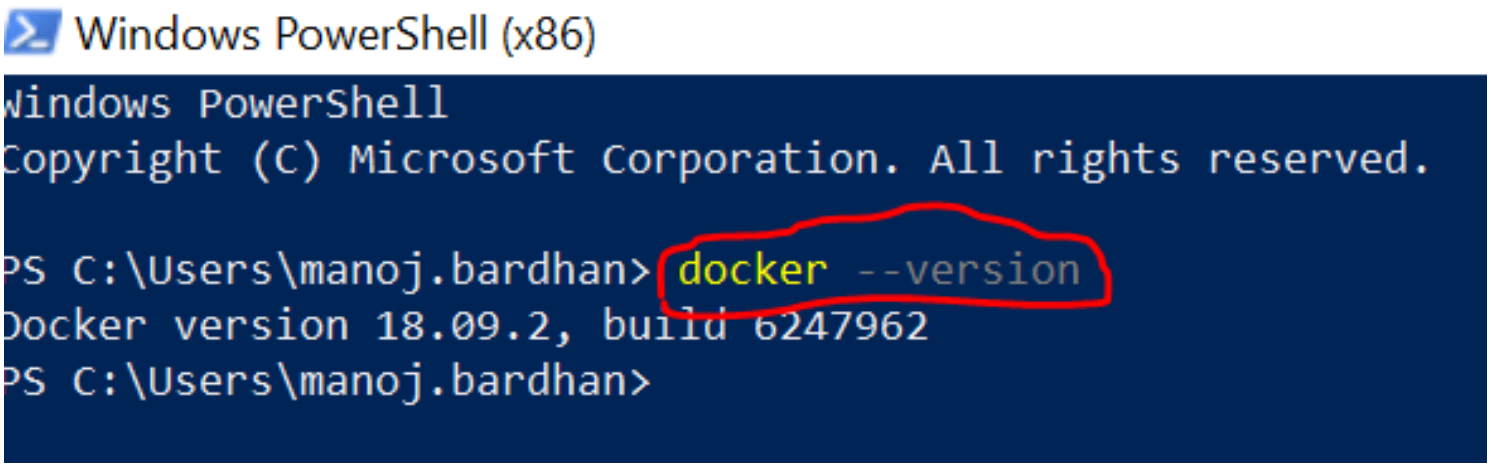
Deploying a Spring Boot Microservice to Docker: A Quick Guide

In this article, we will deploy our Spring Boot microservice to Docker. In our previous article, we had created a simple "Hello World" Spring Boot microservice (https://dzone.com/articles/hello-world-program-spring-boot). Now, we will deploy that application into Docker.

Docker with Spring Boot is currently a very popular technology stack which enables organizations to seamlessly develop and make production ready artifacts.

Before deploying the application to Docker, make sure you have installed Docker. In this example, we have used Docker Community Edition (Docker CE) on Windows OS. How to install docker on Windows/MacOS/Linux (https://docs.docker.com/install/). We can also use Alpine Linux image (https://hub.docker.com/_/alpine), as it provides a minimal Linux environment to deploy and run the application. There are a few Docker commands to manage your application. The below sample command and screen shot shows how to check the version of your installed Docker engine.

```
docker --version
```



What We Need to Deploy a Spring Boot application in Docker

First, we need to create an Image file for our application. Docker images are an important component for working with the Docker engine. Docker provides a Docker Hub (https://hub.docker.com/), which is a library and community for container images. We can use Docker Hub to get the most common images. In the below project structure, we have created a "*Dockerfile.txt*". If you put this Dockerfile into the class path, then the Docker engine will automatically identify and load this file.

Dockerfiles' names are very sensitive, so you must follow the naming convention as mentioned in the below screen shot. Docker reads commands/instructions from "*Dockerfile.txt*" and builds the image.

The below Dockerfile contains the commands to create the image. Actually there are many commands used for different purposes. Here we have used a few commands as per our needs.

