App Container

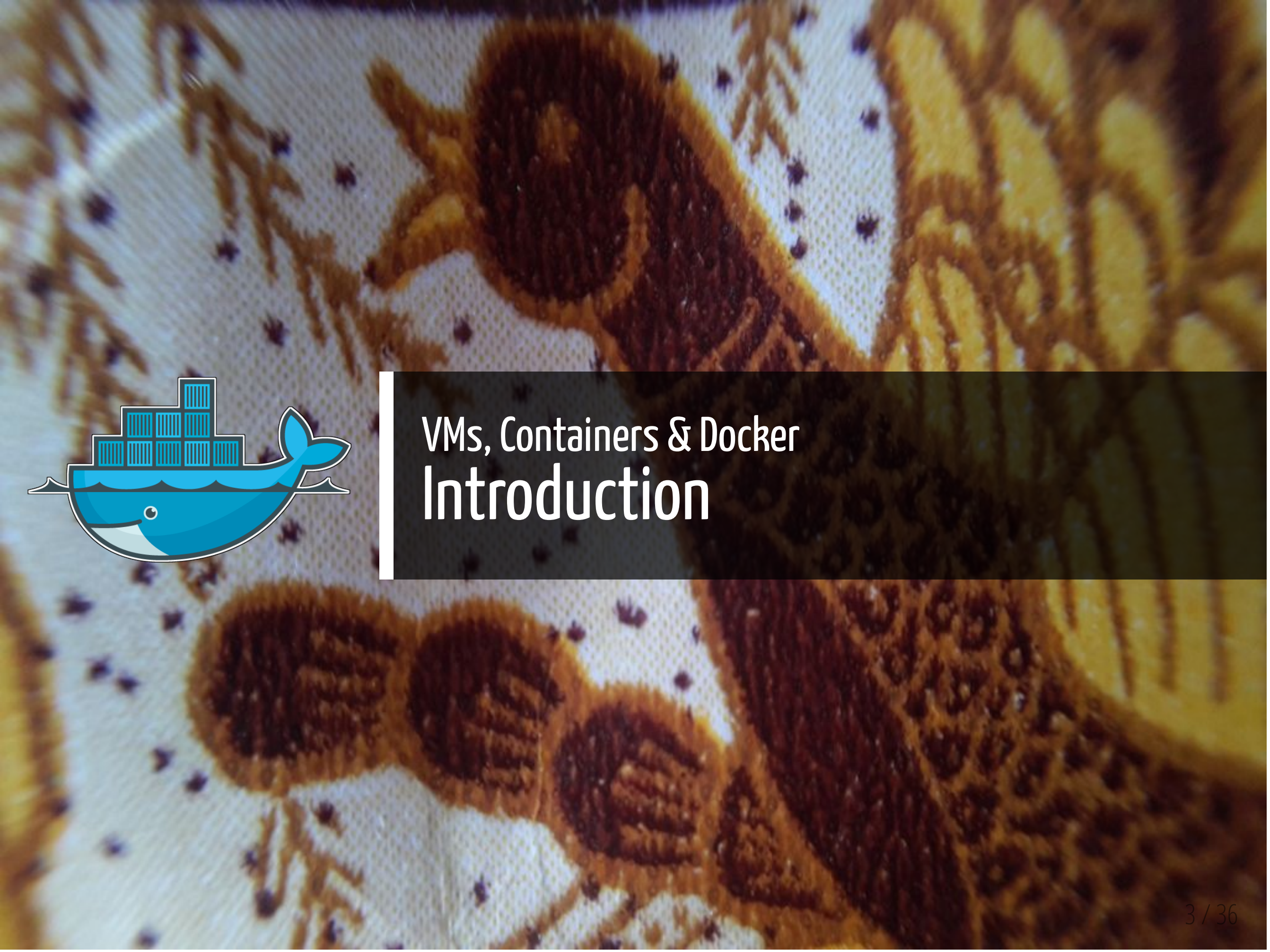
Docker Basics

Eueung Mulyana

http://eueung.github.io/docker-stuff/intro

Outline

- VMs, Containers, Docker
- Getting Started - Docker Engine
- Custom Images
- Docker Compose



VMs, Containers & Docker

Introduction

Virtual Machines vs. Containers

Containers have similar resource isolation and allocation benefits as virtual machines but a different architectural approach allows them to be much more portable and efficient.

Ref: docker.com

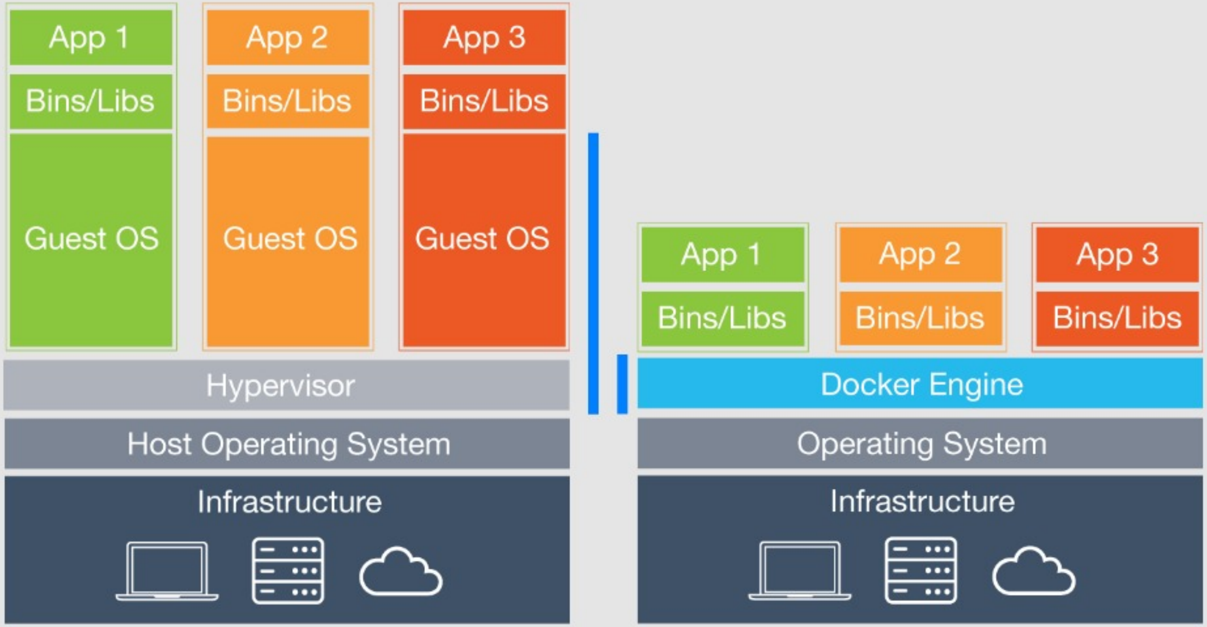
Virtual Machines

Each virtual machine includes the application, the necessary binaries and libraries and an entire guest operating system - all of which may be **tens** of GBs in size.

