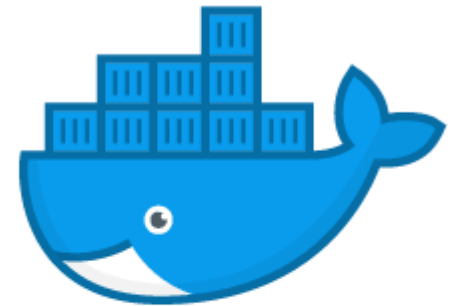


# Docker

Riccardo Cattaneo



docker

# Avvio Spring Application in Docker

1. Creare un file jar eseguibile del progetto
2. Creare un docker file
3. Build dell'immagine Docker
4. Avvio Docker Container

# Creare un file jar eseguibile del Progetto

Per prima cosa dobbiamo creare un file (jar) eseguibile del nostro progetto Spring. Per farlo ci serviamo di Maven. Che cos'è maven ? Vediamolo brevemente...



# Maven

Maven è un progetto open source, sviluppato dalla Apache, che permette di organizzare in modo molto efficiente un progetto java. I vantaggi principali di Maven sono i seguenti:

- Standardizzazione della struttura di un progetto;
- Compilazione;
- Test ed esportazione automatizzate;
- Gestione e download automatico delle librerie necessarie al progetto;

Il componente principale di Maven è il pom.xml (POM, Project Object Model) : file di configurazione che contiene tutte le informazioni su un progetto (dipendenze, test, documentazione, ecc ecc...);

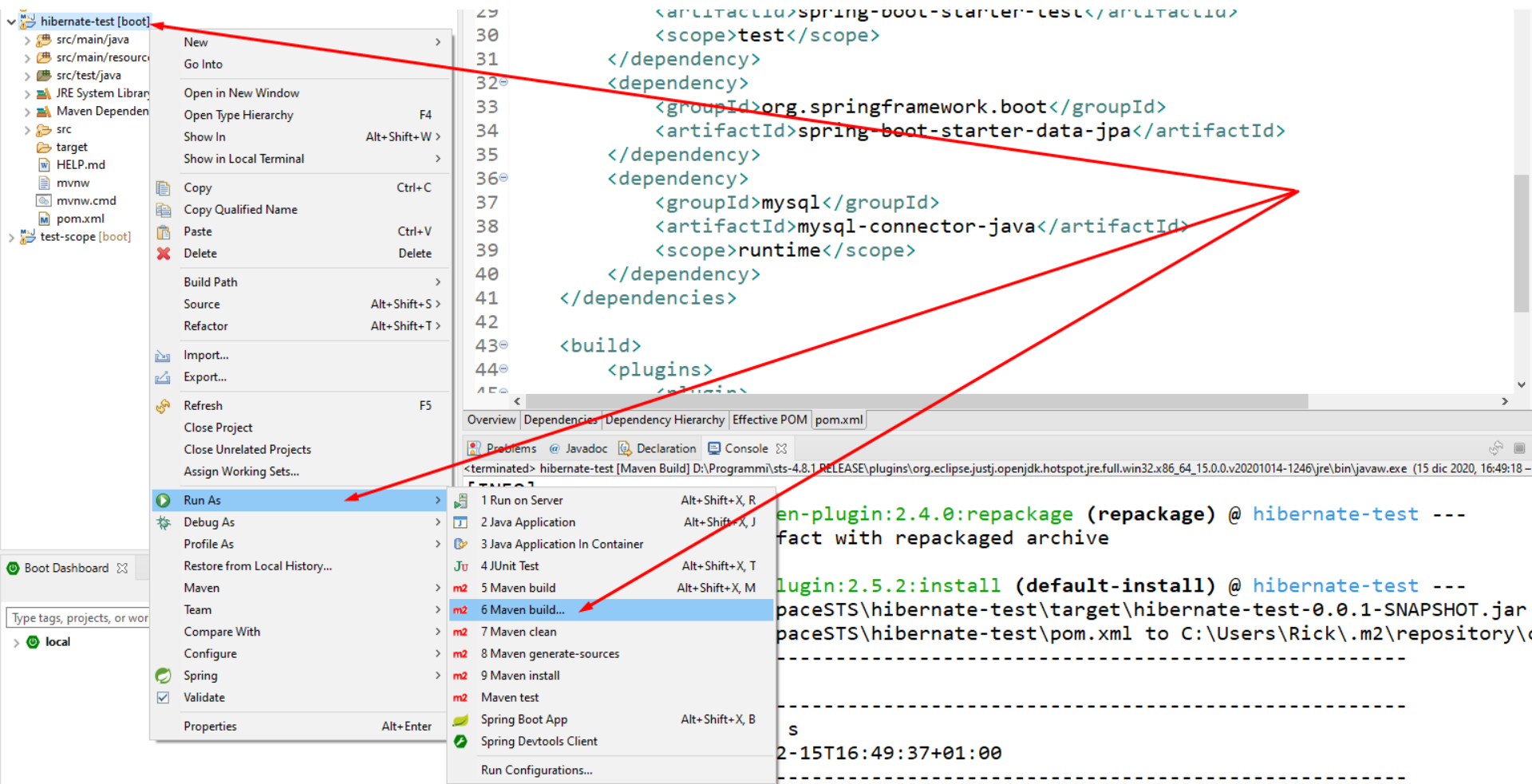
I principali plugin disponibili sono i seguenti:

