GitHub Gist

tonypujals / **docker-orientation-for-node-developers.md**
Last active 8 days ago

Docker Orientation for Node Developers

docker-orientation-for-node-developers.md Raw

Quick Start for getting acquainted with Docker for Node.js developers

Install Dependencies

Install Docker Toolbox

For Mac and Windows users, just install Docker Toolbox. It provides what you need to get started, including:

- Docker Machine - for creating Docker hosts ("machines")
- Docker Client - for communicating with docker hosts
- VirtualBox - to create docker hosts in a Linux-based (boot2docker) virtual machine

https://www.docker.com/toolbox

Working with Docker Machine

List Docker machines

```
$ docker-machine ls
NAME      ACTIVE   DRIVER         STATE     URL                         SWARM
default            virtualbox     Running   tcp://192.168.99.100:2376
```

Create a new machine

```
$ docker-machine create --driver virtualbox dev
Creating VirtualBox VM...
Creating SSH key...
Starting VirtualBox VM...
Starting VM...
To see how to connect Docker to this machine, run: docker-machine env dev
```

Now list machines

```
$ docker-machine ls
NAME      ACTIVE   DRIVER         STATE     URL                         SWARM
default            virtualbox     Running   tcp://192.168.99.100:2376
dev                virtualbox     Running   tcp://192.168.99.101:2376
```

Tell the Docker client to use the new machine

```
$ eval "$(docker-machine env dev)"
```

This command evaluates the command `docker-machine env dev`, which results in updating environment variables used by the Docker client to communicate with a machine.

This means that any `docker` commands you issue will be executed for the docker machine named `dev`.

Stop a machine

```
$ docker-machine stop dev
```

Start a machine

```
$ docker-machine start dev
Starting VM...
Started machines may have new IP addresses. You may need to re-run the `docker-machine env` command.
```

Get the machine host IP

```
$ docker-machine ip dev
192.168.99.101
```

ssh into the machine

```
$ docker-machine ssh dev
                        ##         .
                  ## ## ##        ==
               ## ## ## ## ##    ===
           /"""""""""""""""""\___/ ===
      ~~~ {~~ ~~~~ ~~~ ~~~~ ~~~ ~ /  ===- ~~~
           \______ o           __/
             \    \         __/
              \____\_______/
 _                 _   ____     _            _
| |__   ___   ___ | |_|___ \ __| | ___   ___| | _____ _ __
| '_ \ / _ \ / _ \| __| __) / _` |/ _ \ / __| |/ / _ \ '__|
| |_) | (_) | (_) | |_ / __/ (_| | (_) | (__|   <  __/ |
|_.__/ \___/ \___/ \__|_____\__,_|\___/ \___|_|\_\___|_|
Boot2Docker version 1.8.1, build master : 7f12e95 - Thu Aug 13 03:24:56 UTC 2015
Docker version 1.8.1, build d12ea79
docker@dev:~$
```

Create a machine using other provider drivers

https://docs.docker.com/machine/drivers/

DigitalOcean example

Make sure to create a personal access token: https://cloud.digitalocean.com/settings/applications

Add the access token to your environment:

```
export DIGITALOCEAN_ACCESS_TOKEN='...'
```

Create a machine

```
$ docker-machine create --driver digitalocean demo
Creating SSH key...
Creating Digital Ocean droplet...
To see how to connect Docker to this machine, run: docker-machine env demo
```

List the machine

```
$ docker-machine ls
NAME      ACTIVE   DRIVER         STATE     URL                          SWARM
default            virtualbox     Running   tcp://192.168.99.100:2376
demo               digitalocean   Running   tcp://107.170.201.137:2376
dev                virtualbox     Running   tcp://192.168.99.102:2376
```

Set `docker` client environment to use, just like any other machine

```
$ eval "$(docker-machine env demo)"
```

You can also log into your DigitalOcean console and see that the machine has been created.

Remove the machine if you won't need it

```
$ docker-machine rm demo
Successfully removed demo
```

Working with Docker

Launch a container to run a command, such as the bash builtin `echo` command. The container will be created using the BusyBox image (https://registry.hub.docker.com/_/busybox/). BusyBox is just a single executable that defines many common UNIX utilities. It's very small, so it's convenient to use for demonstration purposes. You can substitute `ubuntu` or another Linux distribution.

```
$ docker run --rm busybox echo hello
hello
```

Launch a BusyBox container and run an interactive shell

```
$ docker run --rm -it busybox sh
```

Working with Node

Docker Hub Node repository

https://hub.docker.com/_/node/

Pull node image

```
$ docker pull node
```

This downloads latest image to your machine.