Containers

(Docker) Containers include the application and all of its dependencies, but **share** the kernel with other containers. They run as an isolated process in userspace on the host operating system. They're also not tied to any specific infrastructure.

(Docker) **Containers** running on a single machine all share the same operating system kernel so they start instantly and make more efficient use of RAM. Images are constructed from layered filesystems so they can share common files, making disk usage and image downloads much more efficient.


App 1
App 2
App 3
Bins/Libs
Bins/Libs
Bins/Libs
Guest OS
Guest OS
Guest OS
Hypervisor
Host Operating System
Infrastructure
App 1
App 2
App 3
Bins/Libs
Bins/Libs
Bins/Libs
Docker Engine
Operating System
Infrastructure

Containers

Container (lightweight process virtualization) technology is not new, mainstream support in the **vanilla kernel** however is, paving the way for widespread adoption (Linux Kernel 3.8 - released in February 2013 - cf. Rami Rosen).

FreeBSD has **Jails**, Solaris has **Zones** and there are other (Linux) container technologies: OpenVZ, VServer, Google Containers, LXC/LXD, Docker, etc.

Ref: Flockport

LXC

LXC owes its origin to the development of **cgroups** and **namespaces** in the Linux kernel to support lightweight virtualized OS environments (containers) and some early work by Daniel Lezcano and Serge Hallyn dating from 2009 at IBM.

The LXC Project provides tools to manage containers, advanced networking and storage support and a wide choice of minimal container OS templates. It is currently led by a 2 member team, Stephane Graber and Serge Hallyn from Ubuntu. The LXC project is supported by Ubuntu.

Docker

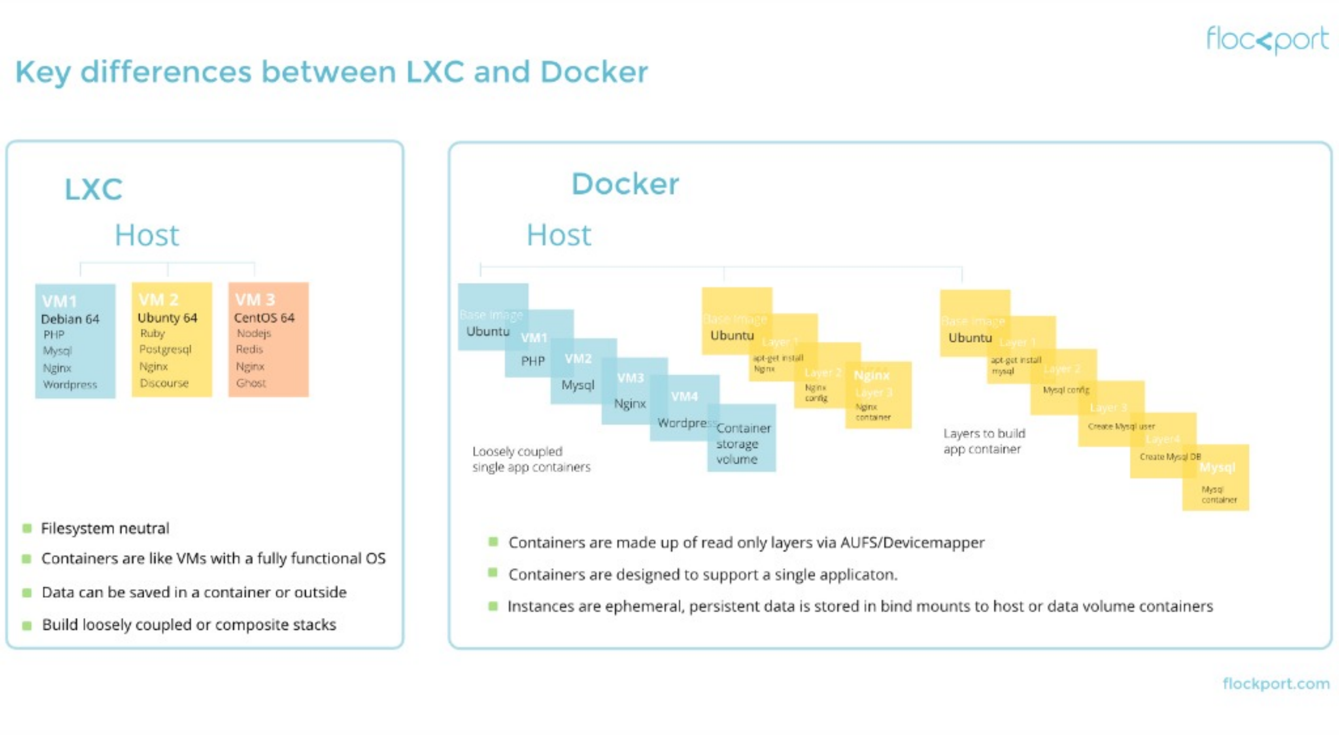
Docker is a project by **dotCloud** now **Docker Inc** released in March 2013, initially based on the LXC project to build **single** application containers. Docker has now developed their own implementation libcontainer that uses kernel namespaces and cgroups directly.

LXC vs. Docker

Ref: Flockport

Both **LXC** and **Docker** are userland container managers that use kernel namespaces to provide end user containers. We also now have **Systemd-Nspawn** that does the same thing.

