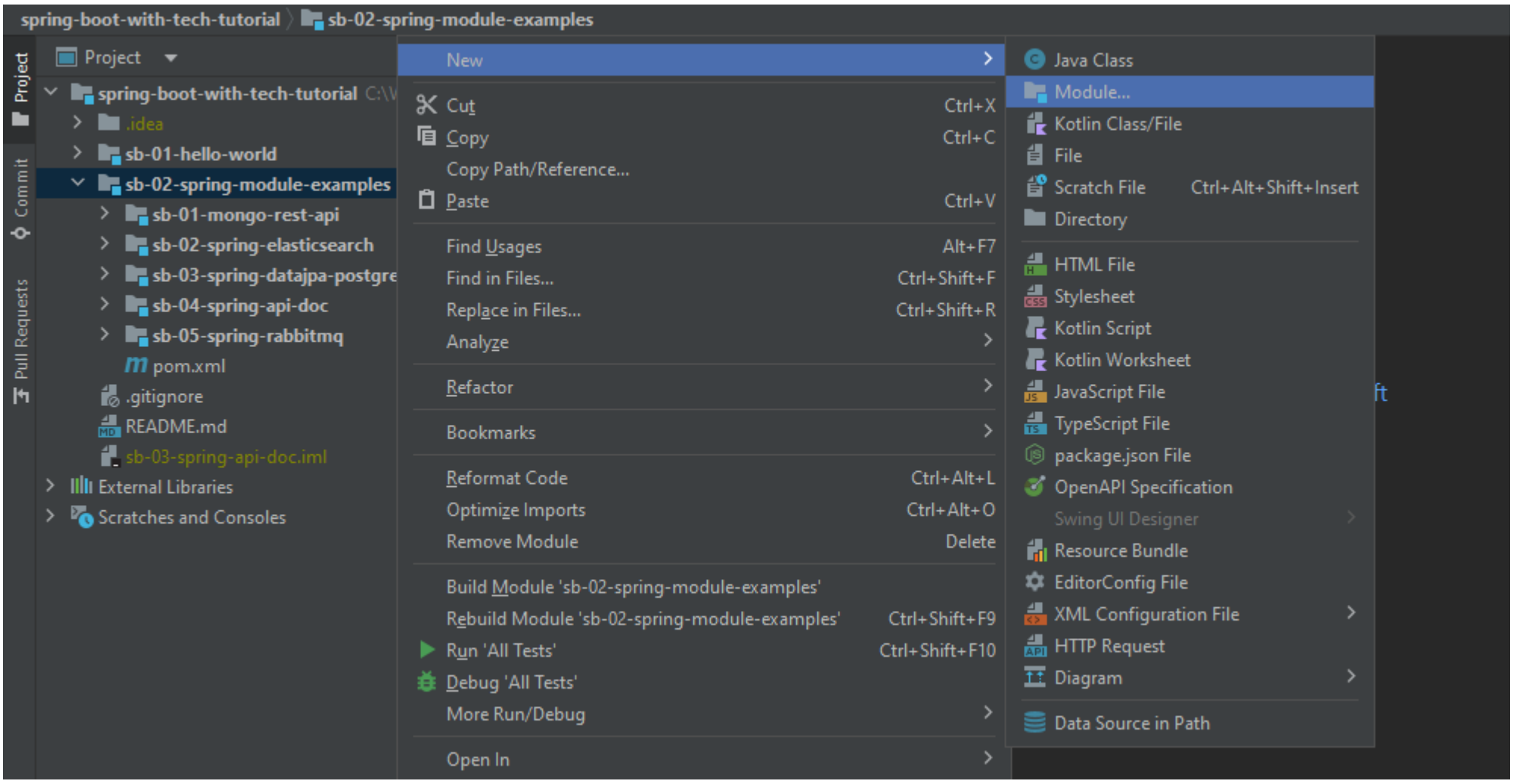
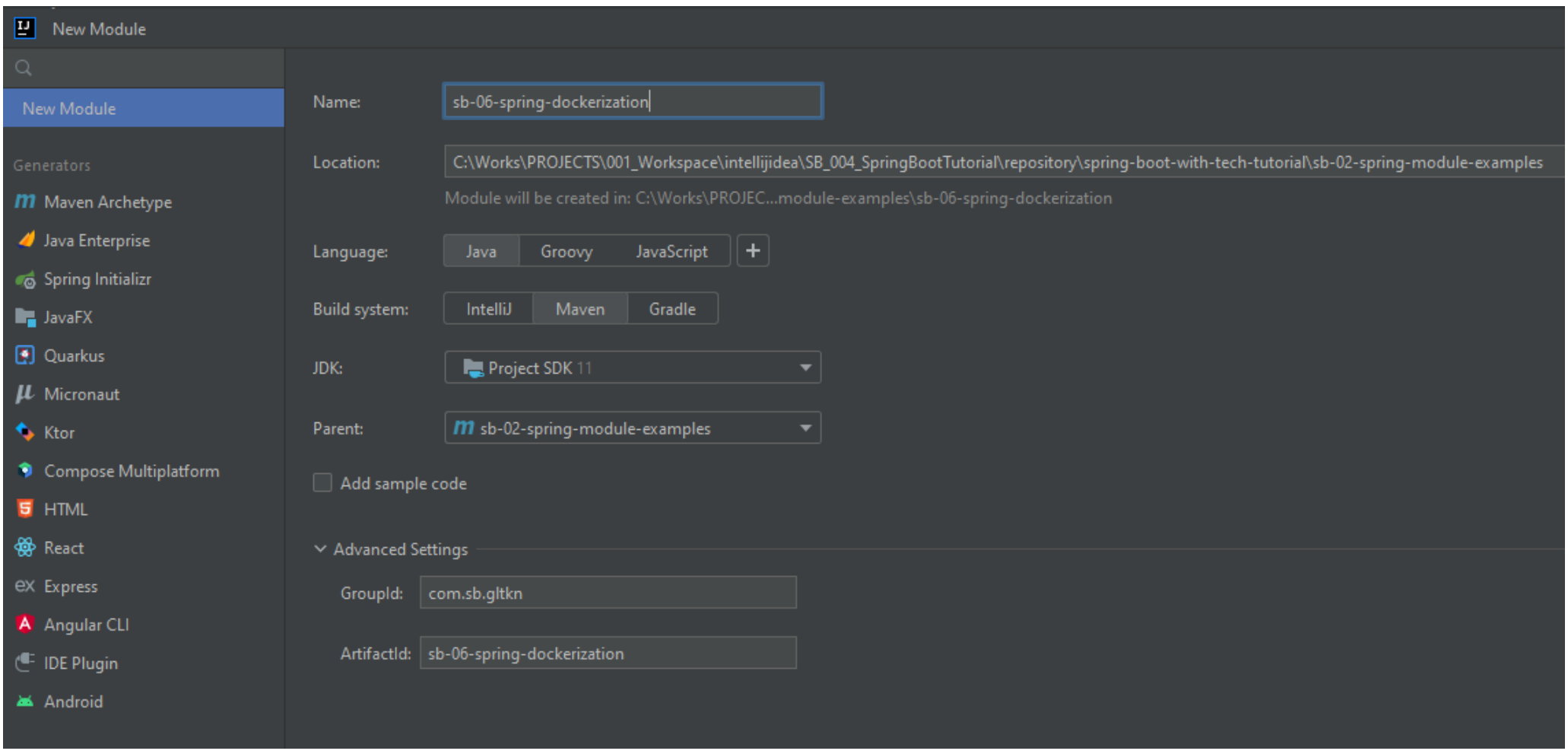
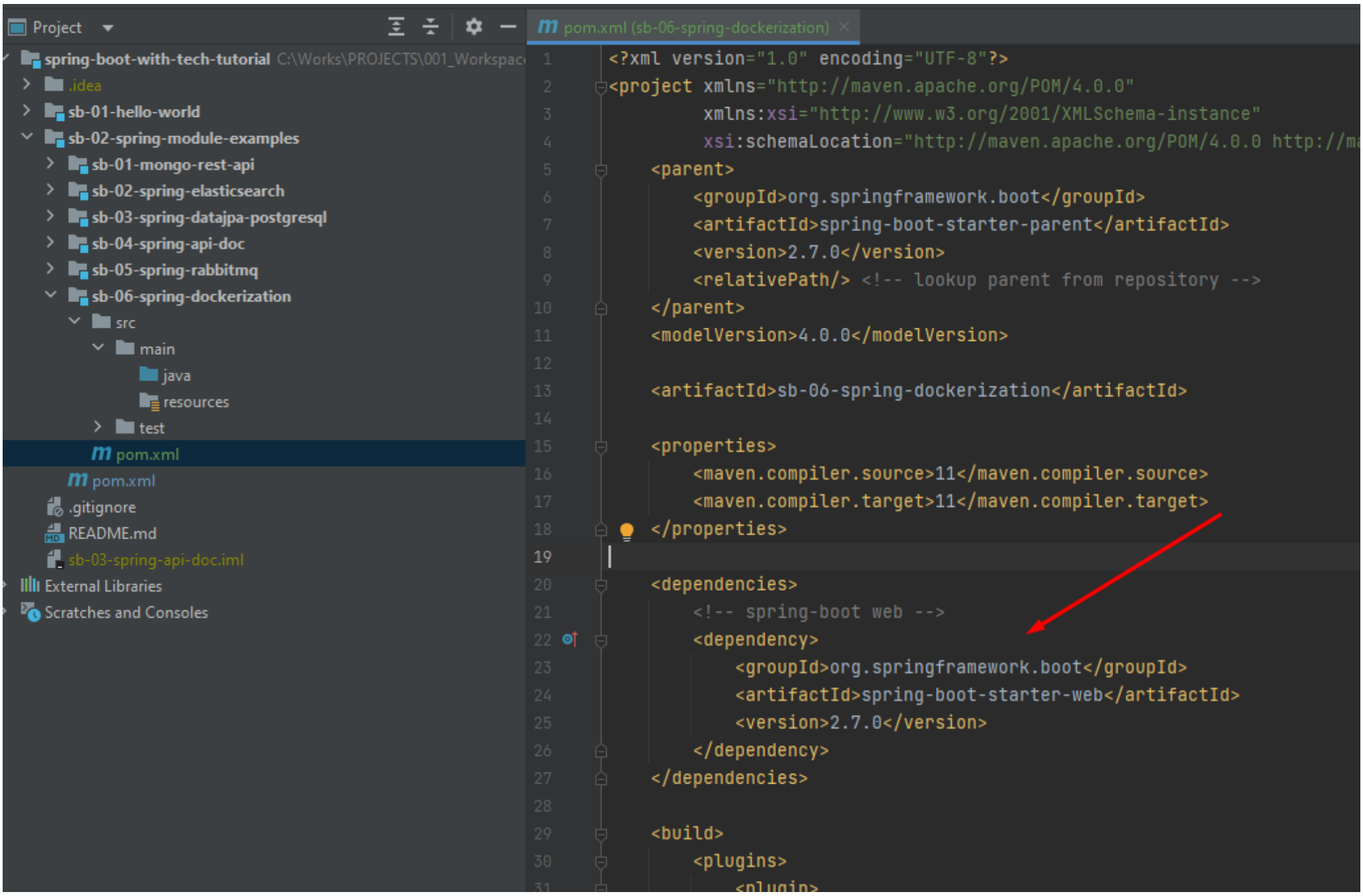
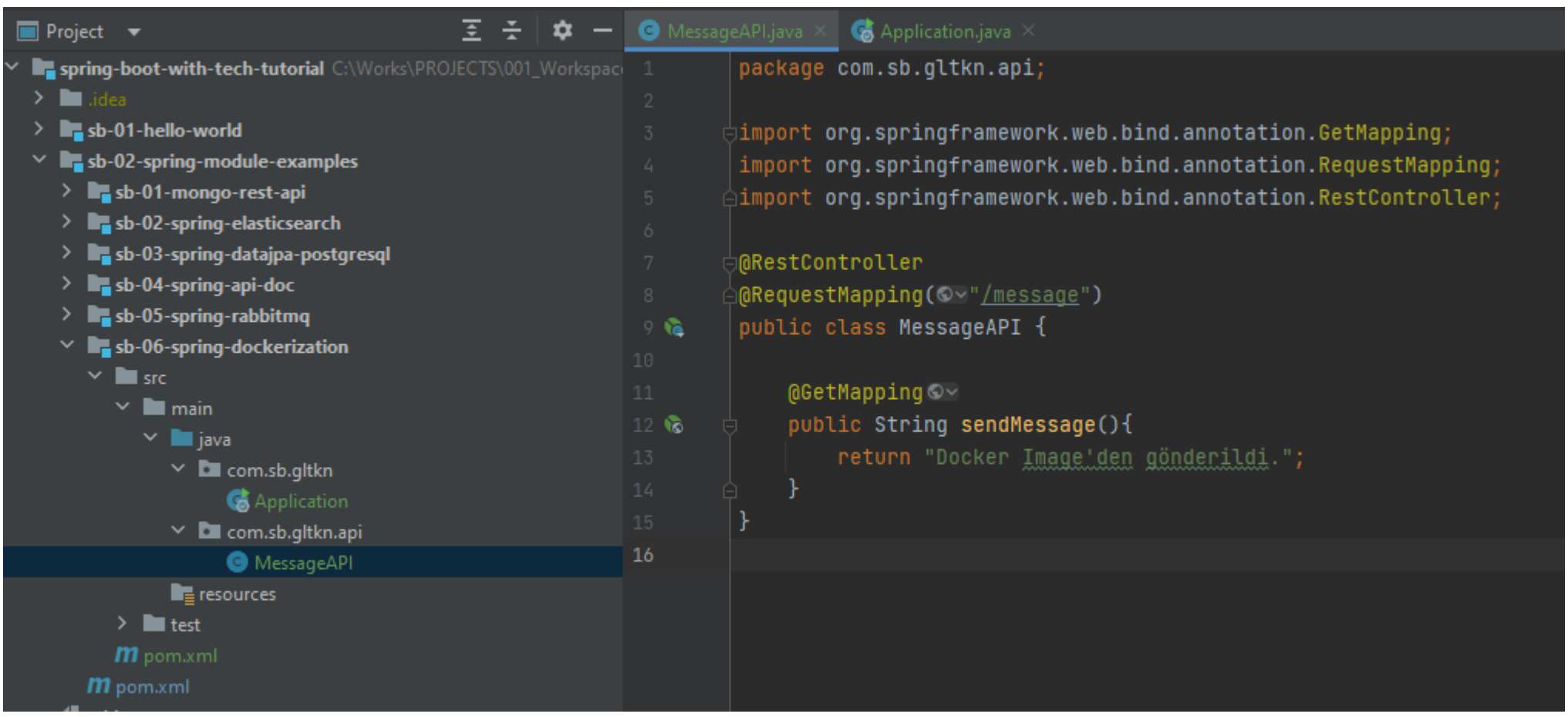
Bu bölümde SpringBoot uygulamamızı nasıl docker image haline getirebileceğimiz göreceğiz.






Project
spring-boot-with-tech-tutorial C:\Works\PROJECTS\001_Workspac
.idea
sb-01-hello-world
sb-02-spring-module-examples
sb-01-mongo-rest-api
sb-02-spring-elasticsearch
sb-03-spring-datajpa-postgresql
sb-04-spring-api-doc
sb-05-spring-rabbitmq
sb-06-spring-dockerization
src
main
java
resources
test
pom.xml
pom.xml
.gitignore
README.md
sb-03-spring-api-doc.iml
External Libraries
Scratches and Consoles
pom.xml (sb-06-spring-dockerization)
