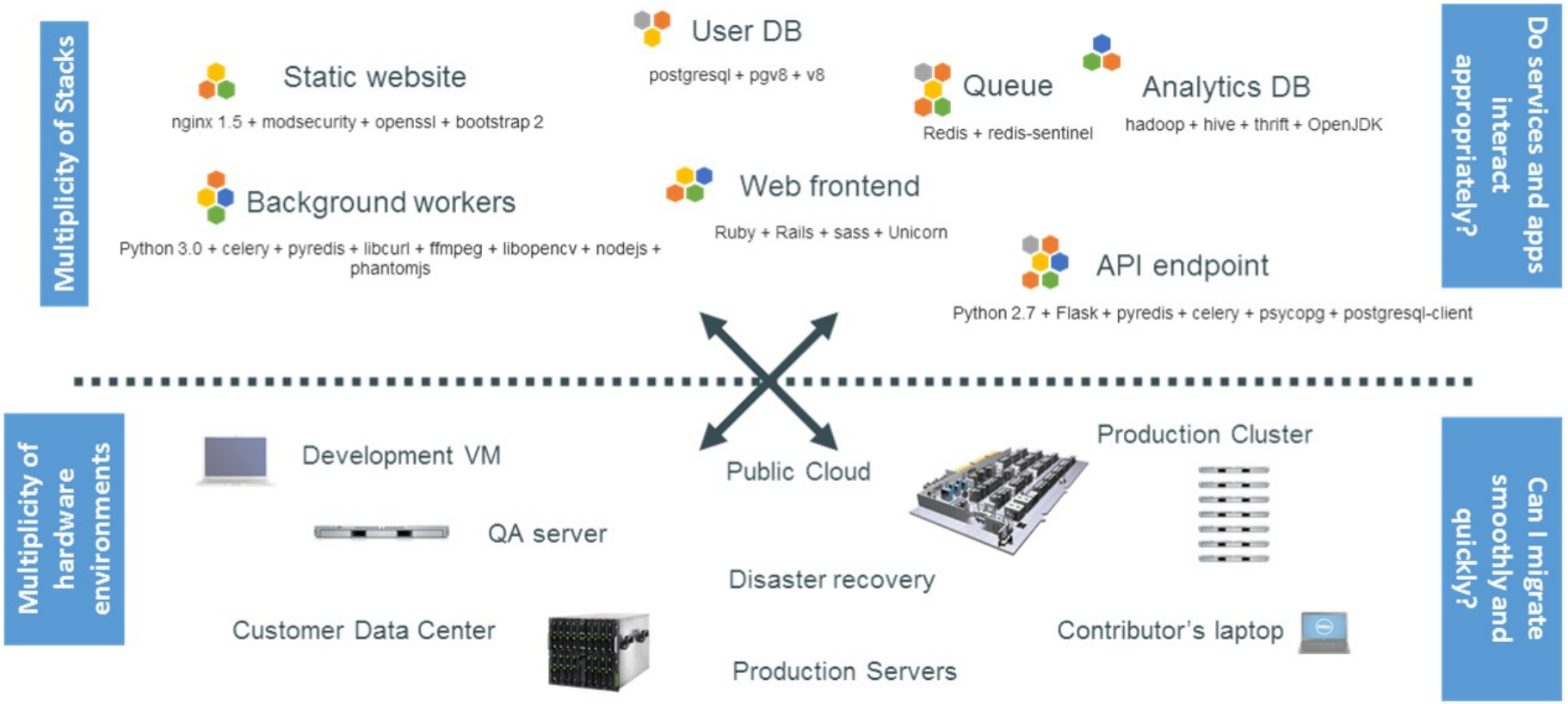
HANDSON INTRODUCTION

Created by Konrad Klimaszewski (2016)

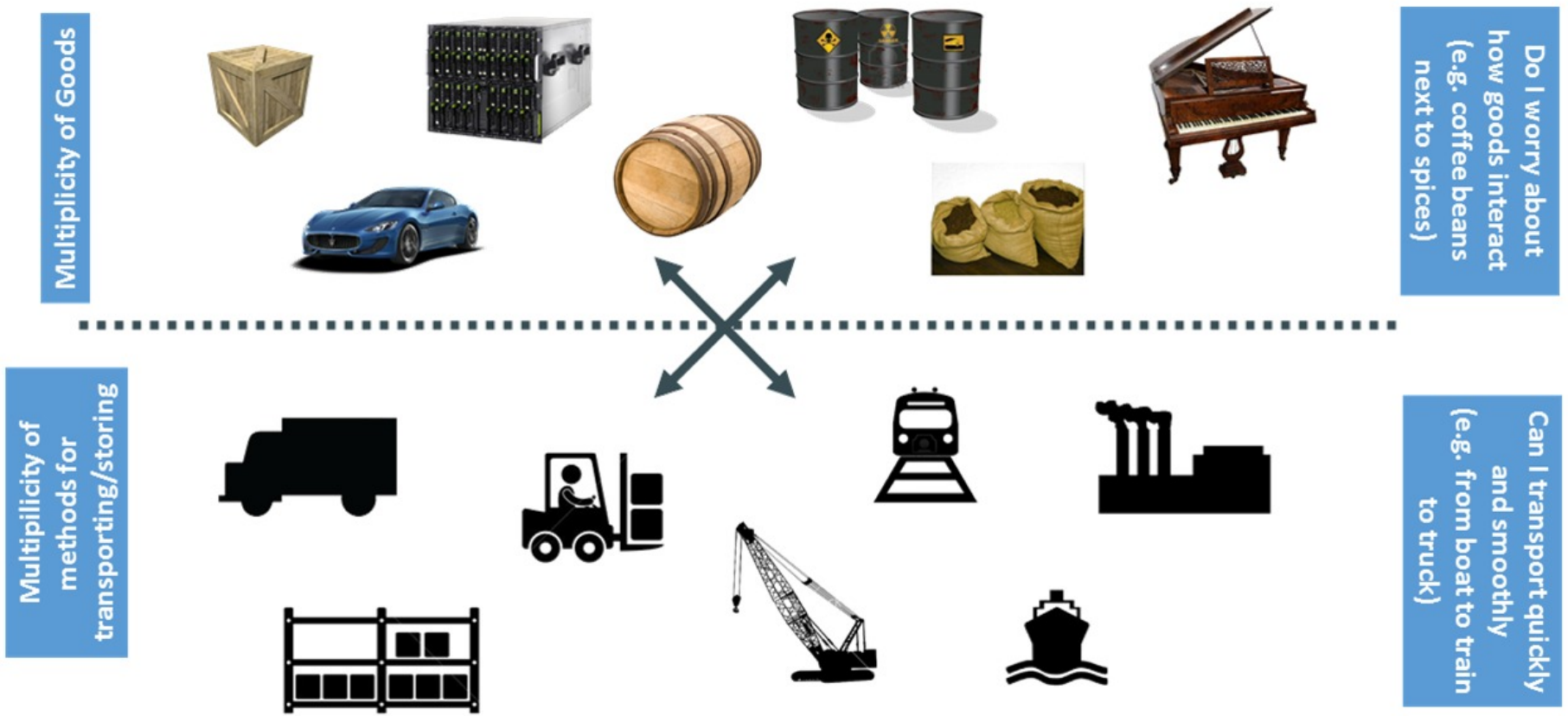
- Containers
- Docker
- Hands-on
- Docker tools

CONTAINERS

THE CHALLENGE

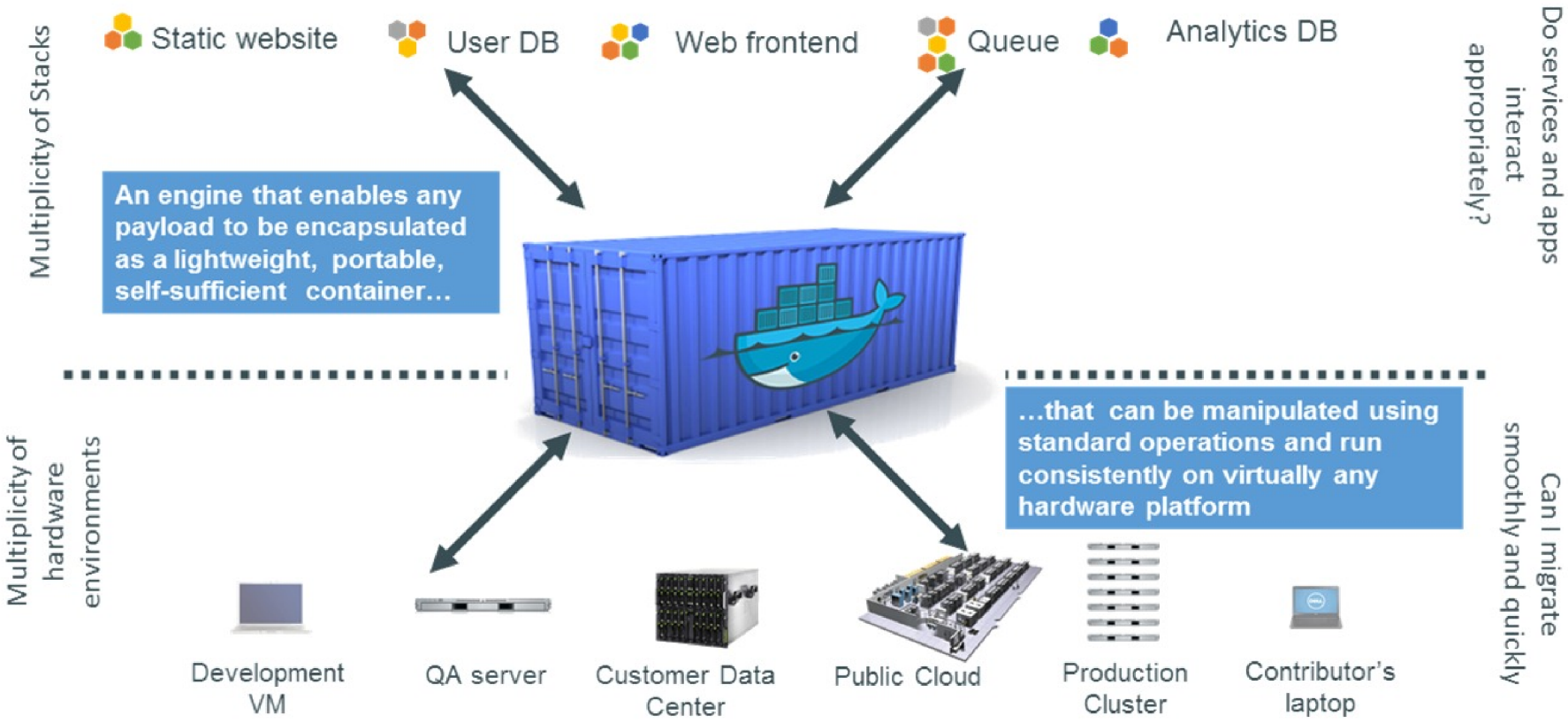


CARGO TRANSPORTATION PRE 1960 ...