The only difference is LXC containers have an an init and can thus run multiple processes and Docker containers do not have an init and can only run single processes.



flocport

Key differences between LXC and Docker

LXC

Host

VM1
Debian 64
PHP
Mysql
Nginx
Wordpress

VM 2
Ubunty 64
Ruby
Postgresql
Nginx
Discourse

VM 3
CentOS 64
Nodejs
Redis
Nginx
Ghost

- Filesystem neutral
- Containers are like VMs with a fully functional OS
- Data can be saved in a container or outside
- Build loosely coupled or composite stacks

Docker

Host

Base image
Ubuntu
VM1
PHP
VM2
Mysql
VM3
Nginx
VM4
Wordpress
Container
storage
volume

Loosely coupled
single app containers

Base image
Ubuntu
Layer 1
apt-get install
Nginx
Layer 2
Nginx
config
Nginx
Layer 3
Nginx
container

Base image
Ubuntu
Layer 1
apt-get install
mysql
Layer 2
Mysql config
Layer 3
Create Mysql user
Layer4
Create Mysql DB
Mysql
Mysql
container

Layers to build
app container

- Containers are made up of read only layers via AUFS/Devicemapper
- Containers are designed to support a single applicaton.
- Instances are ephemeral, persistent data is stored in bind mounts to host or data volume containers

flockport.com

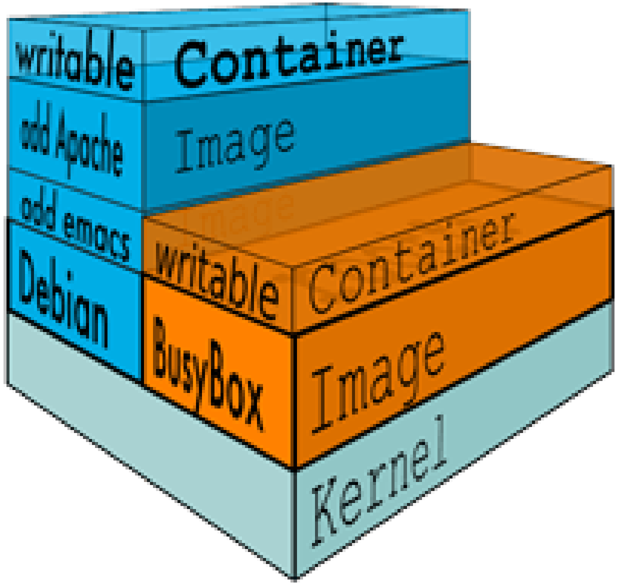
Docker

Docker allows you to package an application with all of its dependencies into a standardized unit for software development.

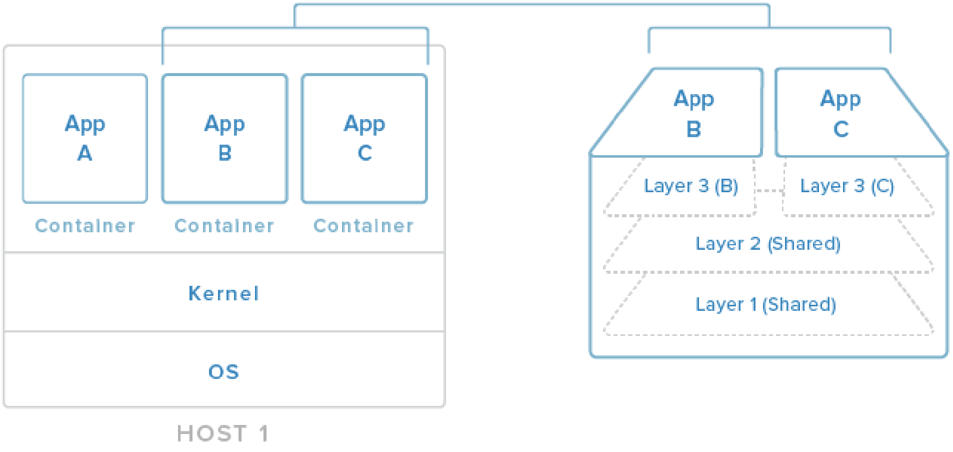
Docker containers wrap up a piece of software in a complete filesystem that contains **everything** it needs to run: code, runtime, system tools, system libraries - anything you can install on a server. This guarantees that it will always run the same, regardless of the environment it is running in.

Docker containers run on any computer, on any infrastructure and in any cloud.

Ref: docker.com



CONTAINER OVERVIEW



Containers isolate individual applications and use operating system resources that have been abstracted by Docker. Containers can be built by "layering", with multiple containers sharing underlying layers, decreasing resource usage.

Ref: Docker Ecosystem - DO

Docker

Typically, when designing an application or service to use Docker, it works best to break out functionality into individual containers, a design recently known as micro-service architecture.

This gives you the ability to easily scale or update components independently in the future.

Having this flexibility is one of the many reasons that people are interested in Docker for development and deployment.

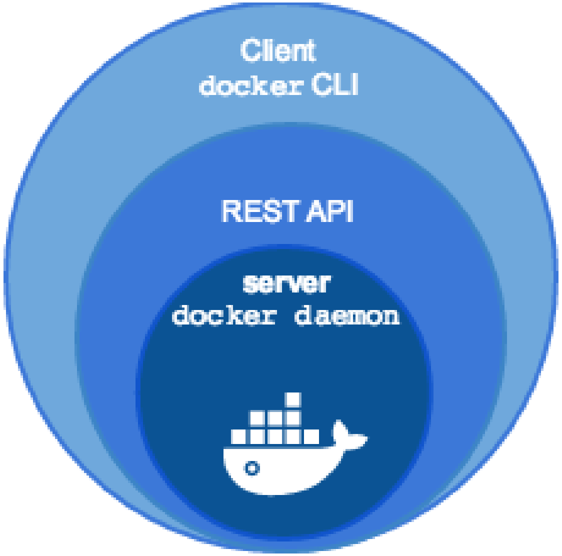
Ref: Docker Ecosystem - DO

Advantages

- **Lightweight** resource utilization: instead of virtualizing an entire operating system, containers isolate at the process level and use the host's kernel.
- **Portability**: all of the dependencies for a containerized application are bundled inside of the container, allowing it to run on any Docker host.
- **Predictability**: The host does not care about what is running inside of the container and the container does not care about which host it is running on. The interfaces are standardized and the interactions are predictable.