- **clean**: che permette di cancellare i compilati dal progetto;
- **compiler**: che permette di compilare i file sorgenti;
- **deploy**: che permette di depositare il pacchetto generato nel repository remoto;
- **install**: che permette di depositare il pacchetto generato nel repository locale;
- **site**: che permette di generare la documentazione del progetto;

A questo punto apriamo il nostro progetto Spring e diamo un'occhiata al nostro pom.xml che sta nella root principale del progetto ed assicuriamoci che sia presente la build di spring boot :

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

A questo punto click con il tasto destro sul progetto → Run as → maven build...



hibernate-test/pom.xml

27</depen28<g29<a30<s31</depe32<depen33<g34<a35</depe36<depen37<g38<a39<s40</depe41</dependen4243<build>44<plugi

OverviewDependenciesDependency

ProblemsJavadocDeclarat

<terminated> hibernate-test [Maven Bu

[INFO] --- spring- ---  
[INFO] Replacing m  
[INFO] --- maven-i  
[INFO] Installing D:\workspaces\hibernate-test\target\hibernate-test-0.0.1-SNAPSHOT.jar

Edit Configuration

Edit configuration and launch.

hibernate-test (2)

MainJRERefreshSourceLaunch ExtensionsEnvironment

Base directory:  
\${project\_loc:hibernate-test}  
Workspace...File System...Variables...

Goals: clean compile install  
Profiles: pom.xml  
User settings: C:\Users\Rick\.m2\settings.xml  
Workspace...File System...Variables...

☐ Offline☐ Update Snapshots  
☐ Debug Output☐ Skip Tests☐ Non-recursive  
☐ Resolve Workspace artifacts  
1 Threads

Parameter Name	Value

Add...  
Edit...  
Remove

Maven Runtime: EMBEDDED (3.6.3/1.16.0.20200610-1735) Configure...

