Lab 3 - Docker File

Now that you have the basic of getting images from Docker, next we will have to try something extra by using Dockerfile, a script that describes the steps to build Docker image.

A docker container is basically created just by using images. An image can be as basic as nothing but just the operating-system fundamentals, or containing a set of application stack ready for launch.

When building your images with docker, each action taken forms a new **layer** on top of the previous one. Then we can use it to create base images that can be used to create new containers.

Let's see how we can use a Docker file to automate this process.

What is a Docker file ?

A Dockerfile is essentially just a script file. It contains a series of commands (also known as actions), to be performed on a base image in order to create (or form) a new one. A Dockerfile simplifies the deployment process and ensure a consistent workflow.

Dockerfiles begin with defining an image FROM which the build process starts. Followed by various other methods, commands and arguments (or conditions), in return, provide a new image which is to be used for creating docker containers.

They can be used by providing a Dockerfile's content - in various ways - to the docker daemon to build an image (as explained in the "How To Use" section).

Create a Docker NodeJS app

Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js is an event-driven, non-blocking I/O, and suitable for writing lightweight and asynchronous applications.

Create a NodeJS app

Create a new folder named **docker-nodejs**, inside the folder, create a file name `package.json` and insert the content below. A `package.json` describes our nodejs package, and the dependencies required.

```
{
  "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"
  }
}
```

Next, create a file `server.js`, which defines a web app that uses ExpressJS framework.

```
'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);
```

Compose Dockerfile

Then we create a file named `Dockerfile` (case-sensitive, so make sure you get the capitalization correct). With the following content.

```
FROM node:argon

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

COPY package.json /usr/src/app/
RUN npm install

COPY . /usr/src/app

EXPOSE 8080
CMD [ "npm", "start" ]
```

Build the image

Now execute the following command, by replacing `<your-own-name>` with your name (in my case `cherhan`). Also, watch out for the dot at the end of the command line. It means that we want to build a docker image, and tag it (option `-t`) with our name and the image name, using the files in current folder where we are at.

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

All the commands should be running by itself in autopilot mode without our intervention. After a while, you should see the following message.

```
Step 8 : CMD npm start
 ---> Running in 66bdf99ef022
 ---> 72f3d8a1450a
Removing intermediate container 66bdf99ef022
Successfully built 72f3d8a1450a
```

> The container ID is generated so you will see a different ID.

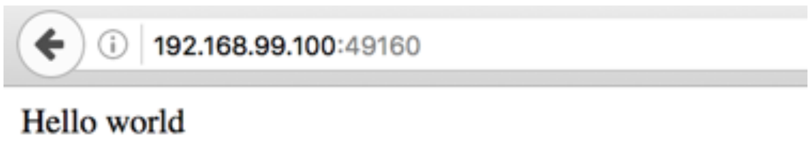
If there is no error message shown, your new Docker image is now created. Use the `docker image` command to verify.

Start a container using the new image

Let's start a container using the new image we have just built, using the following command.

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

This is to map the port 8080 from the container to the 49160 of the VM. Browse to http://192.168.99.100:49160 and you should see the following.



Hello world

This indicates that your image is loaded successfully.

What is in the Dockerfile?

Let's spend a few minutes to understand what we wrote in the Dockerfile.

```
FROM node:argon
```

This is usually the first thing we do. We define the base image we are using, which is the `argon` version of `node`, from docker hub.

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

Next we create a directory in our container, and set it as our working directory.

```
COPY package.json /usr/src/app/
RUN npm install
```

Then we copy the local file `package.json` to the working directory, and `RUN` executes a command. The `npm install` is a node package command that install all the required dependencies.

```
COPY . /usr/src/app
```

Next we copy our source files into the working directory, and lastly, we expose the port `8080` and using `CMD` to execute the command to start our node server, using:

```
EXPOSE 8080
CMD [ "npm", "start" ]
```

