Docker Orientation

Tony Pujals

Working with Docker Machine

List machines

```
docker-machine ls
```

```
$ docker-machine ls
NAME       ACTIVE   DRIVER       STATE     URL                         SWARM
default    *        virtualbox   Saved
$
```

Create a new machine (Docker host)

```
docker-machine create --driver driver-name machine-name
docker-machine create -d driver-name machine-name
```

```
$ docker-machine create --driver virtualbox machine1
Creating VirtualBox VM...
Creating SSH key...
Starting VirtualBox VM...
Starting VM...
To see how to connect Docker to this machine, run: docker-machine env machine1
$
```

List machines again

Can see the new Docker machine is running

```
$ docker-machine ls
NAME        ACTIVE   DRIVER        STATE     URL                         SWARM
default     *        virtualbox    Saved
machine2             virtualbox    Running   tcp://192.168.99.100:2376
$
```

Tell Docker client to use the new machine

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

```
$ docker-machine env machine1
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.99.100:2376"
export DOCKER_CERT_PATH="/Users/tony/.docker/machine/machines/machine1"
export DOCKER_MACHINE_NAME="machine1"
# Run this command to configure your shell:
# eval "$(docker-machine env machine1)"
$
```

Tell Docker client to use the new machine

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

```
$ docker-machine env machine1
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.99.100:2376"
export DOCKER_CERT_PATH="/Users/tony/.docker/machine/machines/machine1"
export DOCKER_MACHINE_NAME="machine1"
# Run this command to configure your shell:
# eval "$(docker-machine env machine1)"
$
```

Displays environment settings you should use to configure your shell

Tell Docker client to use the new machine

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

```
$ docker-machine env machine1
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://192.168.99.100:2376"
export DOCKER_CERT_PATH="/Users/tony/.docker/machine/machines/machine1"
export DOCKER_MACHINE_NAME="machine1"
# Run this command to configure your shell:
# eval "$(docker-machine env machine1)"
$ eval "$(docker-machine env machine1)"
$
```

Displays environment settings you should use to configure your shell

Evaluates the environment settings in the current shell

Stop and start a machine

docker-machine stop|start *machine-name*

```
$ docker-machine stop machine1
$ docker-machine start machine1
Started machines may have new IP addresses. You may need to re-run the `docker-machine env` command.
$
```

ssh into the machine

`docker-machine ssh` *`machine-name`*

```
$ docker-machine ssh machine1
                        ##         .
                  ## ## ##        ==
               ## ## ## ## ##    ===
           /"""""""""""""""""""\___/ ===
      ~~~ {~~ ~~~~ ~~~ ~~~~ ~~~ ~ /  ===- ~~~
           \_____ o           __/
             \    \         __/
              \____\_______/

 _                 _   ____     _            _
| |__   ___   ___ | |_|___ \ __| | ___   ___| | _____ _ __
| '_ \ / _ \ / _ \| __| __) / _` |/ _ \ / __| |/ / _ \ '__|
| |_) | (_) | (_) | |_ / __/ (_| | (_) | (__|   <  __/ |
|_.__/ \___/ \___/ \__|_____\__,_|\___/ \___|_|\_\___|_|
Boot2Docker version 1.8.2, build master : aba6192 - Thu Sep 10 20:58:17 UTC 2015
Docker version 1.8.2, build 0a8c2e3
docker@machine1:~$
```

Create a machine using other drivers

`docker-machine create -d` *`driver-name machine-name`*

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

- Amazon Web Services (**amazonec2**)
- DigitalOcean (**digitalocean**)
- Exoscale (**exoscale**)
- Google Compute Engine (**google**)
- Generic (**generic**) - *for existing host with ssh*
- Microsoft Azure (**azure**)
- Microsoft Hyper-V (**hyper-v**)
- OpenStack (**openstack**)
- Rackspace (**rackspace**)
- IBM Softlayer (**softlayer**)
- Oracle VirtualBox (**virtualbox**)
- VMWare vCloud Air (**vmwarevcloudair**)

