Dockerize your Java Application

A Dockerfile is a fundamental building block used when dockerizing your Java applications, and it is how you can create a Docker image that can be used to create the containers you need for automatic builds.

Introduction to Dockerfiles

Docker builds images by reading instructions from a Dockerfile. A Dockerfile is a simple text file that contains instructions that can be executed on the command line. Using `docker build`, you can start a build that executes all of the command-line instructions contained in the Dockerfile.

Common Dockerfile instructions start with `RUN`, `ENV`, `FROM`, `MAINTAINER`, `ADD`, and `CMD`, among others.

- `FROM` - Specifies the base image that the Dockerfile will use to build a new image. For this post, we are using phusion/baseimage as our base image because it is a minimal Ubuntu-based image modified for Docker friendliness.

- `MAINTAINER` - Specifies the Dockerfile Author Name and his/her email.

- `RUN` - Runs any UNIX command to build the image.

- `ENV` - Sets the environment variables. For this post, `JAVA_HOME` is the variable that is set.

- `CMD` - Provides the facility to run commands at the start of container. This can be overridden upon executing the `docker run` command.

- `ADD` - This instruction copies the new files, directories into the Docker container file system at specified destination.

- `EXPOSE` - This instruction exposes specified port to the host machine.

Writing a Dockerfile for a simple Java application

It is not necessary to write a Dockerfile for OpenJDK in order to run a simple Java application, because you can obtain the official image of OpenJDK from the Docker Hub repository.

Let's create a Dockerfile for the Oracle JDK, which is not available on Docker Hub.

To begin this process, create a new folder and then create a file in it named "Dockerfile" with the following content.

```
# Dockerfile

FROM  phusion/baseimage:0.9.17

MAINTAINER  Author Name <author@email.com>
```

1. Update the package repository

```
RUN echo "deb http://archive.ubuntu.com/ubuntu trusty main universe" >

RUN apt-get -y update
```

2. Install python-software-properties

This enables add-apt-repository for use later in the process.

```
RUN DEBIAN_FRONTEND=noninteractive apt-get install -y -q python-softwar
```

3. Install Oracle Java 8

```
ENV JAVA_VER 8
ENV JAVA_HOME /usr/lib/jvm/java-8-oracle

RUN echo 'deb http://ppa.launchpad.net/webupd8team/java/ubuntu trusty m
    echo 'deb-src http://ppa.launchpad.net/webupd8team/java/ubuntu trus
    apt-key adv --keyserver keyserver.ubuntu.com --recv-keys C2518248EE
    apt-get update && \
    echo oracle-java${JAVA_VER}-installer shared/accepted-oracle-licens
    apt-get install -y --force-yes --no-install-recommends oracle-java$
    apt-get clean && \
    rm -rf /var/cache/oracle-jdk${JAVA_VER}-installer
```

4. Set Oracle Java as the default Java

```
RUN update-java-alternatives -s java-8-oracle

RUN echo "export JAVA_HOME=/usr/lib/jvm/java-8-oracle" >> ~/.bashrc
```

5. Clean Up APT when finished

```
RUN apt-get clean && rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*
```

6. Use baseimage-docker's init system

```
CMD ["/sbin/my_init"]
```

View the complete Dockerfile

```
# Dockerfile

FROM  phusion/baseimage:0.9.17

MAINTAINER  Author Name <author@email.com>

RUN echo "deb http://archive.ubuntu.com/ubuntu trusty main universe" >

RUN apt-get -y update

RUN DEBIAN_FRONTEND=noninteractive apt-get install -y -q python-softwar

ENV JAVA_VER 8
ENV JAVA_HOME /usr/lib/jvm/java-8-oracle

RUN echo 'deb http://ppa.launchpad.net/webupd8team/java/ubuntu trusty m
    echo 'deb-src http://ppa.launchpad.net/webupd8team/java/ubuntu trus
    apt-key adv --keyserver keyserver.ubuntu.com --recv-keys C2518248EE
    apt-get update && \
    echo oracle-java${JAVA_VER}-installer shared/accepted-oracle-licens
    apt-get install -y --force-yes --no-install-recommends oracle-java$
    apt-get clean && \
    rm -rf /var/cache/oracle-jdk${JAVA_VER}-installer

RUN update-java-alternatives -s java-8-oracle

RUN echo "export JAVA_HOME=/usr/lib/jvm/java-8-oracle" >> ~/.bashrc

RUN apt-get clean && rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*

CMD ["/sbin/my_init"]
```

Building the image from the Dockerfile

Now that we have written a Dockerfile, we can build the corresponding Docker image.