_

Push your image to DockerHub

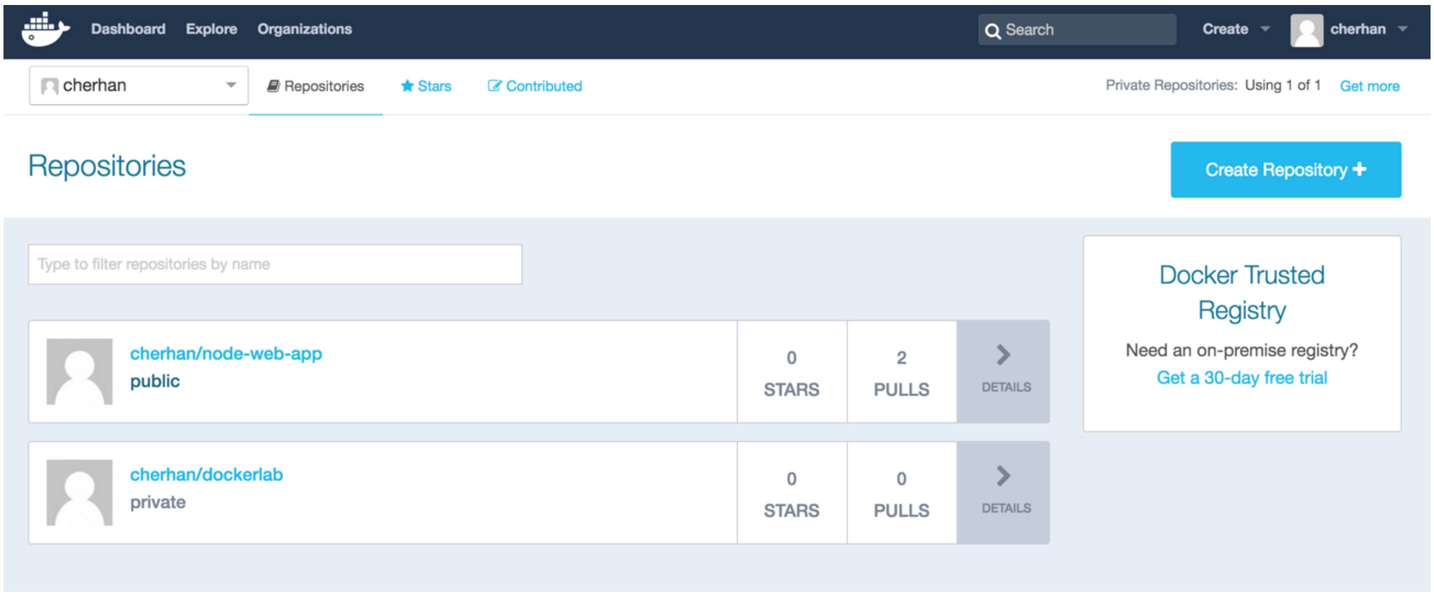
Please make sure that you have signed up for a docker hub account before beginning this lab exercise. Once you have confirmed, type the following command in command prompt (replace the `<username>` with your username):

```
$ docker login --username=<username>
```

Once it is done, enter the command:

```
$ docker push <username>/node-web-app
```

Wait for a few minutes for the upload to be completed. Open a browser and login to http://hub.docker.com, you will see the following screen:



Repositories

cherhan/node-web-app public	0 STARS	2 PULLS
cherhan/dockerlab private	0 STARS	0 PULLS

Docker Trusted Registry

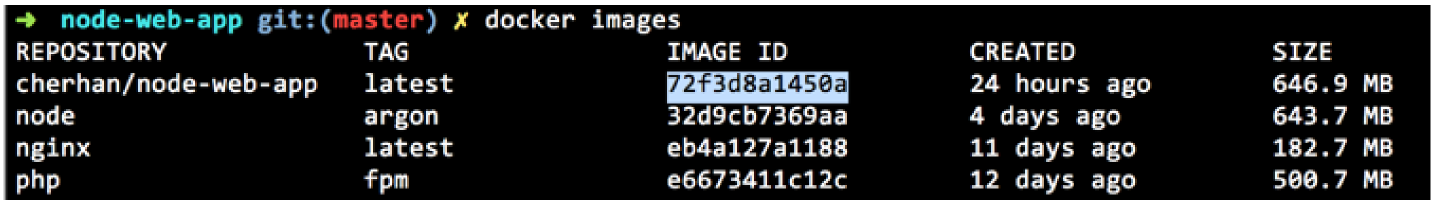
Need an on-premise registry?

Get a 30-day free trial

Tag your image

Now we have decided to make some modification to the current image but want to keep track of versioning. We can use `tag` feature in docker. First we use docker image to find the `image id` of our `<username>/node-web-app` image.

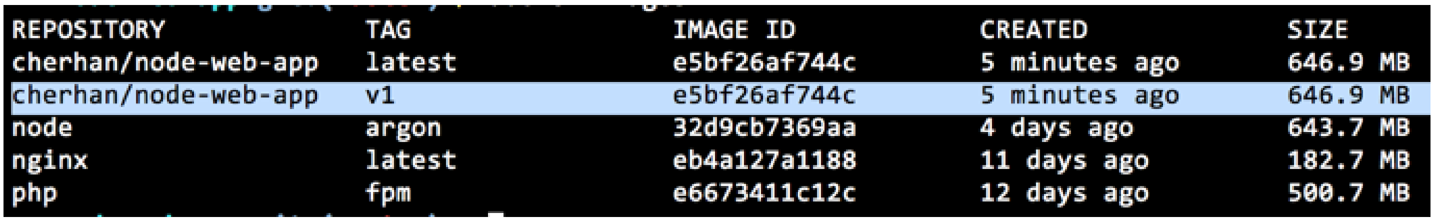
```
$ docker images
```



```
➜  node-web-app git:(master) ✗ docker images
REPOSITORY            TAG              IMAGE ID           CREATED           SIZE
cherhan/node-web-app  latest           72f3d8a1450a       24 hours ago      646.9 MB
node                  argon            32d9cb7369aa       4 days ago        643.7 MB
nginx                 latest           eb4a127a1188       11 days ago       182.7 MB
php                   fpm              e6673411c12c       12 days ago       500.7 MB
```

```
$ docker tag 72f3d8a1450a <username>/node-web-app:v1
```

After this, run the `docker images` command again and make sure both images exist, and there is an image tagged with `v1`.