DigitalOcean Example

Create a DigitalOcean personal access token:

https://cloud.digitalocean.com/settings/applications

1

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

Create a DigitalOcean personal access token:

https://cloud.digitalocean.com/settings/applications

1

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

Create a machine

2

```
$ 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
```

Create a DigitalOcean personal access token:

https://cloud.digitalocean.com/settings/applications

1

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

Create a machine

2

```
$ 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
```

Set docker client shell environment

3

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

Create a DigitalOcean personal access token:

https://cloud.digitalocean.com/settings/applications

1

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

Create a machine

2

```
$ 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
```

Set docker client shell environment

3

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

List the machine

4

```
NAME      ACTIVE   DRIVER         STATE     URL                         SWARM
demo               digitalocean   Running   tcp://107.170.201.137:2376
```

Working with Docker

Launch a container to run a command

docker run --rm *image* [*cmd*]

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

Launch a container to run a command

`docker run --rm image [cmd]`

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

Create a container from this image

Launch a container to run a command

`docker run --rm image [cmd]`

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

Create a container from this image

Run this command in the container

Launch a container to run a command

`docker run --rm image [cmd]`

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

Automatically clean up (remove the container's file system) when the container exits

Create a container from this image

Run this command in the container

Launch a container to run an interactive command

```
docker run --rm -i -t image [cmd]
docker run --rm -it image [cmd]
```

```
docker $ docker run -it --rm alpine sh
/ # ls -l
total 48
drwxr-xr-x    2 root     root         4096 Jun 12 19:19 bin
drwxr-xr-x    5 root     root          380 Oct 13 02:18 dev
drwxr-xr-x   15 root     root         4096 Oct 13 02:18 etc
drwxr-xr-x    2 root     root         4096 Jun 12 19:19 home
drwxr-xr-x    6 root     root         4096 Jun 12 19:19 lib
lrwxrwxrwx    1 root     root           12 Jun 12 19:19 linuxrc -> /bin/busybox
drwxr-xr-x    5 root     root         4096 Jun 12 19:19 media
drwxr-xr-x    2 root     root         4096 Jun 12 19:19 mnt
dr-xr-xr-x  150 root     root            0 Oct 13 02:18 proc
drwx------    2 root     root         4096 Oct 13 02:18 root
drwxr-xr-x    2 root     root         4096 Jun 12 19:19 run
drwxr-xr-x    2 root     root         4096 Jun 12 12 19:19 sbin
dr-xr-xr-x   13 root     root            0 Oct 13 02:18 sys
drwxrwxrwt    2 root     root         4096 Jun 12 19:19 tmp
drwxr-xr-x    7 root     root         4096 Jun 12 19:19 usr
drwxr-xr-x    9 root     root         4096 Jun 12 19:19 var
```

Run this command in the container

sh is an interactive command...

Launch a container to run an interactive command

`docker run --rm -i -t` *`image [cmd]`*

`docker run --rm -it` *`image [cmd]`*

```
docker $ docker run --rm -it alpine sh
/ # ls -l
total 48
drwxr-xr-x    2 root     root         40          19:19 bin
drwxr-xr-x    5 root     root          3
drwxr-
drwxr
drwxr
lrwxr                                                  /bin/busybox
drwxr
drwxr
dr-xr
drwx-
drwxr
drwxr
dr-xr
drwxr
drwxr-
drwxr-xr-x
```

By default, the console is attached to all 3 standard streams of the process.

`-i (--interactive)` keeps STDIN open

`-t` allocates a pseudo-TTY (expected by most command line processes) so you can pass signals, like Ctrl-C (SIGINT)

The combination is needed for interactive processes, like a shell

Run this command in the container

`sh` is an interactive command...