Docker Engine

Getting Started



Docker Engine

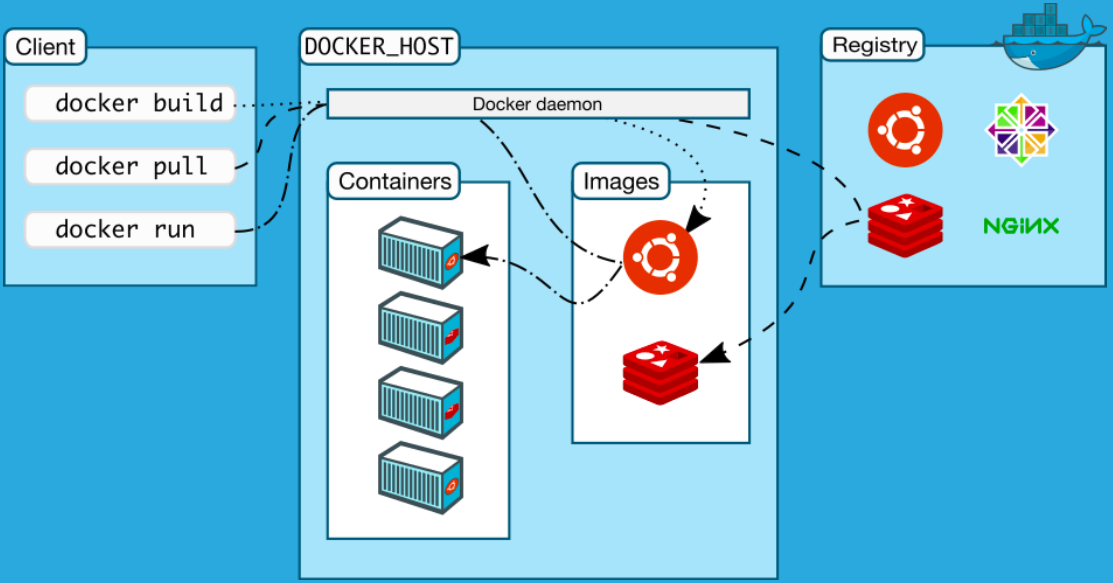
When people say "Docker" they typically mean Docker Engine, the client-server application made up of the Docker daemon, a REST API that specifies interfaces for interacting with the daemon, and a command line interface (CLI) client that talks to the daemon (through the REST API wrapper).

Docker Engine accepts docker commands from the CLI, such as `docker run <image>`, `docker ps` to list running containers, `docker images` to list images, and so on.

Engine is the core of Docker and nothing else will run without it.

Ref: docker.com

Docker Architecture



Docker Architecture

Docker uses a client-server architecture. The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers.

Both the Docker client and the daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon.

The Docker client and daemon communicate via sockets or through a RESTful API.

Docker daemon

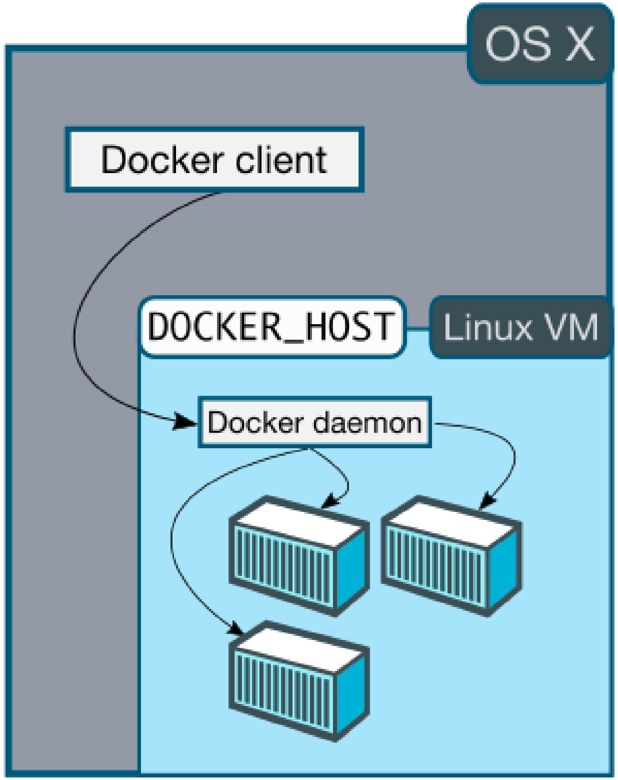
The Docker daemon runs on a host machine. The user does not directly interact with the daemon, but instead through the Docker client.

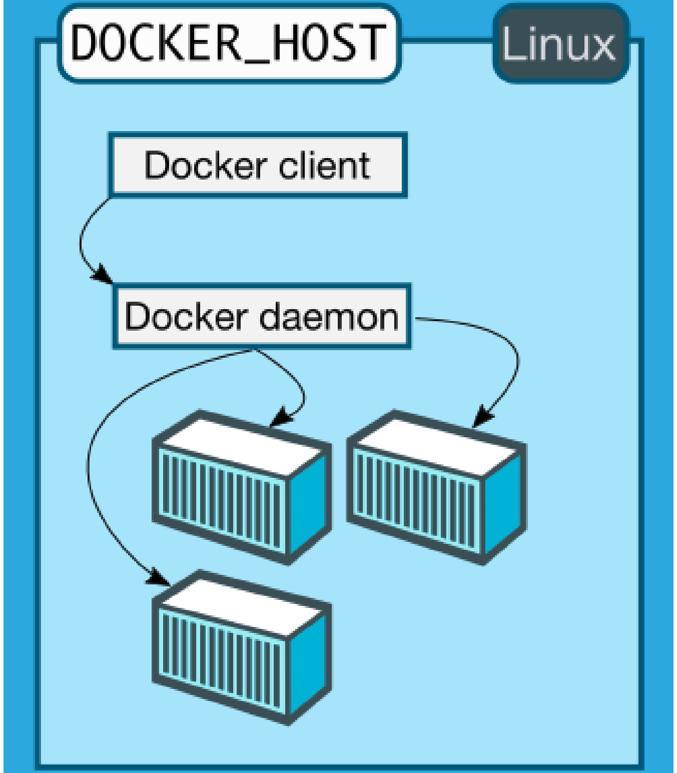
Docker client

The Docker client, in the form of the docker binary, is the primary user interface to Docker.

