# Docker

## An Introduction to Application Deployment

University of Fribourg
Department of Informatics
Software Engineering Group

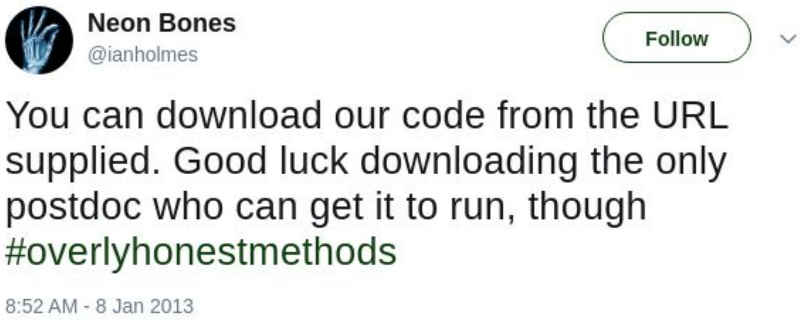
October 29, 2020

# Outline

1. Introduction
2. Concepts
3. Usage
4. Example
5. Conclusion

# Software Applications I

- Two important aspects of software applications are:
  - Reproducibility: How do I make sure that my application is functional, regardless of where it is running or who is running it?
  - Robustness to change: How do I make sure that my application will still be running if I change something?



Neon Bones
@ianholmes

You can download our code from the URL supplied. Good luck downloading the only postdoc who can get it to run, though #overlyhonestmethods

8:52 AM - 8 Jan 2013

https://twitter.com/ianholmes/status/288689712636493824

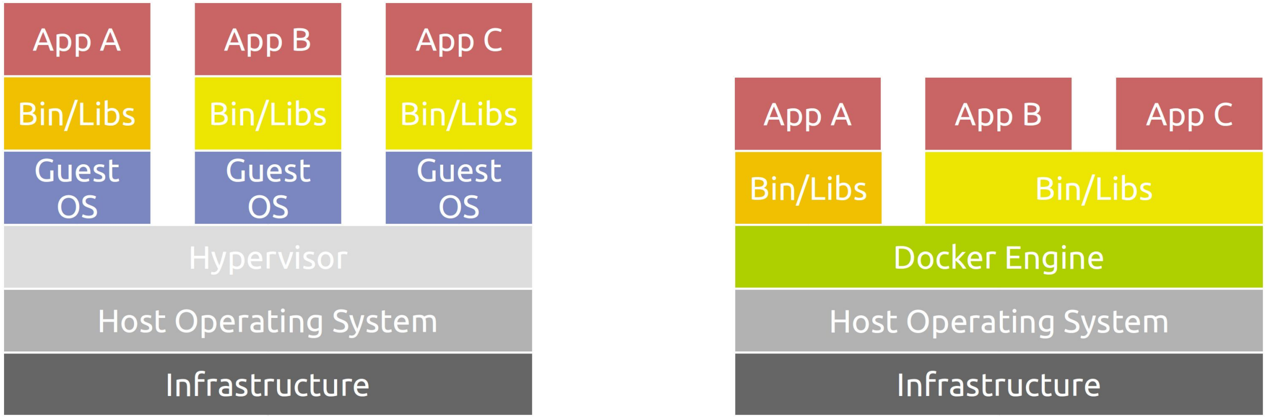
# Software Applications II

- Possible challenges to those aspects:
  - It may be complicated to set up the environment for the application (e.g. server setup).
  - The steps to set up the application may be unclear or difficult to reproduce (e.g. server migration, releasing to clients).
  - Different applications running on a same environment may need different versions of a library.
  - Updating server libraries may break retro-compatibility of applications.

- Is there a solution?

# Introducing Docker

- Open-source project released in 2013.
- Simplifies software deployment by guaranteeing consistency among different running environments.
  - Docker creates an abstraction of an application.
  - Applications are self-contained in (sort of) minimalistic Virtual Machines.



