Express Application

Table of Contents

Introduction Node with Express.js running on Docker

In this section we are working towards a server side application combining, node, express and mongoDb, but first we start with a hello world from express.

Node is described as "as an asynchronous event driven JavaScript runtime, Node is designed to build scalable network applications". In a browser javascript is to some degree sandboxed, but when it is running on Node it can draw on the many modules which Node provides and that includes networking capabilities.

node js

ExpressJS is a small web framework which sits on top of node.js

Express 4 is a breaking change from express 3 which means that express 3 applications will not work if you try to update it simply by incrementing the dependency version in package.json, it also means that you should avoid any tutorials which describe express version 3. Express 5 is coming soon and likewise code for express 4 may not urn in express 5, however this is still in beta so we will stick with express 4.

Express Js

Mongo is document based database. It can be downloaded and run on a machine, but there is a cloud version mongo Atlas which has a free account useful for development.

Mongo logo

Setting up with docker for a node server

Follow the outline of the instructions from node js on Dockerizing a Node.js web app

Make a new folder to work in named express1. Change directories in the terminal into this folder.

In this folder create four files.

1. package.json

```
{
    "name": "docker_express",
    "version": "1.0.0",
    "description": "Simple express",
    "author": "First Last <first.last@example.com>",
    "main": "server.js",
    "scripts": {
      "start": "nodemon index.js"
    },
    "dependencies": {
      "express": "^4.17.1"
    }
  }
```

This contains general information fields which you can edit to suit and also a reference the latest version of express. The node package manager running in a docker container will use this to pull all the files it needs to install express.

2. server.js

```
'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}`);
```

This is the express application. All this will do is return 'Hello world' if it recieves a GET request to port 8080 from a browser. Subsequently we can work in this file to develop the app.

3. dockerfile

Do not upgrade to the current version of node version 14 - thes just lead to dependancy disaster and a node nightmare.

```
FROM node:10

# Install nodemon
RUN npm install -g nodemon

# Create app directory
WORKDIR /usr/src/app

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

RUN npm install && mv /usr/src/app/node_modules /node_modules
# If you are building your code for production
# RUN npm ci --only=production

# Bundle app source
COPY . .

VOLUME ["/usr/src/app"]
EXPOSE 8080
CMD [ "nodemon", "server.js" ]
```

This will: * make a directory to work in. * write the node version to the terminal * copy package json and install all the modules listed as dependancies * move the node modules to another folder so that they will not be mounted to the users local file system. * copy the apps source code * Expose a port (may need to fix this for Heroku) * start node module server.js with nodemon a utility to automatically restart the server when the app is updated.

4. .dockerignore

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

This will prevent modules and logs copying from local files into docker where they may overwrite more up to date files.

Move to the project directory and start up Docker Desktop and build the image using the pattern:

docker build -t <your username>/imageName .

docker build -t dt/express_starter .