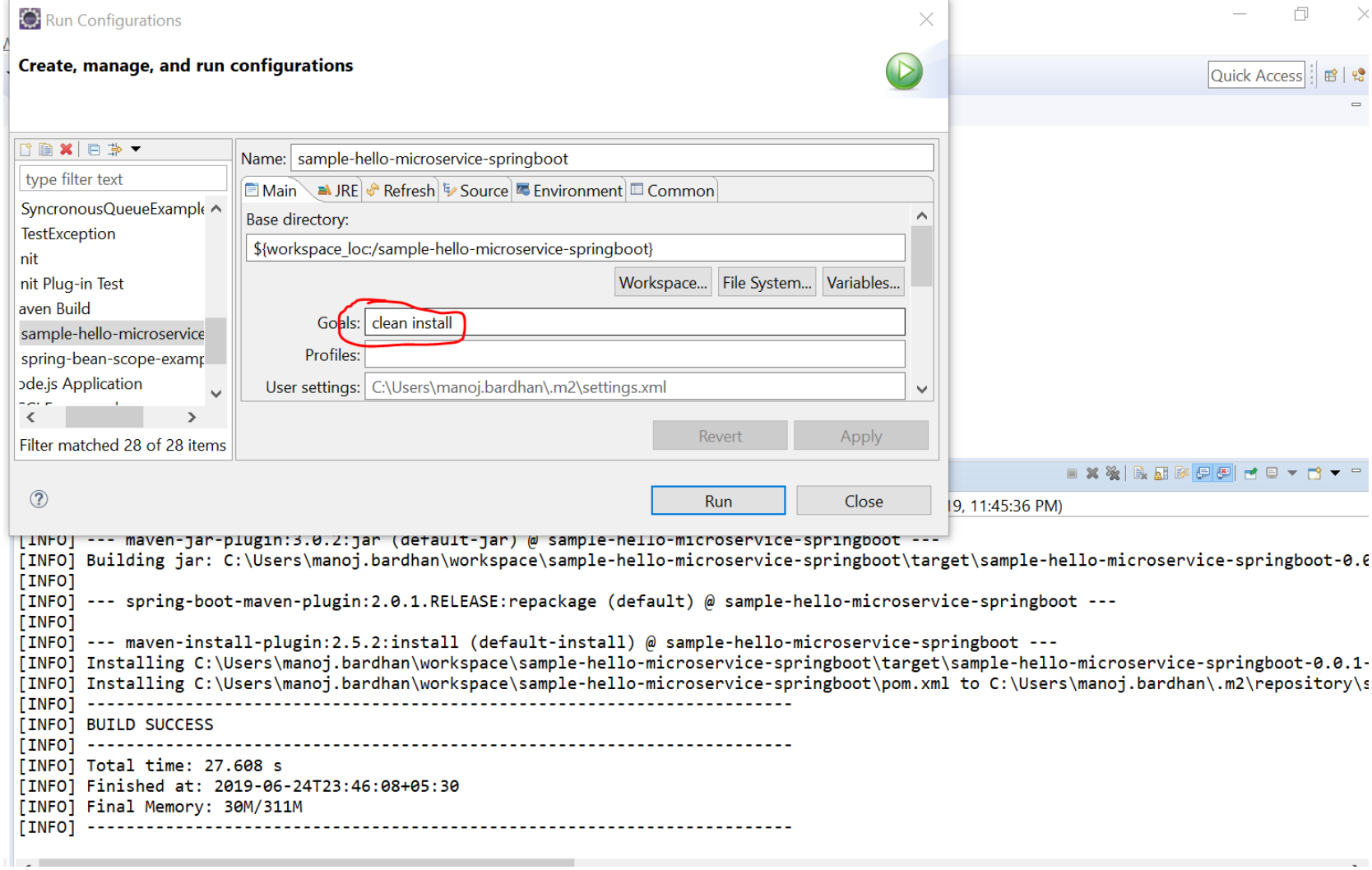
Dockerfile

```
FROM java:8
EXPOSE 8080
ADD /target/sample-hello-microservice-springboot-0.0.1-SNAPSHOT.jar sample-hello-microservice-springboot.jar
ENTRYPOINT ["java","-jar","sample-hello-microservice-springboot.jar"]
```