It accepts commands from the user and communicates back and forth with a Docker daemon.

Ref: docker.com


OS X
Docker client
DOCKER_HOST
Linux VM
Docker daemon


DOCKER_HOST
Linux
Docker client
Docker daemon

Let's Try It ...

My Case: amd64 Machine, Ubuntu 16.04

First Step

```
$ curl -fsSL https://get.docker.com/ | sh
$ docker info
Containers: 1
...
Images: 15
Server Version: 1.11.1
Storage Driver: aufs
...
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
...
Kernel Version: 4.4.0-21-generic
Operating System: Ubuntu 16.04 LTS
...
$ docker version

Client:
 Version:      1.11.1
 API version:  1.23
 Go version:   go1.5.4
 Git commit:   5604cbe
 Built:        Tue Apr 26 23:43:49 2016
 OS/Arch:      linux/amd64

Server:
 Version:      1.11.1
 API version:  1.23
 Go version:   go1.5.4
 Git commit:   5604cbe
 Built:        Tue Apr 26 23:43:49 2016
 OS/Arch:      linux/amd64
```

```
$ docker run hello-world

Hello from Docker.
This message shows that your installation appears to be wo

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from
 3. The Docker daemon created a new container from that im
    executable that produces the output you are currently
 4. The Docker daemon streamed that output to the Docker c
    to your terminal.

To try something more ambitious, you can run an Ubuntu con
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Doc
 https://hub.docker.com

For more examples and ideas, visit:
 https://docs.docker.com/userguide/
```

Ref: Quickstart, Install Docker

Try Some Commands

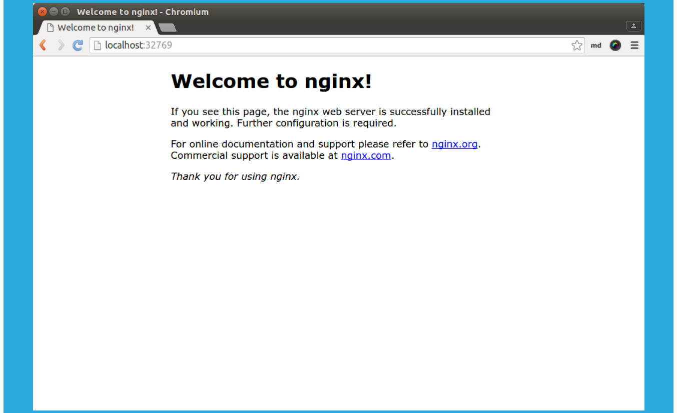
```
$ docker images
REPOSITORY                  TAG                 SIZE
em/notebook                 v1                  864.9 MB
ubuntu                      16.04               120.1 MB
alpine                      3.3                 4.798 MB
busybox                     latest              1.113 MB
firecyberice/whalesay       latest              47.25 MB
hello-world                 latest              960 B
docker/whalesay             latest              247 MB
```

```
$ JOB=$(docker run -d ubuntu /bin/sh -c "while true; do ec

$ docker stop $JOB
$ docker start $JOB
$ docker restart $JOB

$ docker kill $JOB

$ docker stop $JOB   # Container must be stopped to remove
$ docker rm $JOB
$ docker rm -f $JOB  # Running container
```

```
$ docker run --rm firecyberice/whalesay Hello Docker
 ______________
< Hello Docker >
 --------------
    \
     \
      \
                    ##         .
              ## ## ##        ==
           ## ## ## ## ##    ===
       /"""""""""""""""""\___/ ===
  ~~~ {~~ ~~~~ ~~~ ~~~~ ~~~ ~ /  ===- ~~~
       \______ o           __/
         \    \         __/
          \____\_______/

$ docker ps -a
CONTAINER ID        IMAGE               COMMAND
2f6f337530d5        hello-world         "/hello"
e71dbedafb57        em/notebook:v1      "tini -- jupyter n

$ docker rm -f 2f6f
2f6f
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND
e71dbedafb57        em/notebook:v1      "tini -- jupyter n

$ docker run -it --entrypoint /bin/sh firecyberice/whalesa
```

nginx

```
$ docker run -d -P --name web nginx
224a61ea84cfbf468bd090aebbd0ba534e9b07bb8e7e0068bfaeca1ba7

$ docker ps
CONTAINER ID        IMAGE               COMMAND
224a61ea84cf        nginx               "nginx -g 'daemon
e71dbedafb57        em/notebook:v1      "tini -- jupyter n

$ docker port web
443/tcp -> 0.0.0.0:32768
80/tcp -> 0.0.0.0:32769

$ docker stop web
$ docker rm web
```



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

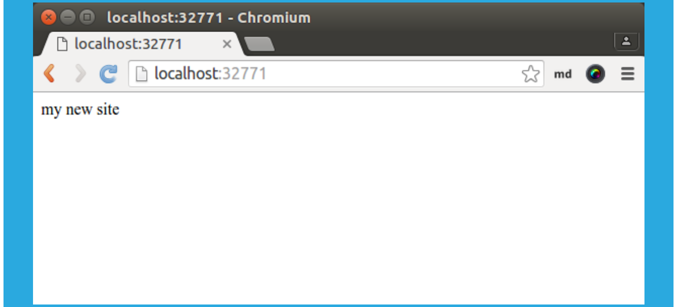
For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

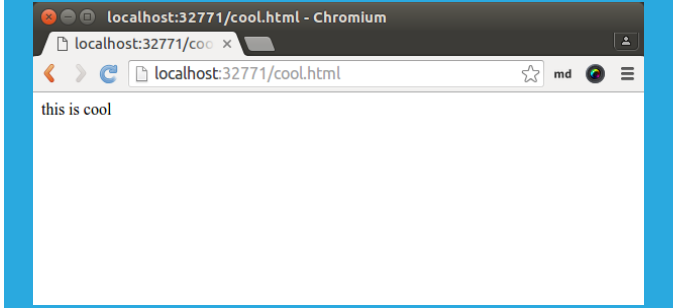
Mount a Volume on the Container