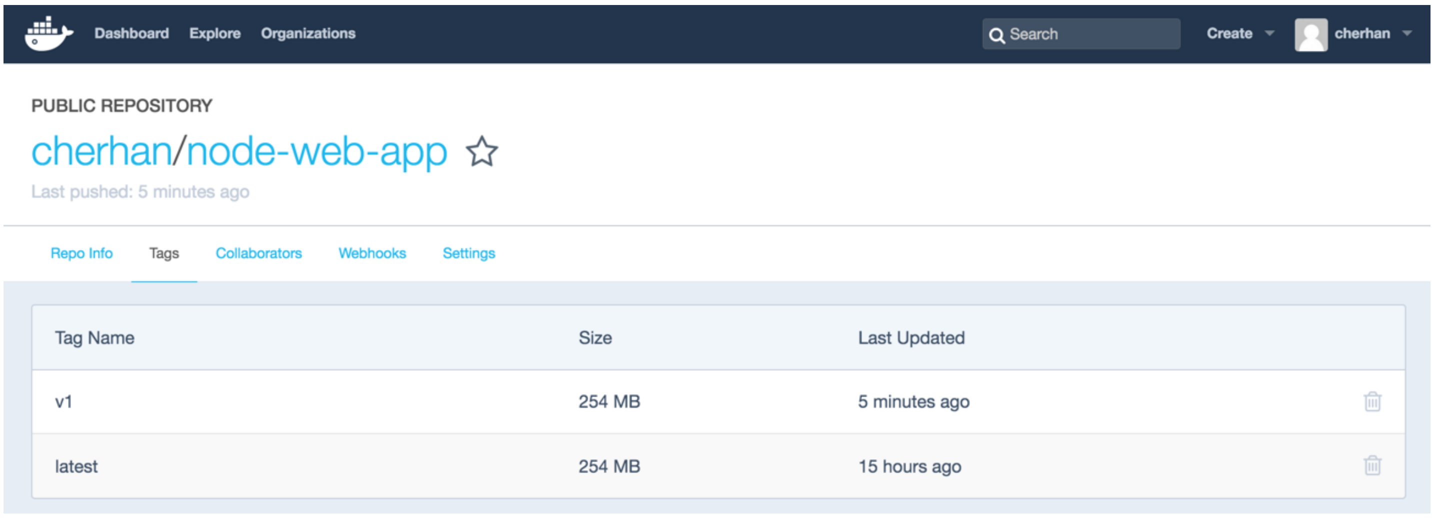


```
REPOSITORY            TAG              IMAGE ID           CREATED           SIZE
cherhan/node-web-app  latest           e5bf26af744c       5 minutes ago     646.9 MB
cherhan/node-web-app  v1               e5bf26af744c       5 minutes ago     646.9 MB
node                  argon            32d9cb7369aa       4 days ago        643.7 MB
nginx                 latest           eb4a127a1188       11 days ago       182.7 MB
php                   fpm              e6673411c12c       12 days ago       500.7 MB
```

Next we will have to push the image to our repository with the tag.

```
$ docker push <username>/node-web-app:v1
```

After the push is completed, browse to https://hub.docker.com/r//node-web-app/tags/ and verify the image is pushed to repository successfully.



Tag Name	Size	Last Updated
v1	254 MB	5 minutes ago
latest	254 MB	15 hours ago

Update our image

Now let's open the `server.js` using a text editor. Let's just change line 11 to:

```
res.send('Hello world\n I am version 2!');
```

Then we rebuild the image

```
$ docker build -t cherhan/node-web-app:latest .
```

Push it to server again.

Lab Exercise

Start the entire lab all over again, create another image, feel free to change the content of NodeJS, using HTML script, CSS or write additional NodeJS code using ExpressJS framework. Update the image and tag it with another version number. Once both version of images are uploaded, share it here https://docs.google.com/spreadsheets/d/1OtZ3ElSFMuIFhmoodbJUDvTHfsJHC6eIuwbGWvOYhDo/edit?usp=sharing and get your other classmates to download the image and use.

The next challenge is to download both version (i.e. v1 and latest), and run them concurrently, using different port and see the differences.