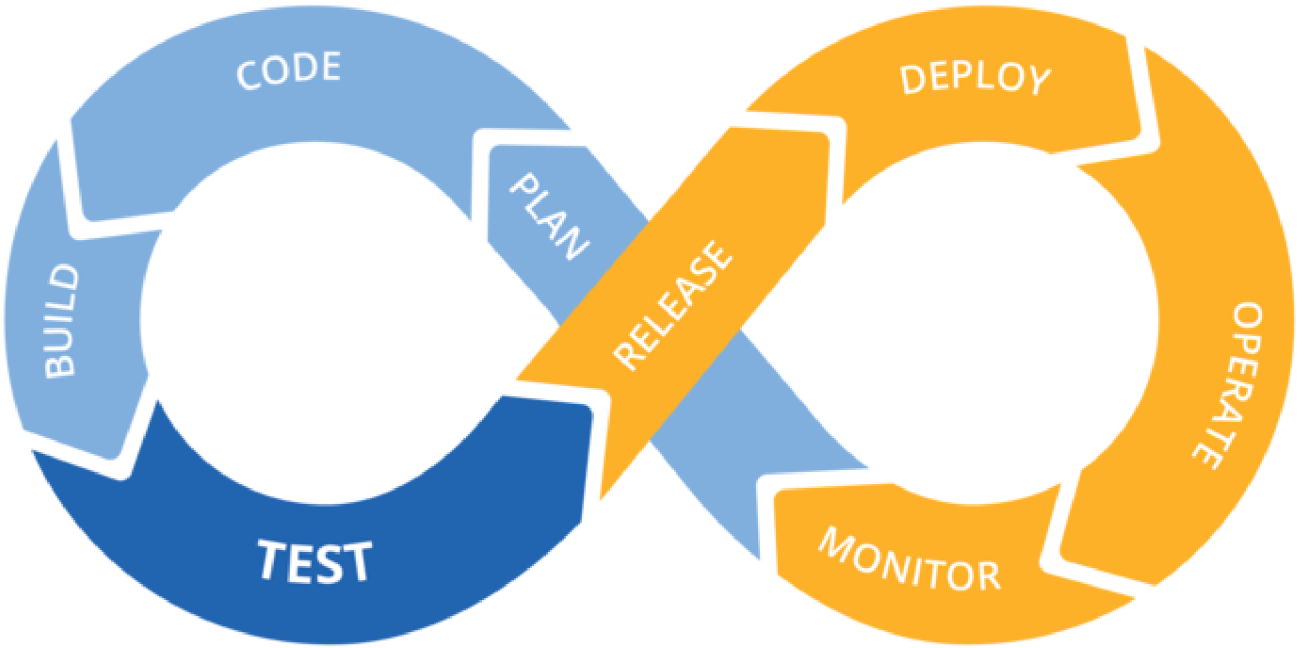
SOLUTION: CONTAINERS

DOCKER IS A CONTAINER SYSTEM FOR CODE ...



USE CASES

DEVOPS



ISOLATION

SERVICE UPGRADES

TESTING

- Lightweight
- Fast
- Disposable
  - Run
  - Test
  - Remove

CAVEATS

SECURITY

CONTAINERS DO NOT CONTAIN

Dan Walsh (Mr SELinux)

SECURITY

Major kernel subsystems are not namespaced like:

- SELinux
- Cgroups
- file systems under /sys
- /proc/sys, /proc/sysrq-trigger, /proc/irq, /proc/bus

Devices are not namespaced:

- /dev/mem
- /dev/sd* file system devices
- Kernel Modules

SECURITY

- Drop privileges

  Docker drops a lot of root capabilities

- Run as user inside container
- Use SELinux or AppArmor
- Run images only from trusted sources

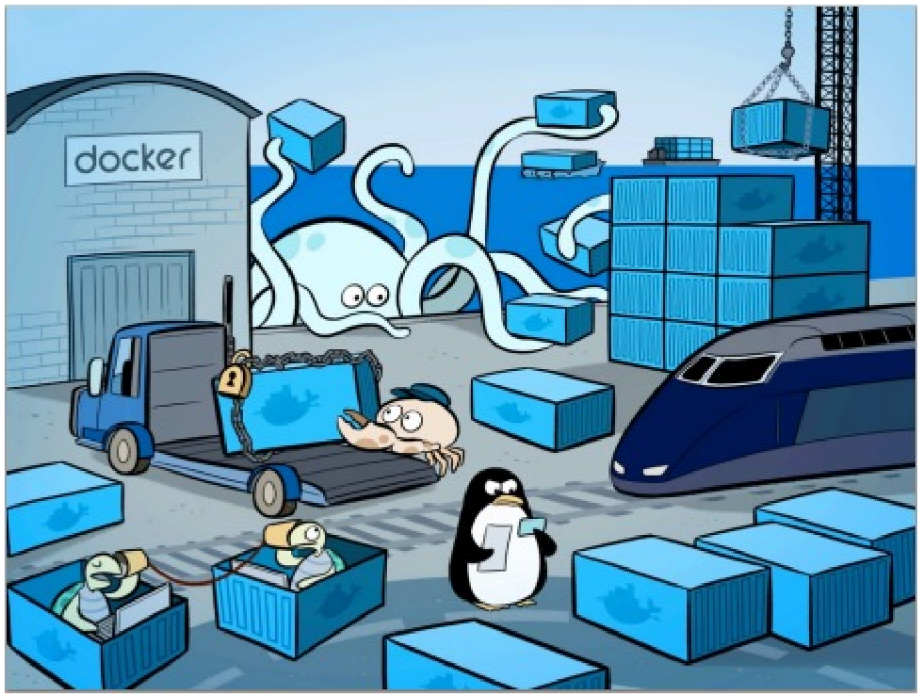
DOCKER IO

- Docker supports several storage drivers for the layered FS
  - **AUFS** - Ubuntu: *slow for big files, not mainline*
  - **Device Mapper** - Red Hat: *very slow or slow, complex and troublesome*
  - **ZFS**: *not mainline, SELinux??*
  - **Btrfs**: *slow??, no page cache, no SELinux*
  - **Overlayfs**: *fast, very new, not mainline, no SELinux*
- **Use volumes for all data written by the container**

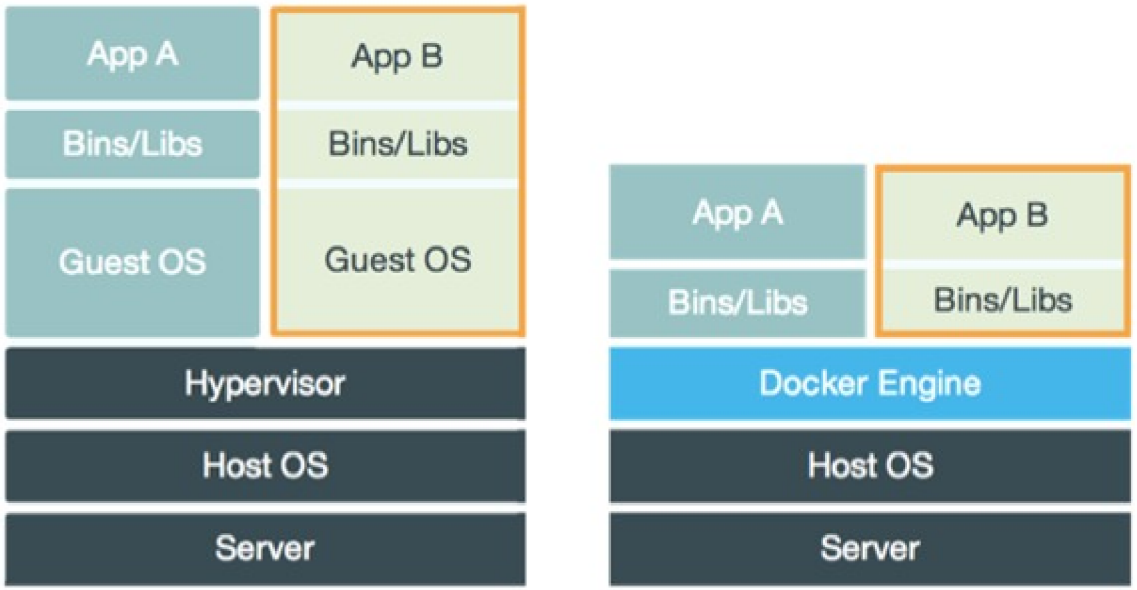
FURTHER READING

- http://www.projectatomic.io/blog/2015/06/notes-on-fedora-centos-and-docker-storage-drivers
- https://opensource.com/business/14/7/docker-security-selinux

