

# Java-Success.com

Prepare to fast-track, choose & go places with 800+ Java & Big Data Q&As with lots of code & diagrams.

[Home](#) [Why? ▾](#) [300+ Java FAQs ▾](#) [300+ Big Data FAQs ▾](#) [Courses ▾](#)

[👤 Membership ▾](#) [Your Career ▾](#)

[Home](#) › [bigdata-success.com](#) › [Tutorials - Big Data](#) › [TUT - Spark on Docker](#) › 01:

Docker tutorial with Java & Maven

## 01: Docker tutorial with Java & Maven

 Posted on [May 17, 2018](#)

**Pre-requisite:** Docker is installed on your machine for Mac OS X (E.g. \$ brew cask install docker) or Windows 10. [Docker interview Q&As](#).

**Step 1:** Create a Java project “**docker-test**” with “**HelloDocker.java**” file under “src/main/java” in a package named “com/mypkg”, and “**pom.xml**” in the base project folder. Using the **Visual Studio Code** editor.

### 300+ Java Interview FAQs

300+ Java FAQs



16+ Java Key Areas Q&As



150+ Java Architect FAQs



80+ Java Code Quality Q&As



150+ Java Coding Q&As



### 300+ Big Data Interview FAQs

300+ Big Data FAQs

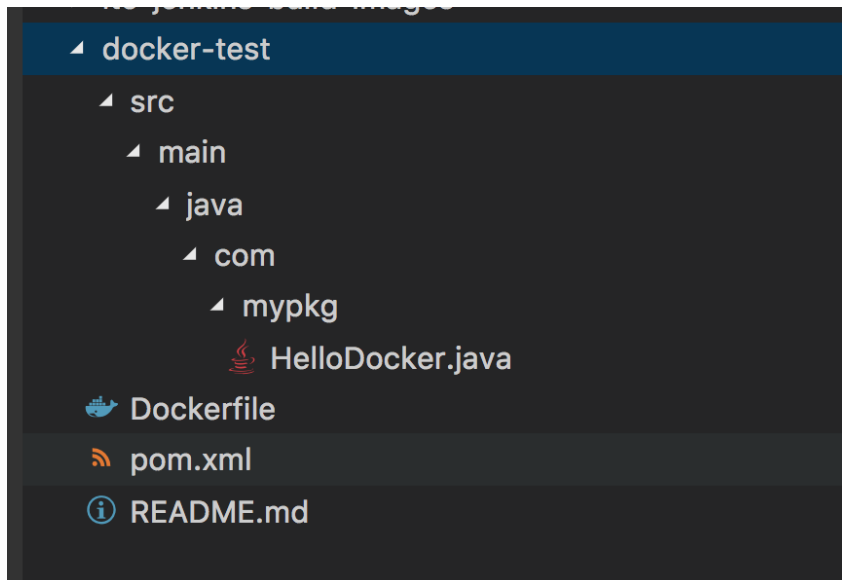


Tutorials - Big Data



TUT -  Starting Big Data

TUT - Starting Spark & Scala



Basic Mavenized Java folder structure with Dockerfile

## HelloDocker.java

```
1 package com.mypkg;
2
3 public class HelloDocker {
4     public static void main(String[] args) throws Exception {
5         System.out.println("Hello Docker");
6     }
7 }
8
```

## pom.xml

```
1
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4         http://maven.apache.org/xsd/maven-4.0.0.xsd">
5     <modelVersion>4.0.0</modelVersion>
6     <groupId>com.myproject</groupId>
7     <artifactId>hellodocker</artifactId>
8     <version>1.0</version>
9
10    <build>
11        <plugins>
12            <plugin>
13                <!-- Build an executable JAR -->
14                <groupId>org.apache.maven.plugins</groupId>
15                <artifactId>maven-jar-plugin</artifactId>
16                <version>3.1.0</version>
17                <configuration>
18                    <archive>
19                        <manifest>
```

[TUT - Starting with Python](#)[TUT - Kafka](#)[TUT - Pig](#)[TUT - Apache Storm](#)[TUT - Spark Scala on Zeppelin](#)[TUT - Cloudera](#)[TUT - Cloudera on Docker](#)[TUT - File Formats](#)[TUT - Spark on Docker](#)[TUT - Flume](#)[TUT - Hadoop \(HDFS\)](#)[TUT - HBase \(NoSQL\)](#)[TUT - Hive \(SQL\)](#)[TUT - Hadoop & Spark](#)[TUT - MapReduce](#)[TUT - Spark and Scala](#)[TUT - Spark & Java](#)[TUT - PySpark on Databricks](#)[TUT - Zookeeper](#)

