| GROUP3 MEMBERS MESUT BUNALDI  MUHAMMED KANDUR  HİKMET TÜTÜNCÜ  ONGUN ALP BABA MUHAMMET BURAK ÖZÇELİK  LEMYE CEREN GÜMÜŞ  GÖKTUĞ ALİ AKIN  FERDİ SÖNMEZ  OSMAN ÇETİN  EMRE DURAK  MELİKE SERRA KALYON HASNAIN ALI | | DEPLOY12 JANUARY 2019 | | --- |  REPORT 8 The group member who did not attend the meeting is LEMYE CEREN GUMUS  Demo  └── src  | ├── main  | │ ├── java  | │ │ └── org  | │ │ └── demo  | │ │ └── Application.java  | │ └── resources  | │ └── application.properties  | └── test  | └── java  | └── org  | └── demo  | └── Application.java  ├──── Dockerfile  ├──── pom.xml  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">  <modelVersion>4.0.0</modelVersion>  <groupId>org.executablejar</groupId>  <artifactId>demo</artifactId>  <packaging>jar</packaging>  <version>0.0.1-SNAPSHOT</version>  <name>demo Maven Webapp</name>  <url>http://maven.apache.org</url>  <properties>  <java-version>1.8</java-version>  <docker.image.prefix>springDemo</docker.image.prefix>  </properties>  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.3.3.RELEASE</version>  </parent>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-tomcat</artifactId>  </dependency>  <dependency>  <groupId>org.apache.tomcat.embed</groupId>  <artifactId>tomcat-embed-jasper</artifactId>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  <finalName>demo</finalName>  </build>  Content of Dockerfile  FROM jav  FROM java:8  EXPOSE 8080  ADD /target/app.jar demo.jar  ENTRYPOINT ["java","-jar","demo.jar"]  Commands to build and run image  Go to the directory of project.Lets say D:/Demo  $ cd D/demo  $ mvn clean install  $ docker build demo .  $ docker run -p 8080:8080 -t demo  There is a difference between images and containers.   * Images will be build ONCE * You can start containers from Images   In your case:  Change your image:  FROM anapsix/alpine-java  MAINTAINER myNAME  COPY testprj-1.0-SNAPSHOT.jar /home/testprj-1.0-SNAPSHOT.jar  CMD ["java","-jar","/home/testprj-1.0-SNAPSHOT.jar"]  Build your image:  docker build -t imageName .  Now invoke your program inside a container:  docker run --name myProgram imageName  Now restart your program by restarting the container:  docker restart myProgram  Your program changed? Rebuild the image!:  docker rmi imageName  docker build -t imageName .  The main application class is shown below and as you can see I haven't added anything else to the project. I'm planning to use the default actuator health endpoint to test the application later.  package com.blog.samples.docker;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  @SpringBootApplication  public class Application {  public static void main(String[] args) {  SpringApplication.run(Application.class, args);  }  } Defining the Docker Image With the Spring Boot end of things covered, let's move on to the Docker image. The image is defined in the Dockerfile below and although it's compact, there's plenty going on. I'll explain each line in detail below.  FROM maven:3.5.2-jdk-8-alpine AS MAVEN\_BUILD  MAINTAINER Brian Hannaway  COPY pom.xml /build/  COPY src /build/src/  WORKDIR /build/  RUN mvn package  FROM openjdk:8-jre-alpine  WORKDIR /app  COPY --from=MAVEN\_BUILD /build/target/docker-boot-intro-0.1.0.jar /app/  ENTRYPOINT ["java", "-jar", "docker-boot-intro-0.1.0.jar"]  FROM maven:3.5.2-jdk-8-alpine AS MAVEN\_BUILD tells Docker to use the Maven image maven:3.5.2-jdk-8-alpine as the base image for the first stage of the build. Docker will look for this image locally and if it isn't available it will be pulled from DockerHub. Although Maven will be discarded from the final image (see COPY --from command later) I've used the Alpine-flavored Maven image as its faster to download.  MAINTAINER Brian Hannaway isn't essential but improves maintainability by providing a point of contact for the image author.  COPY pom.xml /build/ creates a build directory in the image and copies the pom.xml into it.  COPY src /build/src/ copies the src directory into the build directory in the image.  WORKDIR /build/ sets build as the working directory. Any further commands will run from the build directory.  RUN mvn package runs the mvn package command to compile and package the application as an executable JAR. The first time the image is built, Maven will pull all required dependencies from the public Maven repo and cache them locally in the image. Subsequent builds will use a cached version of this image layer which means that the dependencies will be referenced locally and won't have to be pulled down again.  At this point, I've defined the image up to the point where it builds an executable JAR. This is the first part of the multi-stage build finished. The next stage will take the JAR and run it.  FROM openjdk:8-jre-alpine tells Docker you want to use the openjdk:8-jre-alpine base image for the next stage of the multi-stage build. Again, I've used a lightweight Alpine image for the Java 8 JRE. Using an Alpine image here is more important than using it for the Maven image earlier. While the Maven image will be discarded, openjdk:8-jre-alpine will form part of the final image. So choosing a lightweight JRE image is important if you want to keep the final image as small as possible.  WORKDIR /app tells Docker to create a new working directory in the image called /app. All further commands will run from this directory.  COPY --from=MAVEN\_BUILD /build/target/docker-boot-intro-0.1.0.jar /app/ tells Docker to copy docker-boot-intro-0.1.0.jar from the /build/target directory in the MAVEN\_BUILD stage to the /app directory of the current stage. As mentioned earlier, multi-stage builds are great because they allow you to copy specific artefacts from one build stage to another and discard everything else. If you were to retain everything from the MAVEN\_BUILD stage you'd end up with an image that contains Maven, a local Maven repo. and all the class files generated in the target directory. By cherrypicking what you want from the MAVEN\_BUILD stage and discarding everything else, you end up with a much smaller image.  ENTRYPOINT ["java", "-jar", "app.jar"] tells Docker what command to run when a container is started from this image. The individual parts of the command are comma-separated. In this instance, you want to run the executable JAR you copied to the /app directory. Building the Image Now that you have a Docker image defined, [it's time to build it.](https://dzone.com/articles/how-to-build-your-own-docker-images) Open a directory containing the Dockerfile (project root). To build the image run the following command docker image build -t docker-boot-intro . The -t option allows you to specify a name and optionally a tag. If you don't specify a tag Docker will automatically tag the image as latest.  $ docker image build -t docker-boot-intro  RESOURCES :  <https://www.overleaf.com/4889241498nphcffswcqtf>  <https://stackoverflow.com/questions/34170216/run-springboot-based-docker-image-return-error-messageinvalid-or-corrupt-jarfil>  <https://stackoverflow.com/questions/35061746/run-jar-file-in-docker-image>  <https://dzone.com/articles/deploying-spring-boot-on-docker>  <https://medium.com/@wkrzywiec/how-to-put-your-java-application-into-docker-container-5e0a02acdd6b>  DEPLOY |
| --- | --- | --- |