QA Consulting

Docker Multistage Builds

TRAINING MATERIALS - MODULE HANDOUT

Contacts

robert.crutchley@qa.com
team.qac.all.trainers@qa.com

www.consulting.qa.com

Contents

Overview

Likely the reason your learning about Docker is to be able to deploy an application into it and get all the benefits from Docker. There are a some good practices adhere to when developing applications with Docker, multi stage builds is one of them. Multistage builds are aimed at keeping the image size for your application down. Another thing we want to be looking for when building and running applications is consistency between environments such as a developer environment and a build server environment, Multistage Builds will also aid us with this.

Multistage Build with Java & Maven

How Java and Maven work

We'll be using the Java programming language and the Maven dependency and build tool for usage examples. When we want to build a Java application it will need to be compiled and likey packed into a JAR file so that it can be executed by Java, Maven does this for us.

Our options

This means we have to decide how we are going to get a JAR file running in a Docker container, there are 3 options below and we'll be going for number 3.

1. Build the project on the host machine with Maven, use the Dockerfile to copy the built JAR file into the Docker Image, set an entrypoint that runs the JAR file.
   This requires Maven and Java to be installed on the host machine. Host machine environments can vary and effect the build.
2. In the Dockerfile copy everything into the image, build the project and then run the JAR file in an entrypoint.

Unless we delete it, all the dependencies for build the project will be in the .m2 folder for Maven and the source code will be sat in the image for no good reason as well.

3. Use a Multistage build. Build from a Maven image, copy the code and build it, Build from a Java image, copy the JAR file from the previous build stage, create an entrypoint to run the JAR file. This means that Docker will build the Image the same everywhere, whether its a developers laptop or a build server. Also the image which runs the application will be kept as small as possible.

Tasks

Overview

We are going to be implementing the 3 option from the section above. Create a new folder (**~/docker/multistage_builds_exercise**) for this exercise. We'll only need three files to get this Java project working (excluding the Dockerfile):

- pom.xml
  The config file for Maven.
- src/main/java/com/example/helloworld/HelloWorldApplication.java
  The Java code.
- src/main/resources/static/index.html
  The static webpage that will be served.

Creating the Java Application

Create the Java application which will run a Spring Boot server, don't forget to put it in the correct directory.

src/main/java/com/example/helloworld/HelloWorldApplication.java

```
package com.example.helloworld;

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

@SpringBootApplication
public class HelloWorldApplication {

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

Create the Static Webpage

Just a simple static web page for the application to serve, remember not to forget the folders on this one also.

src/main/resources/static/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Java Spring Boot Server</title>
</head>
<body>
    Hello from Docker
</body>
</html>
```

Create the Config File for Maven

For Maven to understand what to compile and how to package the application (a JAR file in our case) we need to create a pom.xml file at the root of the project.

pom.xml

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>hello-world</artifactId>
  <version>1.0.0</version>
  <packaging>jar</packaging>
  <name>hello-world</name>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.0.RELEASE</version>
    <relativePath/>
  </parent>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
    <java.version>1.8</java.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>
</project>
```

Create the Dockerfile

The Dockerfile is where we are going to be able to implement the Multistage Build, using a Maven image to compile the code and create a JAR file, then a Java image to run the code in.

Dockerfile

```
# build from the Maven image
# which has a maven environment configured already
FROM maven:latest
# copy our application in
COPY . /build
# change the working directory to where we are building
# the application
WORKDIR /build
# use maven to build the application
RUN mvn clean package
# create a new build stage from the Java image
# which has java installed already
FROM java:8
# change the working directory to where the application
# is going to be installed
WORKDIR /opt/hello-world
# copy the JAR file that was created in the previous
# build stage to the application folder in this build stage
COPY --from=0 /build/target/hello-world-1.0.0.jar app.jar
# create an entrypoint to run the application
ENTRYPOINT ["/usr/bin/java", "-jar", "app.jar"]
```

Clean Up

Make sure all the containers have been stopped and removed, delete all the images that have been created.