To build the image, start with the following command:

```
$ docker build -f Dockerfile -t demo/oracle-java:8 .
```

- `-f` specifies the Dockerfile. This can be skipped if the filename used at the beginning of this process is `Dockerfile`.

- `-t` specifies the name of the image. The name `demo/oracle-java`, and the `8` after the colon, specify the image tag. The tag `8` is used because we are using Java 8. This can be changed to any tag name that makes sense.

NOTE: Do not forget the `.`(dot) at the end of command; it specifies the context of the build. The `.`(dot) at the end of the command specifies the current directory. The files and directories of current directory will be sent to Docker daemon as a build artifact.

We have built our Java 8 image successfully, now we need to test it using sample Java application.

Testing the Image

Create a project folder and then create a file called *Main.java* inside this folder with following content:

```
public class Main
{
     public static void main(String[] args) {
        System.out.println("Hello, World");
    }
}
```

Now execute the following commands from the current project directory.

- To compile your Main.java file.

```
$ docker run --rm -v $PWD:/app -w /app demo/oracle-java:8 javac Mair
```

- To run your compiled Main.class file.

```
$ docker run --rm -v $PWD:/app -w /app demo/oracle-java:8 java Main
```

`Hello, World` output should be displayed.

Conclusion

You have seen in the above example that we used the Oracle Java image to successfully run a sample Java application.

If you need to use the `OpenJDK` for your application, you don't always need to write a Dockerfile and build an image. You can use the official Docker Hub repository version of Java.

Run the following commands to run your application using OpenJDK.

```
$ docker run --rm -v $PWD:/app -w /app java:8 javac Main.java

$ docker run --rm -v $PWD:/app -w /app java:8 java Main
```

Writing a Dockerfile for a Maven-based Java application

If you're using the OpenJDK with Maven, you don't necessarily need to write a Dockerfile because you can use the official Docker Hub repository's version of Maven. However, if you're using the Oracle JDK with Maven, then you'll need to write your own Dockerfile.

We will use the `demo/oracle-jdk:8` image as our base image because we have built this image in our previous example.

1. Create the Dockerfile

```
# Dockerfile
FROM demo/oracle-java:8

ENV MAVEN_VERSION 3.3.9

RUN mkdir -p /usr/share/maven \
  && curl -fsSL http://apache.osuosl.org/maven/maven-3/$MAVEN_VERSION/b
    | tar -xzC /usr/share/maven --strip-components=1 \
  && ln -s /usr/share/maven/bin/mvn /usr/bin/mvn

ENV MAVEN_HOME /usr/share/maven

VOLUME /root/.m2

CMD ["mvn"]
```

In this Dockerfile we have used the command `VOLUME`. This command is used to expose to the host machine the volume from the container. We can map this volume to any host directory.

2. Build the Docker image

Build the docker image from the above Dockerfile using this command:

```
$ docker build -f Dockerfile -t demo/maven:3.3-jdk-8 .
```

This will build an image with the name of `demo/maven` and tag of `3.3-jdk-8`. Name and tag your images clearly so that you can easily identify each image.

3. Run a test application

Run a test Maven application using this image that we created.

If you don't have a Maven project, create one using this command:

```
y.app -DartifactId=my-app -DarchetypeArtifactId=maven-archetype-quickstar
```

This will create a Maven project in the current directory under the *my-app* directory.

4. Build the project and test the JAR file

Go to the project directory.

```
$ cd  my-app
```

Build the project.

```
$ docker run -it --rm -v "$PWD":/app -w /app demo/maven:3.3-jdk-8 mvn p
```

Test the newly compiled and packaged JAR with the following command.

```
$ docker run -it --rm -v "$PWD":/app -w /app demo/maven:3.3-jdk-8 java
```

This should print the following output:

```
Hello World!
```

Write a Dockerfile for Spring MVC web application

There are two approaches we can take for Spring-based applications.

1. Use the existing Maven-based image as the base image and install Tomcat on it to run the web application.

2. Use the existing Maven-based image to only compile and package the application. And then use a different Tomcat-based Docker container to deploy the application.

We'll cover both approaches next.

