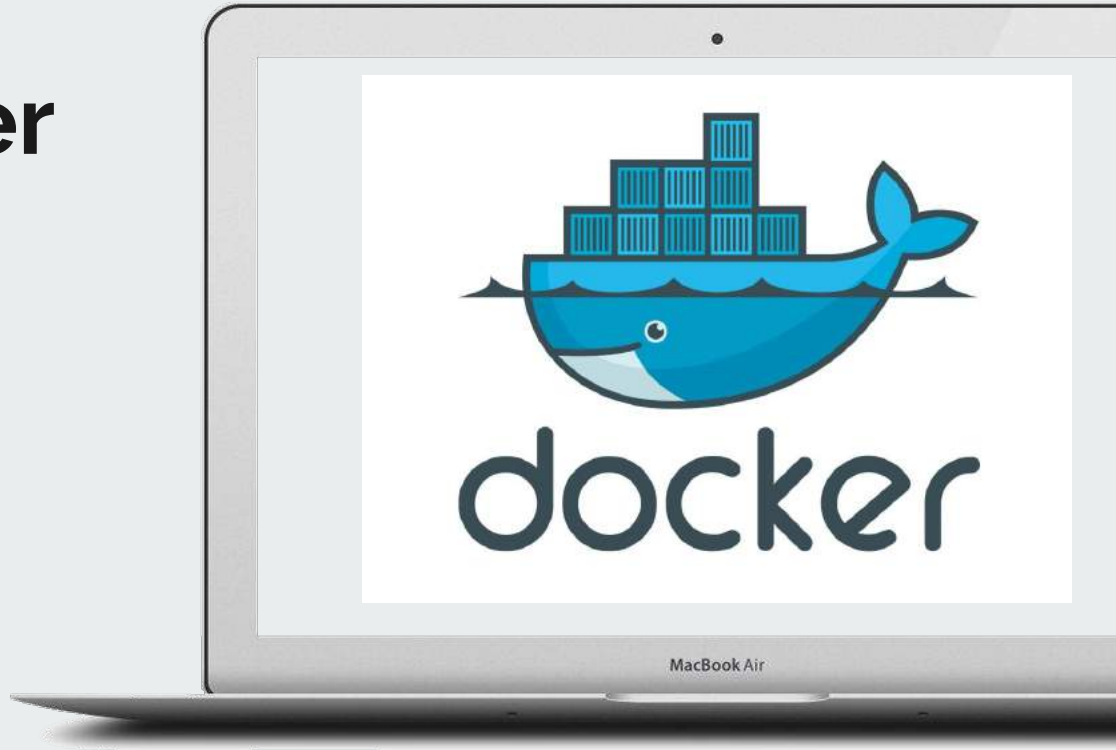
Smooth DevOps

Intro to Docker & Containers

A practical workshop

by
Naomi Pentrel



@naomi_pen

I’m Naomi

@naomi_pen

Who is this for?

- No prior knowledge of docker & containers needed
- You will need to be able to work on the commandline
  - not much experience with this? pair up!

Workshop Goals

- Understand Docker and Containers
- Download and run our first container
- Build and run a container from scratch
- Try Docker Compose

ACTIVITY

Activities

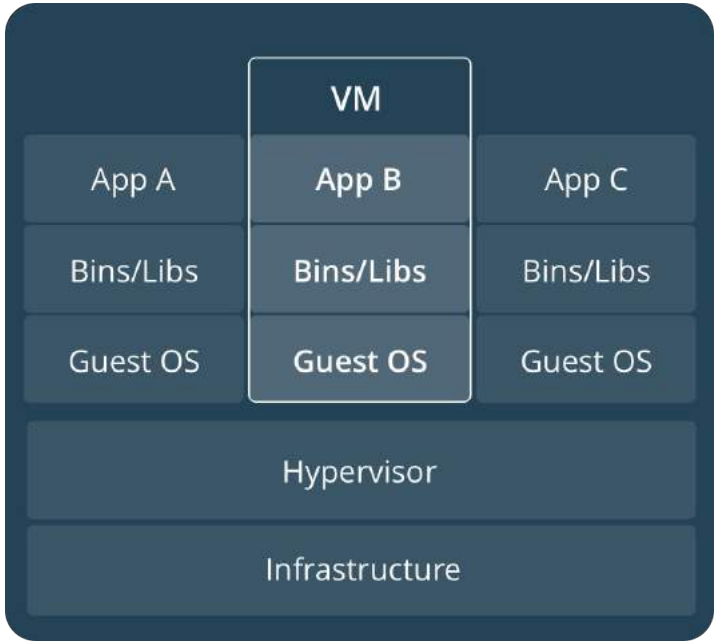
Whenever you are expected to do something, the slide will have an activity badge on the top left corner! Like here!

While we're here, let's get to know someone new in this room!

1. Intro to Docker & Containers
2. Setting up Docker
3. Running your First Container
4. Web Apps with Docker
5. Docker Compose

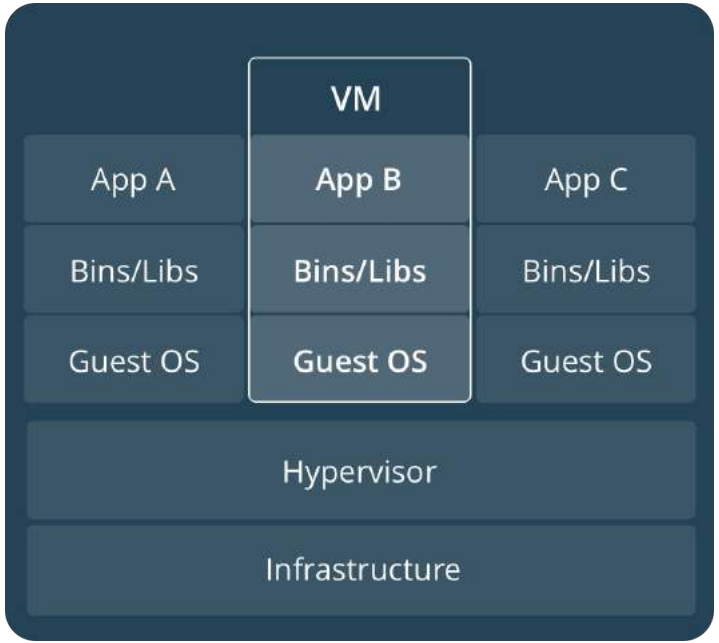
What is a VM?