<?xml version="1.0" encoding="UTF-8"?>
<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://ma
<parent>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version>2.7.0</version>
<relativePath/> <!-- lookup parent from repository -->
</parent>
<modelVersion>4.0.0</modelVersion>
<artifactId>sb-06-spring-dockerization</artifactId>
<properties>
<maven.compiler.source>11</maven.compiler.source>
<maven.compiler.target>11</maven.compiler.target>
</properties>
<dependencies>
<!-- spring-boot web -->
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
<version>2.7.0</version>
</dependency>
</dependencies>
<build>
<plugins>
<plugin>


Project
spring-boot-with-tech-tutorial C:\Works\PROJECTS\001_Workspac
.idea
sb-01-hello-world
sb-02-spring-module-examples
sb-01-mongo-rest-api
sb-02-spring-elasticsearch
sb-03-spring-datajpa-postgresql
sb-04-spring-api-doc
sb-05-spring-rabbitmq
sb-06-spring-dockerization
src
main
java
com.sb.gltkn
Application
com.sb.gltkn.api
MessageAPI
resources
test
pom.xml
pom.xml
MessageAPI.java
Application.java
package com.sb.gltkn.api;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
@RestController
@RequestMapping("/message")
public class MessageAPI {
@GetMapping
public String sendMessage(){
return "Docker Image'den gönderildi.";
}
}

Docker image'ını **docker file** veya **maven plugin** yöntemiyle oluşturabiliriz.