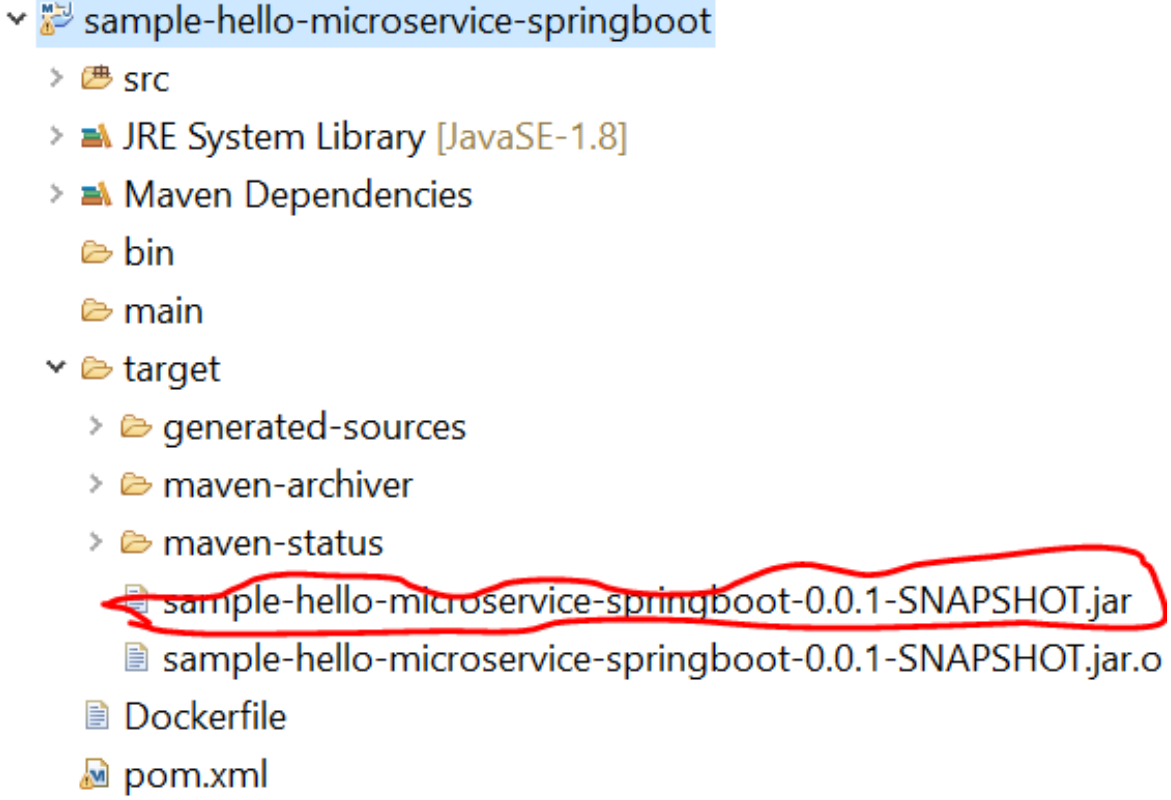
- **FROM** - Must be the first non-comment instruction in the Dockerfile. This command creates a layer from the Docker image. In our case, we have used `java:8` which means this application will run on Java 8.
- **EXPOSE** - Exposing the port for the endpoint. In this example, we have configured 8080.
- **ADD** - This command helps to take a source and destination. Normally, the source is your local copy. The `COPY` command also does same thing, but there is a small difference between the `COPY` and `ADD` commands.
- **ENTRYPOINT** - It's similar to CMD, where our command/jar file will be executed.
- **FROM** - Must be the first non-comment instruction in the Dockerfile. This command creates layer from the Docker image. In our case we have used `java:8`, which means this application will run on Java 8.

Now, run the command to build the image and deploy it to Docker. Before running the Docker command, we need to create the .jar file. This is because we are creating a jar file and then creating the jar file as a Docker image. So, here we have used the `mvn clean install` command to create the jar file. The below Maven command is used to create the jar file.

```
clean install
```



Now we can see the below jar has been created.