A Virtual Machine is an Operating System installed on software which imitates dedicated hardware



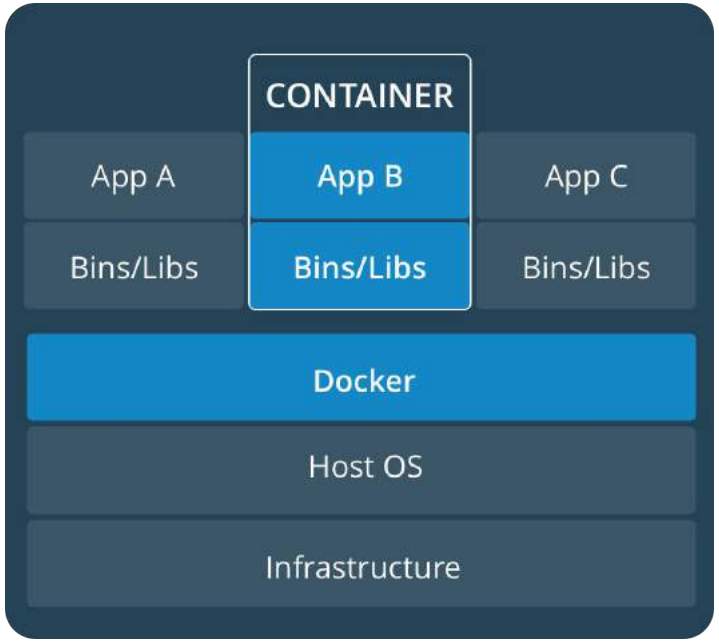
Why VMs?

They allow you to maximize resource utilization by allowing you to run multiple VMs on a single machine



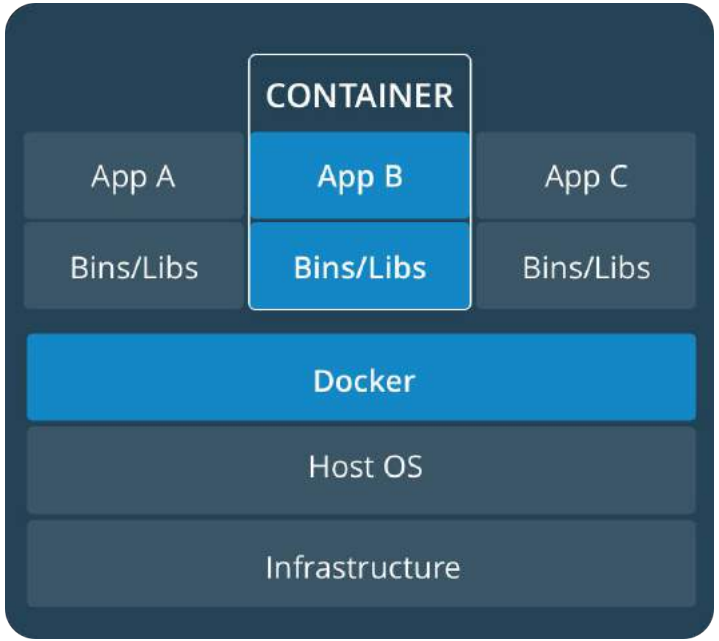
What is a Container?

A **Container** packages code and dependencies together creating an abstracted, isolated, and portable app



Why Containers?

- They improve resource utilization
- They facilitate modularity, portability, and simplicity when provisioning
- Fast to provision



Why Containers for me?

- **For devs:** fast paced, iterative development and testing without environment-specific bugs

- **For ops:**
  - less focus on dependency management and application specific configuration
  - More on runtime tasks (logging, monitoring...)

Containers are not VMs

While they share some characteristics, Virtual Machines and Containers are very different under the hood.

Major similarities:

- Both provide isolated environment for applications to run inside
- Both are easily movable between different hosts

Think Houses vs. Apartments

House = Virtual Machine (VM)

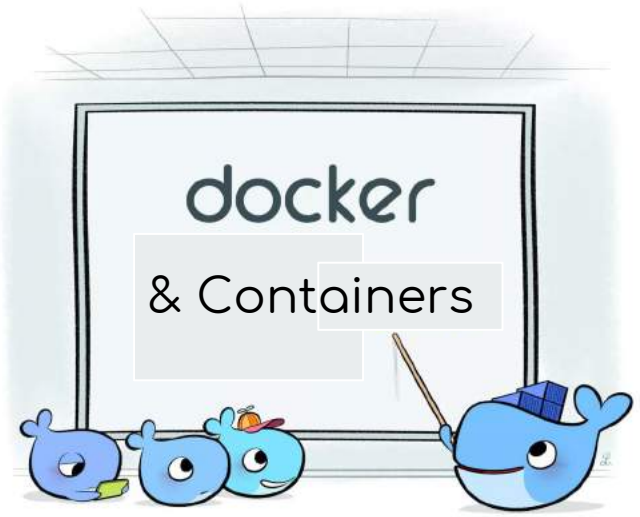
- Each VM has a full copy of the OS with dedicated resources
- Each new VM creates a full copy of this environment
- Virtualization technology
- Shares CPU, storage, RAM,...

Apartment = Container

- Contains exactly what they need to run their application
- Application delivery technology
- Shares OS resources: kernel, filesystem, process tree, network stack,...
- Immutable, not persistent (for persistent data volumes should be used)

What is Docker?

Docker is a platform that helps developers **build, ship, and run any application, in any environment**



Build, Ship, Run - Any App Anywhere

An app written on your laptop will **run exactly the same anywhere:**

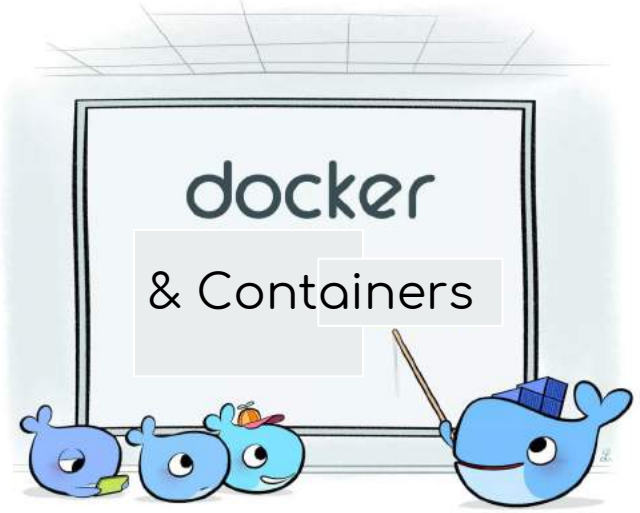
- Your Friend's Laptop
- Bare Metal Servers
- The Cloud
- A Raspberry Pi

Linux Containers on Windows and Mac?

