RUNNING YOUR NODE APP IN DOCKER

@ajafik - Terragon Group

Goal

The goal of this sharing is to do hands-on on how to get a Node.js application into a Docker container.

Requirement:

- Docker installed on your computer. (Docker for Mac, Linux, Windows)
- Basic understanding of how a Node.js application is structured.

Part 1

- In the first part we will create a simple web application in Node.js, then we will build a Docker image for that application, and lastly we will run the image as a container.

Why Docker?

- Docker allows you to package an application with all of its dependencies into a standardized unit, called a container, for software development. A container is a stripped-to-basics version of a Linux operating system. An image is software you load into a container.

Step 1: Create NodeJS app

- Create a directory called “dockerdemo” => mkdir dockerdemo
- On your terminal, run “cd dockerdemo”
- Run => npm init
- Run npm i -S express
- Run “touch server.js”
- Edit script section of your package.json to => "start": "node server.js" NB: with the quotes

Step 1: Create NodeJS app

- Then, create a server.js file that defines a web app using the Express.js framework:
-

```
'use strict';

const express = require('express');
// Constants
const PORT = 8080;
const HOST = '0.0.0.0';
// App
const app = express();
app.get('/', (req, res) => {
  res.send('Hello world\n');
});
app.listen(PORT, HOST);
console.log(`Running on http://${HOST}:${PORT}`);
```

Step 2: Create NodeJS app

- Create an empty file called Dockerfile: => touch Dockerfile
- The first thing we need to do is define from what image we want to build from. Here we will use the latest LTS (long term support) version boron of node available from the Docker Hub: https://hub.docker.com/r/library/node/tags/

Step 2: Create NodeJS app

- FROM node:boron => The Node Base Image to use.
- Next we create a directory to hold the application code inside the image, this will be the working directory for our application:
-
```
# Create app directory

RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app
```
-

Step 2: Create NodeJS app

- Install app dependencies:
- ```
  COPY package.json /usr/src/app/

  RUN npm install
  ```
- Bundle app source to the working directory.
- ```
  COPY . /usr/src/app
  ```
-

Step 2: Create NodeJS app

- Our app binds to port 8080 so you'll use the EXPOSE instruction to have it mapped by the docker daemon:
- `EXPOSE 8080`
- Last but not least, define the command to run your app using CMD which defines the runtime (in this case NPM). Here we will use the basic npm start which will run node server.js to start your server:
- `CMD [ "npm", "start" ]`

Step 2: Create NodeJS app

- Create a **.dockerignore** file in the same directory as your Dockerfile with following content.

NB: This will prevent our local modules and debug logs from being copied onto your Docker image and possibly overwriting modules installed within your image.

node_modules

npm-debug.log

And other files you do not want to bundle with your image.

Step 2: Building your image

In the directory that has your Dockerfile and run the following command to build the Docker image. The -t flag lets you tag your image so it's easier to find later using the docker images command:

On your terminal run :

docker build -t ajafik/node-web-app .

Your image will now be listed by Docker.

Step 2: Building your image

Run => “docker images” to check for your list of built images.

Step 2: Running the image

Running your image with -d runs the container in detached mode, leaving the container running in the background. The -p flag redirects a public port to a private port inside the container. Run the image you previously built:

Run =>

docker run -p 49160:8080 -d ajafik/node-web-app

- To get the container ID, Run: docker ps
- To check the logs of your container, Run: docker logs container_id

Step 2: Running the image

- In case you need to enter your container:
-
```
docker exec -it <container id> /bin/bash
```
- To test the app, we need to get the port mapped from the machine to Docker/container port : Run “docker ps”
- To see if everything works fine, go to http://localhost:<exposed_port> on your machine.

Chata! We are good to go.

Questions and Contributions

Thanks