## 800+ Java Interview Q&As

[300+ Core Java Q&As](#)[300+ Enterprise Java Q&As](#)[150+ Java Frameworks Q&As](#)[120+ Companion Tech Q&As](#)[Tutorials - Enterprise Java](#)

```
19         <addClasspath>true</addClasspath>
20         <classpathPrefix>lib/</classpathPrefix>
21         <mainClass>com.mypkg.HelloDocker</mainClass>
22     </manifest>
23 </archive>
24 </configuration>
25 </plugin>
26 </plugins>
27 </build>
28 </project>
```

**Step 2:** Create a “**Dockerfile**” in the project base directory.

## Dockerfile

```
1
2 FROM maven:3.5-jdk-8-alpine
3 MAINTAINER java-success.com
4
5 WORKDIR /app
6
7 CMD ["/bin/sh"]
```

**Step 3:** Docker has two steps. Firstly, “**build**” an image based on the “Dockerfile” and then “**run**” the container, which is an instance of the image.

## From the docker-test folder build the image

it uses the “Dockerfile” in the “.” (i.e. current directory)

```
1
2 $ docker build --tag=docker-test .
```

once the docker image is built, you can list it with

```
1
2 $ docker images
```

```
1
2 REPOSITORY          TAG
```

```
3 | docker-test latest
```

It has built an image of size 119MB. The image name will be “docker-test:latest”

You can remove an image with “docker rmi 25a4f7356b10”.

**From the docker-test create a container, which is an instance of the image**

```
1  
2 | $ docker run --rm -it -v "$(pwd):/app:consistent" c
```

Present working directory is mapped to Docker container’s “/app” and synchronised.

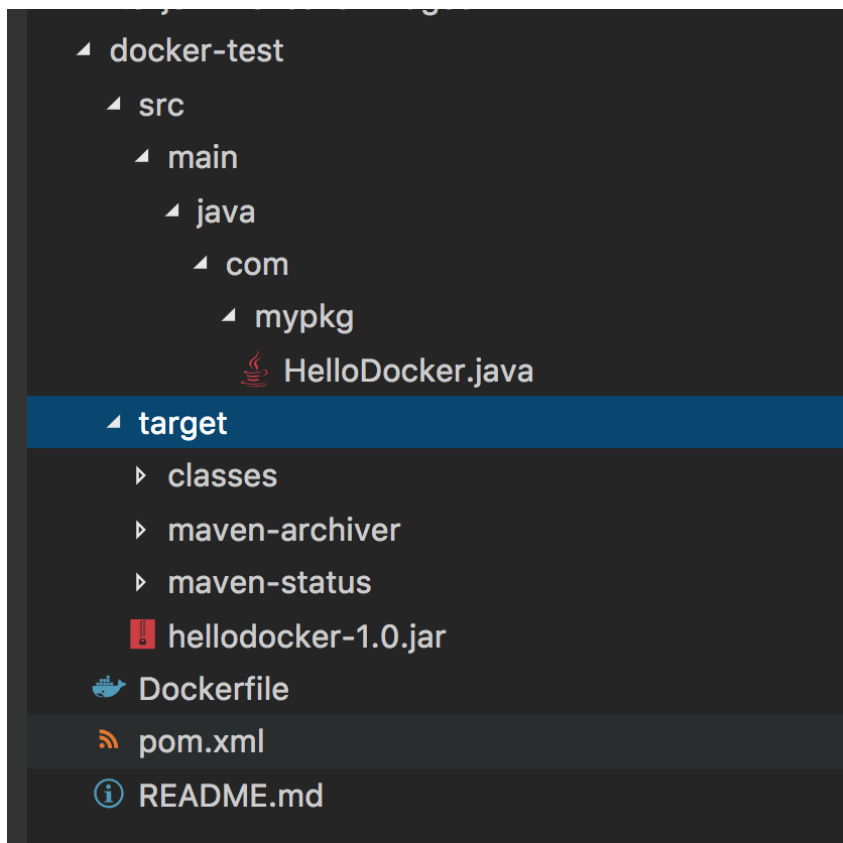
Now, you are in the Docker container prompt

```
1  
2 | /app #
```

**Step 4:** You can run “mvn package” to build the jar file “hellodocker-1.0.jar” in “target” folder.

```
1  
2 | /app # mvn -Dmaven.repo.local="/app/mvn-repository"
```

After packaging, there will be a target folder with “hellodocker-1.0.jar” file.



After packaging.

“-v “\$(pwd):/app:consistent” keeps the folders & files in the Docker container in the “/app” folder in sync with the current working directory “docker-test” shown above.