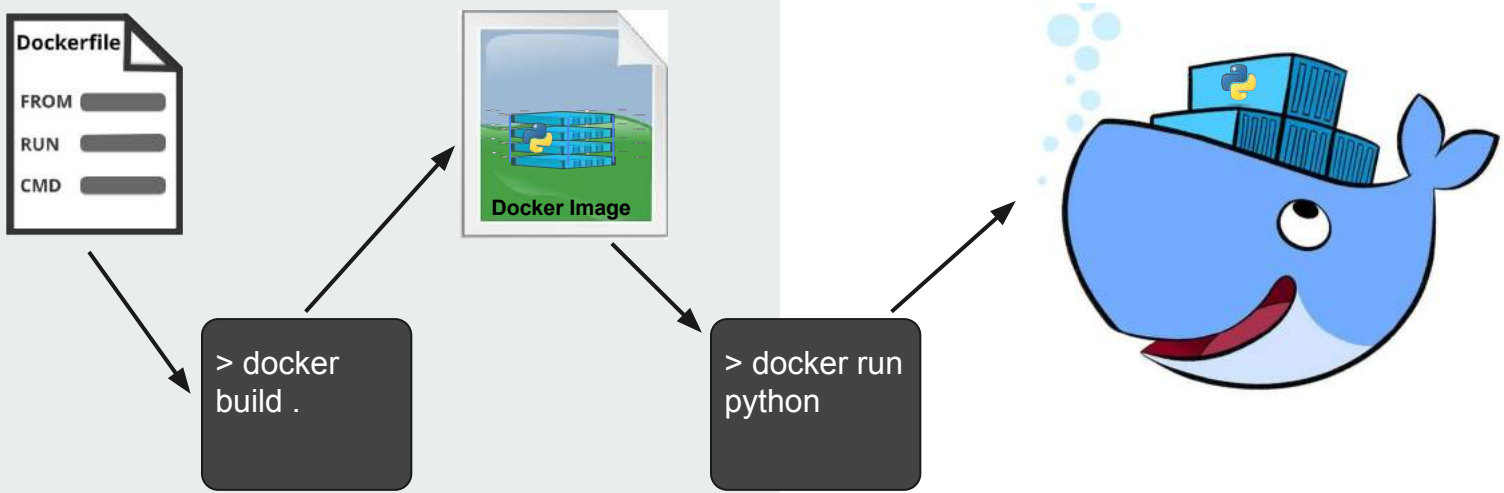
- Docker uses a Linux VM to run Linux containers
- the Linux VM is run using
  - xhyve for Mac or
  - Hyper-V - a Windows-native hypervisor
- On older Windows versions Docker Toolbox installs VirtualBox (a different hypervisor)

Why Docker?

- Docker is managed with a common toolset
- Docker runs on Windows, Mac, and Linux

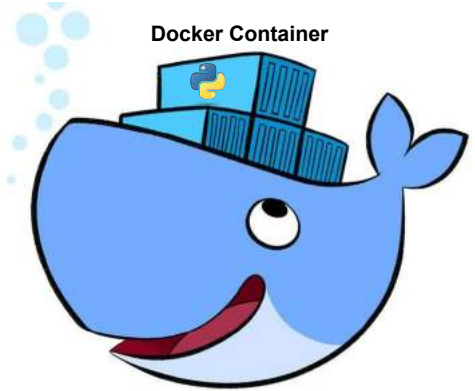


Terms: From Dockerfile to Container



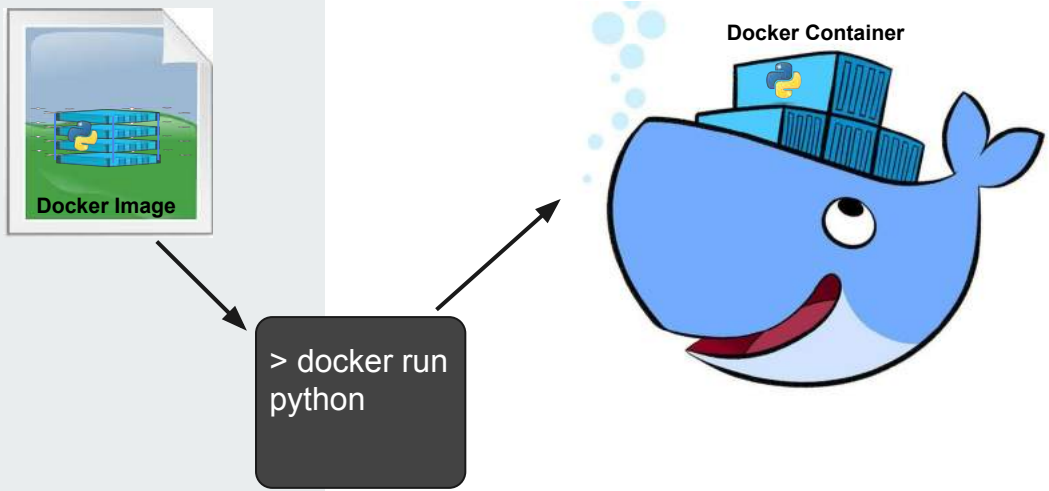
Docker Container -

Packaged code and dependencies running in an isolated portable environment

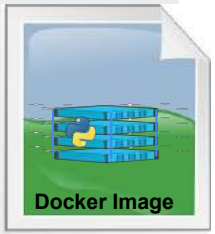


Docker Image -

Blueprint of an application that forms the basis of a container



Docker Image vs Docker Container?



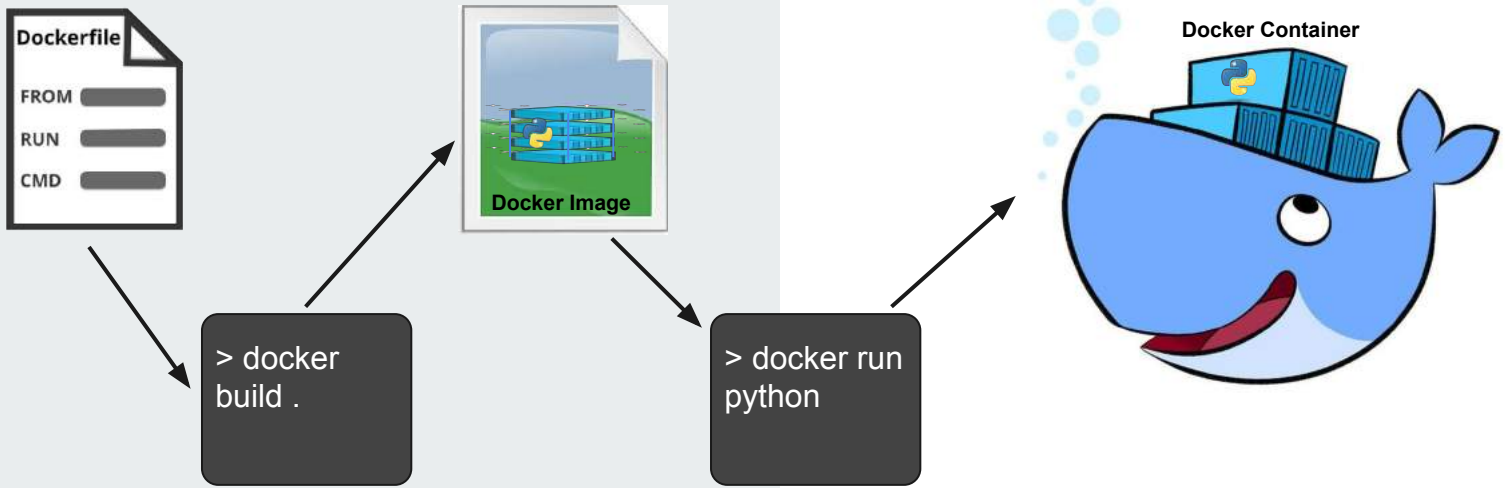
Think of it as a snapshot or a blueprint
from which a container can be built

Ceci n'est pas un container.

