In this article we will see how we can run a stand-alone java application inside a Docker Container.

For this we need to check for a Docker image which has java installed in it. We can manually take a Image like Ubuntu and install java, commit that image and use it. But there will official images available so we will use them.

In this article we will also see how we can build a Docker container manually using the docker configuration file "Dockerfile". For this we need to create a file "Dockerfile" with the below contents as,

```
[root@vx111a dockerDemo]# cat Dockerfile
FROM java:8
MAINTAINER Jagadish Manchala <jagadishm@example.com>
COPY PingPong.java /
RUN javac PingPong.java
EXPOSE 8080
ENTRYPOINT ["java"]
CMD ["PingPong"]
```

Lets see what each line tells us,

From java:8
This defines the base image to use to start the build process. In this file we are asking docker to look for java:8 image and use that to build the container

MAINTAINER Jagadish Manchala <jagadishm@example.com>
The MAINTAINER instruction allows you to set the *Author* field of the generated images.

COPY PingPong.java /
The COPY instruction copies new files or directories from <src> and adds them to the filesystem of the container at the path <dest>. In this case we are copying the PingPong.java class from host machine to the docker container at location /.

RUN javac PingPong.java
The RUN instruction will execute any commands in a new layer on top of the current image. In this case we are asking the docker container once configured to compile the PingPong.java class

EXPOSE 8080
The EXPOSE instruction informs Docker that the container listens on the specified network ports at runtime. In this case we are asking the docker container to listen on 8080 port.

ENTRYPOINT ["java"]
An ENTRYPOINT allows you to configure a container that will run as an executable. We are making the java as an entry point to execute

CMD ["PingPong"]
The main purpose of a CMD is to provide defaults for an executing container. These defaults can include an executable, or they can omit the executable, in which case you must specify an ENTRYPOINT instruction as well. So in this case we passing the java command as entry point to execute the command PingPonng Java class. So in this case we are making the docker to execute the "java PingPong" class.

PingPong is a just a java Class with a Http Server exposed on 8080. The class looks like this,

```
[root@vx111a dockerDemo]# cat PingPong.java
import java.io.IOException;
```

```
import java.io.OutputStream;
import java.net.InetSocketAddress;
import com.sun.net.httpserver.HttpExchange;
import com.sun.net.httpserver.HttpHandler;
import com.sun.net.httpserver.HttpServer;

public class PingPong {

   public static void main(String[] args) throws Exception {
      HttpServer server = HttpServer.create(new InetSocketAddress(8080), 0);
      server.createContext("/ping", new MyHandler());
      server.setExecutor(null);
      server.start();
   }

   static class MyHandler implements HttpHandler {
      @Override
      public void handle(HttpExchange t) throws IOException {
         String response = "pong\n";
         t.sendResponseHeaders(200, response.length());
         OutputStream os = t.getResponseBody();
         os.write(response.getBytes());
         os.close();
      }
   }
}
```

Now make sure the PingPong.java class and Dockerfile exist in the same location so that the copy function above will work. Now once the dockerfile is configured we need to build that using,

```
[root@vx111a dockerDemo]# docker build -t jagadish/java8 .
Sending build context to Docker daemon 3.584 kB
Step 0 : FROM java:8
Trying to pull repository docker.io/library/java ... 8: Pulling from library/java
6d1ae97ee388: Pull complete
8b9a99209d5c: Pull complete
2e05a52ffd47: Pull complete
80887d145531: Pull complete
c973ca63d25f: Pull complete
88a622d91bc9: Pull complete
e05daba205ac: Pull complete
1915a5906f90: Pull complete
34550d1658f3: Pull complete
f629bd96ef4d: Pull complete
83da95e9d912: Pull complete
e9de8c6faf15: Pull complete
Digest: sha256:465e27bf9447f09839c7003bcfb77e13ca015a814f88ebd1d28b7a98920f4830
Status: Downloaded newer image for docker.io/java:8
---> e9de8c6faf15
Step 1 : MAINTAINER Jagadish Manchala <jagadishm@example.com>
---> Running in 4505663ef4cb
---> 455da507df7f
Removing intermediate container 4505663ef4cb
Step 2 : COPY PingPong.java /
---> 0276a9c4a60f
Removing intermediate container aedc19383bc3
Step 3 : RUN javac PingPong.java
```

---> Running in e623866af755
---> 94b91dfae987
Removing intermediate container e623866af755
Step 4 : EXPOSE 8080
---> Running in b379d526b739
---> 49a6d85f8bb8
Removing intermediate container b379d526b739
Step 5 : ENTRYPOINT java
---> Running in 09950b282267
---> 1f4380d93138
Removing intermediate container 09950b282267
Step 6 : CMD PingPong
---> Running in 7e312066b28c
---> e9b5033fc81f
Removing intermediate container 7e312066b28c
Successfully built e9b5033fc81f

Now once the prompt comes back we need to execute the "docker images" to see whether the image was available or not.

```
[root@vx111a dockerDemo]# docker images
REPOSITORY        TAG      IMAGE ID       CREATED          VIRTUAL SIZE
jagadish/java8    latest   e9b5033fc81f   9 minutes ago    641.9 MB
```

We can see that java 8 Docker image with name "Jagadish/java8" is ready.

Now the next step is to run the container with opening the Ports as,

```
[root@vx111a dockerDemo]# docker run -d -p 8080:8080 jagadish/java8
fb994b5f2aa848999cbea43b7c093abfce340b000a0b48308006b7b22136537c
```

Since we have the PingPong.java class running on 8080 inside the java 8 Docker Container we can test that using,

```
[root@vx111a dockerDemo]# wget http://localhost:8080/ping
--2015-12-30 19:15:57-- http://localhost:8080/ping
Resolving localhost (localhost)... ::1, 127.0.0.1
Connecting to localhost (localhost)|::1|:8080... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5
Saving to: 'ping'
100%
```

There will be file available where the wget command runs which containers the output "pong".

This is how we run a java 8 Docker container. More to Come, Happy Coding.