# Testing the application

I have an application running in hyperdoc/node-web-app

Run it locally by pulling

***docker pull hyperdoc/node-web-app***

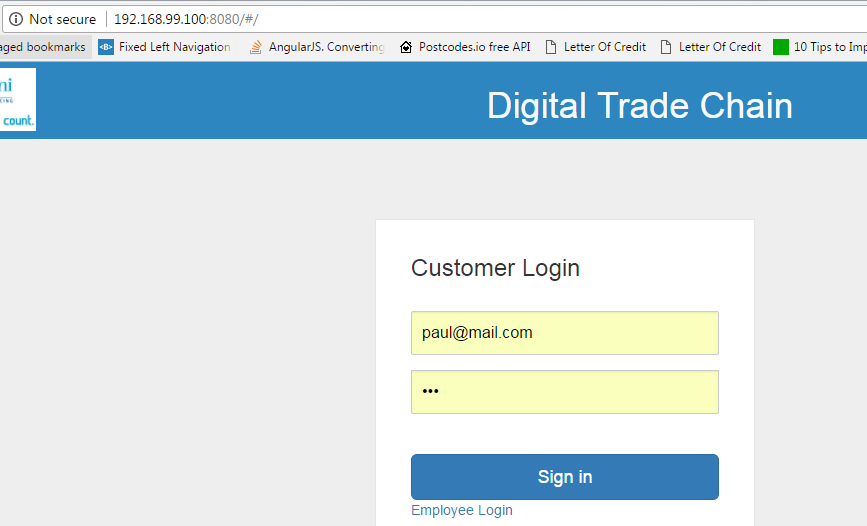
Now

***docker run hyperdoc/node-web-app***

in the browser run

before that run in mysql command prompt

**GRANT ALL PRIVILEGES ON \*.\* TO 'root'@'192.168.99.100' IDENTIFIED BY 'admin';**

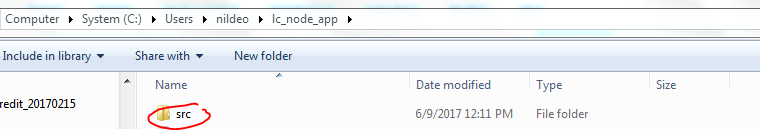


[***http://192.168.99.100:8080***](http://192.168.99.100:8080)

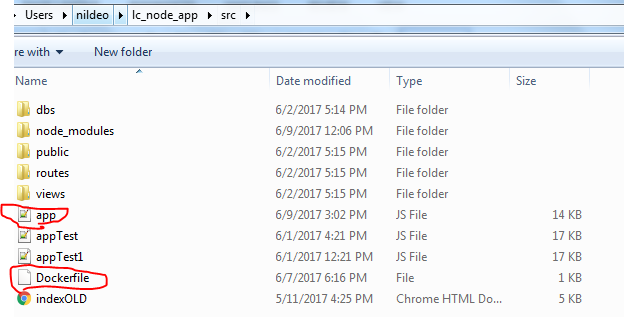
***Refers*** [***here***](#_some_necessary_commands) ***for some useful commands***

***The Steps:***

Create a node js app in a folder *src* locally.



Inside it create a file **package.js,server.js** file and a **Dockerfile**



In side package.json file write the following.

{

"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": "^4.13.3"

}

}

Create a server ***server.js***

'use strict';

const express = require('express');

// Constants

const PORT = 8080;

// App

const app = express();

app.get('/', function (req, res) {

res.send('Hello world\n');

});

app.listen(PORT);

console.log('Running on http://localhost:' + PORT);

# Create a Dockerfile using the following

FROM node:boron

# Create app directory

RUN mkdir -p /usr/src/app

WORKDIR /usr/src/app

# Install app dependencies

COPY package.json /usr/src/app/

RUN npm install

# Bundle app source

COPY . /usr/src/app

EXPOSE 8080

CMD [ "npm", "start" ]

Create a .dockerignore file

node\_modules

npm-debug.log

build the image(note not to ignore the . at the end of the command)

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

ex:***docker build –t nildeo/node-web-app .***

***docker build --no-cache=true -t nildeo/node-web-app .***

*(this will remove intermediate container and better if changes in the applications need to be replaced while running newly again)*

check your images

docker images

run the image in some port of your machine(in docker container the app is running in 8080 and mapped to 49160 on local machine).

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

get the container id using ***docker ps***

now you can see the logs using ***docker logs <container id>***

ex: docker logs b6a20cd61afa

# Publishing for reuse

Create an account in docker hub and create a repository.

Now from inside docker login into the hub(registry) using command ***docker login***

Provide your credentials.