@naomi_pen

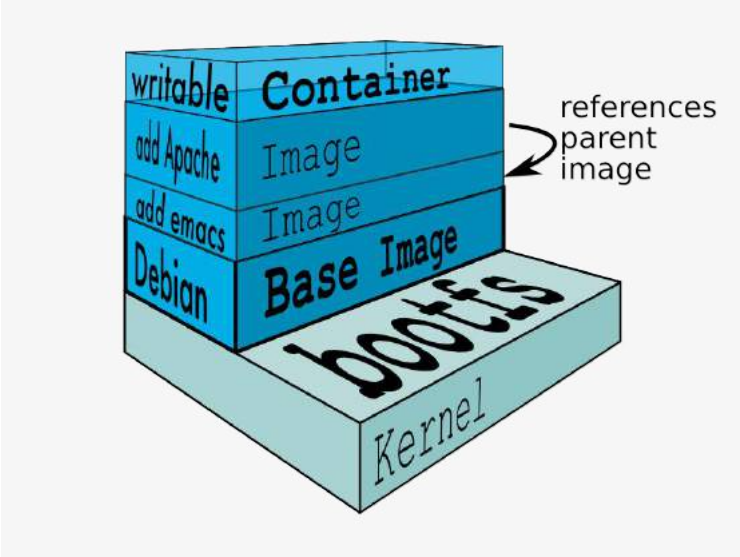
Dockerfile -

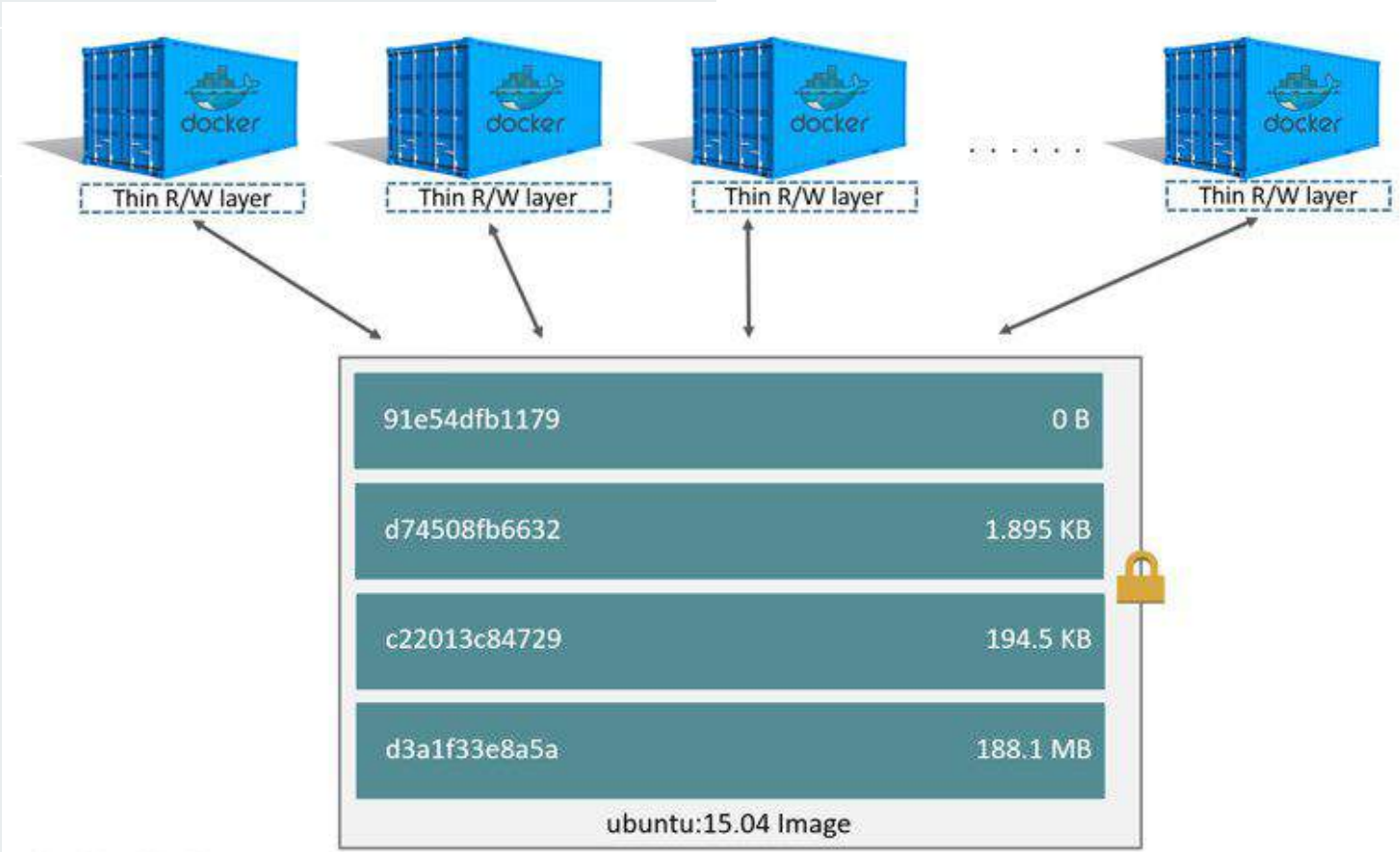
A file on disk that contains instructions for creating an image.



Docker Images: Layers

- An image is made up of layers
- Each layer is a read-only filesystem
- At runtime the Docker engine adds a read-write filesystem on top




docker
docker
docker
docker
Thin R/W layer
Thin R/W layer
Thin R/W layer
Thin R/W layer
91e54dfb1179 0 B
d74508fb6632 1.895 KB
c22013c84729 194.5 KB
d3a1f33e8a5a 188.1 MB
ubuntu:15.04 Image

Docker Engine

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)

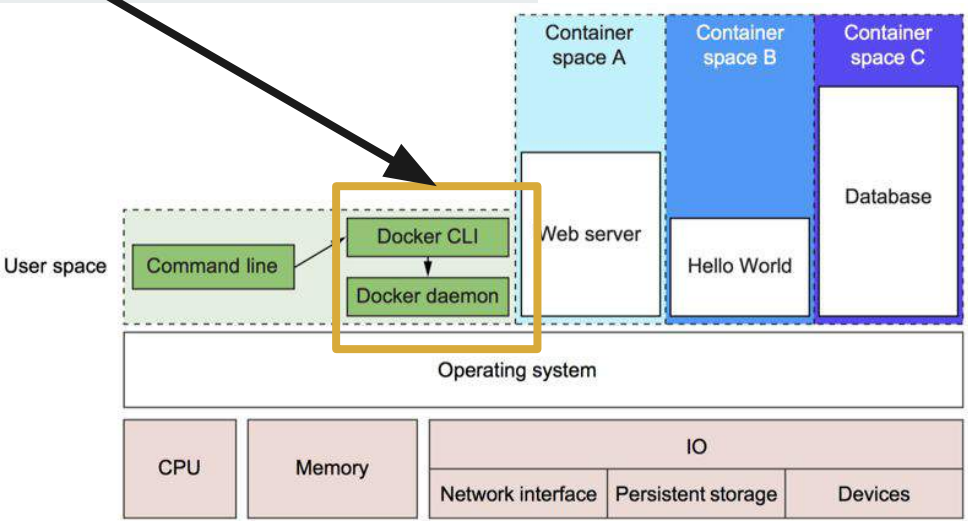


Figure 1.2 Docker running three containers on a basic Linux computer system

Docker daemon

- service that runs on your host operating system
- responsible for creating, running, and monitoring containers, as well as building and storing images
- exposes a REST API