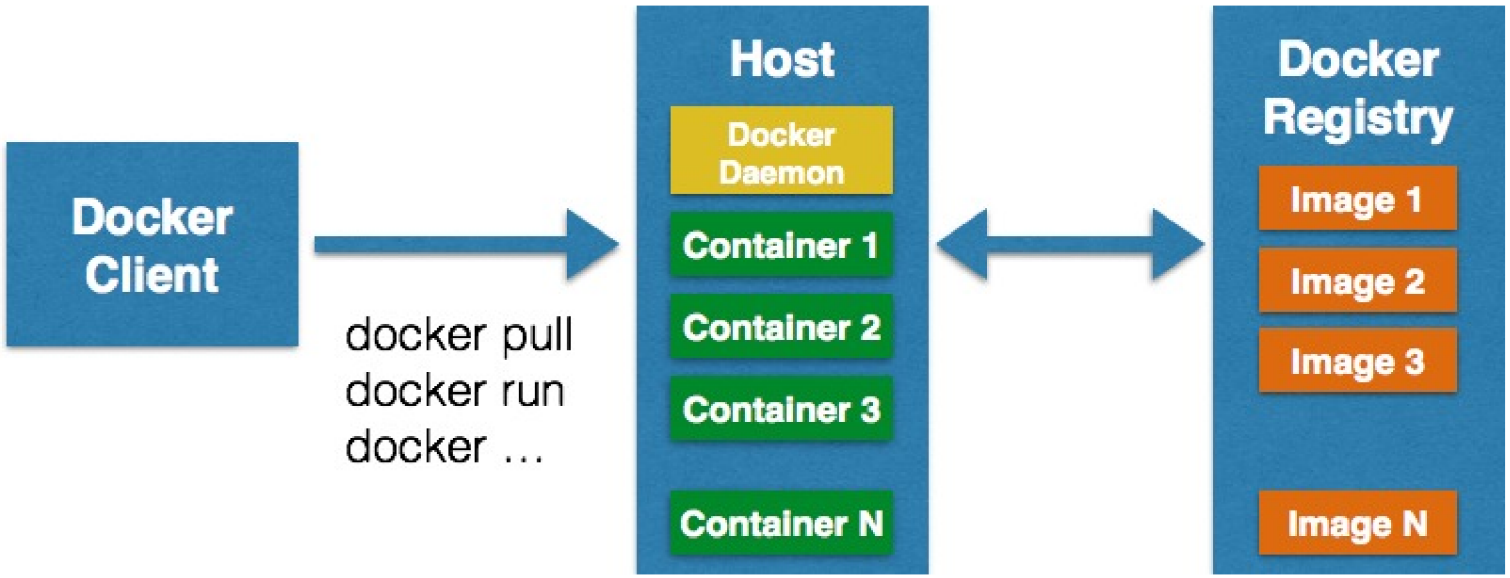
DOCKER



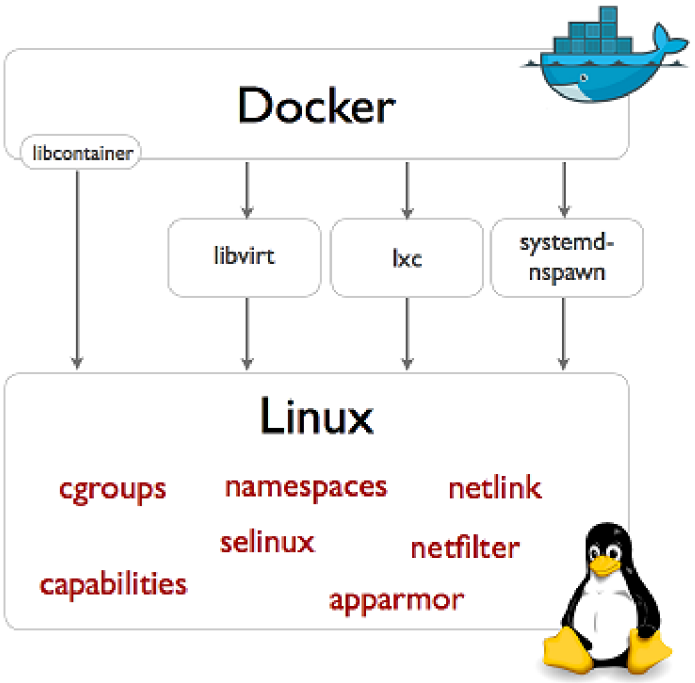
VM VS CONTAINERS



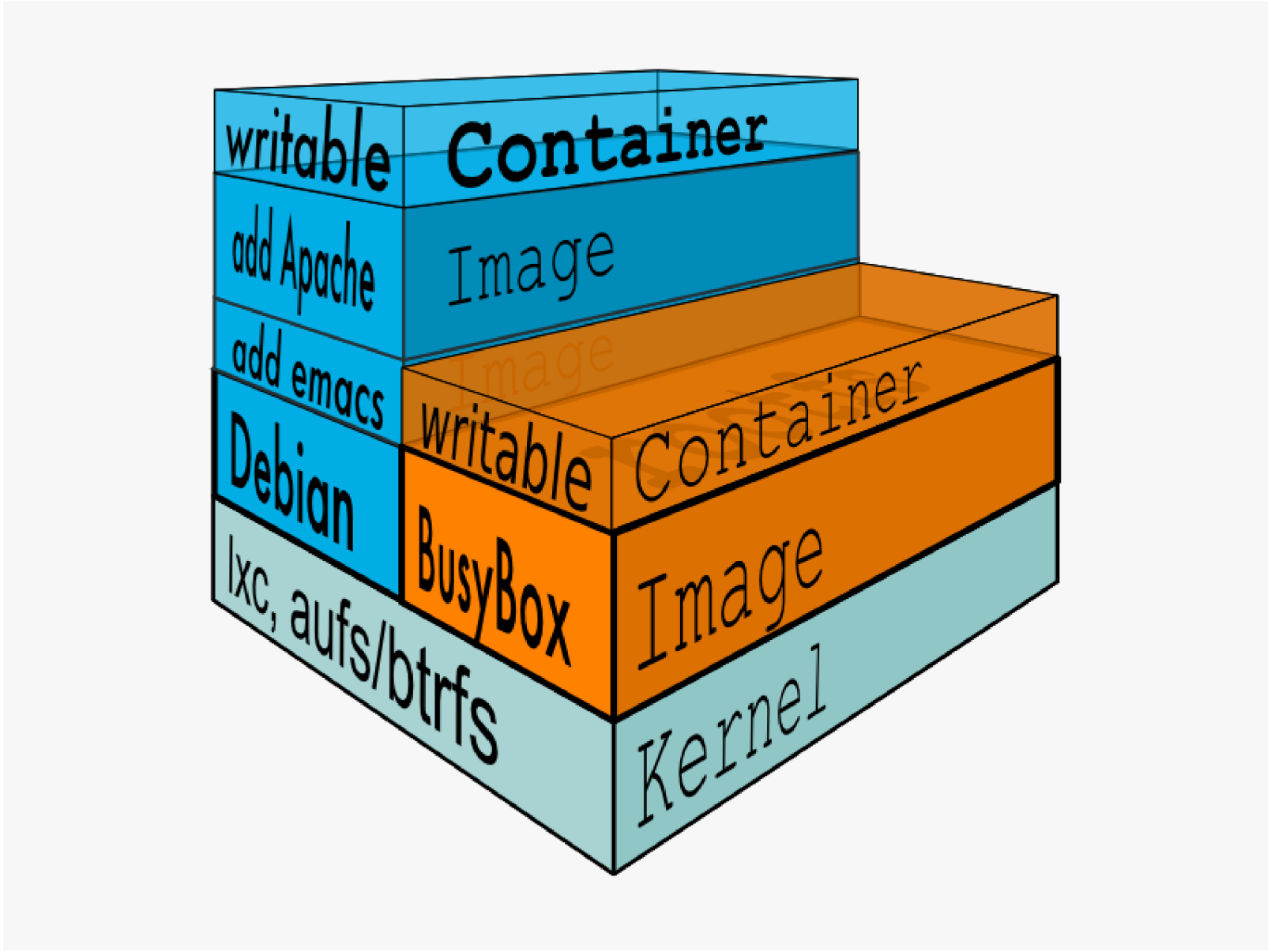
DOCKER ARCHITECTURE



DOCKER ARCHITECTURE



DOCKER IMAGES



THE LIFE OF A CONTAINER

CONCEPTION

BUILD an Image from a Dockerfile

BIRTH

RUN (create+start) a container

REPRODUCTION

COMMIT (persist) a container to a new image
RUN a new container from an image

SLEEP

STOP or KILL a running container

WAKE

START a stopped container

DEATH

RM (delete) a stopped container

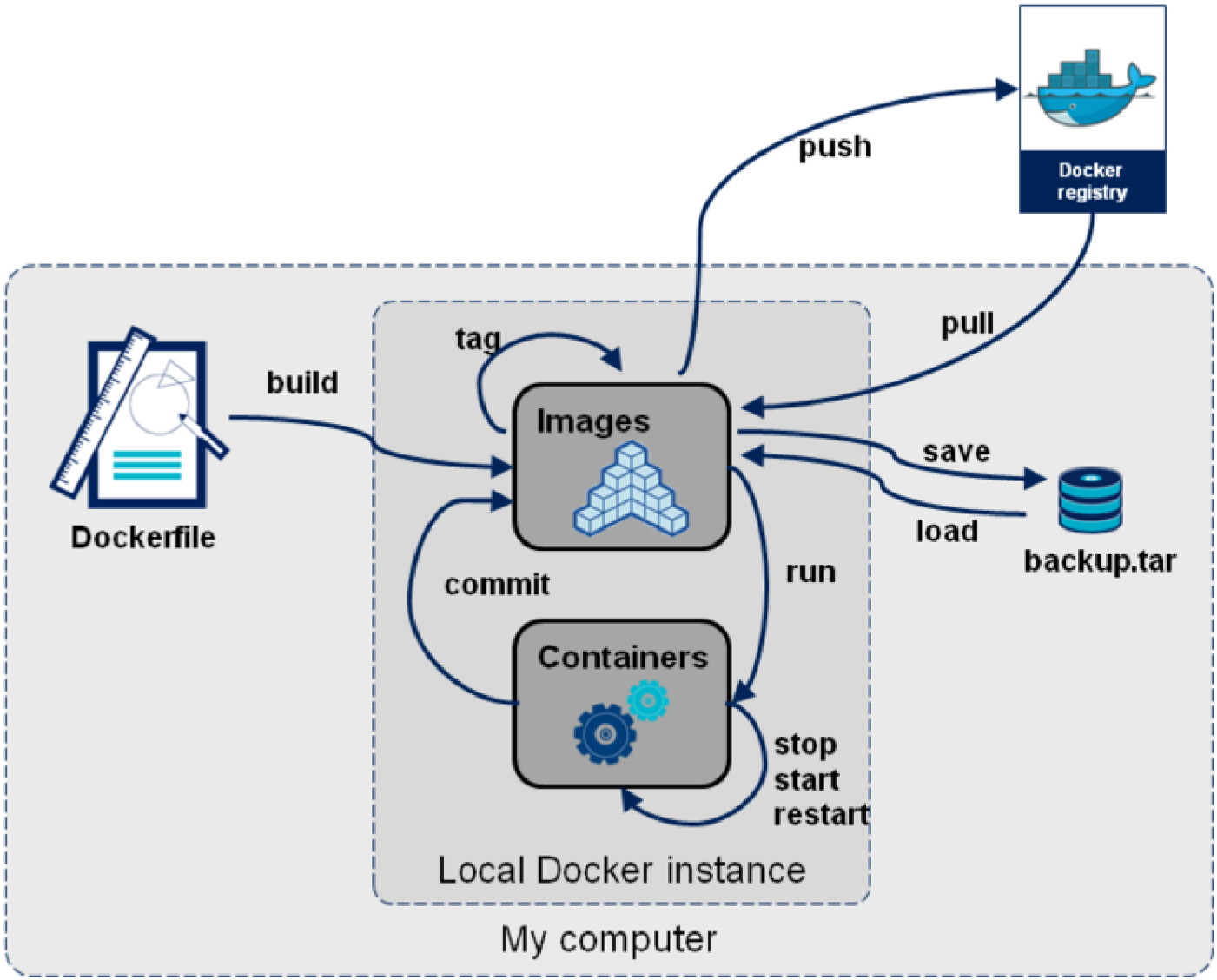
EXTINCTION

RMI a container image (delete image)

THE LIFE OF A CONTAINER BY EXAMPLE.

```
docker pull mongo:latest     # pull the mongo image from the registry
docker inspect mongo:latest  # list information of the container
docker run -p 27017:27017 \
       --name my-mongo -d \
       mongo:latest          # create and start a mongo container
docker inspect my-mongo      # inspect the running container info
docker logs -f my-mongo      # tail the log of the container
docker stop my-mongo         # stop the container
docker rm -v my-mongo        # remove the container
docker rmi mongo:latest      # remove the image from the local repo
```

DOCKER STAGES



INFORMATION OF A CONTAINTER.

command	description
ps	shows running containers
logs	fetch the logs of a container
inspect	return low-level information on a container or image
events	gets events from container
port	list port mappings for the container
top	display the running processes of a container
stats	display a live stream of container(s) resource usage statistics

HANDS ON LABS

PREREQUISITES

SOME HINTS

- You will find the slides here: http://cis-ncbj.github.io/docker-introduction/
- Navigation through the slides is easy, just use your arrow keys. Left and right for going to the previous or next section and up and down for previous or next page. The space bar always give you the next page.
- All code blocks are meant to be executed by your self unless mention otherwise.
- Have fun, take your time, and feel free to ask questions.

SETUP

- For the workshop we use an VM instance @ CIS cluster:
  - Connection details on paper sheets you recieved
  - Requires SSH client (putty on windows)
- Alternatively you can run the handson on your local machine in VirtualBox
  - Requires VirtualBox VirtualBox installation
- Last possibility is to run docker locally on linux machine
  - The titorial was tested with docker 1.10
  - Be aware that part of the tutorial might behave differently
  - Install docker, docker-compose, git and your favourite editor
  - Run as root or add your user to the `docker` group

VIRTUALBOX

- We use a VirtualBox VM to setup a common environment.
- Premade image https://goo.gl/7zzSos
- The VirtualBox appliance contains
  - CentOS 7 Desktop
  - Tools: Docker, Docker Compose and Git (among others)
  - User: cis, Password: cis

HOST SETUP FOR ELASTICSEARCH

Elasticsearch which will be used in the handson has large memory requirements

- Make sure your VM has at least 4GB of RAM
- Increase the hosts mmap count limit:
  - https://www.elastic.co/guide/en/elasticsearch/reference/2.1/setup-configuration.html#vm-max-map-count

```
sysctl -w vm.max_map_count=262144
```

Already set for tutorial VMs

DOCKER @ CIS

- The default docker network clashes with CIS networks
- Change it after the installation

DEBIAN

Edit */etc/default/docker*:

```
DOCKER_OPTS="--bip=10.0.254.1/24"
```

Recreate the docker0 bridge:

```
sudo service docker stop
sudo ip link del docker0
sudo service docker start
```

Already set for tutorial VMs

DEBIAN/UBUNTU WITH UPSTREAM DOCKER

For Debian 8 and later and Ubuntu 15.04 and later the `/etc/default/docker` is ignored when installing directly from docker.com

Create */etc/systemd/system/docker.service.d/network.conf*:

```
[Service]
ExecStart=
ExecStart=/usr/bin/docker daemon -H fd:// --bip=10.0.245.1/24
```

Recreate the docker0 bridge:

```
sudo systemctl daemon-reload
sudo systemctl stop docker
sudo ip link del docker0
sudo systemctl start docker
```

Already set for tutorial VMs

CENTOS

Edit */etc/sysconfig/docker-network*:

```
DOCKER_NETWORK_OPTIONS="--bip=10.0.254.1/24"
```

Recreate the docker0 bridge:

```
sudo systemctl stop docker
sudo ip link del docker0
sudo systemctl start docker
```

Already set for tutorial VMs

LOGIN TO TUTORIAL VMS

VIRTUALBOX

- Import DockerTutorial.ova
- Start DockerTutorial virtual machine
- Login: cis
- Password: cis
- Open terminal