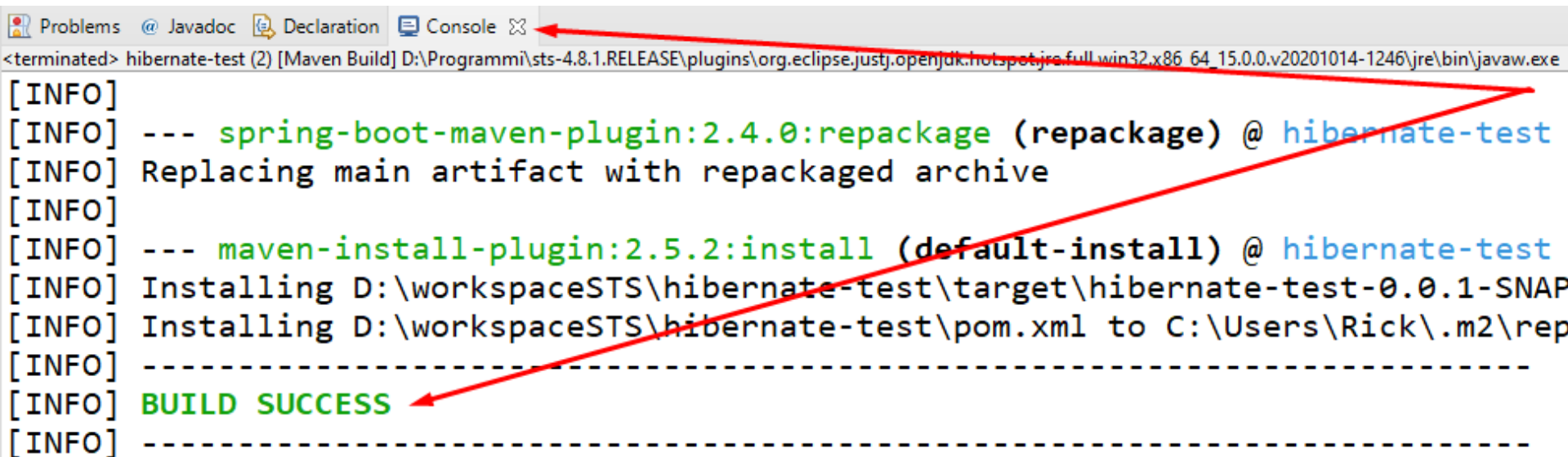
RevertApplyRunClose

ctId>

20201014-1246\jre\bin\javaw.exe (15 dic 2020, 16:49:18 -

@ hibernate-test ---  
@ hibernate-test ---  
@ hibernate-test ---





```
<terminated> hibernate-test (2) [Maven Build] D:\Programmi\sts-4.8.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.0.v20201014-1246\jre\bin\javaw.exe
[INFO]
[INFO] --- spring-boot-maven-plugin:2.4.0:repackage (repackage) @ hibernate-test
[INFO] Replacing main artifact with repackaged archive
[INFO]
[INFO] --- maven-install-plugin:2.5.2:install (default-install) @ hibernate-test
[INFO] Installing D:\workspaceSTS\hibernate-test\target\hibernate-test-0.0.1-SNAPSHOT.jar to C:\Users\Rick\.m2\repository\org\hibernate\hibernate-test\0.0.1-SNAPSHOT.jar
[INFO] Installing D:\workspaceSTS\hibernate-test\pom.xml to C:\Users\Rick\.m2\repository\org\hibernate\hibernate-test\0.0.1-SNAPSHOT.pom
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
```

Verificare sulla console che sia andato tutto bene (BUILD SUCCESS), dopo di che sempre tasto destro sul progetto e facciamo un refresh in modo che si aggiorni la visuale del progetto e vediamo che il contenuto della cartella «target» è cambiato :

The screenshot displays an IDE interface with two main panels. The left panel, titled 'Package Explorer', shows a project structure for 'hibernate-test [boot]'. The right panel shows the 'hibernate-test/pom.xml' file.

**Package Explorer Structure:**

- articoli-web-service [boot] [devtools] [articoli-web-service-start master]
  - demo [boot]
  - demo-mvc [boot]
  - hibernate-test [boot]
    - src/main/java
    - src/main/resources
    - src/test/java
    - JRE System Library [JavaSE-11]
    - Maven Dependencies
    - target/generated-sources/annotations
    - target/generated-test-sources/test-annotations
    - src
    - target
      - generated-sources
      - generated-test-sources
      - maven-archiver
      - maven-status
      - surefire-reports
      - hibernate-test-0.0.1-SNAPSHOT.jar
      - hibernate-test-0.0.1-SNAPSHOT.jar.original
    - HELP.md
    - mvnw
    - mvnw.cmd
    - pom.xml
  - test-scope [boot]