Docker file yöntemi:

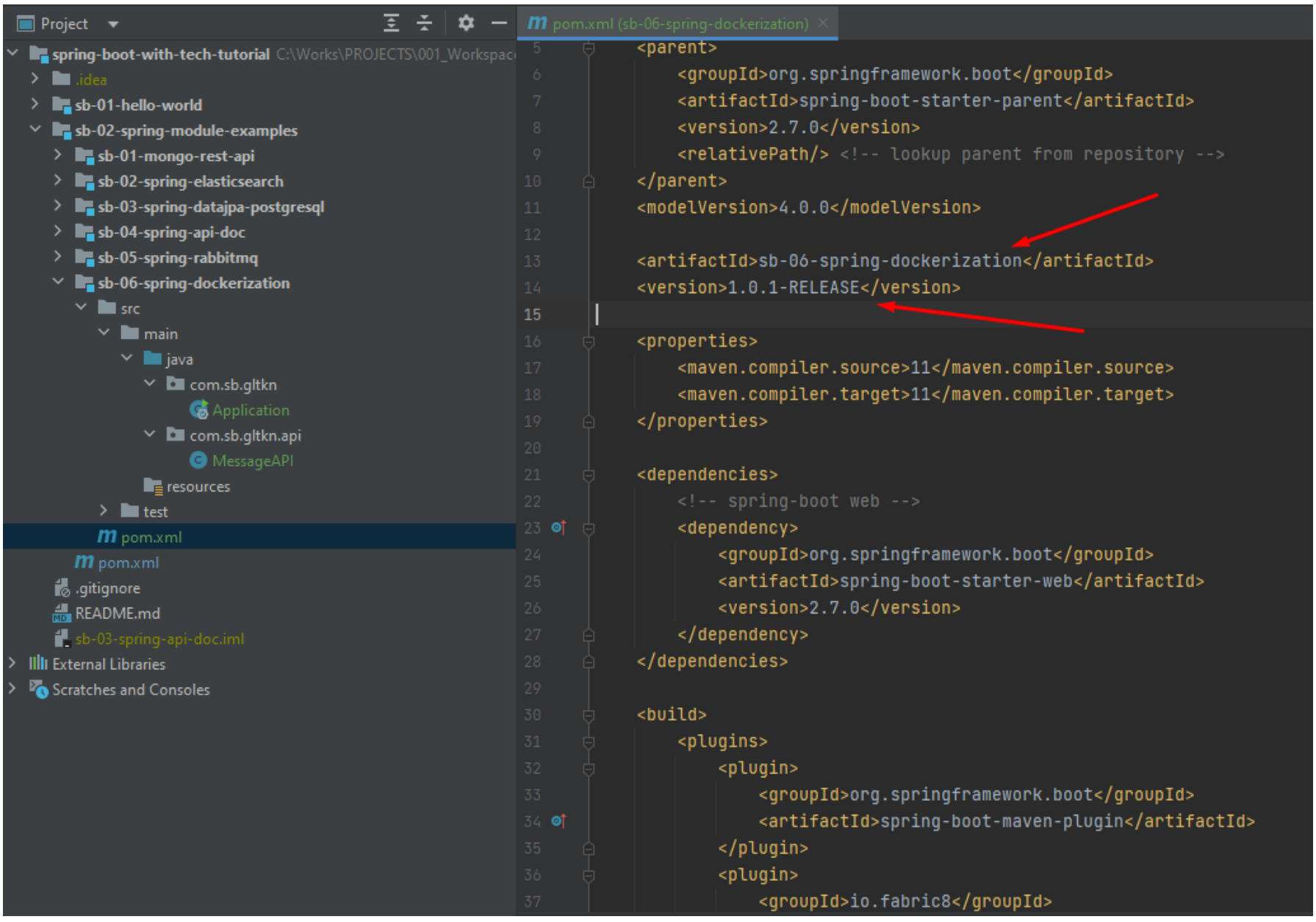
dockerfile
FROM
The base image for building a new image. This command must be on top of the dockerfile.
MAINTAINER
Optional, it contains the name of the maintainer of the image.
RUN
Used to execute a command during the build process of the docker image.
ADD
Copy a file from the host machine to the new docker image. There is an option to use a URL for the file, docker will then download that file to the destination directory.
ENV
Define an environment variable.
CMD
Used for executing commands when we build a new container from the docker image.
ENTRYPOINT
Define the default command that will be executed when the container is running.
WORKDIR
This is directive for CMD command to be executed.
USER
Set the user or UID for the container created with the image.
VOLUME
Enable access/linked directory between the container and the host machine.
Now let's start to create our first dockerfile.