REMOTE VMS

- Login to your VM using ssh
- Hostname and Password are on the paper sheet

```
ssh root@<hostname>
```

PREPARE TUTORIAL MATERIALS

- Open the tutorial slides (for copy & paste)

  http://cis-ncbj.github.io/docker-introduction/

- Download the tutorial materials

```
cd /opt
git clone https://github.com/cis-ncbj/docker-introduction-data
```

VIRTUALBOX

- Execute in terminal

```
yum -y install htop time iptables-services && systemctl enable iptables && syste
```

HANDS ON LAB 1

HELLO WORLD

SOME NOTES

- All steps assume you have a command line open.
  - Almost all steps formatted as code can be executed.
    - `\` is a line break and can be copied and pasted.
    - `< ... >` indicates you should replace the value

DOCKER BASICS

- Below some of the basic docker commands

```
docker help

# Partial output
Commands:
    images    List images
    logs      Fetch the logs of a container
    ps        List containers
    pull      Pull an image or a repository from a Docker registry
    rm        Remove one or more containers
    rmi       Remove one or more images
    run       Run a command in a new container
    start     Start a stopped container
    stop      Stop a running container

Run 'docker help' for all commands.
Run 'docker COMMAND --help' for more information on a command.
```

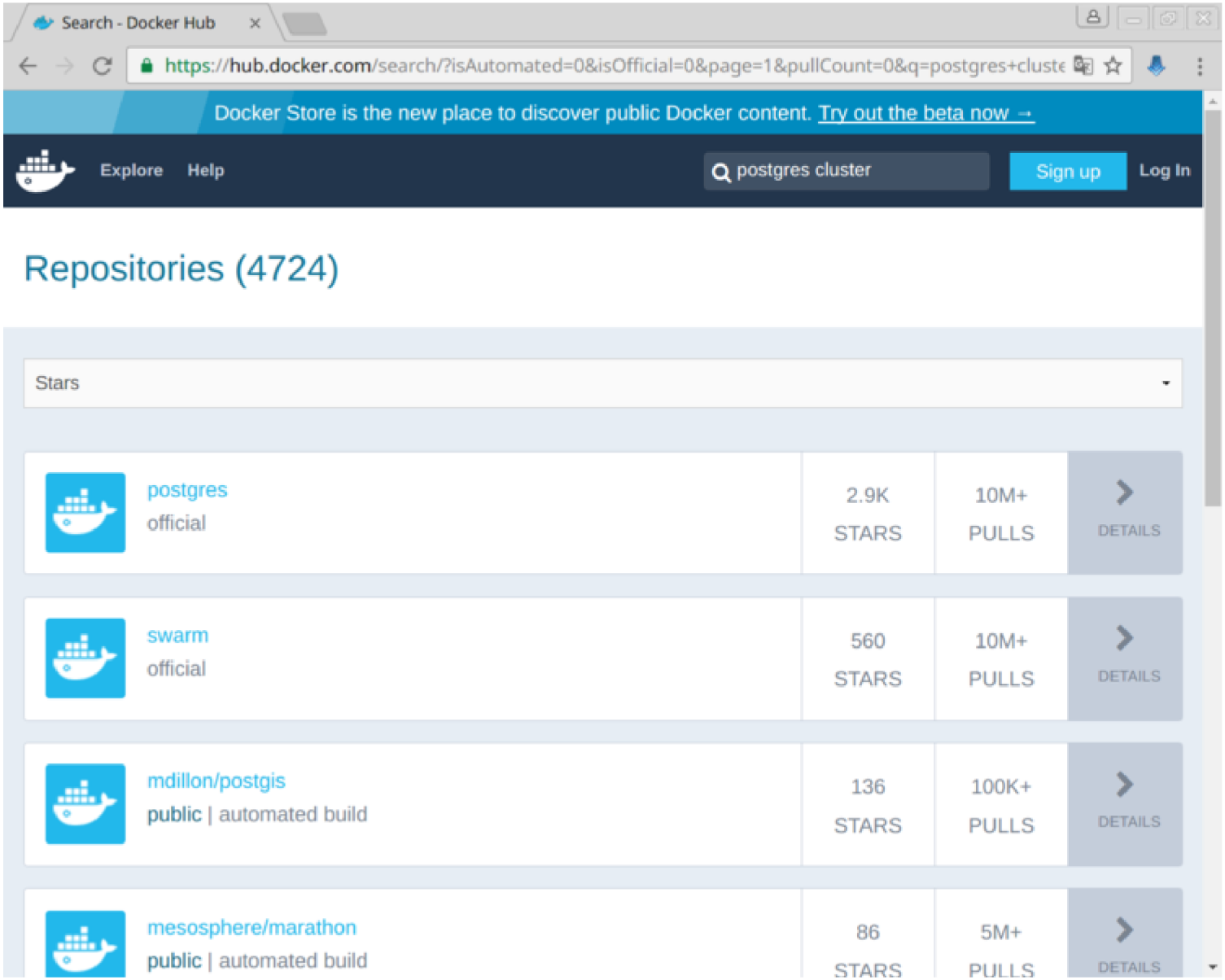
PULL AN IMAGE

- First we pull a base image. We use ubuntu 14.04 latest as base. See Ubuntu repo on docker registry

```
docker pull ubuntu
```

- Now we have the image of ubuntu in our local repository, verify with the command:

```
docker images
```


Search - Docker Hub
https://hub.docker.com/search/?isAutomated=0&isOfficial=0&page=1&pullCount=0&q=postgres+cluste
Docker Store is the new place to discover public Docker content. Try out the beta now →
Explore
Help
postgres cluster
Sign up
Log In
Repositories (4724)
Stars
postgres
official
2.9K
STARS
10M+
PULLS
DETAILS
swarm
official
560
STARS
10M+
PULLS
DETAILS
mdillon/postgis
public | automated build
136
STARS
100K+
PULLS
DETAILS
mesosphere/marathon
public | automated build
86
STARS
5M+
PULLS
DETAILS

START A DOCKER CONTAINER

- Time for hello world.
- With the next command you start an ubuntu container and execute the command to echo some famous string.

```
docker run ubuntu echo "hello world"
```

- Running the command above creates, starts and exits the ubuntu container.
- Observe the output with commands below, remember you can get help by executing `docker help` or `docker help ps`

```
docker ps
docker ps -a
```

- Remove the container

```
docker rm <id or name>
```

KERNEL & ROOT USER

- The container uses the kernel of the *host*

```
docker run --rm ubuntu uname -a
uname -a
```

- Most of the time application in the container runs as root
- However it will not have all capabilities of system administrator

```
docker run --rm ubuntu capsh --print | grep Current
sudo capsh --print | grep Current
```

EXECUTE DOCKER CONTAINER INTERACTIVELY

- Start a container with an interactive shell

```
docker run --name nginx-base -it -e TERM=xterm ubuntu
```

- Install a text editor inside th container

```
apt-get install vim nano
```

- ??? No vim or nano in ubuntu repositories ???

```
apt-get update
apt-get install vim nano
```

- Lets install nginx and edit simple web page

```
apt-get install nginx
ln -sf /dev/stdout /var/log/nginx/access.log
ln -sf /dev/stderr /var/log/nginx/error.log
vim /var/www/html/index.html
exit # Finish when satisfied with the results
```

DATA PERSISTENCY

- Changes made in a container are persisted only in that container. At the moment the container is destroyed the change are lost too.
- There are three ways to obtain data persistence:
  - Commit the changes made in a container to an image, persists the change.
  - Store persistent files in host directory mounted inside the container
  - Store persistent files in data container

CREATE A DOCKER IMAGE

- Image naming conventions
  - `<name>:<tag>` - official docker containers
  - `<user>/<name>:<tag>` - public images @ https://hub.docker.com
  - `local/<name>:<tag>` - local images (stored in local image cache)
- Commit your changes in the container to an (new) image.

```
docker commit nginx-base local/nginx
```

- Inspect your changes.

```
docker images                   # list images stored in local cache
docker history local/nginx      # shows the image history
docker diff nginx-base          # shows the added files
```

- Remove the container.

```
docker rm nginx-base
```

RUN NGINX USING YOUR IMAGE

- Start a new container in daemon mode
  - Publish port 80 to access the web server
  - Run nginx with logging set to stdout/stderr

```
docker run -d --name nginx1 -p 80:80 local/nginx nginx -g "daemon off;"
docker ps
```

- Access your web page
  - Cluster VMs: `<hostname> -> docker<NR>.cis.gov.pl`
  - VirtualBox: `<hostname> -> localhost`

```
firefox http://<hostname> &
```

- Inspect the logging

```
docker logs nginx1
```

- Check `docker help logs for list of very useful options

INTERACTIVE ACCESS TO RUNNING CONTAINER

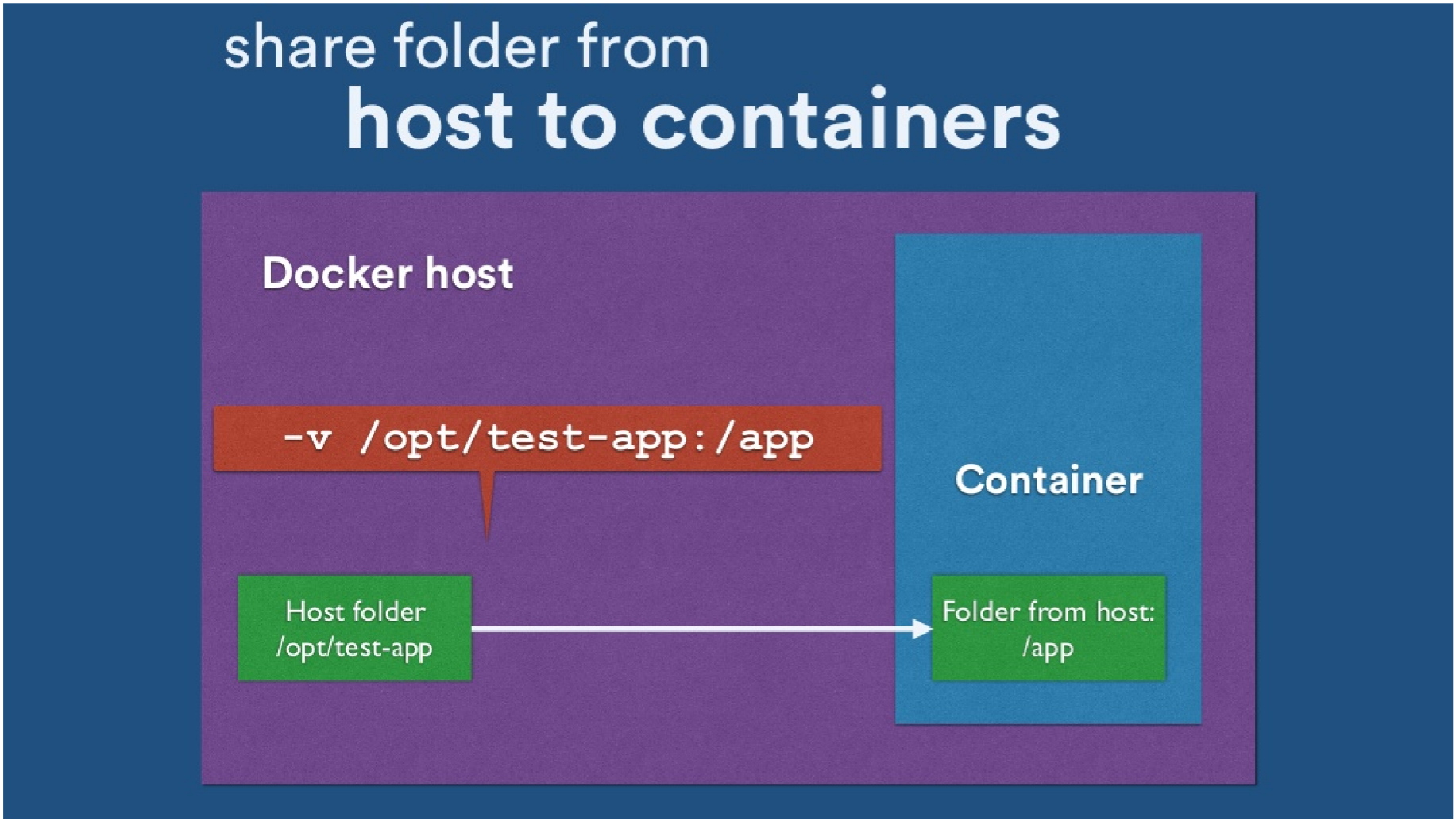
- Use docker exec to access a running container

```
docker exec -it nginx1 bash
vim /var/www/html/index.html
exit
```