Virtual machines

Docker

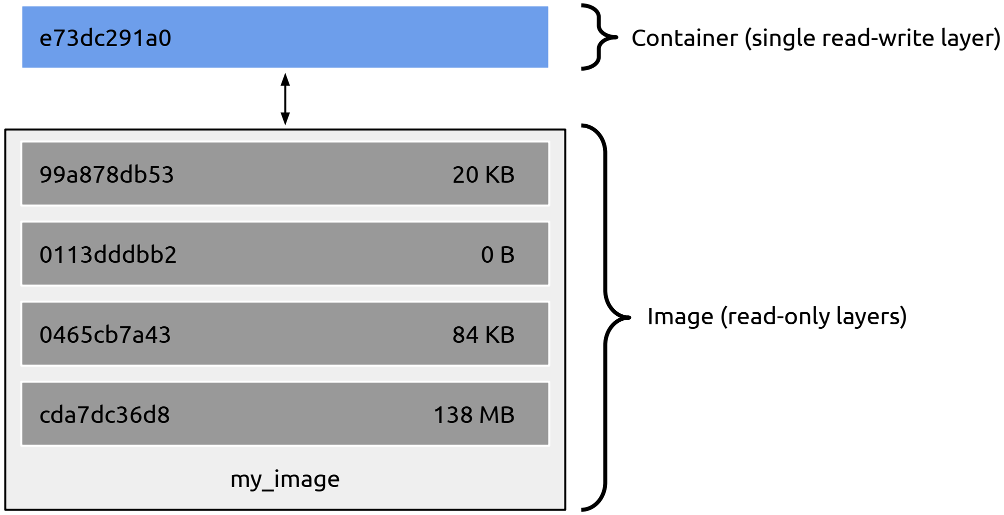
# Concepts

## Images and Containers

- Conceptually, Docker mainly consists of images and containers.
  - Images:
    - Are virtualized application
    - Are immutable
  - Containers:
    - Are an “instance” of an image
    - Can be stopped and restarted
    - Are given an arbitrary name when created