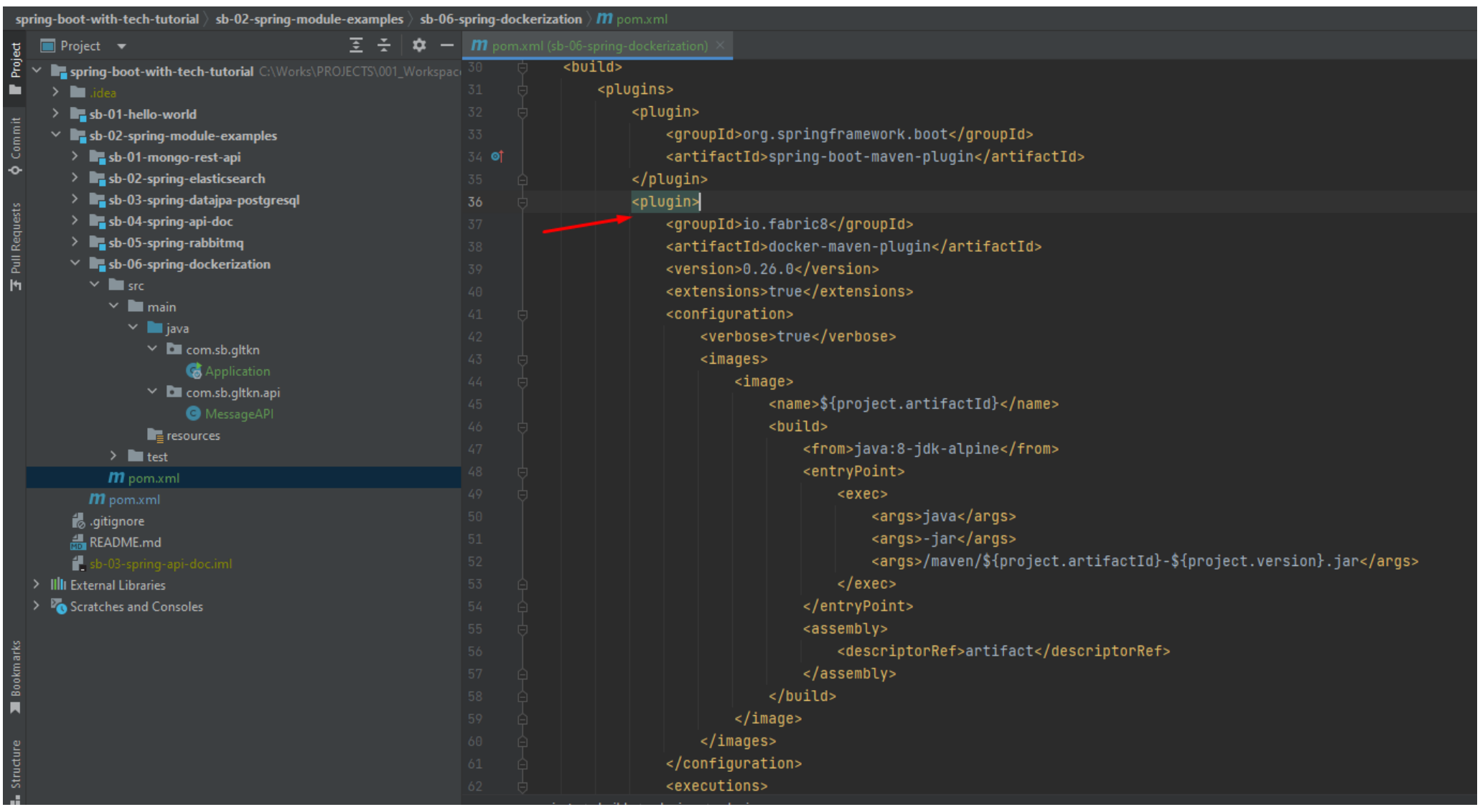
```
FROM java:8-jdk-alpine
COPY ./target/app.jar /usr/app/
WORKDIR /usr/app
RUN sh -c 'touch app.jar'
ENTRYPOINT ["java","-jar",”app.jar"]
```

Biz aşağıdaki maven plugin ile oluşturacağız.

```
<plugin>
  <groupId>io.fabric8</groupId>
  <artifactId>docker-maven-plugin</artifactId>
  <version>0.26.0</version>
  <extensions>true</extensions>
  <configuration>
    <verbose>true</verbose>
    <images>
      <image>
        <name>${project.artifactId}</name>
        <build>
          <from>java:8-jdk-alpine</from>
          <entryPoint>
            <exec>
              <args>java</args>
              <args>-jar</args>
              <args>/maven/${project.artifactId}-${project.version}.jar</args>
            </exec>
          </entryPoint>
          <assembly>
            <descriptorRef>artifact</descriptorRef>
          </assembly>
        </build>
      </image>
    </images>
  </configuration>
  <executions>
    <execution>
      <id>build</id>
      <phase>post-integration-test</phase>
      <goals>
        <goal>build</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

Projemizin pom.xml'inde plugins tag'i arasına eklemeliyiz.





45. Satırdaki ${project.artifactId} alanına pom.xml'deki <artifactId>sb-06-spring-dockerization</artifactId> bilgisi gelecektir. Yani bu docker image ismi olacaktır.

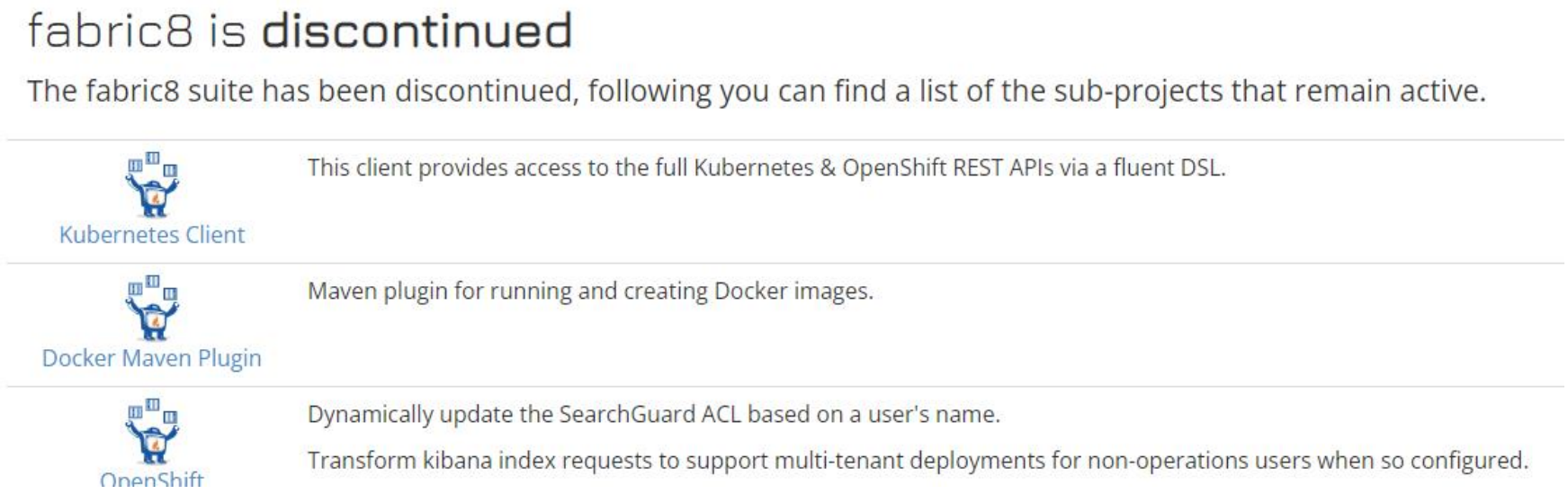
Projemiz build ediltikten sonra aşağıdaki yapıda bir jar dosyası oluşacaktır.