- Copy files from and to the container

```
docker cp nginx1:/var/www/html/index.html /tmp
cat /tmp/index.html
vim /tmp/index.html
docker cp /tmp/index.html nginx1:/var/www/html
```

HOST DIRECTORIES AS VOLUMES



http://www.slideshare.net/durdn/be-a-better-developer-with-docker

HOST DIRECTORIES AS VOLUMES

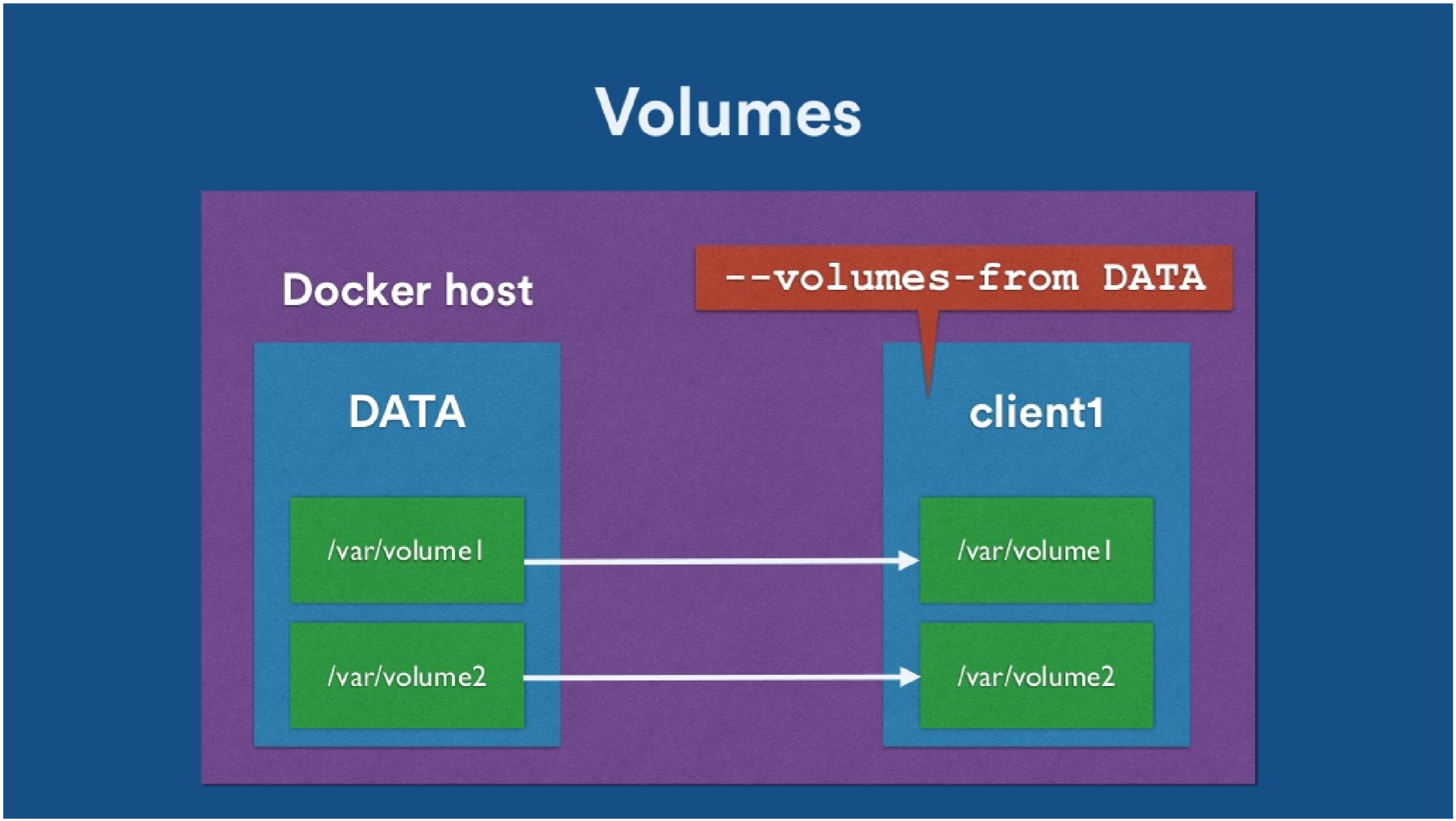
- Run new nginx instance using host directory as a volume
  - Use a different name
  - Publish port 80 as port 888 this time

```
docker run -d -v /opt/docker-introduction-data/static-site:/var/www/html \
    --name nginx3 -p 888:80 local/nginx nginx -g "daemon off;"
```

- Access your web page

```
firefox http://<hostname>:888 &
```

DATA CONTAINERS



http://www.slideshare.net/durdn/be-a-better-developer-with-docker

DATA CONTAINERS

- Create data only container and put inside your web page

```
docker create -v /var/www/html --name nginx-data ubuntu /bin/true
docker cp /tmp/index.html nginx-data:/var/www/html
```

- Run new nginx instance using volume from the data container
  - Use a different name
  - Publish port 80 as port 88 this time

```
docker run -d --volumes-from nginx-data --name nginx2 -p 88:80 \
    local/nginx nginx -g "daemon off;"
```

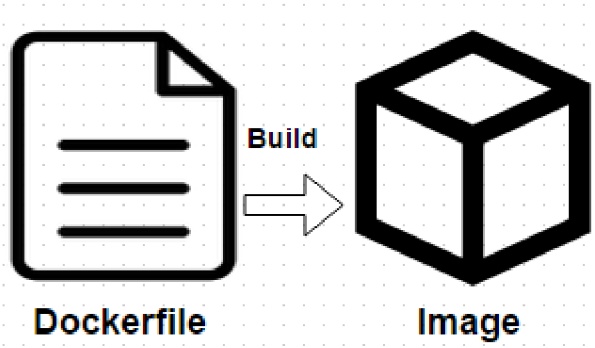
- Access your web page

```
firefox http://<hostname>:88 &
```

- To edit files in data container we need a runing container

```
docker run -it --rm --volumes-from nginx-data -e TERM=xterm local/nginx
vim /var/www/html/index.html
```

HANDS ON LAB 2



BUILDING A IMAGE

DOCKERFILE

- Dockerfile is a script describing how to create an image
- Each command results in a layer of the image
- Behaves a bit similarly to a Makefile
  - Tries to use existing layers if nothing changed
- Key to automatic creation of images
- Docs: Dockerfile reference

DOCKERFILE BY EXAMPLE

```
# Base image
FROM python:3
MAINTAINER Konrad Klimaszewski <konrad.klimaszewski@ncbj.gov.pl>

# Copy to the image pip requirements file
COPY requirements.txt /usr/src/app/
WORKDIR /usr/src/app

# Install dependencies (notice --no-cache-dir)
RUN pip install --no-cache-dir -r requirements.txt

# Copy the rest of the application
COPY . /usr/src/app

# Tell the port number the container should expose
EXPOSE 5000

# Run the command
CMD ["python", "./app.py"]
```

BUILD DOCKER IMAGE

- The Dockerfile and the static web app are already prepared:

```
cd /opt/docker-introduction-data/flask-app
ls
```

- Building the image based on a parent image will create only the diff image.
  - For building the image, you should specify a repository and tag.
  - Specify as repository "local/static-cats" and leave the tag empty.

```
docker build --tag local/static-cats .
```

- Check the result

```
docker images
```

RUN THE IMAGE

- Run new web app instance using `local/static-cats` image
  - Publish port 80 as 8888

```
docker run -d --name cats -p 8888:5000 local/static-cats
```

- Access your web page

```
firefox http://<hostname>:8888 &
```

192.168.99.100:8888

192.168.99.100:8888

CAT GIF OF THE DAY

Courtesy: Buzzfeed

MORE ADVANCED EXAMPLE

PART 1

```
# Use nodejs as base image
FROM node:4.3
MAINTAINER Konrad Klimaszewski <konrad.klimaszewski@ncbj.gov.pl>

# Add local user
RUN groupadd -r flask && useradd -r -g flask flask

## Run all commands in one chain: ##
## - reduces number of layers     ##
## - allows for cleanup step      ##
# Install system-wide deps for python and node
RUN apt-get update \
 && apt-get install -y --no-install-recommends \
      python-pip python-dev dnsutils \
 && rm -rf /var/lib/apt/lists/*

## Separate install of app dependencies from the app           ##
## - During rebuild after code change this step will be skiped ##
# Copy dependency lists
COPY flask-app/requirements.txt flask-app/package.json /opt/flask-app/
WORKDIR /opt/flask-app

# Install app specific dependencies
RUN npm install \
 && pip install -r requirements.txt
```

MORE ADVANCED EXAMPLE

PART 2

```
# Copy our application code
COPY flask-app /opt/flask-app

# Build application (compactify json, css, etc)
RUN npm run build

## If possible drop as many privilidges as possible ##
# Run the application as normal user
USER flask

# expose port
EXPOSE 5000

# start app
CMD [ "python", "./app.py" ]
```

BUILD DOCKER IMAGE FOR FOODTRUCKS

- The Dockerfile and the FoodTrucks web app are already prepared:

```
cd /opt/docker-introduction-data/food-trucks
ls
```

- Build the image

```
docker build --tag local/food-trucks .
```

- Check the result

```
docker images
```

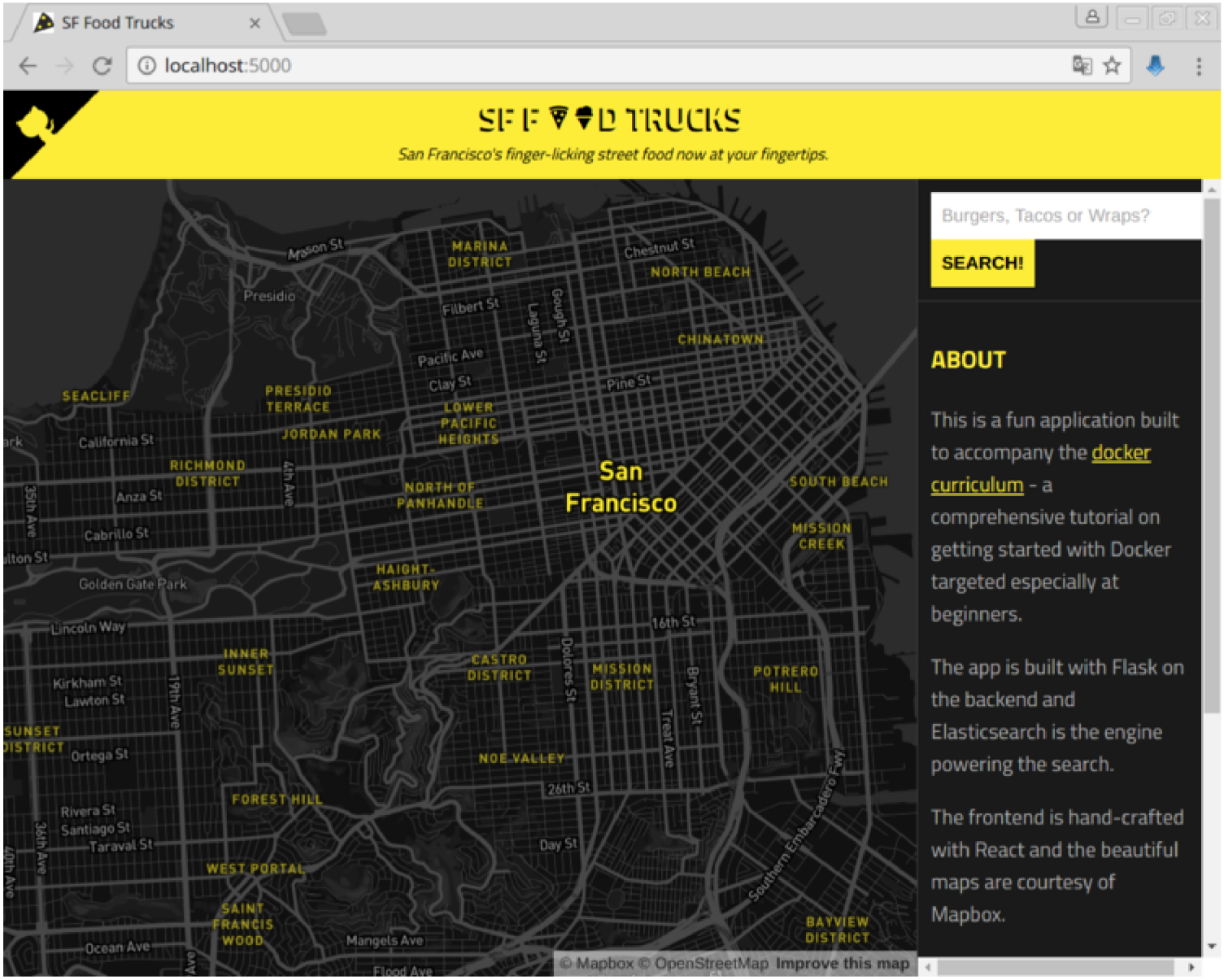
RUN THE APPLICATION

- Run new web app instance using `local/food-trucks` image
  - Publish the port 5000