```
$ mkdir mysite && cd mysite
mysite$ echo "my new site" > index.html
mysite$ docker run -d -P -v $(pwd):/usr/share/nginx/html -
da01817c28bbdb2f3b71275ba7b9560da4e65f8716329c16787c831817

mysite$ docker port myweb
443/tcp -> 0.0.0.0:32770
80/tcp -> 0.0.0.0:32771

mysite$ echo "this is cool" > cool.html
$ docker stop myweb
$ docker rm myweb
```

Ref: docker.com



my new site


localhost:32771/cool.html - Chromium
localhost:32771/coo
localhost:32771/cool.html
md

this is cool

Custom Images

Build Custom Image

```
$ mkdir mydockerbuild && cd mydockerbuild/ && touch Docker
$ docker build -t docker-whale .
Sending build context to Docker daemon 2.048 kB
Step 1 : FROM docker/whalesay:latest
 ---> 6b362a9f73eb
Step 2 : RUN apt-get -y update && apt-get install -y fortu
 ---> Running in 7375f27597d7
...
Step 3 : CMD /usr/games/fortune -a | cowsay
 ---> Running in 09c57e3ebb83
 ---> 428cbace4310
Removing intermediate container 09c57e3ebb83
Successfully built 428cbace4310
```

Ref: Build your own image

Dockerfile

```
FROM docker/whalesay:latest
RUN apt-get -y update && apt-get install -y fortunes
CMD /usr/games/fortune -a | cowsay
```

```
$ docker run docker-whale
 ________________________________________
/ On the other hand, life can be an      \
| endless parade of TRANSSEXUAL QUILTING |
| BEES aboard a cruise ship to           |
\ DISNEYWORLD if only we let it!!        /
 ----------------------------------------
    \
     \
      \
                    ##        .
              ## ## ##       ==
           ## ## ## ##      ===
       /""""""""""""""""___/ ===
  ~~~ {~~ ~~~~ ~~~ ~~~~ ~~ ~ /  ===- ~~~
       \______ o          __/
        \    \        __/
          \____\______/
```

Example Dockerfile

docker/whalesay

```
FROM ubuntu:14.04

RUN apt-get update \
    && apt-get install -y cowsay --no-install-recommends \
    && rm -rf /var/lib/apt/lists/* \
    && mv /usr/share/cowsay/cows/default.cow /usr/share/co

# "cowsay" installs to /usr/games
ENV PATH $PATH:/usr/games

COPY docker.cow /usr/share/cowsay/cows/
RUN ln -sv /usr/share/cowsay/cows/docker.cow /usr/share/co

CMD ["cowsay"]
```

firecyberice/whalesay

```
FROM alpine:3.2

RUN apk update \
    && apk add git perl \
    && cd /tmp/ \
    && git clone https://github.com/jasonm23/cowsay.git \
    && cd cowsay && ./install.sh /usr/local \
    && cd .. \
    && rm -rf cowsay \
    && apk del git

ENV PATH $PATH
COPY docker.cow /usr/local/share/cows/

# Move the "default.cow" out of the way so we can overwrit
RUN \
    mv /usr/local/share/cows/default.cow /usr/local/share/
    && ln -sv /usr/local/share/cows/docker.cow /usr/local/

ENTRYPOINT ["cowsay"]
```

```
$ docker run --name myredis -it ubuntu:16.04 bash

root@ac6002b2a98b:/# apt-get update
root@ac6002b2a98b:/# apt-get install wget
root@ac6002b2a98b:/# apt-get install build-essential tcl8.

root@ac6002b2a98b:/# wget http://download.redis.io/redis-s
root@ac6002b2a98b:/# tar xzf redis-stable.tar.gz
root@ac6002b2a98b:/# cd redis-stable && make && make insta
root@ac6002b2a98b:/# ./redis-stable/utils/install_server.s
...
Selected config:
Port           : 6379
Config file    : /etc/redis/6379.conf
Log file       : /var/log/redis_6379.log
Data dir       : /var/lib/redis/6379
Executable     : /usr/local/bin/redis-server
Cli Executable : /usr/local/bin/redis-cli
Is this ok? Then press ENTER to go on or Ctrl-C to abort.
Copied /tmp/6379.conf => /etc/init.d/redis_6379
Installing service...
Success!
Starting Redis server...
Installation successful
```

Manual Process

test/myredis:v1

```
root@ac6002b2a98b:/redis-stable# ps ax | grep redis
root@ac6002b2a98b:/redis-stable# src/redis-cli
127.0.0.1:6379> set foo bar
OK
127.0.0.1:6379> get foo
"bar"
127.0.0.1:6379> exit

root@ac6002b2a98b:/redis-stable# exit

$ docker ps -a
$ docker commit -m "add redis" -a "em" myredis test/myredi
sha256:9b75a94f67cb47b012f445ed65fb13fc67d05a00ad1bb262d20

$ docker images | grep redis
test/myredis    v1    9b75a94f67cb    39 seconds ago    408.3
```