Working with Docker and Node

Pull the latest node image

`docker pull node`

```
$ docker pull node
Using default tag: latest
latest: Pulling from library/node

843e2bded498: Pull complete
8c00acfb0175: Pull complete
8b49fe88b40b: Pull complete
20b348f4d568: Pull complete
16b189cc8ce6: Pull complete
116f2940b0c5: Pull complete
1c4c600b16f4: Pull complete
971759ab10fc: Pull complete
bdf99c85d0f4: Pull complete
a3157e9edc18: Pull complete
library/node:latest: The image you are pulling has been verified. Important: image verification is a tech preview
feature and should not be relied on to provide security.

Digest: sha256:559f91e2f6823953800360976e42fb99316044e2f9242b4f322b85a4c23f4c4f
Status: Downloaded newer image for node:latest
```

Run a container and display Node/npm version

```
docker run --rm node node -v; npm -v
```

```
$ docker run --rm node node -v; npm -v
v4.1.2
2.14.4
```

Run a container to evaluate a Node statement

docker run --rm node *node --eval "..."*

```
$ docker run --rm node node -e "console.log('hello')"
hello
```

Run the Node REPL in a container

```
docker run -it --rm node node
```

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

Running a simple Node app

Create a package.json file

demo-app/package.json

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

Add the express package

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

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

Create app.js

demo-app/app.js

```
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);
```

Add a Dockerfile

demo-app/Dockerfile

```
FROM node:onbuild
expose 3000
```

Add a .dockerignore

demo-app/.dockerignore

```
Dockerfile
.dockerignore
node_modules/
```

Build the Docker image for the app

`docker build -t image-tag .`

```
$ docker build -t demo-app:v1 .
Sending build context to Docker daemon  5.12 kB
```

The path (or url to a Git repo) defines the Docker build context.

All files are sent to the Docker daemon and are available to Dockerfile commands while building the image.

Build the Docker image for the app

`docker build -t image-tag .`

```
$ docker build -t subfuzion/demo-app:v1 .
Sending build context to Docker daemon  5.12 kB
```

Docker images get assigned Image IDs automatically, but you should also provide a tag in this form if you plan on publishing it:

user/repo:tag

Build the Docker image for the app

`docker build -t image-tag .`

```
$ docker build -t subfuzion/demo-app:v1 .
Sending build context to Docker daemon  5.12 kB
Step 0 : FROM node:onbuild
# Executing 3 build triggers
Trigger 0, COPY package.json /usr/src/app/
Step 0 : COPY package.json /usr/src/app/
 ---> Using cache
Trigger 1, RUN npm install
Step 0 : RUN npm install
 ---> Using cache
Trigger 2, COPY . /usr/src/app
Step 0 : COPY . /usr/src/app
 ---> ab7beb9c0287
Removing intermediate container 676c92cf1528
```

Execution of the 1st statement in the Dockerfile

FROM node:onbuild

Node base image

https://github.com/nodejs/docker-node

4.2/onbuild/Dockerfile

```
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" ]
```

Create a directory in the image and make it the working directory for subsequent commands

Node base image

https://github.com/nodejs/docker-node

4.2/onbuild/Dockerfile

```
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" ]
```

Create a directory in the image and make it the working directory for subsequent commands

When this image is used as a base for another image (child image), these instructions will be triggered.

As separate steps (layers), copy package.json, run npm install, and finally copy all the files (recursively) from the build context.

Node base image

https://github.com/nodejs/docker-node

4.2/onbuild/Dockerfile

```
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" ]
```

Create a directory in the image and make it the working directory for subsequent commands

When this image is used as a base for another image (child image), these instructions will be triggered.

As separate steps (layers), copy package.json, run npm install, and finally copy all the files (recursively) from the build context.

The command to execute when a container is started (can be overridden)

Build the Docker image for the app

`docker build -t image-tag .`

```
$ docker build -t subfuzion/demo-app:v1 .
Sending build context to Docker daemon  5.12 kB
Step 0 : FROM node:onbuild
# Executing 3 build triggers
Trigger 0, COPY package.json /usr/src/app/
Step 0 : COPY package.json /usr/src/app/
 ---> Using cache
Trigger 1, RUN npm install
Step 0 : RUN npm install
 ---> Using cache
Trigger 2, COPY . /usr/src/app
Step 0 : COPY . /usr/src/app
 ---> ab7beb9c0287
Removing intermediate container 676c92cf1528
Step 1 : EXPOSE 3000
 ---> Running in f16d963adcb4
 ---> d785b0f27ffa
Removing intermediate container f16d963adcb4
Successfully built d785b0f27ffa
```