Then ***docker push <your username>/node-web-app***

If found error create a tag for your local image(name should be same as your id/repository name)

Ex:

local image name: ***nildeo/node-web-app***

Remote repository login: ***hyperdoc***

Remote repository name: ***node-web-app***

First run the command: ***docker tag nildeo/node-web-app hyperdoc/node-web-app***

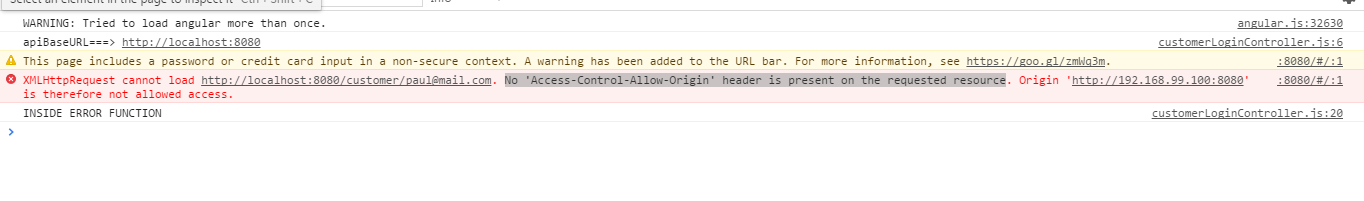
then push it to the cloud using: ***docker push hyperdoc/node-web-app***

good practice while creating an image name it like *dockerhublogin/imagename*

*ex:****hyperdoc/newApp***

# Cross Origin Resource Sharing

* While logging in the app if XMLHttp error shows ,such as the following:



Then it is due to CORS(Cross Origin Resource Sharing),which is disabled unless enabled otherwise.

In node require cors module in the application and use it accordingly.

Ex:

*var cors=require(‘cors’);*

*var app=express();*

*app.use(cors());*

# Some necessary commands

* **docker ps:**running containers
* **docker exec –it <conatainer\_id> bash:**to enter inside the docker container
* **docker images:**to check docker images and tags,names etc.
* **docker attach <container id> :**To see logs in the docker(alternatively docker logs container id can be used)
* **docke stop <container id>:** to stop a docker container
* **docker rm <container id>:** to remove a docker container
* **docker push repo\_login/repo\_name:**to push the local image to registry such as docker hub.
* **docker tag <local\_image\_name> <remote repository login/repository name>:** tag a local repo to same as remote repository.
* **docker login:**login to your remote repository
* **docker run -p 8080:8081 -d <image name>:**expose 8081 in the container and access 8080 from the default docker machine.
* **docker build --no-cache=true -t <image name> . :**build image without using cache
* **docker ip <image name>:** ip of the docker machine
* **docker inspect -f '{{.Name}} - {{.NetworkSettings.IPAddress }}' $(docker ps -aq):** to get all the ips of the containers.
* ***In docker-compose make this as follows:***

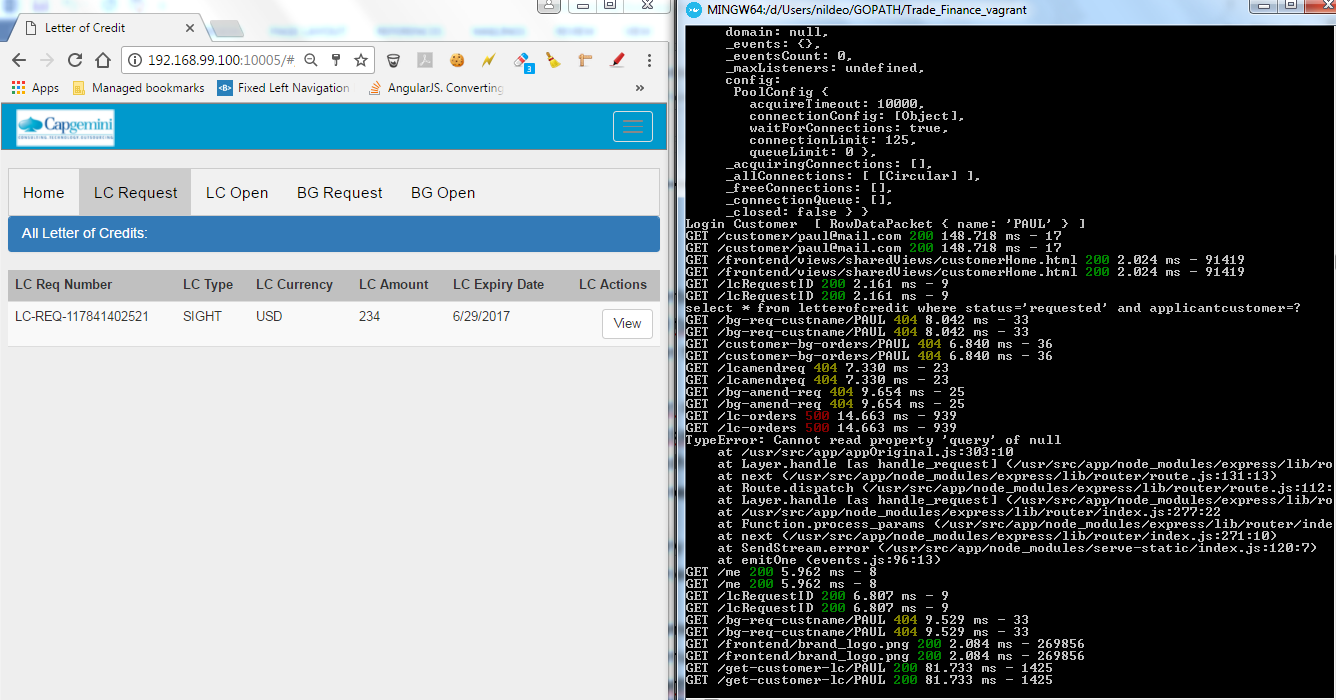
**docker inspect -f '{{.Name}} - {{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' $(docker ps -aq)**