**hibernate-test/pom.xml Content:**

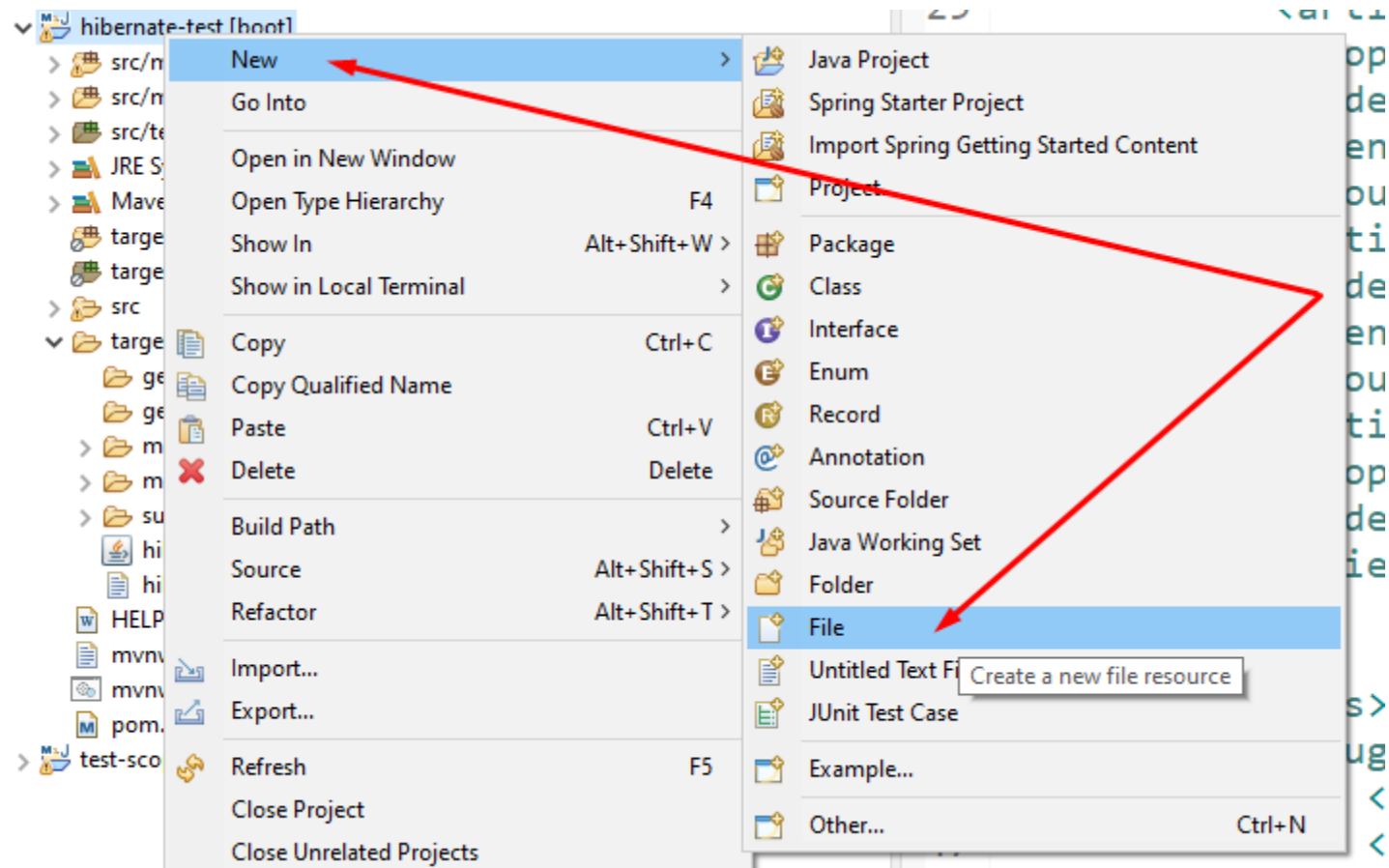
```
2 /  
28     <groupId>org.springframework.boot</groupId>  
29     <artifactId>spring-boot-starter-test</artifactId>  
30     <scope>test</scope>  
31 </dependency>  
32 <dependency>  
33     <groupId>org.springframework.boot</groupId>  
34     <artifactId>spring-boot-starter-data-jpa</artifactId>  
35 </dependency>  
36 <dependency>  
37     <groupId>mysql</groupId>  
38     <artifactId>mysql-connector-java</artifactId>  
39     <scope>runtime</scope>  
40 </dependency>  
41 </dependencies>  
42  
43 <build>  
44     <plugins>  
45         <plugin>  
46             <groupId>org.springframework.boot</groupId>  
47             <artifactId>spring-boot-maven-plugin</artifactId>  
48         </plugin>  
49     </plugins>  
50 </build>
```