Execution of the 2nd statement in the Dockerfile

EXPOSE 3000

The app will be listening to port 3000 in the container, but it can't be accessed outside the container unless exposed

You can also add tags after building an image

docker tag *image-id-or-tag tag*

```
$ docker tag d785b0f27ffa subfuzion/demo-app:latest

or

$ docker tag subfuzion/demo-app:v1 subfuzion/demo-app:latest
```

List the image

```
docker images
```

```
$ docker images
REPOSITORY                  TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
subfuzion/demo-app          latest              d785b0f27ffa        2 minutes ago       644.2 MB
subfuzion/demo-app          v1                  d785b0f27ffa        2 minutes ago       644.2 MB
```

Run the Node app in a container

docker run --rm -t -p *host-port:container-port image*

```
$ docker run --rm -t -p 3000:3000 subfuzion/demo-app:v1
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

listening on port 3000
```

Run the Node app in a container

`docker run --rm -t -p` *`host-port:container-port image`*

```
$ docker run --rm -t -p 3000:3000 subfuzion/demo-app:v1
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

listening on port 3000
```

Create a container from this image and run the default command (**npm start**)

Run the Node app in a container

`docker run --rm -t -p host-port:container-port image`

```
$ docker run --rm -t -p 3000:3000 subfuzion/demo-app:v1
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

listening on port 3000
```

Map a port on the docker machine to the container's exposed port

Run the Node app in a container in detached mode

docker run -d -t -p *host-port:container-port image*

```
$ docker run -d -t -p 80:3000 --name demo subfuzion/demo-app:v1
be76984370dd8e3aa4066af955eb54ab4116495007b7cd45743700804392555a

$ docker logs demo
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

listening on port 3000
```

Run the Node app in a container in detached mode

docker run -d -t -p *host-port:container-port image*

```
$ docker run -d -t -p 80:3000 --name demo subfuzion/demo-app:v1
be76984370dd8e3aa4066af955eb54ab4116495007b7cd45743700804392555a

$ docker logs demo
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

listening on port 3000
```

Good idea to name your containers, especially detached ones

```
$ docker ps
CONTAINER ID        IMAGE                   COMMAND             CREATED             STATUS
PORTS                     NAMES
be76984370dd        subfuzion/demo-app:v1   "npm start"         2 minutes ago       Up 2 minutes
0.0.0.0:8000->3000/tcp   demo

$ docker inspect demo
. . .

$ docker stop demo
demo

$ docker rm demo
```

Accessing the running container

Mapped Docker machine port to container port, the IP address will be the IP address of the machine

```
$ docker-machine ip machine1
192.168.99.100

$ curl http://192.168.99.100:8000
{"message":"hello world"}

# or

$ curl http://$(docker-machine ip machine1):8000
{"message":"hello world"}
```

Link to Mongo container

Create a Docker data volume container

```
$ docker create --name mongo-db -v /dbdata mongo /bin/true
b582fc52d80e62588816683479a99ce5c11d756372e008221e594af9dafd32a3
```

Start Mongo container & attach data volume container

```
docker run -d --name mongo --volumes-from mongo-db mongo
80ecaa6958a6904ee26259b36ae75ed0e2cd6105e1f4f074243ddf868c491f54

# connect from a mongo client container
$ docker run -it --link mongo:mongo --rm mongo sh \
  -c 'exec mongo "$MONGO_PORT_27017_TCP_ADDR:$MONGO_PORT_27017_TCP_PORT/test"'

# backup database
$ docker run --rm --link mongo:mongo -v /root:/backup mongo bash \
  -c 'mongodump --out /backup --host $MONGO_PORT_27017_TCP_ADDR'
$ docker-machine scp -r dev:~/test .
```

Link Node app container to Mongo container

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

Docker provisions Node container environment

```
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
```

Accessing Mongo connection in your app

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

For more information

Docker Machine reference

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

Docker Machine basics

http://atomiq.io/docker-machine/

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://atomiq.io/node-docker-and-microservices/

@subfuzion
https://twitter.com/subfuzion

https://www.linkedin.com/in/tonypujals/