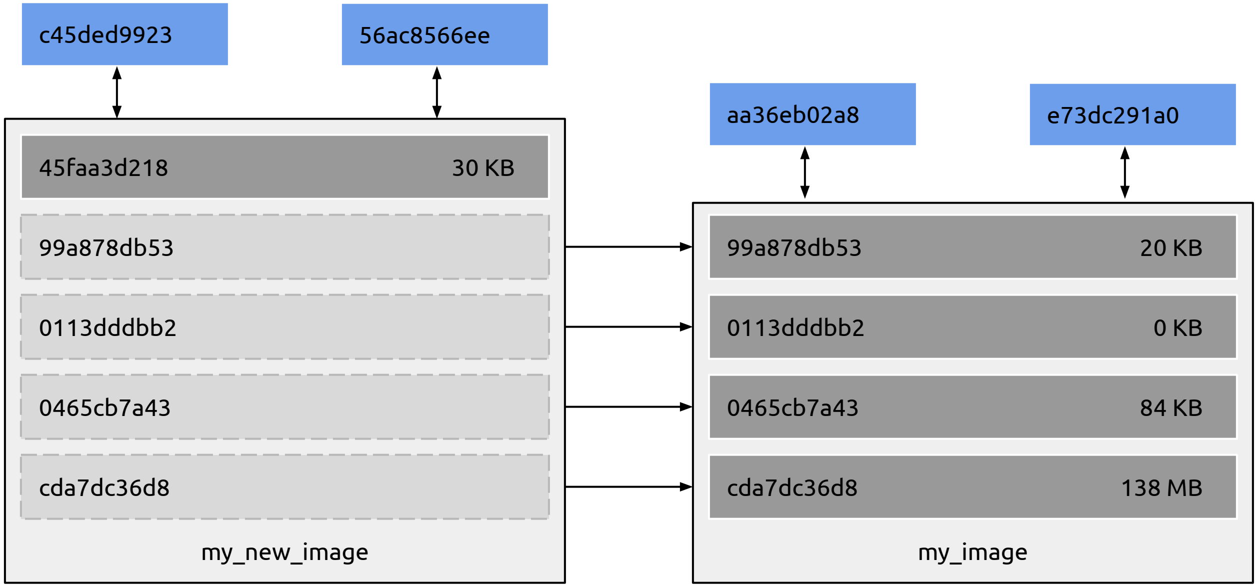
# Image Layers I

- Images are composed by a list of read-only filesystem layers.
- Each layer only contains the diff from the previous layer.
- Each layer is identified using a secure hash.
- A container is an extra writable layer on top of an image.
- There exist layers with no changes in filesystem (0 bytes).



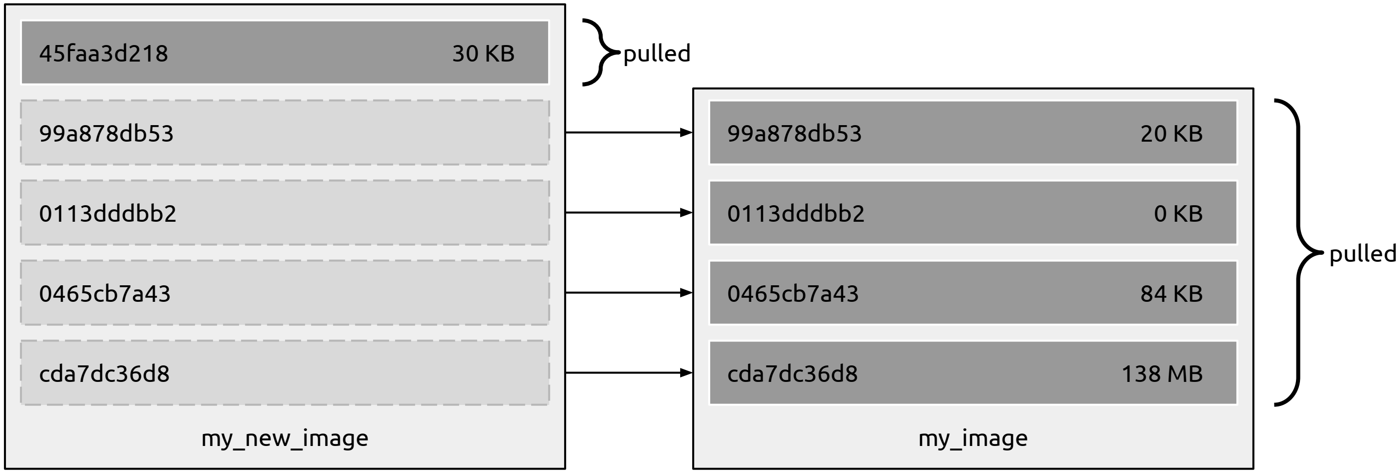
# Image Layers II

- Images are shared between different containers.
- Image layers can also be shared by different images.
  - Great for reusability and for saving memory.



# Image Layers III

- When you pull an image based on an existing image on your machine, Docker uses layer hashes to only pull the missing layers.
- It is not possible for an image to use only the top-layers of another image, since they are only diffs from the lower ones.



# Data Access I

## R/W Filesystem

- When a running container needs to access program data of its image, Docker searches from the top layers until it finds them.
  - If the container then needs to change these data, docker creates a local copy of them in the container layer.
- Containers are not supposed to be long-lived and should not be used to store actual data (i.e. databases, config files, etc).
  - Instead, data volumes are used.

