Developing Angular applications using Docker – The Problem Solver

Using Docker to deploy applications is great but there is so much more you can do with Docker if you want to. And it can solve some interesting problems along the way.

One problem when developing with NPM to manage dependencies is keeping all dependencies in sync. If I pull a repository from GitHub and an NPM install I will get a local copy, on my hard disk, of all dependencies. Fine, great that is just what it should do.

But if you pull exactly the same commit 10 minutes later you might end up with a different result. Sure the package.json is the same but the semantic ranges of packages listed in your as well as dependent packages might result in a different set of packages.

Is NPM doing random things here? No NPM does exactly the same thing every time but the issue is that package authors might have released a new version. And if everyone stuck to proper usage of Semantic Versioning and all code was perfectly tested this would probably be fine. Maybe not deterministic but okay.

Turns out that not everyone is doing the right Semantic Versioning thing, sometimes by accident and sometimes because they choose not to, it isn’t the law after all. The result is that the same *npm install* can have different result and break things. I have seen plenty of CI builds fail all of a sudden because of a buggy package being released on NPM and automagically be fixed again a day later when the bug was fixed. Of course after wasting our time investigating what was wrong. I know NPM shrinkwrap or Yarn are supposed to help here but we can do better.

And that is just one of the things you can fix using Docker.

Instead of installing all NPM packages locally (and on the CI server) install them in a Docker image and share the images between developers and the CI build. That way everyone shares exactly the same code and as a side benefit a *docker pull* is quite a bit faster than a *npm install* 😊 On my machine pulling the image from the Docker hub takes 23 seconds. Compare that to an npm install that takes 177 seconds, a full 7.5 times longer ☹

Developing an Angular application inside a Docker container

Before starting make sure to gave the Angular CLI installed.

Create a new project using the Angular CLI. Make sure to use the –skip-install option because we don’t want to install the NPM dependencies locally.

```
ng new docker-demo --skip-install
```

Next add a Dockerfile with the following content:

```
FROM node:6.10.2-alpine

RUN mkdir -p /app
WORKDIR /app

COPY package.json /app/

RUN ["npm", "install"]

COPY . /app

EXPOSE 4200/tcp

CMD ["npm", "start", "--", "--host", "0.0.0.0", "--poll", "500"]
```

Most of this is quite straightforward. There are two special things to note.

First only the package.json is copied and npm install is executed. Because of the layered system of the Docker file system this means that the slowest part of the Docker build will only execute again if the package.json is updated.

The second part is starting the container. First using *npm start* means that the Angular CLI doesn't need do be installed globally in the container image. The additional parameters ensure that we can access the development web server from the outside and that changes made to the source code on the host are picked up by the Webpack development server running inside of the container.

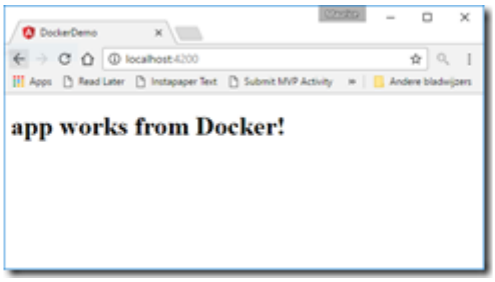
With that done you can build the container with the following command.

```
docker build -t docker-demo-dev .
```

As soon as it is build you can run it.

```
docker run -it --rm -p 4200:4200 -v ${pwd}/src:/app/src docker-demo-dev
```

I am running from Windows 10 using PowerShell here. The –v volume mapping means that I can make changes on the host and these are immediately available inside the Docker container. Open the blower on http://localhost:4200/ and make a change to the title of the AppComponent. You should see the bowser automatically update when you save the changes.



app works from Docker!

Great, push the Docker image to the Docker hub or another Docker repository and all your coworkers can pull and run the same container image with exactly the same NPM packages. No need to update this image until you make a change outside of the *./src* folder.

Creating a distribution version of your app

Regardless of how you intend to deploy your application you can use the same Docker container image to build a distributable version of your application with a simple command.

```
docker run -it --rm -v ${pwd}/src:/app/src -v ${pwd}/dist:/app/dist docker-d
```

Here I added volume mappings for both the *./src* folder as input and the *./dist* folder as output. After running this, preferably on the CI server, you will have the *./dist* folder just like you would have with a normal build.

Want to try it? You can pull this docker container from the Docker hub with the command *docker pull mauricedb/docker-demo-dev*. The source code in on GitHub here.

Cool stuff .