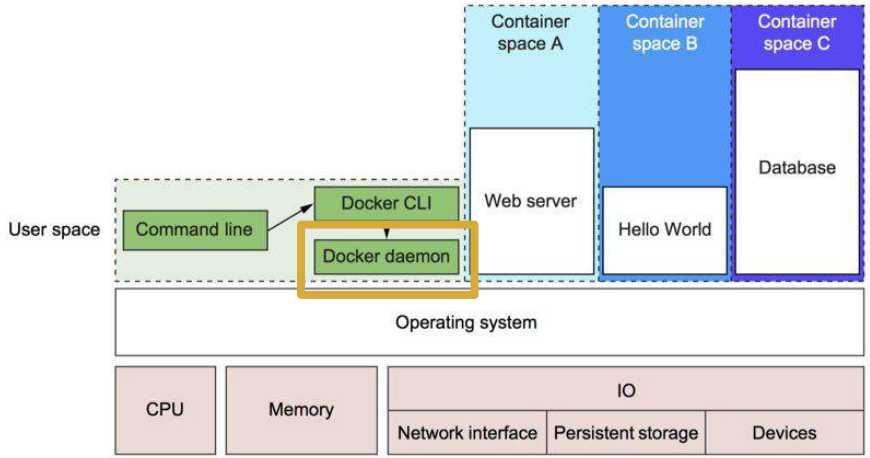


Figure 1.2 **Docker running three containers on a basic Linux computer system**

Docker CLI

- accepts docker commands
- uses the REST API to talk to the daemon

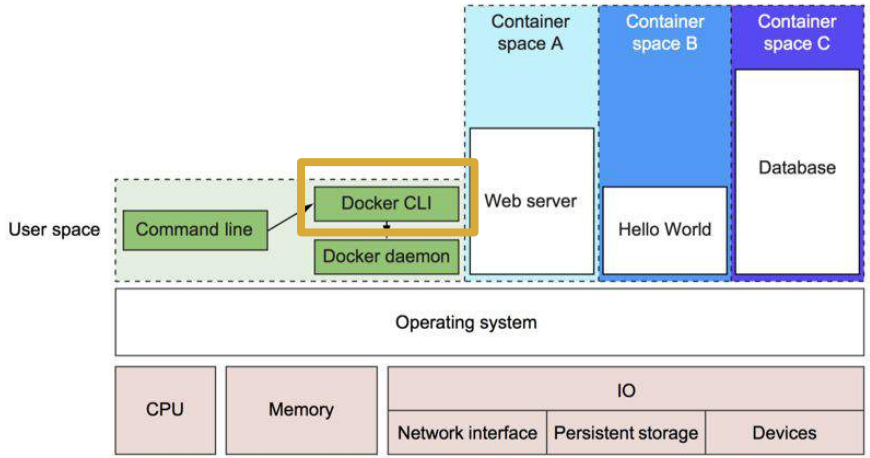


Figure 1.2 **Docker running three containers on a basic Linux computer system**

Image Registries

Where do images live?

In **Image Registries**. You can think of a registry as a directory of all available Docker images.

Examples:

- Docker Hub,
- Google Container Registry,
- Azure Container Registry,
- Private Registries,...

1. Intro to Docker & Containers
2. **Setting up Docker**
3. Running your First Container
4. Web Apps with Docker
5. Docker Compose

ACTIVITY

Use a Docker Playground!

http://bit.ly/dockerplayground

Stuck? *Raise your hand & someone will come help you!*

When installing Docker locally

On Linux machines, the docker user is equivalent to root!

Make sure to restrict access the same way you would for root

1. Intro to Docker & Containers
2. Setting up Docker
3. **Running your First Container**
4. Web Apps with Docker
5. Docker Compose

ACTIVITY

Test Docker

Test your Docker installation by running the following in a command prompt (Terminal, PowerShell, etc):

```
→ docker run hello-world
```

Running your 1st Container

OUTPUT

```
→ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

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 the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash
...
```

Let's Run a more interesting Container!

We're going to run Alpine Linux!

- Alpine Linux is a lightweight Linux distribution
- We can get a **Docker Image** containing Alpine Linux from **Docker Hub**

ACTIVITY

Pull the Alpine Linux Image

The pull command fetches the Alpine Linux image from the Docker registry **(Docker Hub)** and saves it in our system.

```
→ docker pull alpine
```

Running your 1st Container

OUTPUT

```
→ docker pull alpine

Using default tag: latest
latest: Pulling from library/alpine
ff3a5c916c92: Pull complete
Digest: sha256:7df6db5aa61ae9480f52f0b3a06a140ab98d427f86d8d5de0bedab9b8df6b1c0
Status: Downloaded newer image for alpine:latest
```

ACTIVITY

Review the List of Local Docker Images

You can use the **docker images** command to see a list of all images on your system.

```
→ docker images
```

Running your 1st Container

OUTPUT

```
→ docker images

REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
alpine              latest              3fd9065eaf02        4 months ago        4.15MB
hello-world         latest              e38bc07ac18e        7 weeks ago         1.85kB
```

ACTIVITY

Run a command on a container

Let’s now run a Docker container based on this image.

```
→ docker run alpine ls -l
```

Running your 1st Container

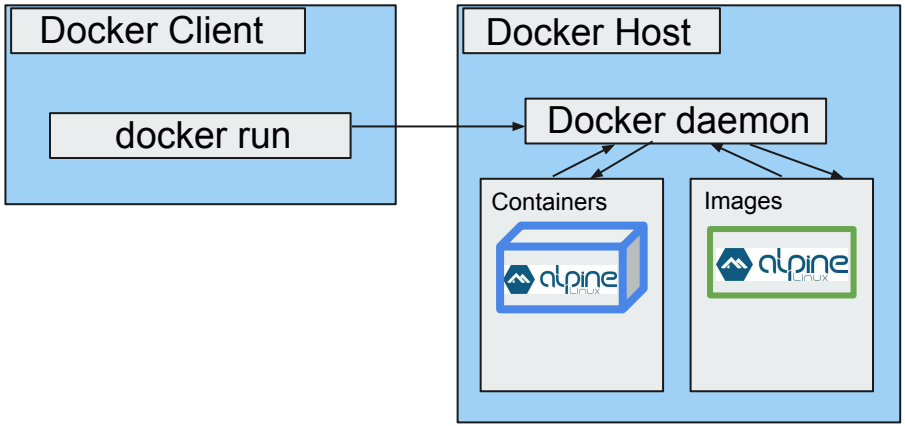
OUTPUT

```
→ docker run alpine ls -l

total 52
drwxr-xr-x    2 root     root          4096 Jan  9 19:37 bin
drwxr-xr-x    5 root     root           340 May 31 22:45 dev
drwxr-xr-x    1 root     root          4096 May 31 22:45 etc
drwxr-xr-x    2 root     root          4096 Jan  9 19:37 home
drwxr-xr-x    5 root     root          4096 Jan  9 19:37 lib
drwxr-xr-x    5 root     root          4096 Jan  9 19:37 media
drwxr-xr-x    2 root     root          4096 Jan  9 19:37 mnt
dr-xr-xr-x  178 root     root             0 May 31 22:45 proc
drwx------    2 root     root          4096 Jan  9 19:37 root
drwxr-xr-x    2 root     root          4096 Jan  9 19:37 run
drwxr-xr-x    2 root     root          4096 Jan  9 19:37 sbin
drwxr-xr-x    2 root     root          4096 Jan  9 19:37 srv
dr-xr-xr-x   13 root     root             0 May 31 22:45 sys
```

What just happened?

```
→ docker run alpine ls -l
```

1. CLI asked the daemon to find alpine
2. daemon created a new container with that image
3. daemon executed the provided command in the container



ACTIVITY

Let's Try Some Other Commands!