Install Tomcat on the existing Maven-based image

Create an empty directory and add a Dockerfile inside it with the following content.

```
# Dockerfile
FROM demo/maven:3.3-jdk-8
MAINTAINER Author <autor@email.com>
RUN apt-get update && \
    apt-get install -yq --no-install-recommends wget pwgen ca-certifica
```

```
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*
ENV TOMCAT_MAJOR_VERSION 8
ENV TOMCAT_MINOR_VERSION 8.0.11
ENV CATALINA_HOME /tomcat
```

1. Install Tomcat

```
RUN wget -q https://archive.apache.org/dist/tomcat/tomcat-${TOMCAT_MAJO
        wget -qO- https://archive.apache.org/dist/tomcat/tomcat-${TOMCA
        tar zxf apache-tomcat-*.tar.gz && \
        rm apache-tomcat-*.tar.gz && \
        mv apache-tomcat* tomcat

ADD create_tomcat_admin_user.sh /create_tomcat_admin_user.sh
RUN mkdir /etc/service/tomcat
ADD run.sh /etc/service/tomcat/run
RUN chmod +x /*.sh
RUN chmod +x /etc/service/tomcat/run

EXPOSE 8080
```

2. Use baseimage-docker's init system

`CMD ["/sbin/my_init"]`

3. Create the Tomcat admin user

Create a file called `create_tomcat_admin_user.sh` in the same directory with following content:

```
!/bin/bash

if [ -f /.tomcat_admin_created ]; then
    echo "Tomcat 'admin' user already created"
```

```
    exit 0
fi
```

4. Generate Password

Continue to add the following script to generate a password.

```
anager-jmx,admin-gui, admin-script\"/>" >> ${CATALINA_HOME}/conf/tomcat-u
```

This file creates the Tomcat admin user. Add one more file in the same directory named as `run.sh` with following content. This will call the create users file and then reload the Tomcat server.

```
!/bin/bash

if [ ! -f /.tomcat_admin_created ]; then
    /create_tomcat_admin_user.sh
fi

exec ${CATALINA_HOME}/bin/catalina.sh run
```

5. Build and Test the Docker Image

Build the docker image using following command.

```
$ docker build -f Dockerfile -t demo/spring:maven-3.3-jdk-8 .
```

Test the image.

Here is the sample Spring Maven-based project that you can use for testing. You can download the zip or clone it by using the following command:

```
$ git clone  https://github.com/atifsaddique211f/spring-maven-sample.gi
```

Go to the project directory:

```
$ cd spring-maven-sample
```

Run the following command to build and package the project:

```
run -it --rm -v "$PWD":/app -w /app demo/spring:maven-3.3-jdk-8 mvn clear
```

The above command will create a war file at `target/springwebapp.war`.

Copy this war file into the Tomcat *webapps* directory:

```
"cp /app/target/springwebapp.war /tomcat/webapps/ & /tomcat/bin/catalina
```

The above command runs a container named `spring` in detached mode. It also provides mapping from the Docker container port `8080` to host machine port `8080`.

You can access your web application at the following url in the browser. http://localhost:8080/springwebapp/car/add.

This will show a sample form to add a car.

Since the Docker container is run in detached mode, it will keep running in the background. You can check the running container using following command:

```
$ docker ps
```

To check all running and non-running containers use this command:

```
$ docker ps -a
```

So to stop our Spring application container, we need to execute following command:

```
$ docker rm -vf spring
```

Run Spring app with a different Tomcat Docker image

Clone the same sample project:

```
$ git clone  https://github.com/atifsaddique211f/spring-maven-sample.gi
```

Go to the project directory:

```
$ cd spring-maven-sample
```

Create a Dockerfile with following content:

```
# Dockerfile
FROM tomcat:8

ADD target/*.war /usr/local/tomcat/webapps/
```

Build and package and the project:

```
$ docker run -it --rm -v "$PWD":/app -w /app demo/maven:3.3-jdk-8 mvn c
```

Build the Dockerfile:

```
$ docker build -f Dockerfile -t demo/tomcat:8 .
```

Run the application by executing this command:

```
$ docker run --rm -p 8080:8080 demo/tomcat:8
```

Now you can access your web application at the following url in the browser. http://localhost:8080/springwebapp/car/add.

This will show a sample form to add a car.

To stop the application, press `Ctrl + C`.

Conclusion

In this article, we've covered the fundamentals of how to go about dockerizing your Java application. We went over the basics of what a Dockerfile is, and how to create a Dockerfile for a simple Java application, a Maven-based Java application, and a Spring MVC web application. We've also learned how to build the Dockerfiles into an image, and to run the image to launch a container.