Create a Docker image file. The below command is used to create the image file.

```
docker build -t sample-hello-microservice-springboot .
```

```
PS C:\Users\manoj.bardhan\workspace\sample-hello-microservice-springboot> docker build -t sample-hello-microservice-springboot .
Sending build context to Docker daemon  16.15MB
Step 1/4 : FROM java:8
 ---> d23bdf5b1b1b
Step 2/4 : EXPOSE 8080
 ---> Using cache
 ---> aa9acc28d0b7
Step 3/4 : ADD /target/sample-hello-microservice-springboot-0.0.1-SNAPSHOT.jar sample-hello-microservice-springboot.jar
 ---> bb4a14b010ba
Step 4/4 : ENTRYPOINT ["java","-jar","sample-hello-microservice-springboot.jar"]
 ---> Running in 98b6f5bfe6fb
Removing intermediate container 98b6f5bfe6fb
 ---> 697d2dd008f0
Successfully built 697d2dd008f0
Successfully tagged sample-hello-microservice-springboot:latest
SECURITY WARNING: You are building a Docker image from Windows against a non-Windows Docker host. All files and directories added to build context will have '-rwxr-xr-x' permissions. It is
recommended to double check and reset permissions for sensitive files and directories.
PS C:\Users\manoj.bardhan\workspace\sample-hello-microservice-springboot>
```

As per the above screenshot, it seems we have created the Docker image successfully. You can check the created image by using the `"docker images"` command.

```
docker images
```

```
PS C:\Users\manoj.bardhan\workspace\sample-hello-microservice-springboot> docker images
REPOSITORY                                TAG                 IMAGE ID            CREATED             SIZE
sample-hello-microservice-springboot      latest              697d2dd008f0        2 minutes ago       659MB
```

Now, our image file is ready. We can push this image to Docker by using the below command.

