Dockerizing a Node.js web app

Edit on GitHub

The goal of this example is to show you how to get a Node.js application into a Docker container. The guide is intended for development, and *not* for a production deployment. The guide also assumes you have a working Docker installation and a basic understanding of how a Node.js application is structured.

In the first part of this guide we will create a simple web application in Node.js, then we will build a Docker image for that application, and lastly we will instantiate a container from that image.

Docker allows you to package an application with its environment and all of its dependencies into a "box", called a container. Usually, a container consists of an application running in a stripped-to-basics version of a Linux operating system. An image is the blueprint for a container, a container is a running instance of an image.

Create the Node.js app

First, create a new directory where all the files would live. In this directory create a `package.json` file that describes your app and its dependencies:

```
{
  "name": "docker_web_app",
  "version": "1.0.0",
  "description": "Node.js on Docker",
  "author": "First Last <first.last@example.com>",
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },
  "dependencies": {
    "express": "^4.16.1"
  }
}
```

With your new `package.json` file, run `npm install`. If you are using `npm` version 5 or later, this will generate a `package-lock.json` file which will be copied to your Docker image.

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');
});
```

```
app.listen(PORT, HOST);
console.log(`Running on http://${HOST}:${PORT}`);
```

In the next steps, we'll look at how you can run this app inside a Docker container using the official Docker image. First, you'll need to build a Docker image of your app.

Creating a Dockerfile

Create an empty file called `Dockerfile`:

```
touch Dockerfile
```

Open the `Dockerfile` in your favorite text editor

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 `10` of `node` available from the Docker Hub:

```
FROM node:10
```

Next we create a directory to hold the application code inside the image, this will be the working directory for your application:

```
# Create app directory
WORKDIR /usr/src/app
```

This image comes with Node.js and NPM already installed so the next thing we need to do is to install your app dependencies using the `npm` binary. Please note that if you are using `npm` version 4 or earlier a `package-lock.json` file will *not* be generated.

```
# Install app dependencies
# A wildcard is used to ensure both package.json AND package-l
# where available (npm@5+)
COPY package*.json ./

RUN npm install
# If you are building your code for production
# RUN npm ci --only=production
```

Note that, rather than copying the entire working directory, we are only copying the `package.json` file. This allows us to take advantage of cached Docker layers. bitJudo has a good explanation of this here. Furthermore, the `npm ci` command, specified in the comments, helps provide faster, reliable, reproducible builds for production environments. You can read more about this here.

To bundle your app's source code inside the Docker image, use the `COPY` instruction:

```
# Bundle app source
COPY . .
```

Your 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 your runtime. Here we will use `node server.js` to start your server:

```
CMD [ "node", "server.js" ]
```

Your `Dockerfile` should now look like this:

```
FROM node:10

# Create app directory
WORKDIR /usr/src/app

# Install app dependencies
# A wildcard is used to ensure both package.json AND package-l
# where available (npm@5+)
COPY package*.json ./

RUN npm install
# If you are building your code for production
# RUN npm ci --only=production

# Bundle app source
COPY . .

EXPOSE 8080
CMD [ "node", "server.js" ]
```

.dockerignore file

Create a `.dockerignore` file in the same directory as your `Dockerfile` with following content:

```
node_modules
npm-debug.log
```

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

Building your image

Go to 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:

```
docker build -t <your username>/node-web-app .
```

Your image will now be listed by Docker:

```
$ docker images

# Example
REPOSITORY                      TAG        ID              CRE
node                            10         1934b0b038d1    5 d
<your username>/node-web-app    latest     d64d3505b0d2    1 m
```

Run 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:

```
docker run -p 49160:8080 -d <your username>/node-web-app
```

Print the output of your app:

```
# Get container ID
$ docker ps

# Print app output
$ docker logs <container id>

# Example
Running on http://localhost:8080
```

If you need to go inside the container you can use the `exec` command:

```
# Enter the container
$ docker exec -it <container id> /bin/bash
```

Test

To test your app, get the port of your app that Docker mapped:

```
$ docker ps

# Example
ID            IMAGE                                COMMAND
ecce33b30ebf  <your username>/node-web-app:latest  npm start
```

In the example above, Docker mapped the `8080` port inside of the container to the port `49160` on your machine.

Now you can call your app using `curl` (install if needed via: `sudo apt-get install curl`):

```
$ curl -i localhost:49160

HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 12
ETag: W/"c-M6tWOb/Y57lesdjQuHeB1P/qTV0"
Date: Mon, 13 Nov 2017 20:53:59 GMT
Connection: keep-alive

Hello world
```

We hope this tutorial helped you get up and running a simple Node.js application on Docker.

You can find more information about Docker and Node.js on Docker in the following places:

- Official Node.js Docker Image
- Node.js Docker Best Practices Guide
- Official Docker documentation
- Docker Tag on Stack Overflow
- Docker Subreddit

© OpenJS Foundation. All Rights Reserved. Portions of this site originally © Joyent.

Node.js is a trademark of Joyent, Inc. and is used with its permission. Please review the Trademark Guidelines of the OpenJS Foundation.

Node.js Project Licensing Information.

Thank you username for being a Node.js contributor **0 contributions**