# Data Access II

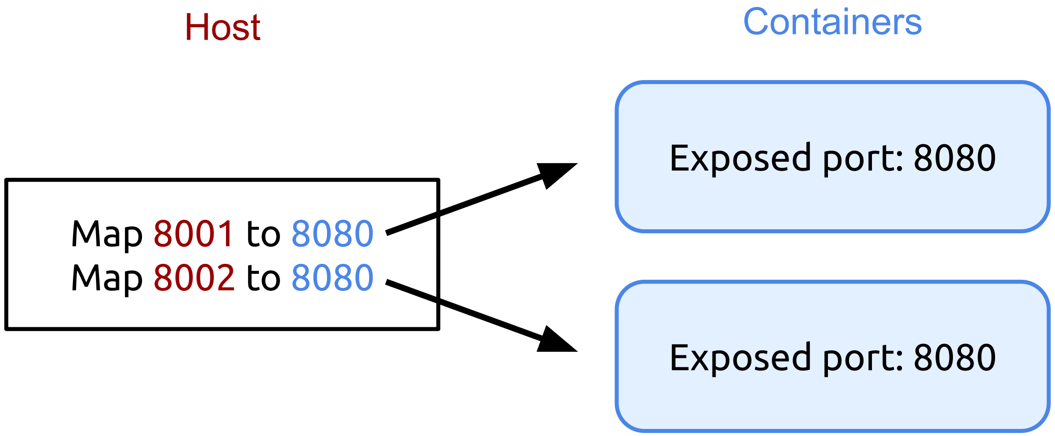
## Data Volumes

- Data volumes are directories (or files) on the host system.
- They have several purposes:
  - Data persistence after a container is deleted
  - Data exchange between a container and the host
  - Data exchange between multiple containers
- Data volumes are mounted directly into one or several containers.
- Multiple data volumes can be mounted in a single container.

# External Communication

## Network Ports

- Ports are used to communicate with a container.
  - As most applications are web-based, this method is well-suited.
- The ports defined inside the container have to be mapped to the outside.
  - This is essential to avoid conflicts.