Run the node shell in a container and execute some JavaScript.

```
$ docker -it --rm node
> console.log('hello')
hello
```

This runs the node REPL (read-evaluate-print-loop shell) in a container created from the node image.

It runs a REPL because the container runs the default command as defined by the node image CMD, as defined here:

https://github.com/nodejs/docker-node/blob/c2a8075f1e3155577c071bf1178c59370bb76d1a/4.0/Dockerfile#L26

```
FROM buildpack-deps:jessie
...
CMD [ "node" ]
```

Run an alternate shell command in the node container

Override the default CMD by supplying a command after the name of the `node` image.

```
$ docker -it --rm node node -v
v4.0.0

$ docker run -it --rm node npm -v
2.14.2
```

Run bash in the container

This will put us into a bash shell in the container

```
$ docker run -it --rm node bash
root@d72e494e756a:/#
```

In the bash shell in the node container

```
# node -v
v4.0.0

# npm -v
2.14.2
```

Create a simple node app

In a new directory...

Create a package.json file

```
{
  "name": "simple-docker-node-demo",
  "version": "1.0.0",
  "scripts": {
    "start": "node app.js"
  }
}
```

Create app.js

```
console.log('hello world!');
```

Create a `Dockerfile`

```
FROM node:onbuild
```

Build an image:

```
$ docker build -t demo-app .
Sending build context to Docker daemon 4.096 kB
Step 0 : FROM node:onbuild
onbuild: Pulling from library/node
843e2bded498: Already exists
8c00acfb0175: Already exists
...
# Executing 3 build triggers
Trigger 0, COPY package.json /usr/src/app/
Step 0 : COPY package.json /usr/src/app/
Trigger 1, RUN npm install
Step 0 : RUN npm install
 ---> Running in 85dc900dbab9
npm ...
npm info ok
Trigger 2, COPY . /usr/src/app
Step 0 : COPY . /usr/src/app
 ---> e90767b20603
...
Successfully built e90767b20603
```

Docker created an image based on the Dockerfile in the current directory and named it `demo-app-1` . As part of creating the image, it ran instructions triggered from the 'onbuild' base image that we specified in our Dockerfile that included copying the contents of the current directory to /usr/src/app/ and running npm install.

You can see what the instructions look like here:

https://github.com/nodejs/docker-node/blob/e763a1065077c580aab4d73945597c0b160b4ee2/4.0/onbuild/Dockerfile

The triggers are the `ONBUILD` statements. These will be run when the image is built that referenced this `Dockerfile` in its `FROM` statement.

```
FROM node:4.0.0

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

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

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

Run the demo app in a container

```
$ docker run --rm demo-app-1
npm info it worked if it ends with ok
npm info using npm@2.14.2
npm info using node@v4.0.0
npm info prestart simple-docker-node-demo@1.0.0
npm info start simple-docker-node-demo@1.0.0

> simple-docker-node-demo@1.0.0 start /usr/src/app
> node app.js

hello world
npm info poststart simple-docker-node-demo@1.0.0
npm info ok
```

The node app is trivial, but the mechanics of how this works is the same for more complicated examples. One thing you will probably want to do is export a port for accessing your node application, which we cover in the next section.

Pushing your image

Save your image for spinning up your app in a node container to Docker repository. You can use Docker Hub.

You will need to create an account on Docker Hub, then login from the command line:

```
$ docker login
```

Only official images (such as iojs) can have simple names. To push your own image, you will need to change the name of the image. Instead of demo-app, you will need to create the image using your login name. Mine is 'subfuzion' so my docker build and docker push commands would look like this:

```
$ docker build -t subfuzion/demo-app .
```

Or when you build, give it a tag:

```
$ docker build -t subfuzion/demo-app:1.0 .
```

Or if you forget, tag it after:

```
$ docker tag <image-id> subfuzion:/demo-app:v1
```

Then push the image to Docker Hub

```
$ docker push subfuzion/demo-app
```

Create Express API app and expose a port

Modify the demo to make it an express app. Install express:

```
$ npm install --save express
```

Update app.js to start an express server

```
const app = require('express')();
const port = process.env.PORT || 3000;

app.use('/', function(req, res) {
 res.json({ message: 'hello world' });
});

app.listen(port);
console.log('listening on port ' + port);
```

And update Dockerfile

```
FROM iojs:onbuild
expose 3000
```

Rebuild the image

```
$ docker build -t demo-app:v2 .
```

Now we can run it, but we want to map port 3000 inside the container to a port we can access from our system. We'll pick port 49100:

```
$ docker run --rm -t -p 49100:3000 demo-app:v2
```

This won't be an interactive session, so no need for `-i` , but the `-t` allows us to send signals from our client, like `CTRL-C` . As usual, `--rm` will remove the container for us when it is stopped.

Now we can access the app via port 49100, which will be mapped to port 3000 in the container.

If you're using Docker on a Mac, the port is actually mapped to the Docker host virtual machine. To determine the IP address, enter this at the command line:

```
$ docker-machine ip dev
192.168.99.102
```

You should be able to test http://192.168.99.102:49100/ in your browser.

The app looks for the environment variable PORT to be set, otherwise it defaults to 3000. We could specify an alternate port via the environment from the command line like this:

```
$ docker run --rm -t -p 49100:8080 -e "PORT=8080" demo-app
```

You will still access the app using the external port 49100, but it is now mapped to port 8080, which is what the app is listening to since the environment variable PORT was set.

Run it as a daemon

```
$ docker run -d -t -p 49100:3000 --name demo demo-app
dea4768765c176b679d1cb944fdf84466be17be7db0f749b47f5ab32a3ff2bc7
```

Naming containers is always a good idea, especially for daemons.

List it

```
$ docker ps
```

Stop it

```
$ docker stop demo
```

Remove it

```
$ docker rm demo
```

Mongo example

Create a Docker data volume container

```
$ docker create --name dbdata -v /dbdata mongo /bin/true
```

Start mongo

-without data volume container

```
$ docker run --name mongo -v /data/db -d mongo
```

-with data volume container

```
$ docker run -d --name mongo --volumes-from dbdata mongo
```

Connect with client container

```
$ docker run -it --link mongo:mongo --rm mongo sh -c 'exec mongo "$MONGO_PORT_27017_TCP_ADDR:$MONGO_PORT_27017_TCP_PORT/t
```

Backup database

```
$ docker run --rm --link mongo:mongo -v /root:/backup mongo bash -c 'mongodump --out /backup --host $MONGO_PORT_27017_TCP

$ docker-machine scp -r dev:~/test .
```

Node app container linking to mongo container

```
$ docker run -p 49100:3000 --link mongo:mongo demo-app
```

When launching your app container, Docker dynamically adds a `mongo` entry to the container's `etc/hosts` file and sets a number of convenient mongo-related environment variables, eliminating the need for you to have to provide configuration information. Here are all the environment variables that Docker created related to the mongo dependency in the previous command:

```
MONGO_ENV_MONGO_MAJOR=3.0
MONGO_PORT=tcp://172.17.0.89:27017
MONGO_ENV_MONGO_VERSION=3.0.3
MONGO_PORT_27017_TCP=tcp://172.17.0.89:27017
MONGO_PORT_27017_TCP_PROTO=tcp
MONGO_PORT_27017_TCP_ADDR=172.17.0.89
MONGO_NAME=/xapp/mongo
MONGO_PORT_27017_TCP_PORT=27017
```

In your app, you can create a connection URL to the Mongo `test` collection as simple as the following:

```
var mongoUrl = util.format('mongodb://mongo:%s/test', process.env.MONGO_PORT_27017_TCP_PORT);
```

Links

Docker Machine reference
https://docs.docker.com/machine/

Docker Machine basics
http://blog.codefresh.io/docker-machine-basics/

Machine presentation (Nathan LeClaire)
https://www.youtube.com/watch?v=xwj44dAvdYo&feature=youtu.be

Docker Machine Drivers
https://docs.docker.com/machine/drivers/

Linking Containers
https://docs.docker.com/userguide/dockerlinks/

Best practices for writing Dockerfiles
https://docs.docker.com/articles/dockerfile_best-practices/

Node, Docker, and Microservices
http://blog.codefresh.io/node-docker-and-microservices/