Title: How to build a docker container for a Java application

Post Body:

What I want to do is build a docker image for my Java application but the following considerations should be true for most compiled languages.

problem

On my build server I want to produce a docker image for my application as the deliverable. For this I have to compile the application using some build tool (typically Gradle, Maven or Ant) and then add the created JAR file to the docker image. As I want the docker image to just execute the JAR file I will of course start from a base image with Java already installed.

There are three ways of doing this:

let the build tool control the process

In this case my build tool controls the whole process. So it prepares the JAR file and after the JAR is created it calls Docker to create the image. This works as the JAR is created beforehand and Docker can be oblivious of the build process needed to create the JAR.

But my Dockerfile is no longer standalone. It depends on steps to happen outside of Docker for it work. In my Dockerfile I will have a `COPY` or `ADD` statement that is supposed to copy the JAR file to the image. This statement will fail when the jar is not created beforehand. So just executing the Dockerfile might not work. This becomes a problem if you want to integrate with services that just build using the present Dockerfile like the auto-build feature on DockerHub.

let Docker control the build

In this case all necessary steps to create the image are added to the Dockerfile so the image can be created by just executing the Docker build.

The main problem with this approach is that there is no way to add to a Dockerfile commands that should be executed outside the docker image being created. This means I have to add my source code and my build tools to the docker image and build my JAR file inside the image. This will result in my image being bigger than it has to be due to all the files added that will be unnecessary at runtime. This will also add extra layers to my image.

Edit:

As @adrian-mouat pointed out if I would add the sources, build the application and deleted the sources in one RUN statement I could avoid adding unnecessary files and layers to the Docker image. This would mean creating some insane chained command.

two separate builds

In this case we split our build in two: first we create the JAR file using our build tool and upload it to a repository (Maven or Ivy repository). We then trigger a separate Docker build that just adds the JAR file from the repository.

conclusion

In my opinion the better way would be *letting the build tool control the process*. This is will result in a clean docker image and as the image is what we want to deliver this is of importance. To avoid having a potentially not working Dockerfile lying around this should be created as part of the build. So no one would accidentally use it to start a broken build.

But this will not allow me to integrate with DockerHub.

question

Is there another way I am missing?

update June 2020

In the years since I first created this question a lot of stuff has changed. At this point I would advocate using Googel's JIB Tool. It integrates with the most common Java Build Tools (Maven and Gradle) and allows you to create container directly from your build. This is much more concise than any of the old approaches I considered all these years ago.

update February 2021

I found this blog post and video from James Ward that reflects better what is currently state of the art.
https://cloud.google.com/blog/topics/developers-practitioners/comparing-containerization-methods-buildpacks-jib-and-dockerfile

Accepted Answer:

The docker registry hub has a Maven image that can be used to create java containers.

Using this approach the build machine does not need to have either Java or Maven pre-installed, Docker controls the entire build process.

Example

```
■■■ Dockerfile ■■■ pom.xml ■■■ src     ■■■ main     ■   ■■■ java     ■   ■   ■■■ org     ■   ■       ■■■ demo     ■   ■
```

Image is built as follows:

```
docker build -t my-maven .
```

And run as follows:

```
$ docker run -it --rm my-maven 0    [main] INFO  org.demo.App  - hello world
```

Dockerfile

```
FROM maven:3.3-jdk-8-onbuild CMD ['java','-jar','/usr/src/app/target/demo-1.0-SNAPSHOT-jar-with-dependencies.jar']
```

Update

If you wanted to optimize your image to exclude the source you could create a Dockerfile that only includes the built jar:

```
FROM java:8 ADD target/demo-1.0-SNAPSHOT-jar-with-dependencies.jar /opt/demo/demo-1.0-SNAPSHOT-jar-with-dependencies.jar CMD [
```

And build the image in two steps:

```
docker run -it --rm -w /opt/maven \    -v $PWD:/opt/maven \    -v $HOME/.m2:/root/.m2 \    maven:3.3-jdk-8 \    mvn clean inst
```

—

Update (2017-07-27)

Docker now has a multi-stage build capability. This enables Docker to build an image containing the build tools but only the runtime dependencies.

The following example demonstrates this concept, note how the jar is copied from target directory of the first build phase

```
FROM maven:3.3-jdk-8-onbuild   FROM java:8 COPY --from=0 /usr/src/app/target/demo-1.0-SNAPSHOT.jar /opt/demo.jar CMD ['java','
```

Highest Rated Answer:

Structure of java aplication

```
Demo ■■■ src |    ■■■ main |    ■   ■■■ java |    ■   ■   ■■■ org |    ■   ■       ■■■ demo |    ■   ■           ■■■ Ap
```

Content of Dockerfile

```
FROM java:8 EXPOSE 8080 ADD /target/demo.jar demo.jar ENTRYPOINT ['java','-jar','demo.jar']
```

Commands to build and run image

- Go to the directory of project.Lets say D:/Demo

```
$ cd D/demo $ mvn clean install $ docker build demo . $ docker run -p 8080:8080 -t demo
```

Check that container is running or not

```
$ docker ps
```

The output will be

```
CONTAINER ID        IMAGE               COMMAND                CREATED             STATUS              PORTS
```