Terminal output:

```
Sending build context to Docker daemon  6.656kB
Step 1/9 : FROM node:10
10: Pulling from library/node
7919f5b7d602: Pull complete
0e107167dcc5: Pull complete
66a456bba435: Pull complete
5435318a0426: Pull complete
8494dd328465: Pull complete
3b01939c6506: Pull complete
cea1862d3fdb: Pull complete
3ff2b5bfcd35: Pull complete
d8d433ddc7ef: Pull complete
Digest: sha256:14fa22a8989cd64ce811db9d47e3ed2910e0f2d95323240e23bc9282
Status: Downloaded newer image for node:10
 ---> 2db91b8e7c1b
Step 2/9 : RUN npm install -g nodemon
 ---> Running in 6884b0d68442
/usr/local/bin/nodemon -> /usr/local/lib/node_modules/nodemon/bin/nodem

> nodemon@2.0.6 postinstall /usr/local/lib/node_modules/nodemon
> node bin/postinstall || exit 0

Love nodemon? You can now support the project via the open collective:
 > https://opencollective.com/nodemon/donate

npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@~2.1.2 (node_m
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for
","arch":"x64"})

+ nodemon@2.0.6
added 119 packages from 53 contributors in 8.8s
Removing intermediate container 6884b0d68442
 ---> 7de5fffa6c38
Step 3/9 : WORKDIR /usr/src/app
 ---> Running in 12850e315918
Removing intermediate container 12850e315918
 ---> b6b4cda7fc05
Step 4/9 : COPY package*.json ./
 ---> 104363140691
Step 5/9 : RUN npm install && mv /usr/src/app/node_modules /node_module
 ---> Running in eea594c8c0ef
```

```
npm notice created a lockfile as package-lock.json. You should commit t
npm WARN docker_express@1.0.0 No repository field.
npm WARN docker_express@1.0.0 No license field.

added 50 packages from 37 contributors and audited 50 packages in 5.073s and
audited 50 packages in 5.073s
found 0 vulnerabilities
                                    4c8c0ef
Removing intermediate container eea594c8c0ef
 ---> b1bb4a0c97a6
Step 6/9 : COPY . .
 ---> c756bdde2120
Step 7/9 : VOLUME ["/usr/src/app"]   fb3b5de
 ---> Running in 0869ffb3b5de
Removing intermediate container 0869ffb3b5de
 ---> 5b6e8fffbab5                   637a73d
Step 8/9 : EXPOSE 8080
 ---> Running in 47512637a73d
Removing intermediate container 47512637a73d
 ---> 3589d868e056
Step 9/9 : CMD [ "nodemon", "server.js" ]
 ---> Running in 70bcb4bd8c62
Removing intermediate container 70bcb4bd8c62
 ---> 9fd7b94ed9cc
Successfully built 9fd7b94ed9cc
Successfully tagged dt/express_starter:latest
SECURITY WARNING: You are building a Docker image from Windows against a non-
Windows Docker host. All files and directories added to build context will
have '-rwxr-xr-x' permissions. It is recommended to double check and reset
permissions for sensitive files and directories.
```

Check the image creation in docker desktop dashboard or by issuing:

docker images

Terminal output:

```
REPOSITORY                                      TAG       IMAGE ID
CREATED       SIZE
dt/express_starter                              latest    9fd7b94ed9cc   5
hours ago   919MB
node                                            10        2db91b8e7c1b   6
days ago    911MB
```

Running in the detatched mode (-d) and passing container port 8000 out to local port 49160:

Local files are mounted to the container volume.

docker run -p 49160:8080 –name express1 –mount type=bind,source=${pwd}/,target=/usr/src/app -d dt/express_starter

Terminal output (yours will differ):

```
b62f212afa96dcc9bcc78287415b623455b498154e217fb0be6a72ddc9803493
```

Check that the container is running in docker desktop dashboard or by issuing:

docker ps

Terminal output:

```
CONTAINER ID   IMAGE                COMMAND                  CREATED
STATUS          PORTS                    NAMES
b62f212afa96   dt/express_starter   "docker-entrypoint.s…"   45 seconds ago
Up 42 seconds   0.0.0.0:49160->8080/tcp   express1
```

Note the container ID b62f212afa96 this is not the same number as the image ID!
Note the port mapping 49160->8000

Check the container logs from the docker desktop dashboard or by issuing (Using the container ID):

docker logs b62f212afa96

Terminal output:

```
[nodemon] 2.0.6
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
Running on http://0.0.0.0:8080
```

The app is running in detached mode, but a terminal into it can be opened just to test:

docker exec -it b62f212afa96 /bin/bash

terminal output:

```
root@b62f212afa96:/usr/src/app#
```

Linux command line commands are now running in the container: >ls

shows the directory contents

```
Dockerfile  Dockerfile  package.json  server.js
```

pwd

shows the current directory path

```
/usr/src/app
```

ps

shows the process IDs which shows that (unlike a virtual machine) there are no processes running other than the ones deliberately started.

```
 PID TTY          TIME CMD
   27 pts/0    00:00:00 bash
   34 pts/0    00:00:00 ps
```

Since a bash shell has been started, try some bash scripting commands.