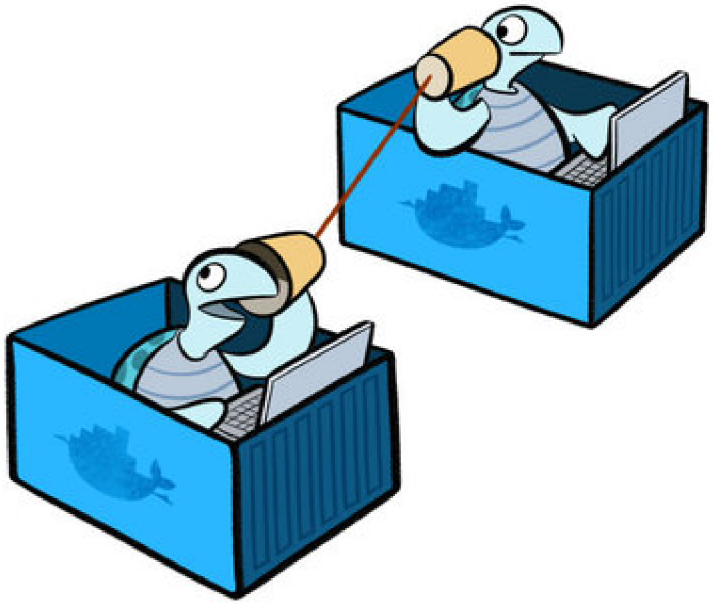
```
docker run -d --name food-trucks -p 5000:5000 local/food-trucks
```

- Access your web page
  - It takes at least 15s for the webapp to start as it tries to connect to elasticsearch database which we will prepare in the next part of the handson

```
firefox http://<hostname>:5000 &
```


SF Food Trucks
localhost:5000
SF FOOD TRUCKS
San Francisco's finger-licking street food now at your fingertips.
Burgers, Tacos or Wraps?
SEARCH!
ABOUT
This is a fun application built to accompany the docker curriculum - a comprehensive tutorial on getting started with Docker targeted especially at beginners.
The app is built with Flask on the backend and Elasticsearch is the engine powering the search.
The frontend is hand-crafted with React and the beautiful maps are courtesy of Mapbox.
MARINA DISTRICT
NORTH BEACH
CHINATOWN
SEACLIFF
PRESIDIO TERRACE
JORDAN PARK
LOWER PACIFIC HEIGHTS
RICHMOND DISTRICT
NORTH OF PANHANDLE
San Francisco
SOUTH BEACH
MISSION CREEK
HAIGHT-ASHBURY
Golden Gate Park
INNER SUNSET
CASTRO DISTRICT
MISSION DISTRICT
POTRERO HILL
NOE VALLEY
FOREST HILL
WEST PORTAL
SAINT FRANCIS WOOD
BAYVIEW DISTRICT
Presidio
Aquatic Park
Fillmore St
Laguna St
Gough St
Pacific Ave
Clay St
Pine St
California St
Anza St
Cabrillo St
Lincoln Way
Kirkham St
Lawton St
Ortega St
Rivera St
Santiago St
Taraval St
Ocean Ave
Mangels Ave
Flood Ave
16th St
26th St
Day St
Dolores St
Treat Ave
Bryant St
Southern Embarcadero Fwy
© Mapbox © OpenStreetMap Improve this map

HANDS ON LAB 3

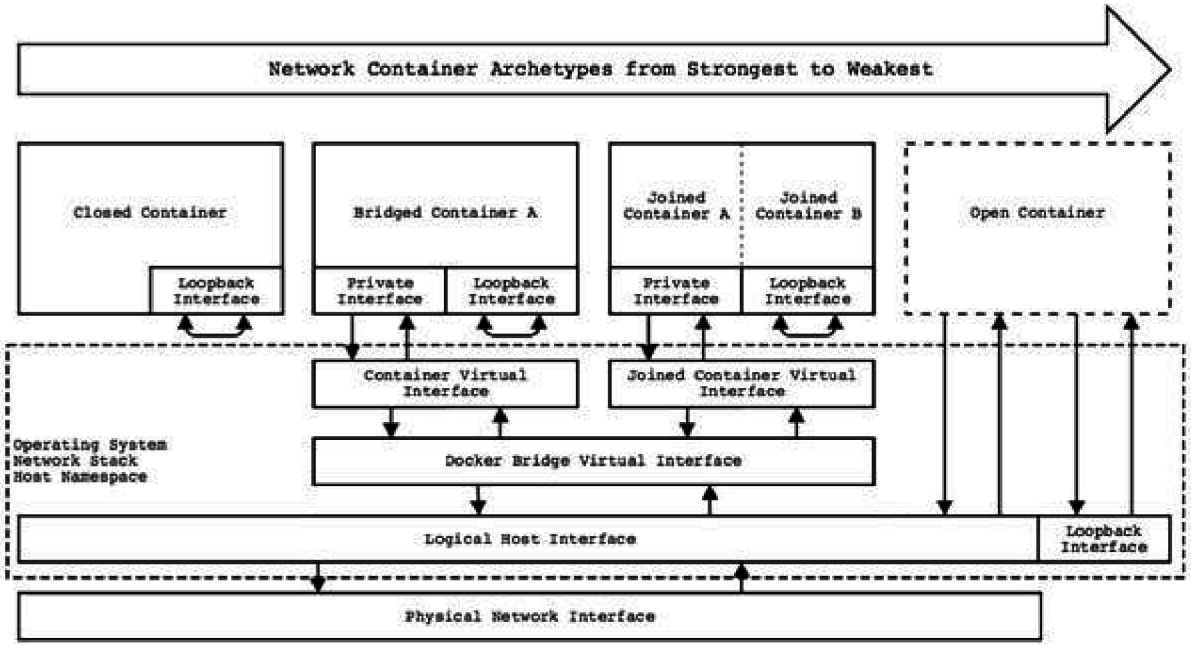


NETWORKING

FOOD TRUCKS

- A simple app to search for food stands in San Francisco
- The app's backend is written in Python (Flask)
- For search it uses Elasticsearch
- Let's try and dockerize it into two containers
  - One container for Elasticsearch
  - One for python backend
  - Containers connected via network

DOCKER NETWORKING TOPOLOGY



- **none**, no networking
- **bridge**, each container has is own
- **shared**, containers share a single network stack
- **hosts**, use the host networking
- **user defined**, user defined network

DOCKER NETWORKING TOPOLOGY BY EXAMPLE

```
# List docker networks
docker network ls

# Run a container with no network
docker run --rm --net none busybox:latest ifconfig

# Run a container in a bridged network (using default docker0 bridge)
docker run --rm --net bridge busybox:latest ifconfig

# or (bridge is the default)
docker run --rm busybox:latest ifconfig

# joined
docker run --name joined1 -d --net none busybox:latest \
  nc -l 127.0.0.1:3333
docker run --rm -it --net container:joined1 busybox:latest netstat -al

# host
docker run --rm --net host busybox:latest ifconfig
```

CONTAINER LINKING

- Docker has a linking system that allows you to link multiple containers together and send connection information from one to another.

```
docker run --link <container_name>:<hostname_alias>
```

- This feature will be **deprecated** in future docker versions.
- Use **`docker network`** instead
  - Provides proper DNS
  - Allows running containers with absent links

USER DEFINED NETWORKS

- You can create your own user-defined networks that better isolate containers
- You can create multiple networks
- You can add containers to more than one network
- Containers can only communicate within networks but not across networks
- A container attached to two networks can communicate with member containers in either network

USER DEFINED NETWORKS

- Let's go ahead and create our own network

```
docker network create --subnet=10.0.100.0/24 foodtrucks
docker network ls
```

- Let's move our running flask web app to the `foodtrucks` network

```
docker network connect foodtrucks food-trucks
docker network disconnect bridge food-trucks
docker network inspect foodtrucks
docker network inspect bridge
docker exec food-trucks ip addr
```

ELASTICSEARCH

- Let's see if we can find pre made Elasticsearch image on the hub

```
docker search elasticsearch
```

- There exists an officially supported image for Elasticsearch
- Let's run it with default settings attached to our `foodtrucks` network

```
docker run -d --net foodtrucks --name es elasticsearch
docker exec food-trucks dig es
docker exec food-trucks curl es:9200
```

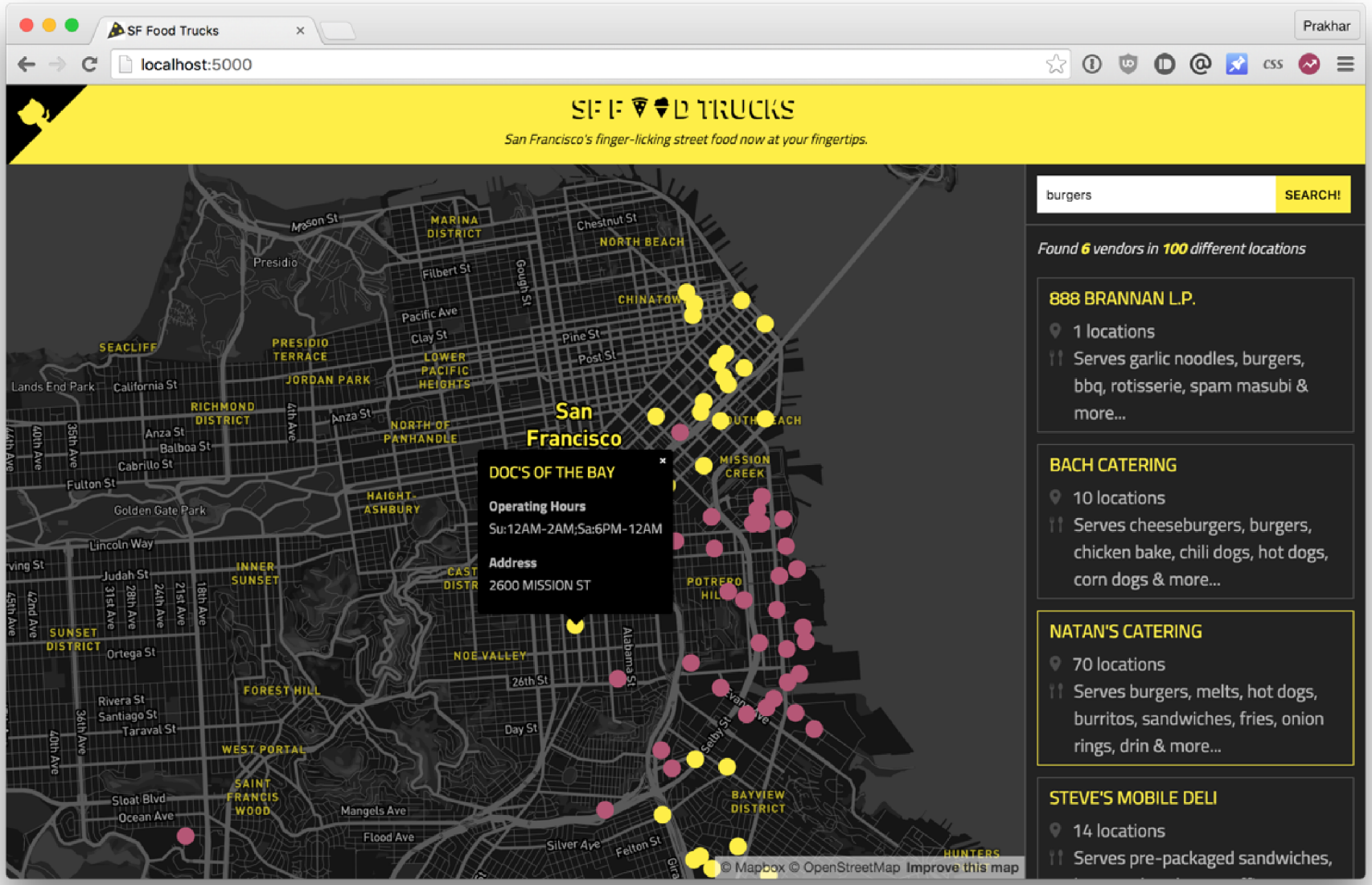
- Let's restart food-trucks container to initialize the ES database

```
docker restart food-trucks
```