For example:

```
→ docker run alpine cal
```

```
→ docker run alpine echo "hello world"
```

```
→ docker run alpine pwd
```

```
→ docker run alpine id -u -n
```

***Note**: Containers are fast. Using a vm to run just one command would take a lot longer.*

ACTIVITY

Let's get Interactive!

All of our commands have exited immediately after running them.
How do we run a container interactively?

```
→ docker run -it alpine /bin/sh
```

Running your 1st Container

OUTPUT

```
→ docker run -it alpine /bin/sh

/ # uname -a
Linux ec71b23e9094 4.9.87-linuxkit-aufs #1 SMP Wed Mar 14 15:12:16 UTC 2018 x86_64 Linux

/ # ls
bin    dev    etc    home   lib    media  mnt    proc   root   run    sbin   srv    sys
tmp    usr    var

/ # cd /usr

/usr # ls
bin    lib    local  sbin   share

/usr # exit
```

SMOOTHIE BREAK

1. Intro to Docker & Containers
2. Setting up Docker
3. Running your First Container
4. **Web Apps with Docker**
5. Docker Compose

Let's Run a Web App with Docker

Time for the good stuff - deploying web applications with Docker! We're going to make a website that looks like this:

ACTIVITY

Let's Run the Image from Docker Hub.

We created an image on Docker Hub that contains the code under **npentrel/smooth-docker:1.0.**

```
→ docker run --name static-site -d -p 8888:9000 npentrel/smooth-docker:1.0
```

Note: The -d flag enables detached mode which detaches the running container from the terminal. The -p flag publishes the website on port 8888

OUTPUT

```
→ docker run --name static-site -d -p 8888:9000 npentrel/smooth-docker:1.0
Unable to find image 'npentrel/smooth-docker:1.0' locally
1.0: Pulling from npentrel/smooth-docker:1.0
ff3a5c916c92: Pull complete
44014a6ad6bc: Pull complete
9e372a7142ef: Pull complete
3ab6d28ced3c: Pull complete
27f34cba021a: Pull complete
3e4e7e551d07: Pull complete
c7166274d0c0: Pull complete
d8e5fc13aeb5: Pull complete
9c665538eb3c: Pull complete
Digest: sha256:8701330992fa5d64f3f5aa68f160ff0cbbf00fb639104e5b2b3b57b6bac0015e
Status: Downloaded newer image for npentrel/smooth-docker-v1:1.0
499bc62640275c995c6b0cbc077d680d8a411dfb7b5b4b45cadb6cafe4791725
```

So, What just happened?

Notice that we didn't run **docker pull** before we ran the new image. Here's what happened:

```
→ docker run --name static-site -d -p 8888:9000 npentrel/smooth-docker:1.0
```

1. Locate the requested image *(not found locally in this case)*
2. Download the missing image from Docker Hub
3. Create a new container using the requested image
4. Name the container using the --name parameter
5. Run it in the background *(detached)* and return to the prompt
6. Expose port 8888 and link it to the container's internal port 9000

ACTIVITY

Try Visiting your Website!

Head over to the following port and you should see the website below:

8888

ACTIVITY

How do we Stop a Detached Image?

We can use docker stop and docker rm to stop and remove a Docker Image that is running in the background.

```
→ docker ps

CONTAINER ID        IMAGE                          COMMAND
499bc6264027        npentrel/smooth-docker:1.0     "python /usr/src/app…"

CREATED             STATUS              PORTS                          NAMES
26 minutes ago      Up 26 minutes       0.0.0.0:8888->9000/tcp         static-site

→ docker stop 499bc6264027
499bc6264027

→ docker rm 499bc6264027
499bc6264027
```

Note: The example above provides the CONTAINER ID on our system; you should use the value that you see in your terminal.

ACTIVITY

Let's Clean Up!

Since we started this process with a name we don't need to use docker ps to figure out the ID.

```
→ docker stop static-site
static-site

→ docker rm static-site
static-site
```

ACTIVITY

Let's Make a Docker Image!

We've run other people's images so far, how do we create our own? Let's find out by creating the Smoothie app from scratch with your favorite smoothie in it.

ACTIVITY

Get the Source Code:

```
→ git clone https://github.com/npentrel/smooth-docker.git
→ cd smooth-docker
→ git checkout v1
```

See the site running live:

smooth-docker-v1.herokuapp.com/

The code for the website we'll run is here:

github.com/npentrel/smooth-docker/tree/v1

app.py

```python
from flask import Flask, render_template

app = Flask(__name__, static_url_path='')

## Feel free to change this!
smoothie = {
    '_id': 1,
    'name': 'Layered Rainbow Smoothies',
    'img': 'http://www.bestofvegan.com/wp-content/uploads/2017/05/layered-rainbow-smoothies-by-@artrawpaulina-1st-layer-banana-mango-spirulina-2nd-layer-banana-mango-.jpg',
    'source': 'http://www.bestofvegan.com/vegan-recipe-285/'
}

@app.route('/')
def index():
    return render_template("index.html", smoothie=smoothie)

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=9000)
```

github.com/npentrel/smooth-docker/tree/v1

requirements.txt

```
Flask==0.10.1
```

github.com/npentrel/smooth-docker/tree/v1

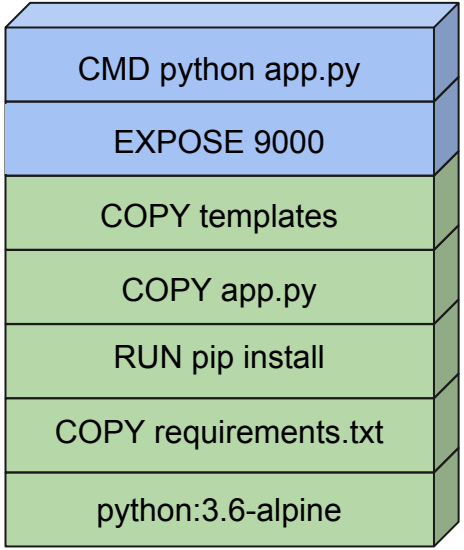
templates/index.html

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <link rel="stylesheet"
href="//netdna.bootstrapcdn.com/bootstrap/4.1.0/css/bootstrap.min.css">
  <link href="//netdna.bootstrapcdn.com/font-awesome/4.0.3/css/font-awesome.css"
rel="stylesheet">
  <style>
  body {
     background-image: linear-gradient(
      rgba(255, 255, 255, 0.45),
      rgba(255, 255, 255, 0.45)
    ),
url("https://upload.wikimedia.org/wikipedia/commons/3/36/Sweet_Summer_Rainbow_Fruit_Salad.j
pg");
     background-color: #cccccc;
  }
  a:link, a:visited, a:hover, a:active {
      color: white;
  }
  </style>
</head>