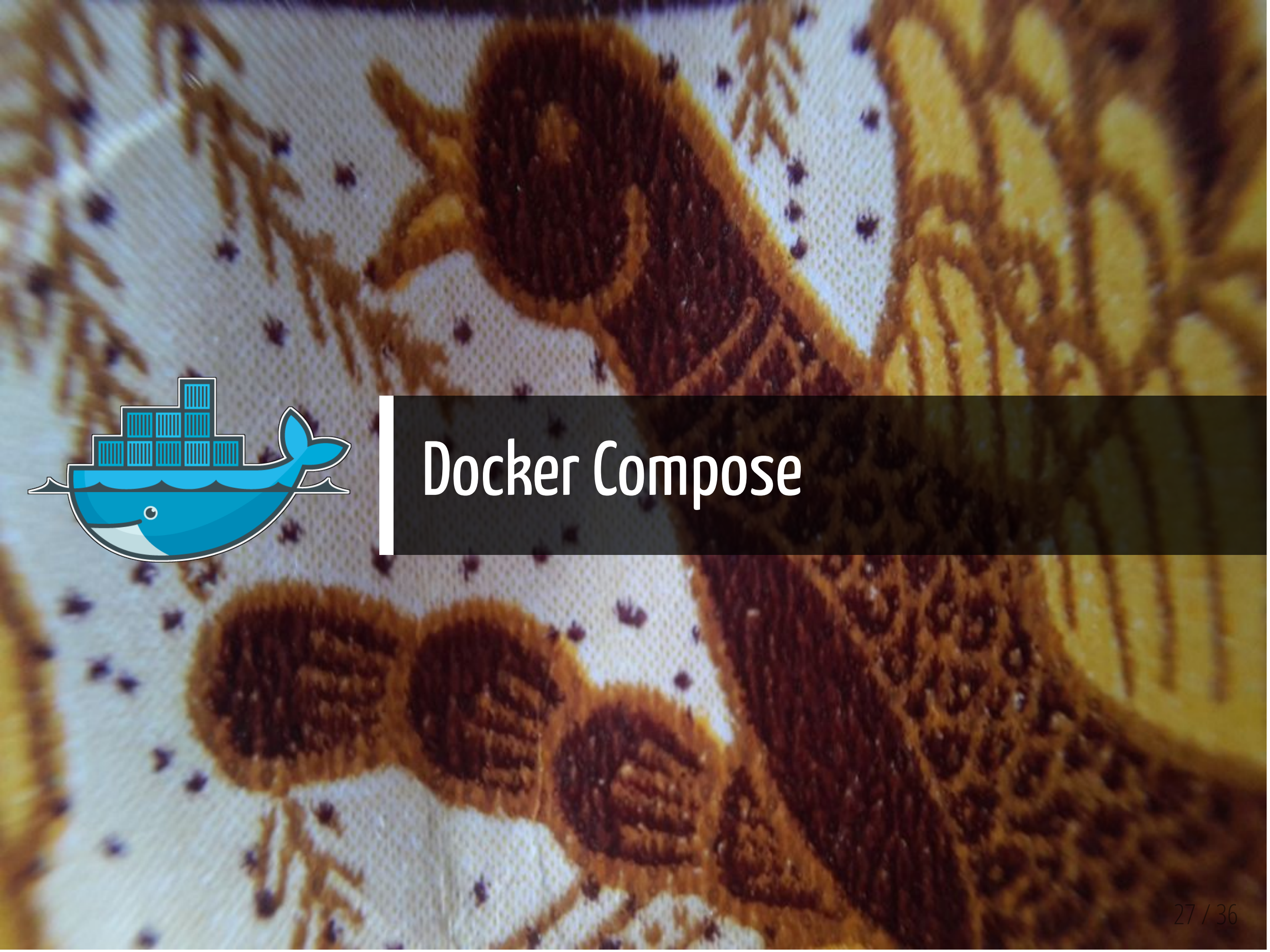
Ref: Getting Started with Docker

Dockerfile

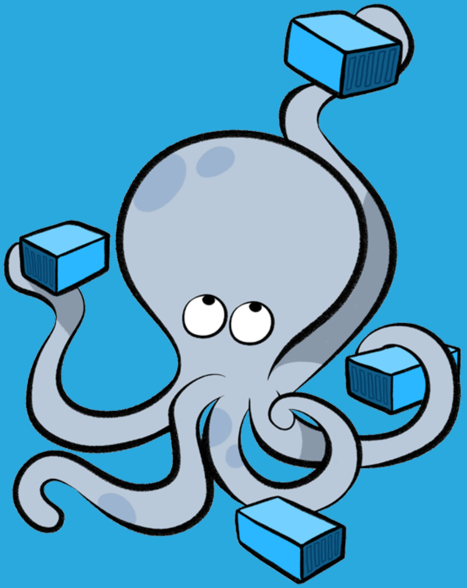
test/myredis:df

```
$ docker build -t test/myredis:df .

$ docker images | grep redis
test/myredis   df   9a450ae418d8   About a minute ago   40
test/myredis   v1   9b75a94f67cb   13 minutes ago       40

$ docker run -d -p 6379:6379 test/myredis:df
1240a12b56e4a87dfe89e4ca4400eb1cafde802ac0187a54776f9ea54b

$ docker ps
CONTAINER ID        IMAGE               COMMAND
1240a12b56e4        test/myredis:df          "redis-server

$ sudo apt-get install redis-tools
$ redis-cli
127.0.0.1:6379> set bat man
OK
127.0.0.1:6379> get bat
"man"
127.0.0.1:6379> quit
```

```
FROM ubuntu:16.04

RUN apt-get update
RUN apt-get install -y wget
RUN apt-get install -y build-essential tcl8.5

RUN wget http://download.redis.io/redis-stable.tar.gz
RUN tar xzf redis-stable.tar.gz
RUN cd redis-stable && make && make install
RUN ./redis-stable/utils/install_server.sh

EXPOSE 6379
ENTRYPOINT  ["redis-server"]
```



Docker Compose



Docker Compose

Compose is a tool for defining and running **multi-container** Docker applications. With Compose, you use a Compose file to configure your application's services. Then, using a single command, you create and start all the services from your configuration.

Using Compose is basically a three-step process:

1. Define your app's environment with a **Dockerfile** so it can be reproduced anywhere.
2. Define the services that make up your app in **docker-compose.yml** so they can be run together in an isolated environment.
3. Lastly, run **docker-compose up** and Compose will start and run your entire app.

Ref: Overview of Docker Compose

Docker Compose

Getting Started

```
$ curl -L https://github.com/docker/compose/releases/downl
$ chmod +x /usr/local/bin/docker-compose
```

```
$ docker-compose version
docker-compose version 1.7.0, build 0d7bf73
docker-py version: 1.8.0
CPython version: 2.7.9
OpenSSL version: OpenSSL 1.0.1e 11 Feb 2013
```

Getting Started

Step #1

app.py

```
from flask import Flask
from redis import Redis

app = Flask(__name__)
redis = Redis(host='redis', port=6379)

@app.route('/')
def hello():
    redis.incr('hits')
    return 'Hello World! I have been seen %s times.' % red

if __name__ == "__main__":
    app.run(host="0.0.0.0", debug=True)
```

requirements.txt

```
flask
redis
```

Ref: Getting Started

Dockerfile

```
FROM python:2.7
ADD . /code
WORKDIR /code
RUN pip install -r requirements.txt
CMD python app.py
```

```
$ docker build -t web .
$ docker images | grep web
web    latest    d6f25a9bf632    2 minutes ago    667.7 MB
```