Host

Containers

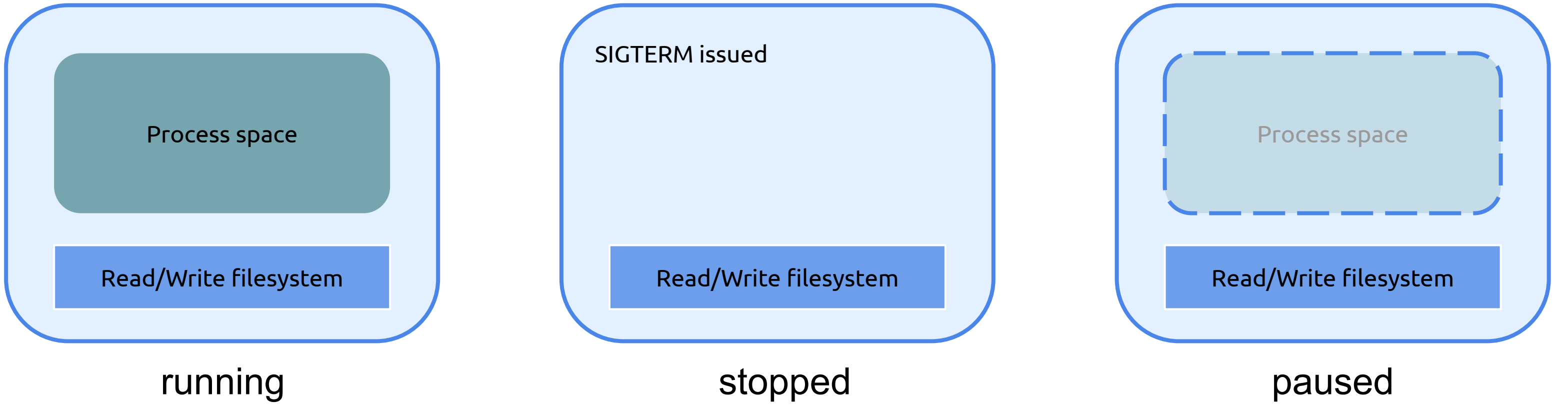
Map 8001 to 8080
Map 8002 to 8080

Exposed port: 8080

Exposed port: 8080

# Container States

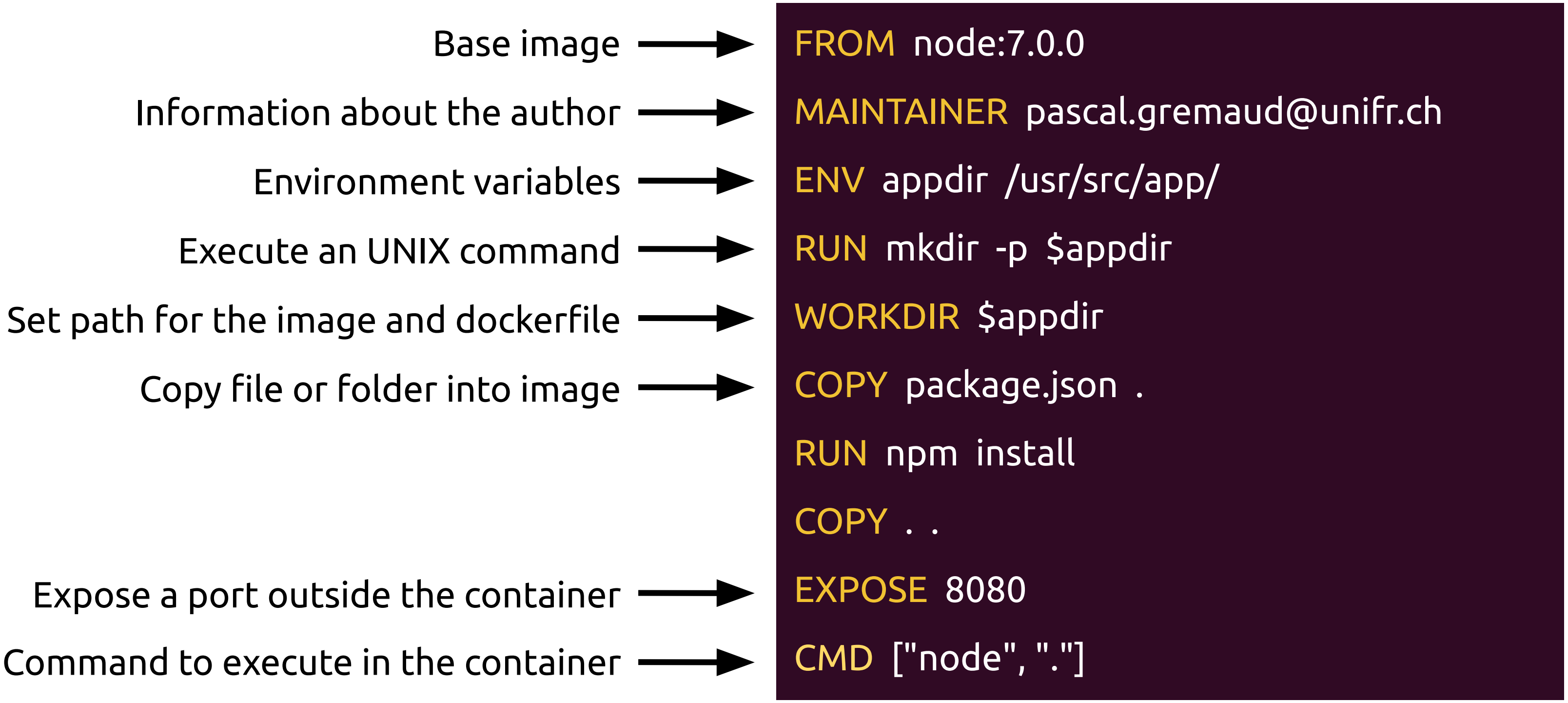
- A container can be in a number of states, mainly:
  - Running
  - Stopped (or Exited)
  - Paused
- A running container has its own process space.
- Containers can be paused, maintaining their process space.



# Building Images I

## Dockerfile

- Images are built using a *Dockerfile*, a set of instructions.
- Each command corresponds to an image layer.
- Your *Dockerfile* should be placed alongside your code.