- Our web app should be fully operational

```
firefox http://<hostname>:5000
```

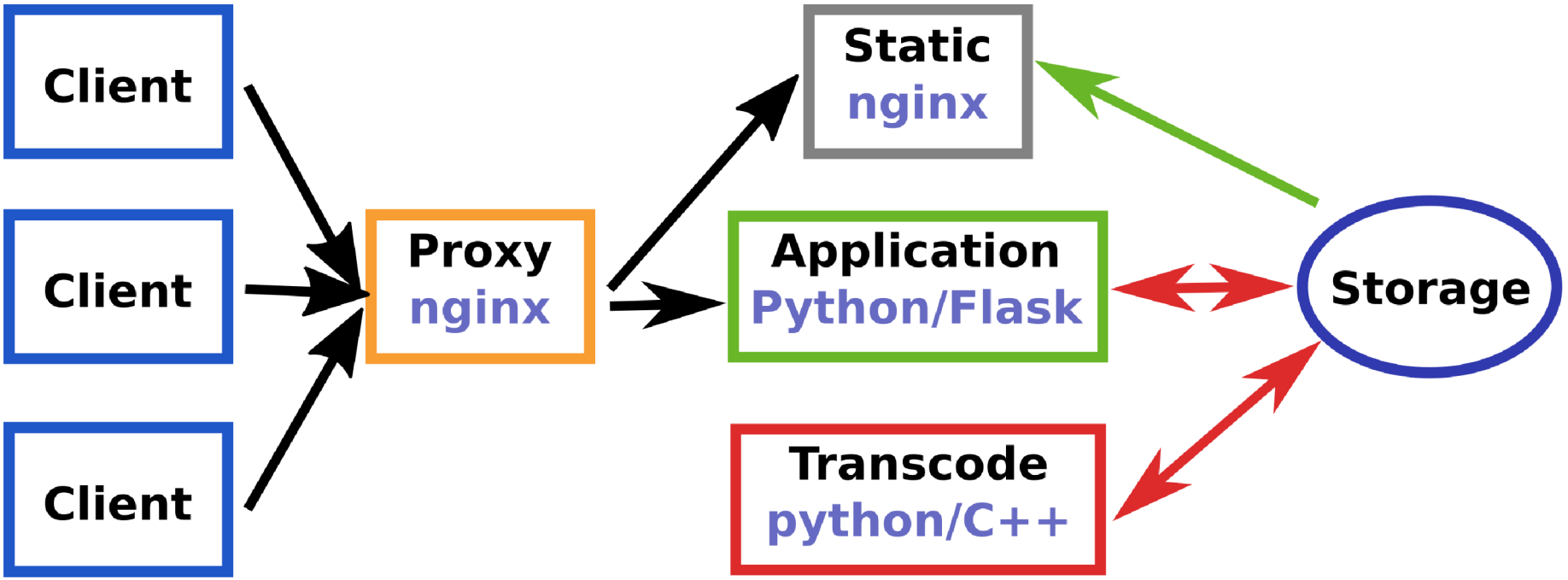
FOOD TRUCKS


SF Food Trucks
Prakhar
localhost:5000
SF FOOD TRUCKS
San Francisco's finger-licking street food now at your fingertips.
burgers
SEARCH!
Found 6 vendors in 100 different locations
888 BRANNAN L.P.
1 locations
Serves garlic noodles, burgers, bbq, rotisserie, spam masubi & more...
BACH CATERING
10 locations
Serves cheeseburgers, burgers, chicken bake, chili dogs, hot dogs, corn dogs & more...
NATAN'S CATERING
70 locations
Serves burgers, melts, hot dogs, burritos, sandwiches, fries, onion rings, drin & more...
STEVE'S MOBILE DELI
14 locations
Serves pre-packaged sandwiches,
DOC'S OF THE BAY
Operating Hours
Su:12AM-2AM;Sa:6PM-12AM
Address
2600 MISSION ST
San Francisco
MARINA DISTRICT
NORTH BEACH
CHINATOWN
PRESIDIO TERRACE
SEACLIFF
JORDAN PARK
LOWER PACIFIC HEIGHTS
RICHMOND DISTRICT
NORTH OF PANHANDLE
MISSION CREEK
HAIGHT-ASHBURY
INNER SUNSET
SUNSET DISTRICT
NOE VALLEY
FOREST HILL
WEST PORTAL
SAINT FRANCIS WOOD
BAYVIEW DISTRICT
POTRERO HILL
Golden Gate Park
Lands End Park
Presidio
Mangels Ave
Flood Ave
Silver Ave
© Mapbox © OpenStreetMap Improve this map

VS

SELINUX VS YOUTUBE

- Let's think how a YouToube like app would work on one machine in docker containers
- We consider only upload, transcode and download of videos/images



SELINUX AND HOST FOLDER SHARING

- Create a shared host folder

```
mkdir /opt/shared
echo HELLO > /opt/shared/hello.txt
```

- Access it from a container

```
docker run --rm -it -v /opt/shared:/data busybox
cd /data
cat hello.txt
echo HELLO2 >> hello.txt
echo HELLO3 > hello3.txt
```

- [Second Terminal]

```
ls -Z /opt
```

- [Container]

```
exit
```

SELINUX AND WRITE ACCESS

- Let's try again
  - Notice the `:Z` parameter for -v option

```
docker run --rm -it -v /opt/shared:/data:Z busybox
cd data
echo HELLO2 >> hello.txt
echo HELLO3 > hello3.txt
```

- [Second Terminal]

```
ls -Z /opt
```

- [Container]

```
exit
```

SELINUX AND SHARED ACCESS FOR MULTIPLE CONTAINERS

- This time let's create several containers
  - Notice `:z` parameter instead of `:Z`

```
docker run -d --name selinux1 -v /opt/shared:/data:z nginx
docker run -d --name selinux2 -v /opt/shared:/data:z nginx
docker run -d --name selinux3 -v /opt/shared:/data:z nginx
ls -Z /opt
```

- Let's check if that works

```
docker exec -it selinux3 /bin/sh
cd data
echo HELLO2 >> hello.txt
echo HELLO3 > hello3.txt
exit
```

- Stop the containers:

```
docker stop selinux1 selinux2 selinux3
```

CLEANUP CONTAINERS

- List active and inactive containers

```
docker ps
docker ps -a
```

- Remove containers filtering by name

```
docker rm $(docker ps -a -q -f "name=.*data")
```

- Remove stopped containers

```
docker rm $(docker ps -a -q -f exited=0)
```

- Remove all containers (forcing stop for running ones)

```
docker rm -f $(docker ps -a -q)
```

CLEANUP IMAGES AND VOLUMES

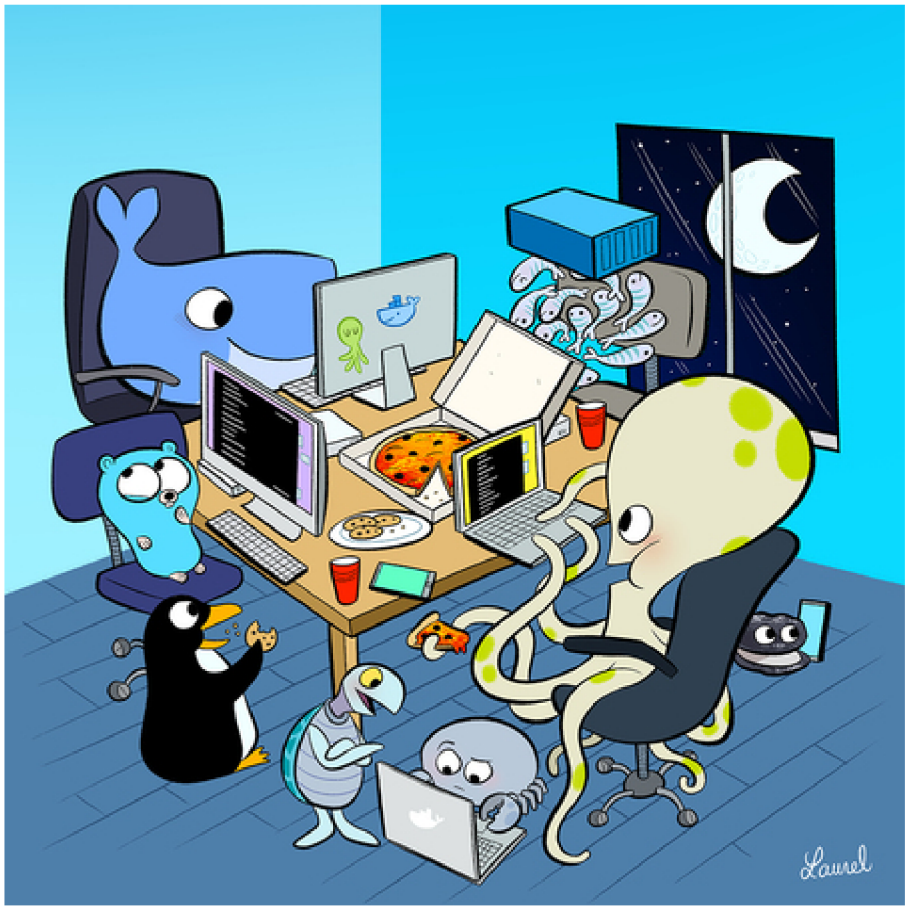
- List images and volumes

```
docker images
docker volume ls
```

- **The volume of the `nginx-data` container still exists!!**
- Remove orphaned images and volumes

```
docker rmi $(docker images -q -f dangling=true)
docker volume rm $(docker volume ls -q -f dangling=true)
```