Getting Started

Step #2

Getting Started

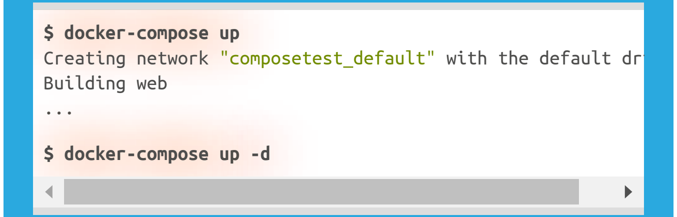
Step #3

docker-compose.yml

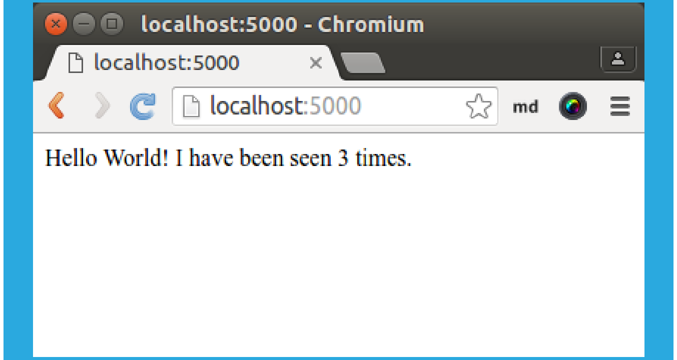
```yaml
version: '2'
services:
  web:
    build: .
    ports:
     - "5000:5000"
    volumes:
     - .:/code
    depends_on:
     - redis
  redis:
    image: redis
```

Getting Started

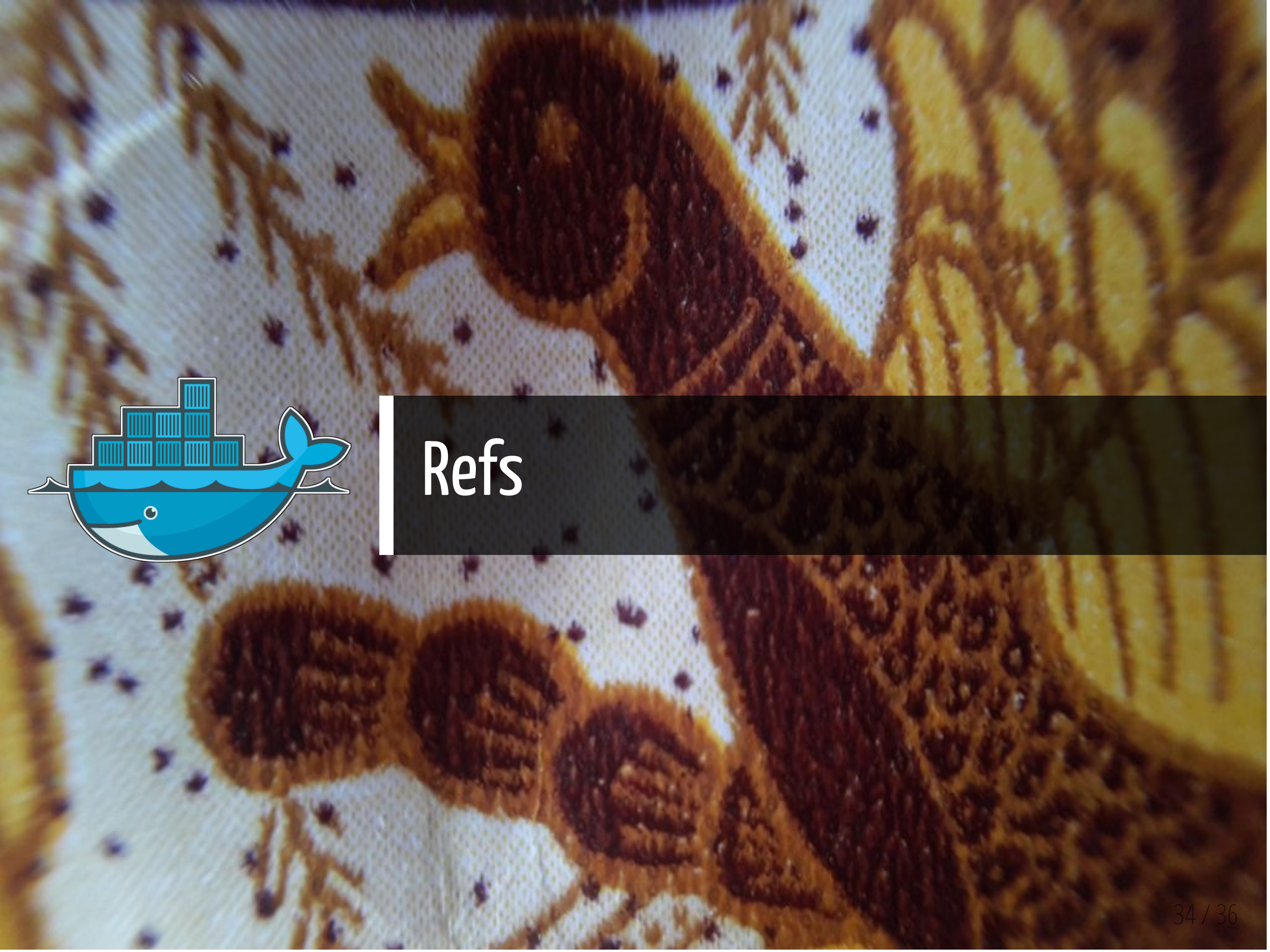
Step #4



```
$ docker-compose up
Creating network "composetest_default" with the default dr
Building web
...

$ docker-compose up -d
```



Hello World! I have been seen 3 times.



Refs

Refs

1. Docker Introduction
2. Docker - Documentation
3. Docker Ecosystem - Digital Ocean
4. LXC vs. Docker - Flockport
5. CLIs Reference docker ps
6. Open Container Initiative
7. Getting Started with Docker
8. Docker Compose - Getting Started

END

Eueung Mulyana

http://eueung.github.io/docker-stuff/intro

CodeLabs |