sb-06-spring-dockerization-1.0.1-RELEASE.jar

Bu sebeple 52 satırda bu jar dosyasını gösteren tanım bulunmaktadır.

<args>/maven/${project.artifactId}-${project.version}.jar</args>

entryPoint içindeki bu komutlar ile docker container çalıştığında default olarak devreye girecektir.

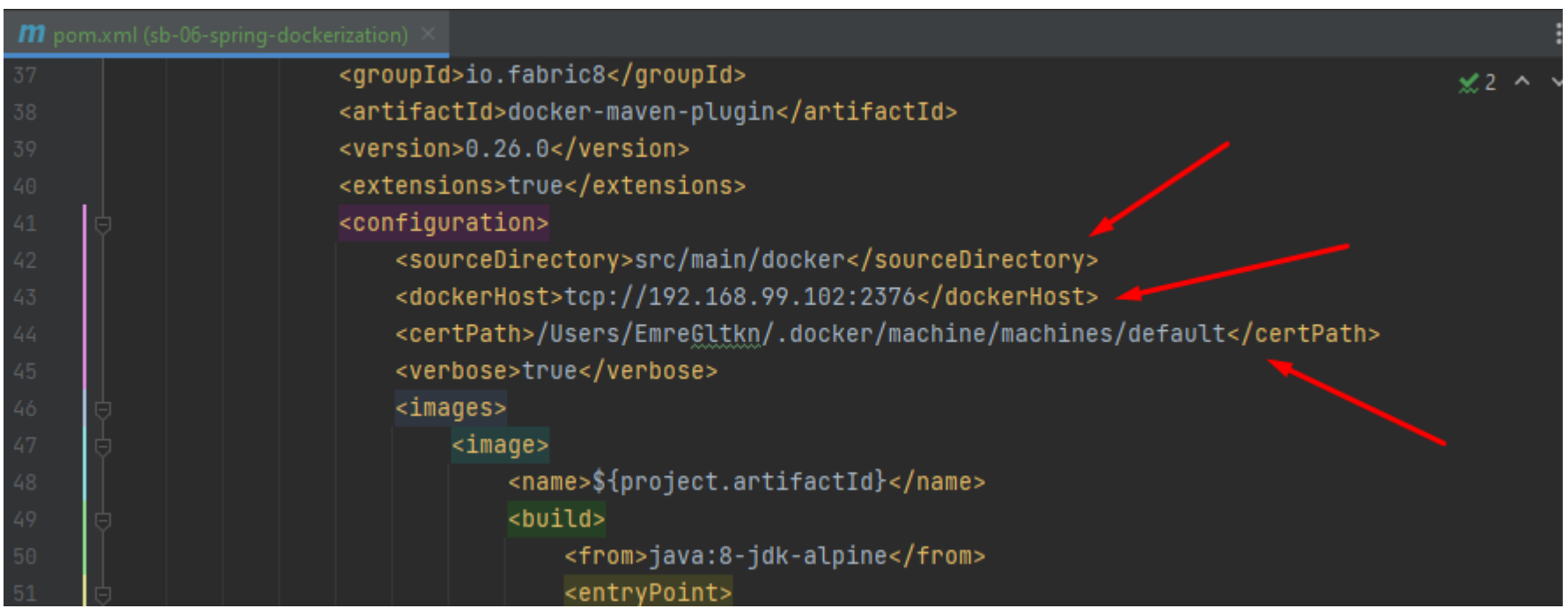
Bu **io.fabric8** plugini için şu siteden yararlanılmıştır.