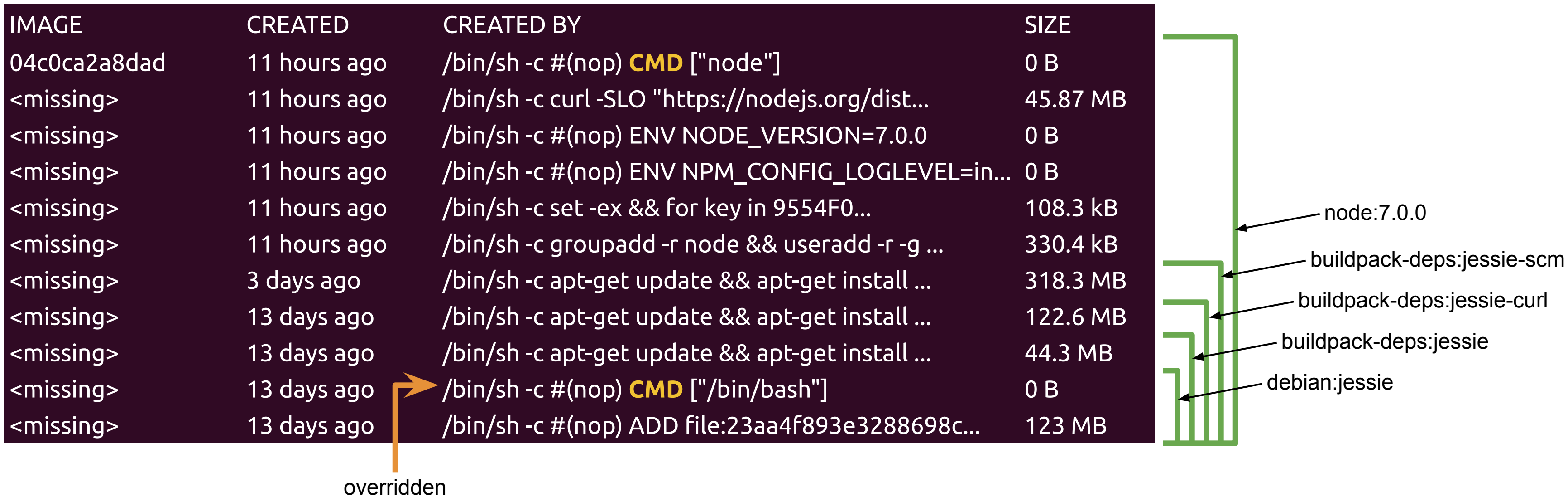
Base image
Information about the author
Environment variables
Execute an UNIX command
Set path for the image and dockerfile
Copy file or folder into image
Expose a port outside the container
Command to execute in the container

```
FROM node:7.0.0
MAINTAINER pascal.gremaud@unifr.ch
ENV appdir /usr/src/app/
RUN mkdir -p $appdir
WORKDIR $appdir
COPY package.json .
RUN npm install
COPY . .
EXPOSE 8080
CMD ["node", "."]
```

# Building Images II

## Commands and Layers

- The “CMD” command is a single, default command to execute when the container starts.
- If an image has a “CMD” command, it will override any “CMD” command from the images below it.
- Even the commands with no effect on the diffs create a layer.
- Here is the result of the docker history command on node:7.0.0



```
IMAGE          CREATED        CREATED BY                                       SIZE
04c0ca2a8dad   11 hours ago   /bin/sh -c #(nop) CMD ["node"]                   0 B
<missing>      11 hours ago   /bin/sh -c curl -SLO "https://nodejs.org/dist...  45.87 MB
<missing>      11 hours ago   /bin/sh -c #(nop) ENV NODE_VERSION=7.0.0         0 B
<missing>      11 hours ago   /bin/sh -c #(nop) ENV NPM_CONFIG_LOGLEVEL=in...  0 B
<missing>      11 hours ago   /bin/sh -c set -ex && for key in 9554F0...       108.3 kB
<missing>      11 hours ago   /bin/sh -c groupadd -r node && useradd -r -g ... 330.4 kB
<missing>      3 days ago     /bin/sh -c apt-get update && apt-get install ... 318.3 MB
<missing>      13 days ago    /bin/sh -c apt-get update && apt-get install ... 122.6 MB
<missing>      13 days ago    /bin/sh -c apt-get update && apt-get install ... 44.3 MB
<missing>      13 days ago    /bin/sh -c #(nop) CMD ["/bin/bash"]              0 B
<missing>      13 days ago    /bin/sh -c #(nop) ADD file:23aa4f893e3288698c... 123 MB
```

