# Creating Docker Images

## Overview

In this lab, you'll define a Dockerfile to build an image for an existing Java Spring Boot web application. You will then use the Dockerfile to run some containers based on this image.

## Getting started

Start a 'Play with Docker' session and run the following command to download all the sample code into the environment:

wget http://olsensoft.com/DockerOpenShift/DoshDev.zip -O temp.zip; \

unzip temp.zip; \

rm temp.zip

## Source folders for this lab

* DoshDev/Student/02-CreatingDockerImages
* DoshDev/Solutions/02-CreatingDockerImages

## Roadmap

There are 4 exercises in this lab, of which the last exercise is "if time permits". Here is a brief summary of the tasks you will perform in each exercise; more detailed instructions follow later:

1. Familiarization
2. Writing the Dockerfile
3. Running some containers
4. (If Time Permits) Additional suggestions

## Exercise 1: Familiarization

In the 'Play with Docker' environment, go to the following directory:

DoshDev/Student/02-CreatingDockerImages

Then go to the SpringBootWebApp subdirectory, where we've implemented a complete Java Spring Boot web application. Here's an overview of the application:

* The src/main/java/mypackage folder contains the source code for the web app. Spring Boot is a remarkable framework – you can achieve a lot of functionality from a small amount of code. For example, take a look at Application.java (located in the src/main/java/mypackage directory) – this is the complete implementation of a REST service that maps to the relative URL "/".
* The src/main/resources/static folder contains static content for the web application (HTML files, CSS stylesheets, JavaScript script files, etc.). There aren’t any static resources in this simple web app.

## We've already compiled the application and packaged it into a JAR file (via by running the command mvn package in the course virtual machines). Run the following command, to see the JAR file:

## ls -l target/demoapp-0.0.1.jar

## This JAR file contains everything needed to run the web application, including an embedded Tomcat web server (the Tomcat web server listens on port 8080). Spring Boot encourages you to create JAR files that contain an embedded web server, because these applications are standalone and easily containerized ☺.

## Exercise 2: Writing the Dockerfile

In this exercise you'll create a Dockerfile to containerize the Spring Boot application. The easiest way to create files in the 'Play with Docker' environment is to actually create files outside of the environment on your actual Windows machine, and then drag-and-drop the file onto the 'Play with Docker' terminal window.

So as we said, your task is to create a Dockerfile for the Spring Boot application. The Dockerfile must reside in the root folder of the application (i.e. in the SpringBootWebApp folder). Here are the details for the Dockerfile:

* The Dockerfile should build an image based on openjdk:11.0. OpenJDK is an open-source implementation of the Java Standard Edition Platform, upon which you can run Java applications.
* The Dockerfile should receive a build argument (ARG) that specifies the name of the JAR file to be executed within the container.
* The Dockerfile should copy this JAR file into the image, and give it a local name such as myapp.jar within the image.
* The Dockerfile should expose port 8080 from the image (remember, the JAR file contains an embedded Tomcat web server that listens on ort 8080).
* The Dockerfile should specify an ENTRYPOINT instruction to run the JAR file on the Java platform. Here's the normal command to do this:

java –jar /myapp.jar

Save your Dockerfile. Then run a docker build command to build a Docker image. Here are some hints (the *solution* folder has a readme1.txt file if you need some help):

* Specify an image tag name such as myspringbootwebapp.
* Pass a build-arg into the Dockerfile, specifying the JAR file to be copied into the image and executed within it. The name of the JAR file is target/demoapp-0.0.1.jar.

Verify that the docker build command runs successfully and creates an image with your specified name, e.g. myspringbootwebapp.