```
EmreGltkn@DESKTOP-DN9PH1A MINGW64 ~
$ docker-machine ls
NAME      ACTIVE   DRIVER       STATE     URL   SWARM   DOCKER    ERRORS
default   -        virtualbox   Stopped                 Unknown
```

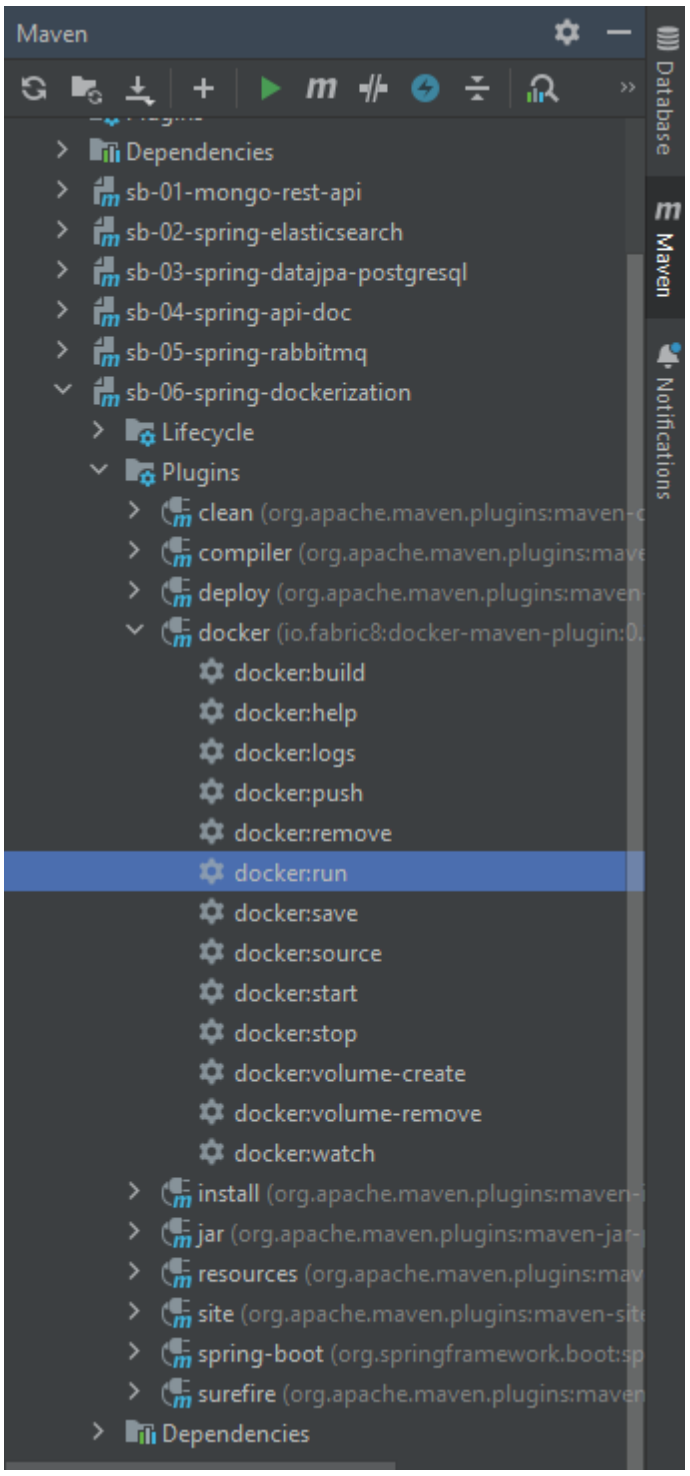
```
EmreGltkn@DESKTOP-DN9PH1A MINGW64 ~
$ docker-machine start default
Starting "default"...
(default) Check network to re-create if needed...
(default) Windows might ask for the permission to configure a dhcp server. Sometimes, such co
(default) Waiting for an IP...
```

Ben windows 10 Home'da çalıştığım için aşağıdaki 3 tanımı da maven plugin'e ekliyorum.

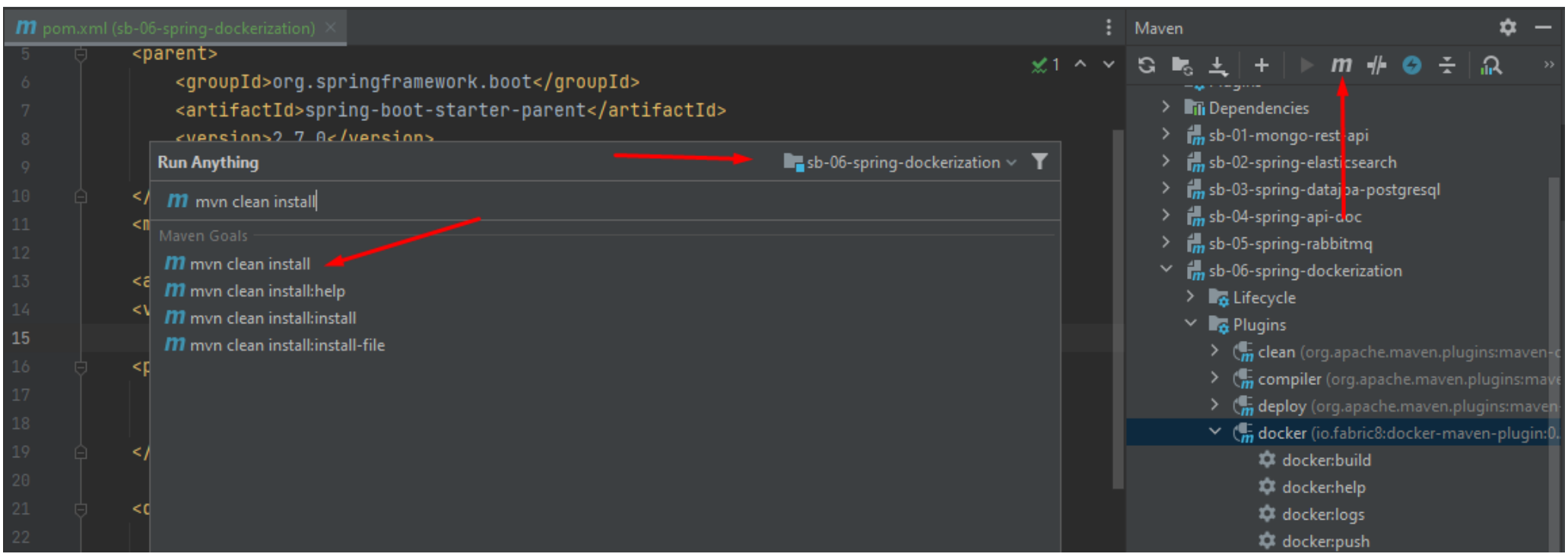


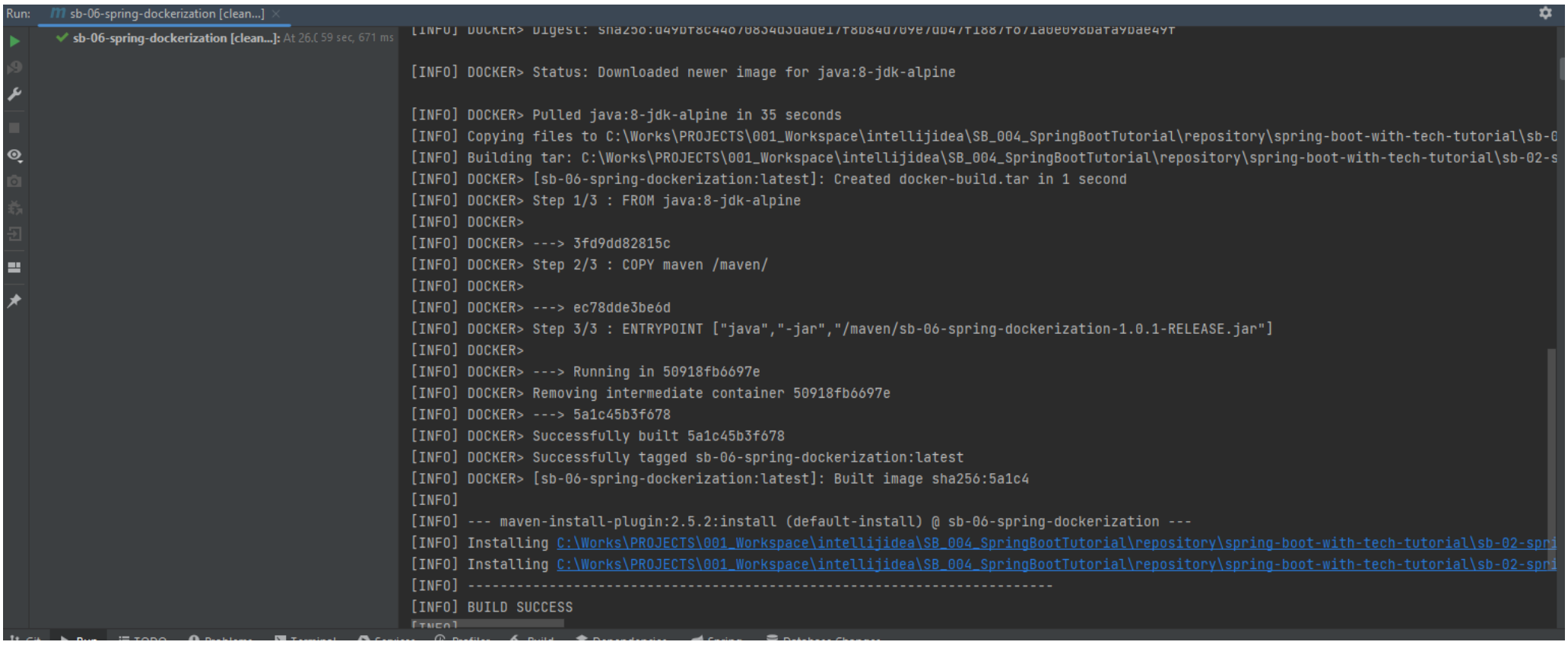

```
<sourceDirectory>src/main/docker</sourceDirectory>
<dockerHost>tcp://192.168.99.102:2376</dockerHost>
<certPath>/Users/EmreGltkn/.docker/machine/machines/default</certPath>
```

Docker image'ı aşağıdaki gibi run diyerek oluşturabiliriz.



Ancak biz aşağıdaki gibi clean yaparak çalıştıralım.





Buradaki image'larda 8-jdk-alpine 145 mb'mış. Bununla birlikte bizim image'ımız 163 mb olmuştur. Aslında bizim jar dosyamız 163-145 = 18 mb'mış.

```
EmreGltkn@DESKTOP-DN9PH1A MINGW64 ~
$ docker-machine env default
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.99.102:2376"
export DOCKER_CERT_PATH="C:\Users\EmreGltkn\.docker\machine\machines\default"