### Step 5: Run the “hellodocker-1.0.jar”

```
1  
2 /app # java -jar target/hellodocker-1.0.jar
```

### Output:

```
1  
2 Hello Docker
```

## You could also have directly run

### Create an image

```
1  
2 docker run --rm -it -v "$(pwd):/app:consistent" do
```

Create a container, which is an instance of the image

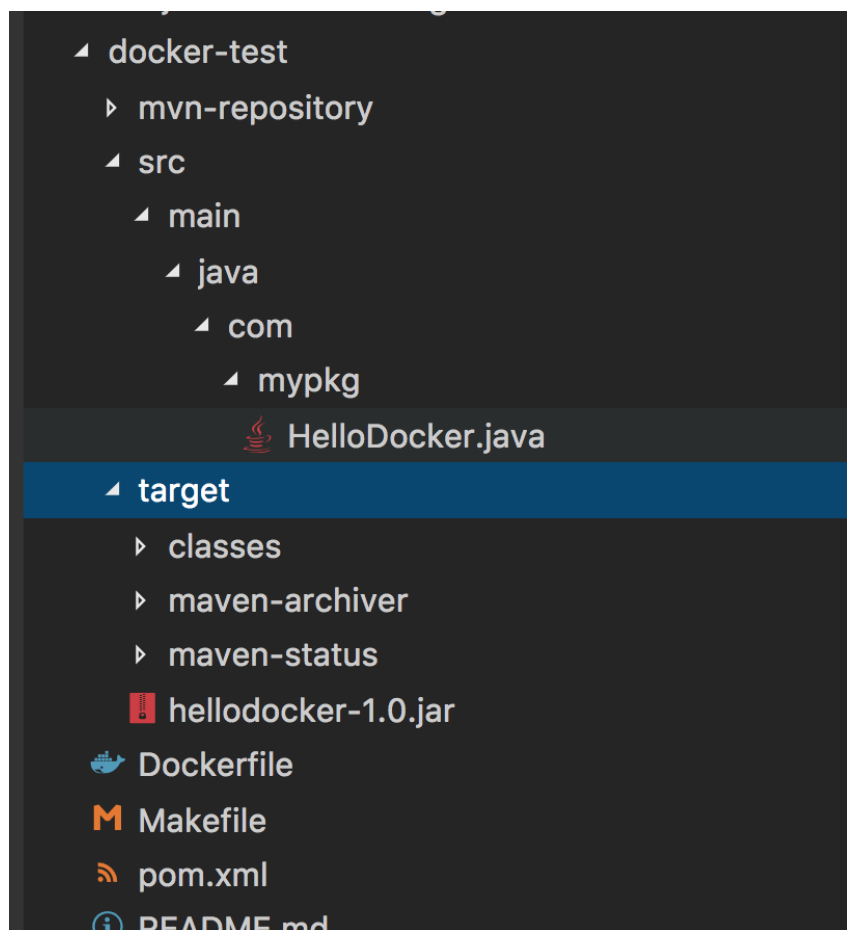
```
1  
2 docker run --rm -it -v "$(pwd):/app:consistent" do
```

**Notes:** Look at the “Docker hub” for the images that you can use.

## GNU make file to automate docker runs

Assumes that GNU Make is installed on your machine.

**Step 6:** Create a “Makefile” in the base project folder.



Docker with makefile

## Makefile

```
1
2 help:
3     @cat $(MAKEFILE_LIST) | grep -e "^[a-zA-Z_\-]"
4
5 docker-build:
6     docker build --tag=docker-test .
7
8 docker-run: docker-build
9     docker run --rm -it -v "$(pwd):/app:consistent"
10
11 app-run: docker-build
12     docker run --rm -it -v "$(shell pwd):/app:consistent"
13     mvn -Dmaven.repo.local="/app/mvn-repository"
14     java -jar target/hellodocker-1.0.jar
```

You can run the commands like:

```
1
2 $ make
3 $ make docker-run
4 $ make app-run
```

If you want to run the app, just execute

```
1
2 $ make app-run
```

◀ 17: Spark interview Q&As with coding examples in pyspark (i.e. python)

02: Apache Spark – local mode on Docker tutorial with Java & Maven ▶

## Disclaimer

The contents in this Java-Success are copyrighted and from EmpoweringTech pty Ltd. The EmpoweringTech pty Ltd has the right to correct or enhance the current content without any prior notice. These are general advice only, and one needs to take his/her own circumstances

into consideration. The EmpoweringTech Pty Ltd will not be held liable for any damages caused or alleged to be caused either directly or indirectly by these materials and resources. Any trademarked names or labels used in this blog remain the property of their respective trademark owners. Links to external sites do not imply endorsement of the linked-to sites. [Privacy Policy](#)