# Registries

- Registries are used to share images.
- Different versions of an image are grouped in a repository, as with git.
- The most commonly used registry is *Dockerhub* (*www.dockerhub.com*)
  - It is used by default when pulling an image.
- You can also push your own built images to *Dockerhub*.

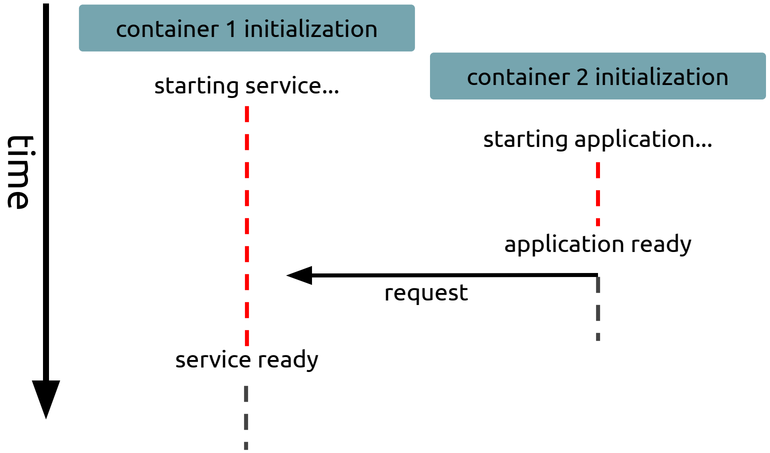
# Docker Compose I

- Sometimes it is better (or needed) to separate application components in different containers.
- Docker Compose is used to orchestrate these elements.
- It uses a *docker-compose.yml* file.
- If a container uses a service in another container (e.g. a database), some extra scripts may be needed to ensure that the service is already up and running before the application starts.

```
first_container:
  build: my_custom_image
  ports:
    - 8002:8080
  depends_on:
    - second_container
second_container:
  image: image1
  ports:
    - 8001:8080
  volumes:
    - “/data:/data/db”
```

# Docker Compose II

- Docker starts your containers one after the others, without waiting.
  - This can lead to dependency issues (e.g. a container trying to communicate via another, not set up yet container).
  - This issue needs to be solved in your code, not by Docker.



# Additional Features

- Networks
  - It is possible to ask docker to create networks for “direct” communication between containers, even on different hosts.
- Docker Swarm
  - Grouping of several distant host machines into a single virtual host in order to scale up applications.
- Automated builds with *Github* (or *Bitbucket*)
  - Image automatically built and published on *Dockerhub* (not possible to use with Docker-Compose).

# Outline

1. Introduction
2. Docker - Concepts
3. **Docker - Usage**
4. Example

# Containers Manipulation I

## Basic Commands

- Docker commands: docker [options] [command] [arg..]
- Starting a container (and possibly pulling it):
  docker run -d -p <host_port>:<container_port> <image>
  - -d is used to run in detached mode
  - -p is used to map ports
- Once started, the container is associated with a hash and a name.
  - To interact with a container, use either of them.
- See the list of non-stopped containers:
  docker ps
  - -a to show all containers
  - -q to show only the hash of each container