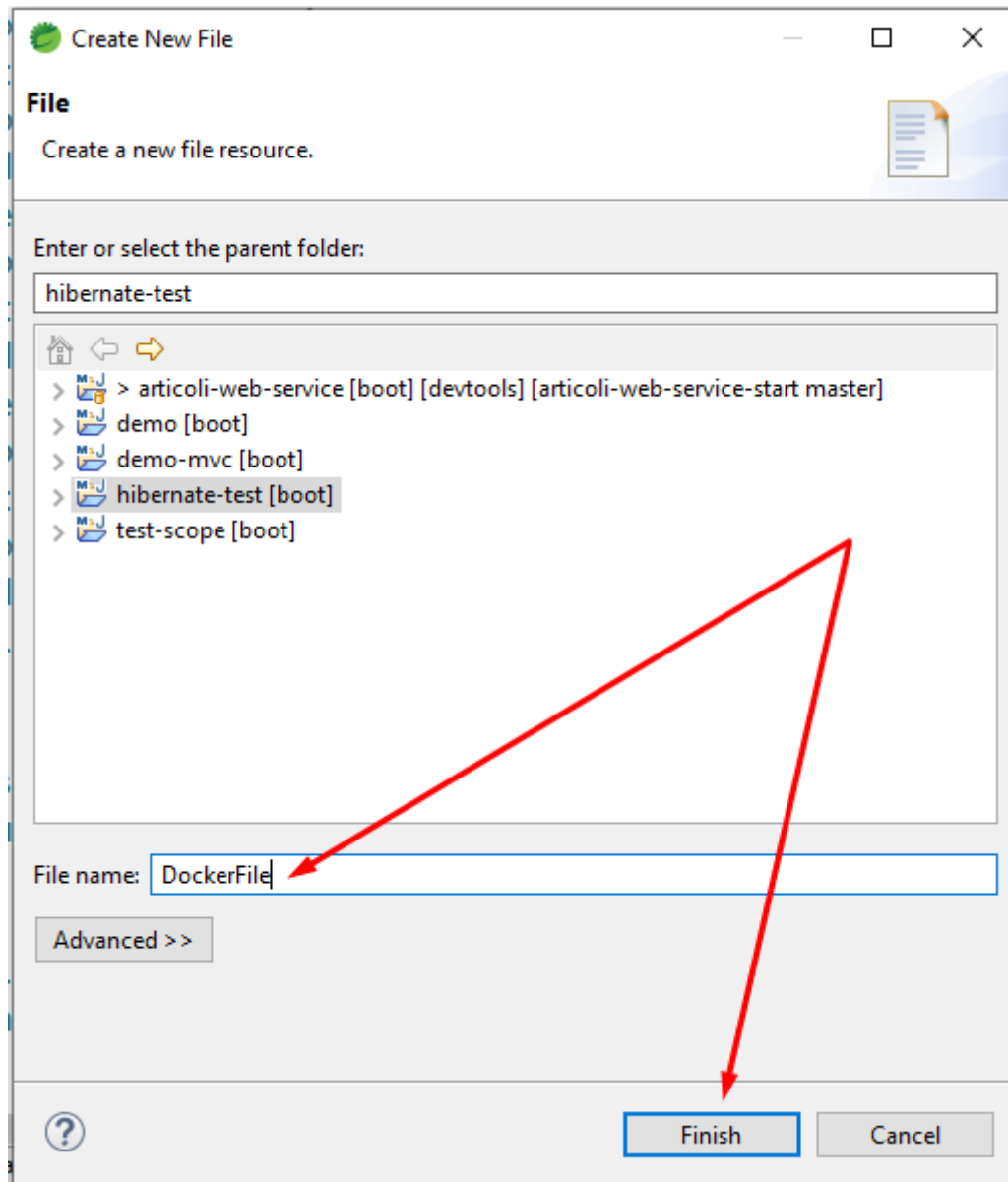
Red arrows indicate the mapping between the Package Explorer and the pom.xml file:

- From 'hibernate-test [boot]' to the root of the pom.xml.
- From 'target' to the 'dependencies' section.
- From 'target/hibernate-test-0.0.1-SNAPSHOT.jar' to the 'mysql-connector-java' dependency.

# Creare un dockerfile

Andiamo ora a creare il dockerfile direttamente nella root principale del progetto :





workspaceSTS - hibernate-test/DockerFile - Spring Tool Suite 4

File Edit Navigate Search Project Run Window Help



Package Explorer

- > demo
- > demo-mvc
- > hibernate-test [boot]
  - > src/main/java
  - > src/main/resources
  - > src/test/java
  - > JRE System Library [JavaSE-11]
  - > Maven Dependencies
    - target/generated-sources/annotations
    - target/generated-test-sources/test-annotations
  - > src
  - > target
    - DockerFile
    - HELP.md
    - mvnw
    - mvnw.cmd
    - pom.xml

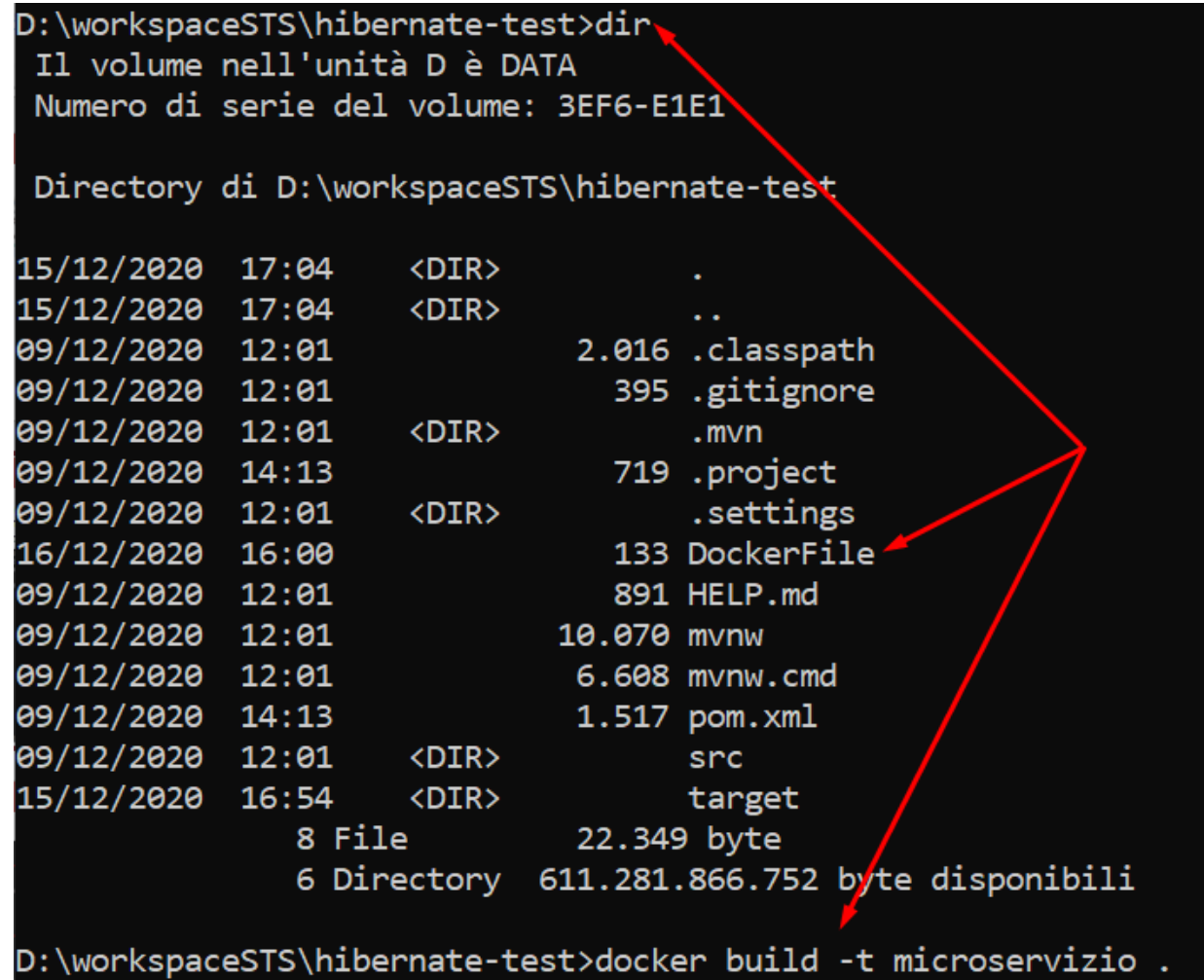
hibernate-test/pom.xml

DockerFile

```
1 FROM openjdk
2 VOLUME /tmp
3 ADD target/hibernate-test-0.0.1-SNAPSHOT.jar app.jar
4 ENTRYPOINT ["java", "-jar", "/app.jar"]
5 |
```

Docker

Da riga di comando posizioniamoci nella root principale dove è presente il nostro dockerfile e lanciamo il comando docker per fare la build che dovremmo conoscere :