export DOCKER_MACHINE_NAME="default"
export COMPOSE_CONVERT_WINDOWS_PATHS="true"
# Run this command to configure your shell:
# eval $("C:\Users\EmreGltkn\bin\docker-machine.exe" env default)

EmreGltkn@DESKTOP-DN9PH1A MINGW64 ~
$ eval "$(docker-machine env default)"

EmreGltkn@DESKTOP-DN9PH1A MINGW64 ~
$ docker images
REPOSITORY                  TAG             IMAGE ID       CREATED          SIZE
sb-06-spring-dockerization  latest          5a1c45b3f678   44 minutes ago   163MB
rabbitmq                    3-management    77d763f31ce4   4 days ago       257MB
adminer                     latest          6261a306e265   2 weeks ago      90.6MB
elasticsearch               8.2.3           59306705ed62   2 weeks ago      1.2GB
postgres                    latest          5b21e2e86aab   4 weeks ago      376MB
mongo                       4.4.6           61ea24dc52c6   11 months ago    423MB
elasticsearch               7.4.2           b1179d41a7b4   2 years ago      855MB
java                        8-jdk-alpine    3fd9dd82815c   5 years ago      145MB

EmreGltkn@DESKTOP-DN9PH1A MINGW64 ~
$
```

Şimdi bu oluşturduğumuz docker image'ımızı çalıştıralım.

```
EmreGltkn@DESKTOP-DN9PH1A MINGW64 ~
$ docker run -p 8080:8080 sb-06-spring-dockerization
```

http://localhost:8080/message