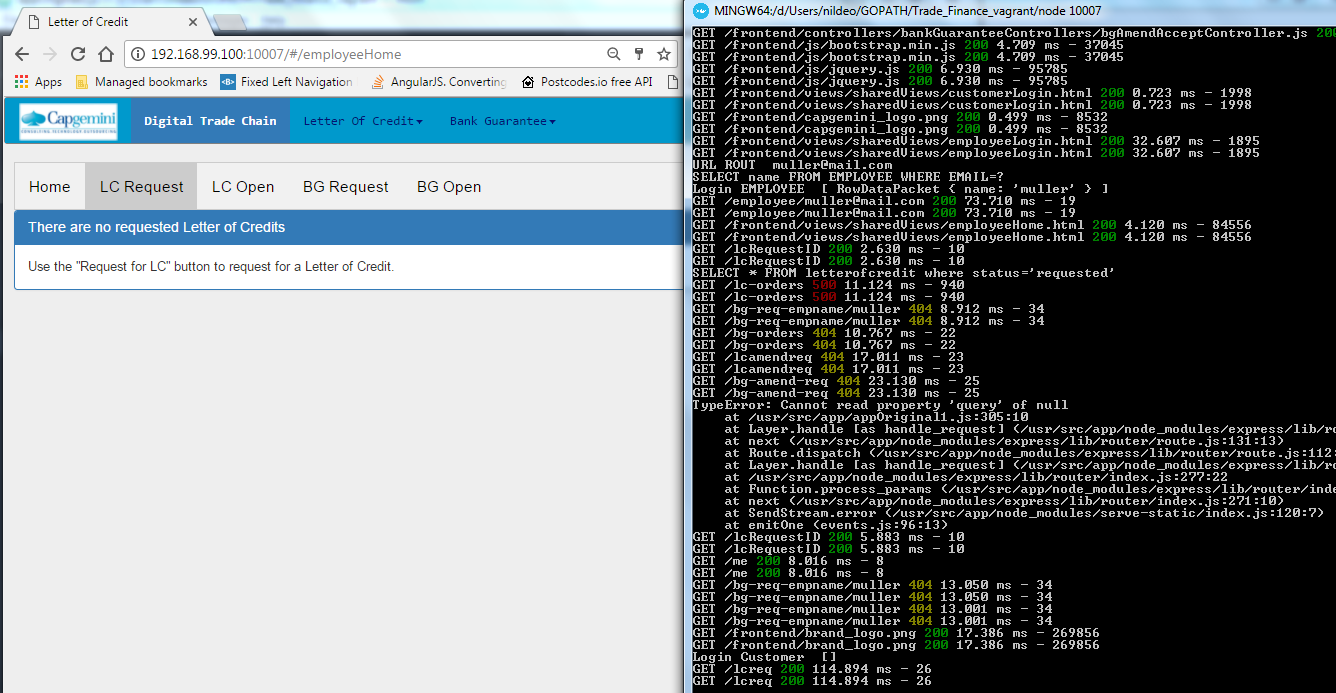
* **docker-machine ip <machine id>:** to check the ip of the running machine**(ex:docker-machine ip default)**
* **docker rmi $(docker images -f dangling=true -q):** remove dangling docker images(that are not linked to any tagged images)
* **docker rmi <image\_iD>:** removes docker image(image without having any child dependent image)

To debug in docker, run the container in ***attached*** mode using

docker attach <container\_id>







docker rmi 2a9ca0051b39 27f7cebb4137 b6a05e2cc431 0bd3ee538779 31872821a9fa 31872821a9fa 31872821a9fa 31872821a9fa 12730f3e5f0e 29b70d934a5c 3e474c7bfab1 fd44f8cea059 73d56223641a 828e588b3b50 b4f91e3a1a45 b4f91e3a1a45 d36d62b9c570 23ad5216b0bd 23ad5216b0bd 23ad5216b0bd 23ad5216b0bd 3f3928767182 7d8b4a8257f3 7d8b4a8257f3 7d8b4a8257f3 a3ac03980abf a3ac03980abf 958a7ae9e569 37f0ecae85a6 75b63e430bd1 da2b86c1900b 6c792d919591 ebcd9d4fca80 21cb00fb27f4 21cb00fb27f4