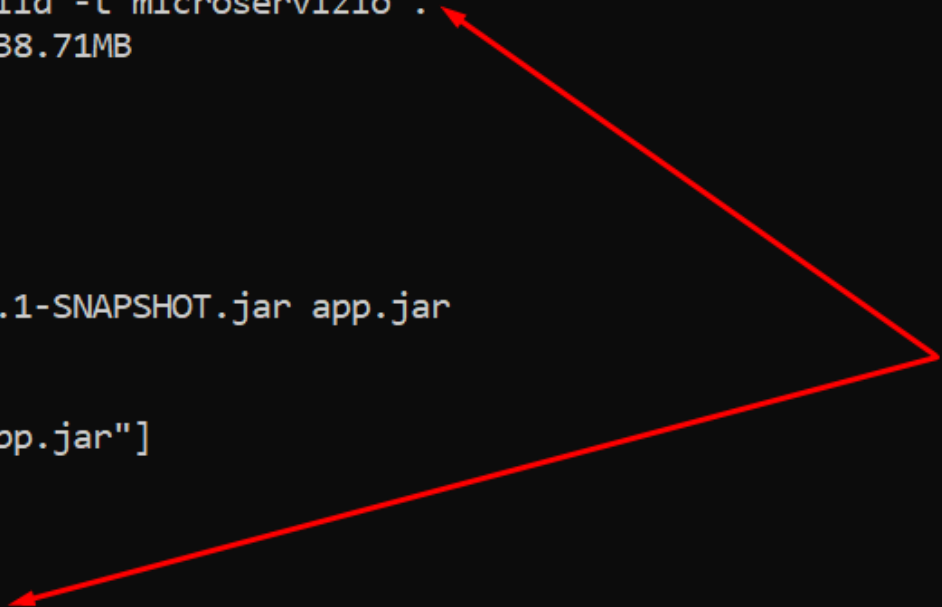
```
D:\workspaceSTS\hibernate-test>dir
Il volume nell'unità D è DATA
Numero di serie del volume: 3EF6-E1E1

Directory di D:\workspaceSTS\hibernate-test

15/12/2020  17:04    <DIR>          .
15/12/2020  17:04    <DIR>          ..
09/12/2020  12:01             2.016 .classpath
09/12/2020  12:01             395 .gitignore
09/12/2020  12:01    <DIR>          .mvn
09/12/2020  14:13             719 .project
09/12/2020  12:01    <DIR>          .settings
16/12/2020  16:00             133 DockerFile
09/12/2020  12:01             891 HELP.md
09/12/2020  12:01          10.070 mvnw
09/12/2020  12:01             6.608 mvnw.cmd
09/12/2020  14:13             1.517 pom.xml
09/12/2020  12:01    <DIR>          src
15/12/2020  16:54    <DIR>          target
                        8 File             22.349 byte
                        6 Directory  611.281.866.752 byte disponibili

D:\workspaceSTS\hibernate-test>docker build -t microservizio .
```

```
D:\workspaceSTS\hibernate-test>docker build -t microservizio .
Sending build context to Docker daemon 38.71MB
Step 1/4 : FROM openjdk:8-jdk-alpine
---> a3562aa0b991
Step 2/4 : VOLUME /tmp
---> Using cache
---> c6032943f177
Step 3/4 : ADD target/hibernate-test-0.0.1-SNAPSHOT.jar app.jar
---> Using cache
---> 4d6c95b52f00
Step 4/4 : ENTRYPOINT ["java","-jar","/app.jar"]
---> Using cache
---> 231b0402ef3b
Successfully built 231b0402ef3b
Successfully tagged microservizio:latest
SECURITY WARNING: You are building a Docker image from Windows against a non-Windows Docker host.
The image
will have '-rwxr-xr-x' permissions. It is recommended to double check and reset permissions for
D:\workspaceSTS\hibernate-test>
```



```