<body>
  <div class="bg"></div>
```

github.com/npentrel/smooth-docker/tree/v1

Dockerfile

```
1  # our base image
2  FROM python:3.6-alpine
3
4  # install Python modules needed by the Python app
5  COPY requirements.txt /usr/src/app/
6  RUN pip install --no-cache-dir -r /usr/src/app/requirements.txt
7
8  # copy files required for the app to run
9  COPY app.py /usr/src/app/
10 COPY templates /usr/src/app/templates/
11
12 # tell the port number the container should expose
13 EXPOSE 9000
14
15 # run the application
16 CMD ["python", "/usr/src/app/app.py"]
```



github.com/npentrel/smooth-docker/tree/v1

Let's look for the python:3.6-alpine image on Docker Hub

- Many official images are on https://hub.docker.com/
- You can see the Dockerfiles for each image there

ACTIVITY

Build the Docker Image

Now that you have your Dockerfile, you can build your image. The docker build command does this. Replace npentrel with your Docker ID

```
→ docker build -t npentrel/smooth-docker:1.0 .
```

OUTPUT

```
→ docker build -t npentrel/smooth-docker:1.0 .
Sending build context to Docker daemon   7.68kB
Step 1/7 : FROM python:3.6-alpine
3.6-alpine: Pulling from library/python
ff3a5c916c92: Pull complete
44014a6ad6bc: Pull complete
9e372a7142ef: Pull complete
3ab6d28ced3c: Pull complete
27f34cba021a: Pull complete
Digest: sha256:3f9e4710fc0dfb2aeaa32016bd8a0805f90612e61b5fc5b1194e1d9d1f7edca2
Status: Downloaded newer image for python:3.6-alpine
 ---> 5be6d36f77ee
Step 2/7 : COPY requirements.txt /usr/src/app/
 ---> 47b67ec6e1ab

(...)

Successfully built 3c3b419cd135
Successfully tagged npentrel/smooth-docker:1.0
```

ACTIVITY

Run your Docker Image

Use the docker run command to run the image we just built. Replace npentrel with your Docker ID

```
→ docker run --name static-site -d -p 8888:9000 npentrel/smooth-docker:1.0
05eee9db8db23e1b6da5b9ecff031817b95189cd267751a0fc733159042c4d18
```

Head over to the following port to see your work:

8888

ACTIVITY

Try Visiting your Website

Head over to the following port and you should see the website below:

8888

ACTIVITY

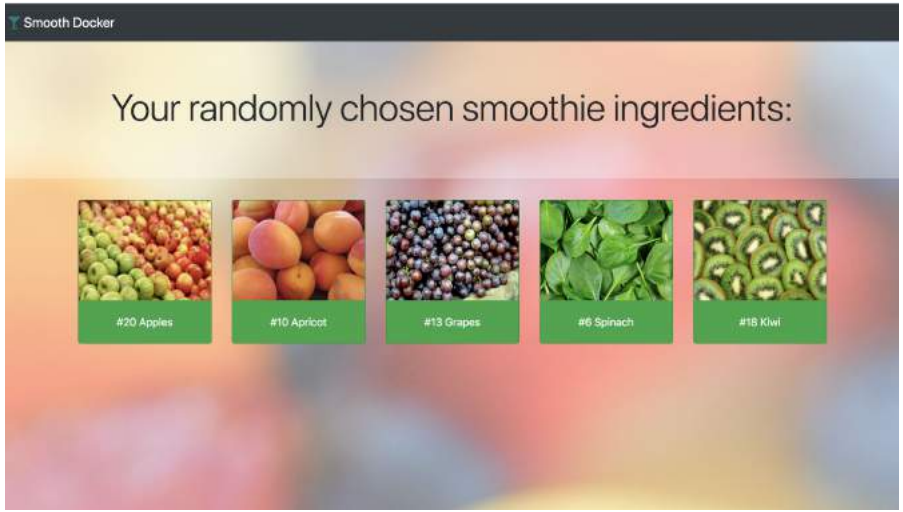
Let's Clean Up!

```
→ docker stop static-site
static-site

→ docker rm static-site
static-site
```

1. Intro to Docker & Containers
2. Setting up Docker
3. Running your First Container
4. Web Apps with Docker
5. **Docker Compose**

Let's Run Two Containers

Containers can also connect to other containers. We're going to create a service that uses MongoDB and flask to select ingredients for a smoothie for you:



ACTIVITY

Get the Source Code:

```
→ git clone https://github.com/npentrel/smooth-docker.git
→ cd smooth-docker
→ git checkout v2
```

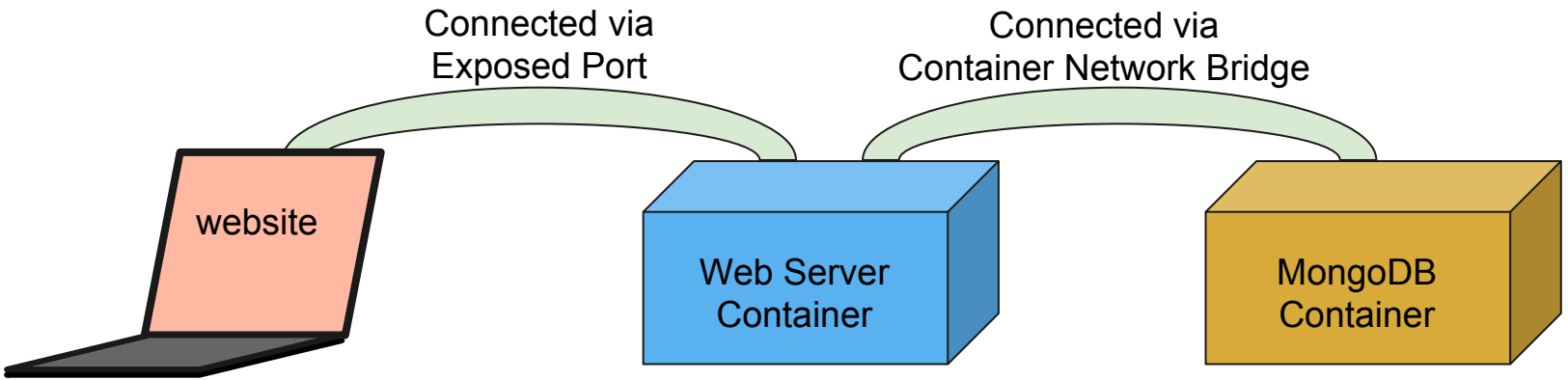
See the site running live:

smooth-docker-v2.herokuapp.com/

The code for the website we'll run is here:

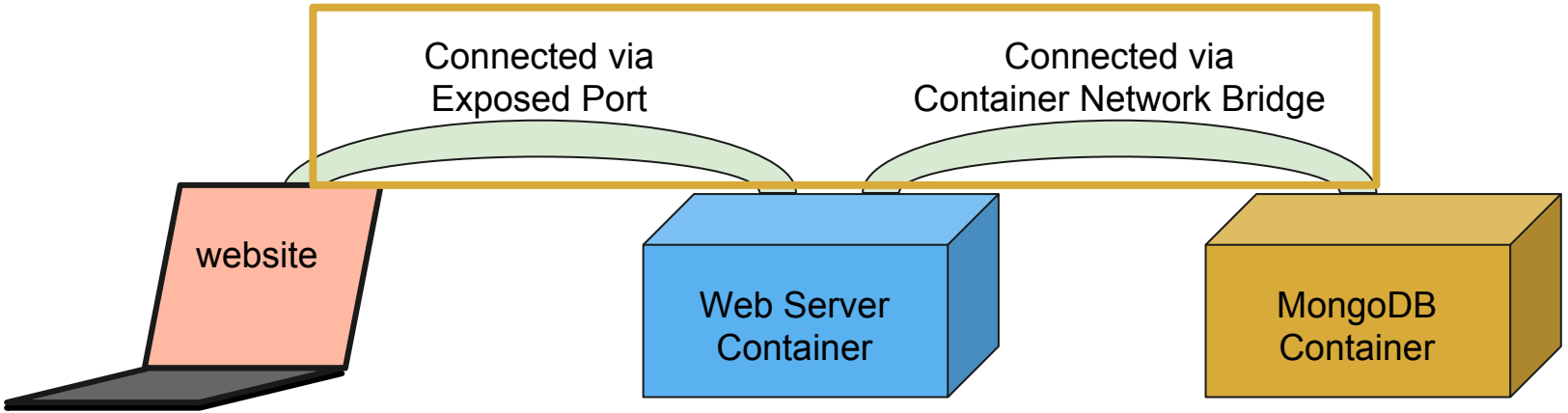
github.com/npentrel/smooth-docker/tree/v2

The Containers



Build and Run the application manually

If you are just using Docker you will have to set up networks



ACTIVITY

Docker-compose.yml (based on yaml)