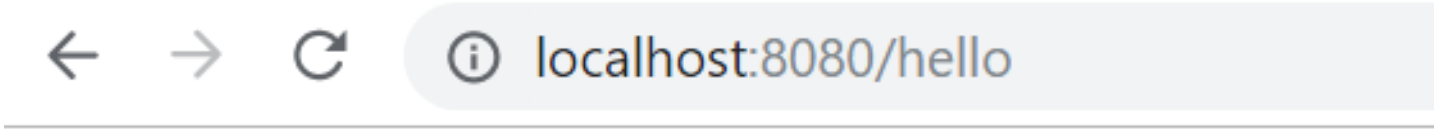
```
docker run -p 8080:8080 -t sample-hello-microservice-springboot
```

```
Windows PowerShell (x86)
PS C:\Users\manoj.bardhan\workspace\sample-hello-microservice-springboot> docker run -p 8080:8080 -t sample-hello-microservice-springboot

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.0.1.RELEASE)

2019-06-24 19:50:09.616  INFO   --- [           main] c.j.l.s.application.HelloApplication     : Starting HelloApplication v0.0.1-SNAPSHOT on f8654ffc2f1a with PID 1 (/sample-hello-microser
vice-springboot.jar started by root in /)
2019-06-24 19:50:09.625  INFO   --- [           main] c.j.l.s.application.HelloApplication     : No active profile set, falling back to default profiles: default
2019-06-24 19:50:09.916  INFO   --- [           main] ConfigServletWebServerApplicationContext : Refreshing org.springframework.boot.web.servlet.context.AnnotationConfigServletWebServerAppl
icationContext@4141d797: startup date [Mon Jun 24 19:50:09 UTC 2019]; root of context hierarchy
2019-06-24 19:50:12.623  INFO   --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2019-06-24 19:50:12.709  INFO   --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2019-06-24 19:50:12.711  INFO   --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet Engine: Apache Tomcat/8.5.29
2019-06-24 19:50:12.739  INFO   --- [ost-startStop-1] o.a.catalina.core.AprLifecycleListener   : The APR based Apache Tomcat Native library which allows optimal performance in production en
vironments was not found on the java.library.path: [/usr/java/packages/lib/amd64:/usr/lib/x86_64-linux-gnu/jni:/lib/x86_64-linux-gnu:/usr/lib/x86_64-linux-gnu:/usr/lib/jni:/lib:/usr/lib]
2019-06-24 19:50:12.874  INFO   --- [ost-startStop-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2019-06-24 19:50:12.875  INFO   --- [ost-startStop-1] o.s.web.context.ContextLoader            : Root WebApplicationContext: initialization completed in 2988 ms
2019-06-24 19:50:13.081  INFO   --- [ost-startStop-1] o.s.b.w.servlet.ServletRegistrationBean  : Servlet dispatcherServlet mapped to [/]
2019-06-24 19:50:13.087  INFO   --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'characterEncodingFilter' to: [/*]
2019-06-24 19:50:13.088  INFO   --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'hiddenHttpMethodFilter' to: [/*]
2019-06-24 19:50:13.088  INFO   --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'httpPutFormContentFilter' to: [/*]
2019-06-24 19:50:13.088  INFO   --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'requestContextFilter' to: [/*]
2019-06-24 19:50:13.503  INFO   --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/**/favicon.ico] onto handler of type [class org.springframework.web.servle
t.resource.ResourceHttpRequestHandler]
2019-06-24 19:50:13.878  INFO   --- [           main] s.w.s.m.m.a.RequestMappingHandlerAdapter : Looking for @ControllerAdvice: org.springframework.boot.web.servlet.context.AnnotationConfig
ServletWebServerApplicationContext@4141d797: startup date [Mon Jun 24 19:50:09 UTC 2019]; root of context hierarchy
2019-06-24 19:50:14.003  INFO   --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/hello]}" onto public java.lang.String com.javadevelopersguide.lab.springboot.busi
ness.DeliverHello.sayHello()
2019-06-24 19:50:14.016  INFO   --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/error],produces=[text/html]}" onto public org.springframework.web.servlet.ModelAn
dView org.springframework.boot.autoconfigure.web.servlet.error.BasicErrorController.errorHtml(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletResponse)
2019-06-24 19:50:14.018  INFO   --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/error]}" onto public org.springframework.http.ResponseEntity<java.util.Map<java.l
ang.String, java.lang.Object>> org.springframework.boot.autoconfigure.web.servlet.error.BasicErrorController.error(javax.servlet.http.HttpServletRequest)
2019-06-24 19:50:14.072  INFO   --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/webjars/**] onto handler of type [class org.springframework.web.servlet.re
source.ResourceHttpRequestHandler]
2019-06-24 19:50:14.072  INFO   --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/**] onto handler of type [class org.springframework.web.servlet.resource.R
esourceHttpRequestHandler]
2019-06-24 19:50:14.256  INFO   --- [           main] o.s.j.e.a.AnnotationMBeanExporter        : Registering beans for JMX exposure on startup
2019-06-24 19:50:14.342  INFO   --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2019-06-24 19:50:14.357  INFO   --- [           main] c.j.l.s.application.HelloApplication     : Started HelloApplication in 6.23 seconds (JVM running for 9.383)
```

Now our microservice has been deployed to Docker and its exposed on port 8080 as per our configuration in the Dockerfile. We can check the running container and its status.

```
PS C:\Users\manoj.bardhan\workspace\sample-hello-microservice-springboot> docker ps -a
CONTAINER ID        IMAGE                                   COMMAND                  CREATED             STATUS              PORTS               NAMES
84e837d13638        sample-hello-microservice-springboot    "java -jar sample-he…"   4 minutes ago       Up 4 minutes        8080/tcp            flamboyant_euler
```

Now we can also access from browser as below.



Hello Developer. I am running on Docker!!!

Below are a few more useful commands:

```
docker stop
docker ps --help
docker run --help
docker system prune
docker ps -a
```

There are many commands with help options to get the help about each command. Read more. (https://docs.docker.com/)

Happy learning!