## Exercise 3: Running some containers

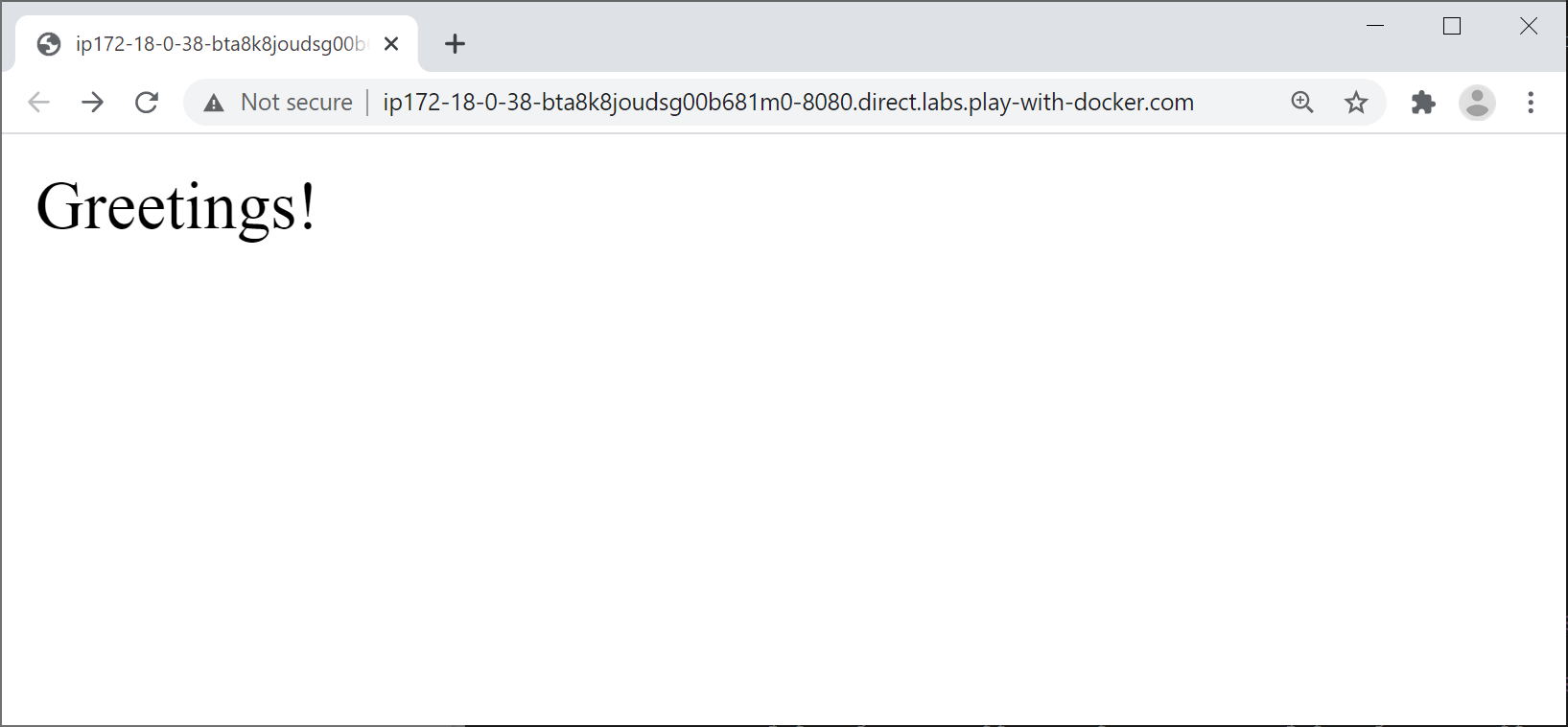
Run a container based on your myspringbootwebapp image. Give your container a name such as myspringbootwebapp1, and map port 8080 within the container to a suitable port on your host computer, e.g. 8080 (the *solution* folder has a readme2.txt file if you need help).

Run a curl command as follows:

curl localhost:8080

Verify it returns an HTML page with a “Greetings!” message.

Then open port 8080 in the 'Play with Docker' environment. You should see the following message, indicating your container is running your Spring Boot web application correctly:



## Exercise 4 (If time permits): Additional suggestions

## Create some more containers with different names, mapped to different ports on your host computer. Verify you can navigate to each web application successfully via curl, using the appropriate mapped port number.