NAME="John"
echo $NAME

```
John
```

echo "${NAME}!"

```
John!
```

To exit type >CTRL d

or

exit

Terminal returns to host machine:

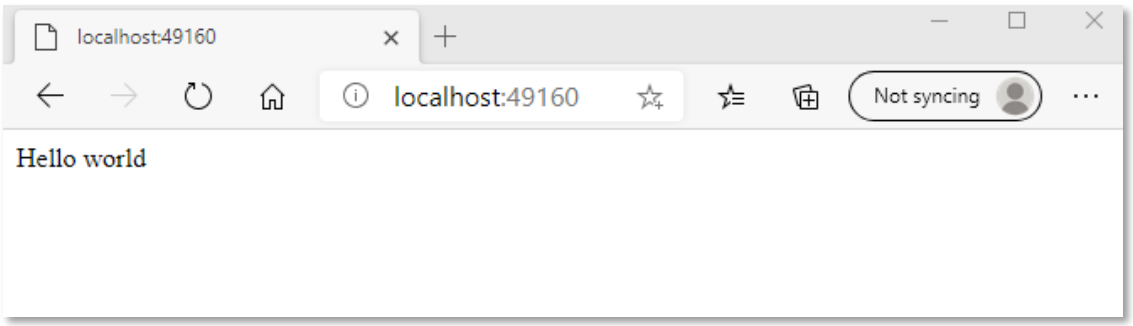
```
PS C:\Users\derek\Documents\Github\InternetTechnologies\express1>
```

The dt/node-web-app container is still running. Check this with

docker ps

```
CONTAINER ID   IMAGE                COMMAND                  CREATED
STATUS         PORTS                     NAMES
b62f212afa96   dt/express_starter   "docker-entrypoint.s…"   5 minutes ago
Up 4 minutes   0.0.0.0:49160->8080/tcp   express1
```

To check that the app is running point a browser at it on port 49160.



Hello world from node server

If it is installed, you can use curl to test the app. This will show the text returned as an HTTP response header and body.

curl http://localhost:49160

Terminal response:

```
StatusCode        : 200
StatusDescription : OK
Content           : Hello world

RawContent        : HTTP/1.1 200 OK
                    Connection: keep-alive
                    Keep-Alive: timeout=5
                    Content-Length: 12
                    Content-Type: text/html; charset=utf-8
                    Date: Mon, 23 Nov 2020 22:51:47 GMT
                    ETag: W/"c-M6tWOb/Y57lesdjQuHeB1P/qTV0"...
Forms             : {}
Headers           : {[Connection, keep-alive], [Keep-Alive, timeout=5],
[Content-Length, 12], [Content-Type, text/html; charset=utf-8]...}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : mshtml.HTMLDocumentClass
RawContentLength  : 12
```

Editing the application

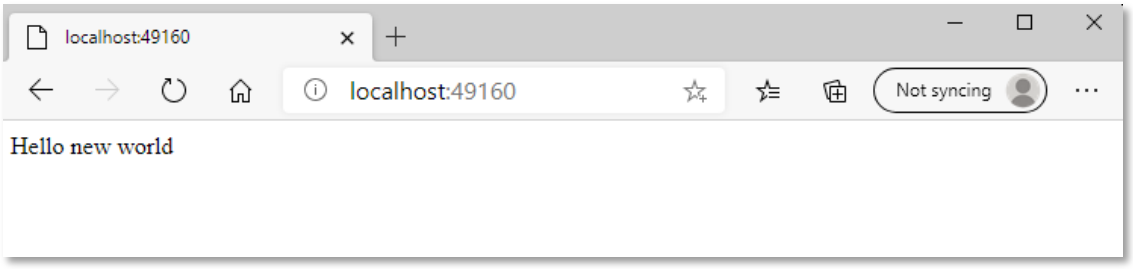
Change one line in server.js

```
app.get('/', (req, res) => {
  res.send('Hello new world \n');
});
```

If nodemon were working properly, this should update but at the moment it doesnt so:

docker restart express1

Refresh the browser.



hello new world

So a basic express server has been run in a container. Back to Nodemon, according to Remy the reason Nodemon does not work is down to express starting as .bin/www before server.js. To fix this later.

A simple example of express is running in a container.

References

Building a Simple CRUD Application with Express and MongoDB

Linux command line cheat sheet

Bash scripting cheatsheet

nodemon inside docker container