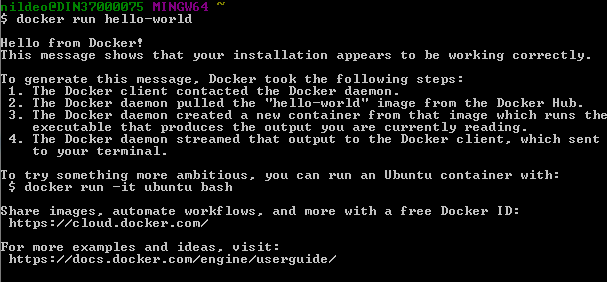
Download the official Ubuntu image:

$ docker pull ubuntu



Start a new container based on our image and pass a command to it:

$ docker run ubuntu /bin/echo ‘Hello world’

run a new container with Ubuntu and connect to it:

$ docker run -i -t Ubuntu

# Steps

Pull a sample docker-site image(docker-site is a sample image already on the hub)

***docker run --name static-site -e AUTHOR="Nilakantha" -d -P dockersamples/static-site***



Shows the port docker has exposed

***docker port static-site***



Default machine ip to run docker

***docker-machine ip default***



Try this on the browser (*note that it takes the default ip:192.168.99.100*)

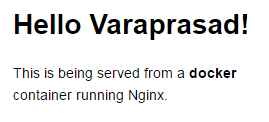
***http://192.168.99.100:32770/***



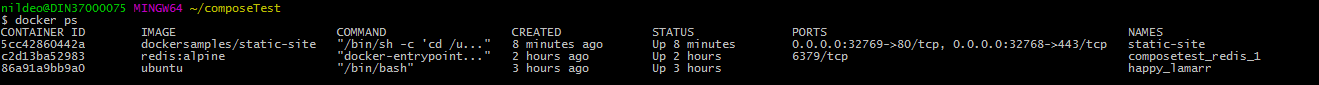
Same instance of the image can be made to run on a different port(here 8888 of vm mapped to port 80 of the container image static-site and renamed as static-site-2)

***docker run --name static-site-2 -e AUTHOR="Varaprasad" -d -p 8888:80 dockersamples/static-site***

Now running it on ***http://192.168.99.100:8888/***



**TO remove the image**

***Docker ps*** shows the pulled image’s(dockersmaples/static-site) id

Stop the image (passing container id)and remove it

***Docker stop 5cc42860442a***



# Step **Wise:**

Creating our own image

Create 3 files app.py,Dockerfile,requirements.txt,index.html(inside templates folder),inside a folder composeTest folder

touch app.py

touch Dockerfile

touch requirement.txt

mkdir templates

cd templates

touch index.html

**Testing the application**

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

Run it locally by pulling

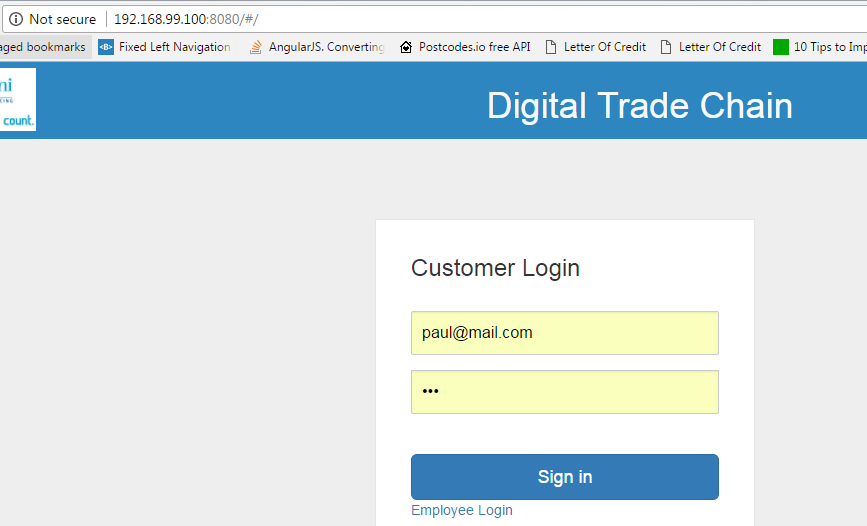
***docker pull hyperdoc/node-web-app***

Now

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

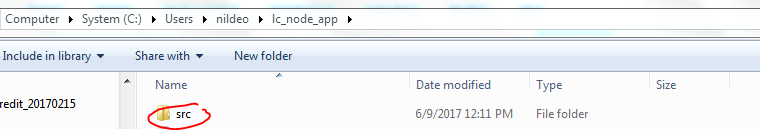
in the browser run

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



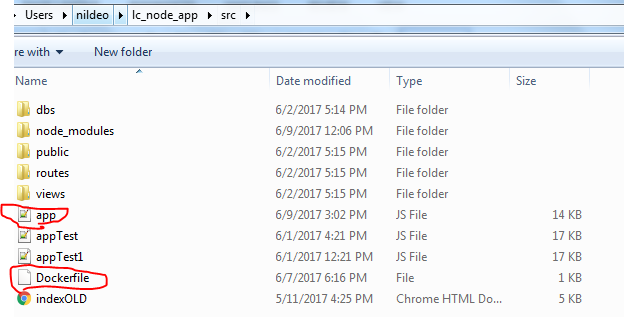
***The Steps:***

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



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

*(in my case app.js file is the server.js file and package.json removed)*



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***

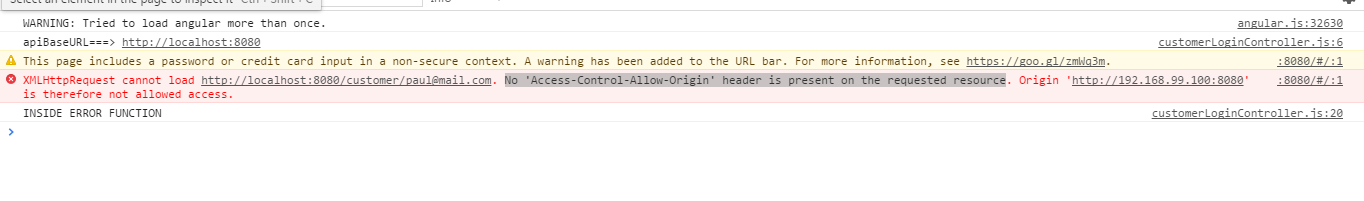
then push it to the cloud using: ***docker push hyperdoc/node-web-app***

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 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

# **Creating a swarm**

Use Docker machine

# Docker machine (https://docs.docker.com/machine/get-started/#create-a-machine)

If you want to create multiple local machines, you still need Docker Machine to create and manage machines for multi-node experimentation. Both Docker for Mac and Docker for Windows include the newest version of Docker Machine, so when you install either of these, you get docker-machine.

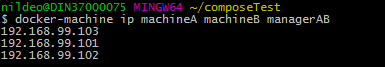
docker-machine create machineA

docker-machine create machine

docker-machine create managerAB



docker-machine ip managerAB machineA machineB



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_