D:\workspaceSTS\hibernate-test>docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
microservizio	latest	231b0402ef3b	19 minutes ago	143MB
dpage/pgadmin4	latest	ae36b8785e03	4 months ago	220MB
postgres	latest	62473370e7ee	4 months ago	314MB
dpage/pgadmin4	<none>	c520f7001785	4 months ago	255MB
hello-world	latest	bf756fb1ae65	11 months ago	13.3kB
dpage/pgadmin4	4.15	048a766c7caa	13 months ago	266MB
openjdk	8-jdk-alpine	a3562aa0b991	19 months ago	105MB

```
D:\workspaceSTS\hibernate-test>
```



# Creare container della nostra App

Creare ora il container della nostra app (sappiamo come fare) e creare un container (sempre all'interno della nostra rete) che esegue il comando curl per chiamare il nostro microservizio e provare il tutto...

```
docker network create retecw
```

```
docker run --name rubricacontainer --network retecw -d  
microservizio
```

```
docker run -it --network retecw ubuntu /bin/bash  
curl "http://INDIRIZZO_IP:8080/elenco"
```

# Deploy su docker-hub

Per completare il tutto andiamo a trasferire l'immagine del nostro microservizio sul server pubblico di docker-hub.

Per farlo è sufficiente eseguire il login dalla root principale della nostra applicazione (la stessa dove è presente il dockerfile) con il comando `docker login -u NOME_UTENTE` e poi eseguire il push con il comando `docker push NOME_UTENTE/NOME_REPOSITORY`

Verifichiamo che è andato tutto bene andando tramite il browser nella pagina del nostro repository.

# Esercizio Finale : Rubrica Telefonica

- Creare un microservizio in java con SpringBoot e SpringWeb che espone un metodo di ricerca per nome di una rubrica telefonica e restituisca il numero corrispondente.
- Buildare e deployare l'applicazione su due container docker configurati in un'apposita rete.