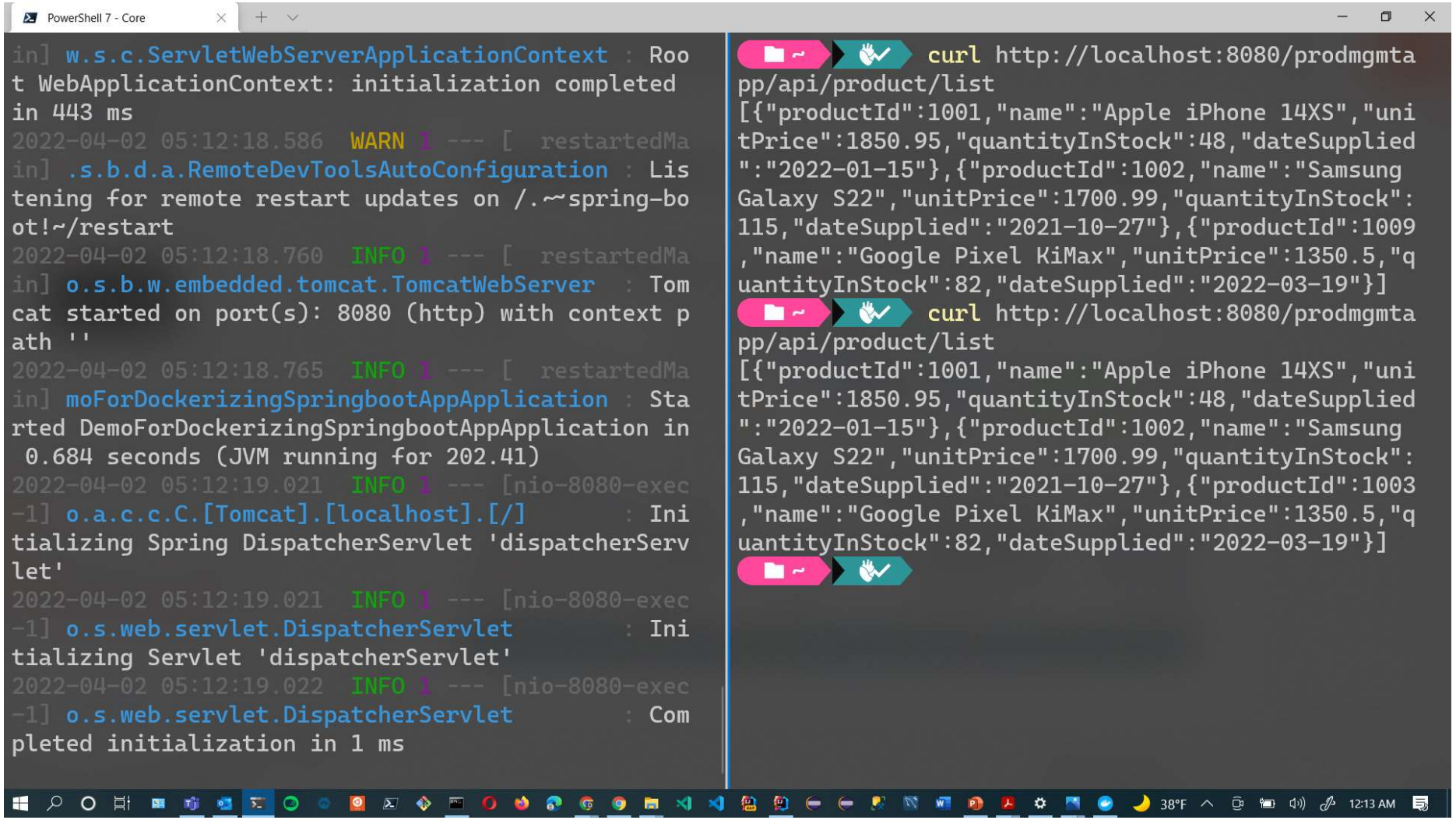
Run the RemoteSpringApplication



Modify the code and see it reload



Verify the output



CS489:
Software Development