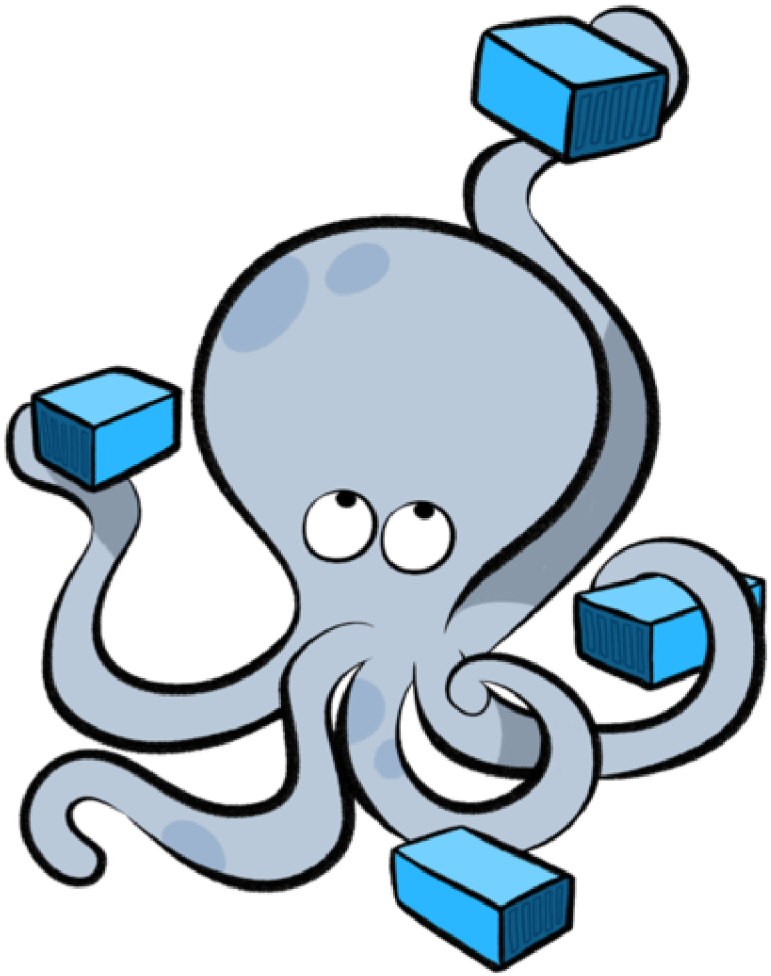
HANDS ON LAB 4



ADVANCED DOCKER

DOCKER COMPOSE

Compose is a tool for defining and running multi-container Docker applications



DOCKER COMPOSE FOOD TRUCKS

```
version: "2"
services:
  es:
    image: elasticsearch:2.3
    container_name: es
    networks:
      - foodtrucks
  web:
    build: .
    command: python app.py
    ports:
      - "5000:5000"
    volumes:
      - .:/code
    networks:
      - foodtrucks
    depends_on:
      - es
networks:
  foodtrucks:
    ipam:
      config:
        - subnet: 10.0.2.0/24
```

RUN

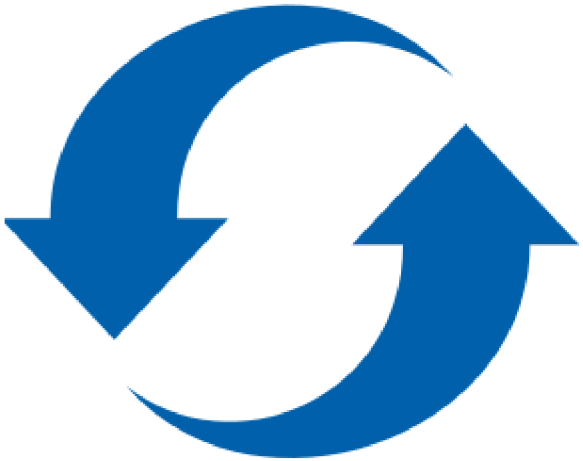
- Build

```
cd /opt/docker-introduction-data/food-trucks
docker-compose build
```

- Start

```
docker-compose up -d
```

CONTAINER UPDATES



TYPICAL CONTAINER ...

Let's have a look at a typical container:

Mongo

Let's have a look at another typical container:

Redis

UPDATES

Component	Update
Kernel	Host
Userspace libraries	Image / Container
Application from deb	Image / Container
Application from tar	Image

UPDATE DEADTIME

Now we'll check the update deadtime

```
docker pull mongo:3.2
docker run -d --name update_test mongo:3.2
docker pull mongo:3.4
/usr/bin/time bash -c "docker rm -f update_test && \
  docker run -d --name update_test mongo:3.4"
```

RESOURCE LIMITS

MEMORY LIMITS

- Docker container limits are based on cgroups: Docs
- We can specify the memory and swap limit. (Default: swap=memory)
  - This container can use 300M memory and 700M swap memory
  - From inside container it is hard to get the limit values

```
docker run -it -m 300M --memory-swap 1G --name mem_test1 ubuntu /bin/bash
free -h
```

- [Second terminal]

```
cat /sys/fs/cgroup/memory/system.slice/docker-$(docker inspect -f '{{.Id}}' mem_test1)
```

- [Container]

```
exit
```

MEMORY LIMITS

- Let's test if that works:

```
cd /opt/docker-introduction-data/stress
docker build --tag local/stress .
docker run --rm -m 128m local/stress --vm 1 --vm-bytes 312M --vm-hang 0
```

CPU LIMITS

option	value	result
--cpu-shares	1-1204	relative process weight used for CPU share allocation
--cpuset-cpus	core_ids: (0-1, 0,3)	CPU cores in which to allow execution
--cpuset-mems	mem_ids: (0-1, 0,3)	Memory nodes in which to allow execution

TEST

```
docker run -d --name cpu_test1 --cpuset-cpus=0 local/stress --cpu 1
docker run -d --name cpu_test2 --cpuset-cpus=1,2 \
  --cpu-shares 1024 local/stress --cpu 2
docker run -d --name cpu_test3 --cpuset-cpus=1,2 \
  --cpu-shares 512 local/stress --cpu 2
htop
docker stop cpu_test1 cpu_test2 cpu_test3
```

FIREWALL

FIREWALL

- Let's inspect the iptable rules

```
iptables -L -n -v
```

- Can you notice what is wrong with the INPUT chain?

FIREWALL

HOW DO WE RESTRICT ACCESS TO SERVICES RUNNING IN DOCKER?

- Put a firewall in front of docker host
- Insert PRE_DOCKER rules: PRE_DOCKER

HANDS ON LAB 5

EXCERCISE

BATCH PROCESSOR

- Create a docker image for automatic resizing of images
  - Reads and writes from current host directory
  - Executed as `docker run <bla> <bla>`
- Tips
  - Remember about SELinux
  - Use debian/ubuntu package imagemagick

```
mogrify -resize x100 -quality 100 -path /data/new-thumbs *
```

STEPS

- Write a docker file that:
  - Is based on ubuntu / debian
  - Installs imagemagick
  - Sets the execution command to `mogrify` with proper options
- Build the image
- Run the container in a host directory with images
  - Set proper volume option

IS THIS USEFUL?

- Not much with imagemagick :)
- Could be usefull when installation of a newer version of your compute application results in problematic dependencies
- Could be usefull to run several isolated instances of a compute application

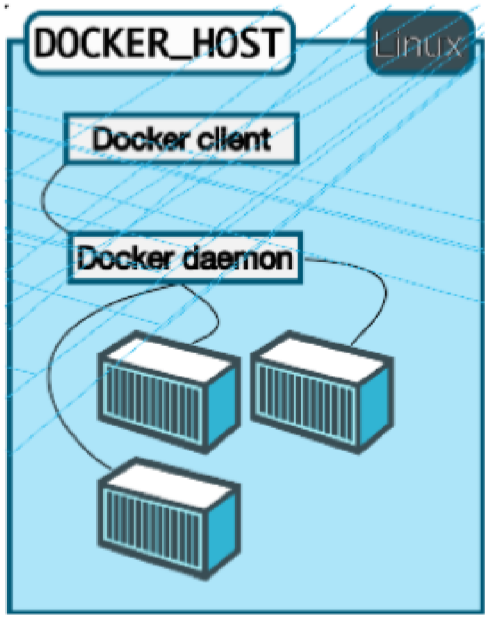
DOCKER TOOLS

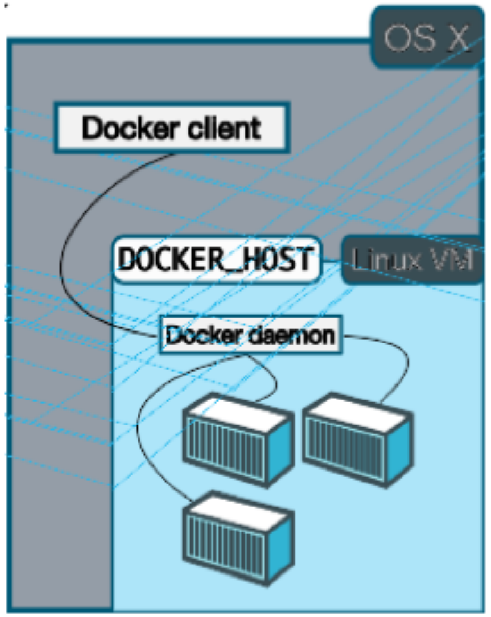
DOCKER REGISTRY @ CIŚ

DOCKER TOOLBOX

- The Docker Toolbox is an installer to quickly and easily install and setup a Docker environment on your computer.
- Available for both Windows and Mac, the Toolbox installs Docker Client, Machine, Compose (Mac only) and Kitematic.

DOCKER TOOLBOX





DOCKER MACHINE

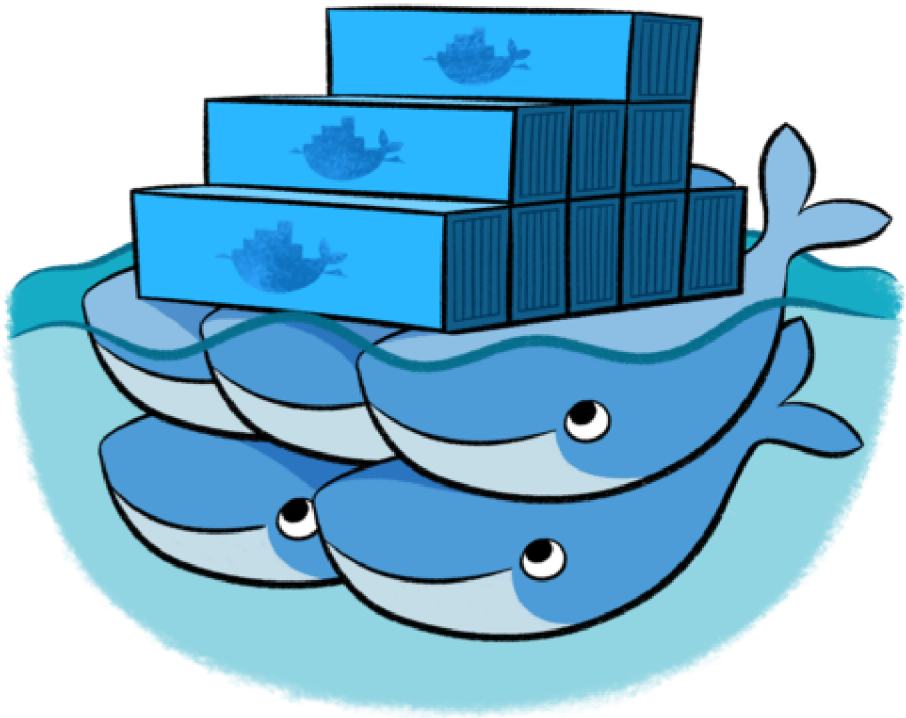
Docker Machine makes it really easy to create Docker hosts on your computer, on cloud providers, and inside your data center. It creates servers, installs Docker on them, then configures the Docker client to talk to them.

DOCKER COMPOSE

Compose is a tool for defining and running complex applications with Docker. With Compose, you define a multi-container application in a single file, and then you spin your application up in a single command which does everything that needs to be done to get it running.

DOCKER SWARM

Docker Swarm is native clustering for Docker. It turns a pool of Docker hosts into a single, virtual host.



THANKS


```
git clone https://github.com/cis-ncbj/docker-introduction.git
cd docker-introduction
docker run -d -p 80:80 --name slides \
  -v ${PWD}:/usr/share/nginx/html nginx
```