```yaml
version: "3.5"
services:
  app:
    restart: always
    build: ./app
    # uncomment for development
    # volumes:
    #   - ./app:/usr/src/app/
    ports:
      - "9000:9000"
    networks:
      - backend

  database:
    image: mongo_database
    build: ./mongo
    volumes:
      - ./data:/data/db
    networks:
      - backend

networks:
  backend:
    driver: bridge
```

github.com/npentrel/smooth-docker/tree/v2

ACTIVITY

Build and Run the Application

The docker-compose build command is used to build the app, followed by docker-compose up to run the app

```
1 → docker-compose build
2
3 database uses an image, skipping
4 Building app
5 (...)
6 Successfully tagged mongo_seed:latest
7
8 → docker-compose up
```

OUTPUT

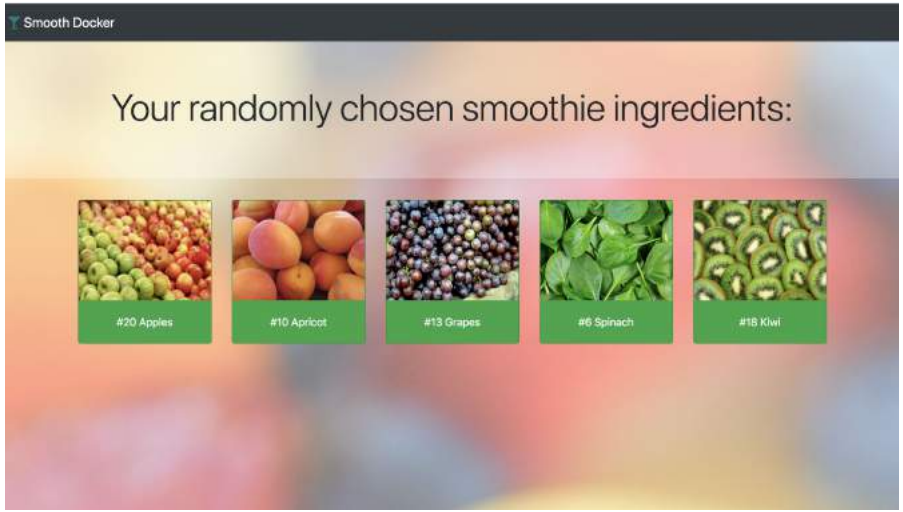
```
1 → docker-compose build
2
3 database uses an image, skipping
4 Building app
5 (...)
6 Successfully tagged mongo_seed:latest
7
8 → docker-compose up
9
10 Starting database ... done
11 Starting python-mongo-example_mongo_seed_1 ... done
12 Starting app                                 ... done
13 Attaching to smooth-docker_database_1, smooth-docker_app_1, smooth-docker_mongo_seed_1
14 (...)
15 mongo_seed_1  | 2018-06-13T00:21:49.579+0000    imported 20 documents
16 database      | 2018-06-13T00:21:49.580+0000 I NETWORK  [conn2] end connection 172.18.0.3:53982 (1
   connection now open)
17 smooth-docker_mongo_seed_1 exited with code 0
```

ACTIVITY

Try Visiting your website

Head over to the following URL and you should see the website below:

9000



1. Intro to Docker & Containers
2. Setting up Docker
3. Running your First Container
4. Web Apps with Docker
5. Docker Compose

Special Thanks to MLH localhost

A special thank you to MLH for letting us base our training on their resources for localhost.mlh.io.

Other Resources

- Docker Deep Dive - Nigel Poulton
- https://docker-curriculum.com/
- https://goto.docker.com/rs/929-FJL-178/images/Docker-for-Virtualization-Admin-eBook.pdf
- http://orhandogan.net/docker/
- https://docs.docker.com/get-started/
- https://docs.docker.com/compose/

Questions?

Thank You!

@naomi_pen

References

- Swarmnado GIF: https://jancelin.github.io/workshop_docker_GIS/swarmnado.gif
- https://nvisium.com/blog/2014/10/15/docker-cache-friend-or-foe.html
- https://www.contino.io/insights/beyond-docker-other-types-of-containers
- https://www.docker.com/what-container#/package_software
- https://searchservervirtualization.techtarget.com/definition/virtual-machine
- https://searchservervirtualization.techtarget.com/definition/hypervisor
- https://cloud.google.com/containers/
- https://stackoverflow.com/questions/41550727/how-does-docker-for-windows-run-linux-containers
- https://codingpackets.com/virtualization/docker/
- https://medium.com/@nagarwal/docker-v1-13-is-pretty-much-awesome-a10a66459acc
- https://docs.docker.com/machine/overview/
- https://docs.docker.com/engine/docker-overview/#docker-architecture
- https://www.reddit.com/r/docker/comments/7xvlye/docker_for_macwindows_performances_vs_linux/
- https://www.docker.com/what-container
- https://nickjanetakis.com/blog/understanding-how-the-docker-daemon-and-docker-cli-work-together
- https://developer.okta.com/blog/2017/10/11/developers-guide-to-docker-part-3#create-a-docker-network