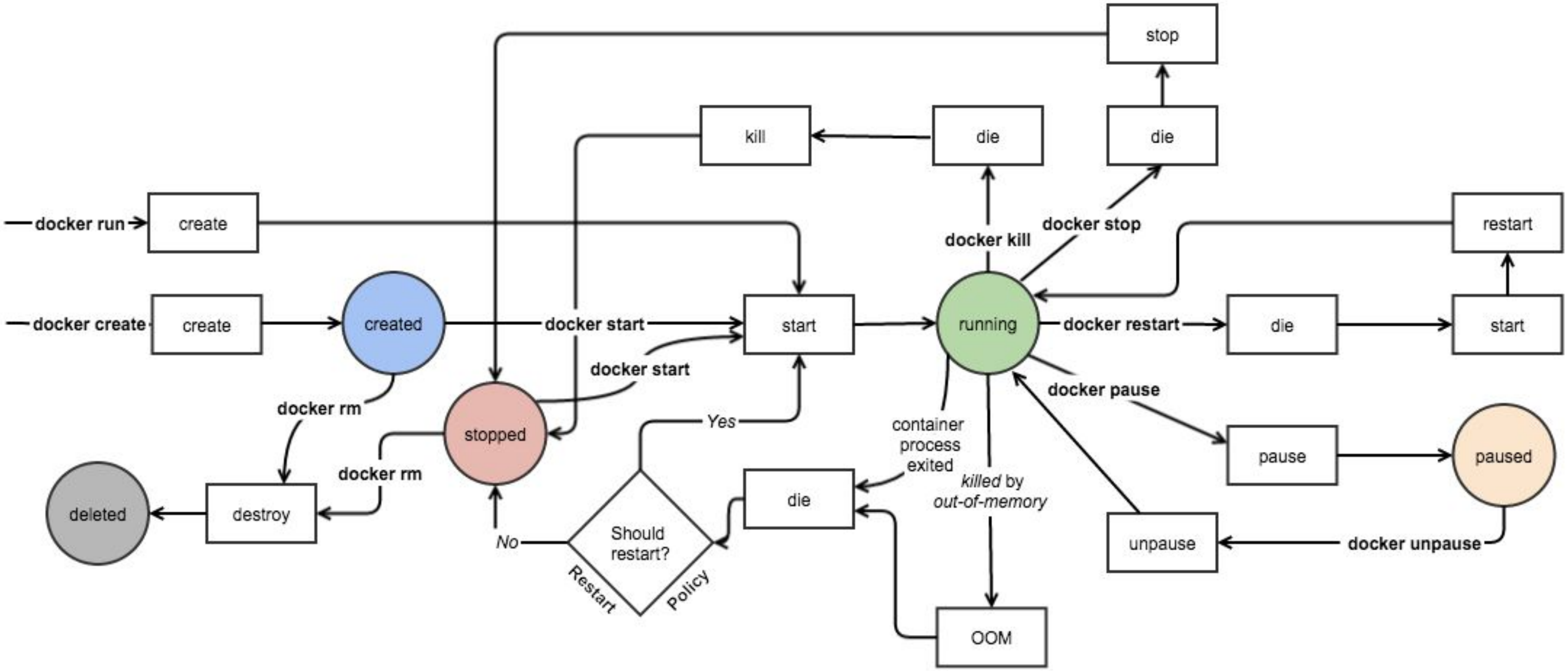
# Containers Manipulation II

## Other docker Commands

- create: create a new container from an image (no running)
  - If no local image is found and no registry is specified, the image is pulled from *Dockerhub*.
- run: create a new container from an image and run it
  - Same pulling mechanism
- rm: destroy a stopped container
- start: run a stopped container
- stop: stop a running container
- restart: stop and start a running container
- pause: pause a running container
- unpause: unpause a paused container

# Containers Manipulation III

## Container States and Transition Commands



https://medium.com/@nagarwal/lifecycle-of-docker-container-d2da9f85959

# Containers Manipulation IV

- Stop all containers:
  `docker stop $(docker ps -q)`
  - `$(docker ps -q)` gives the list of non-stopped containers
- Destroy all containers (stopped):
  `docker rm $(docker ps -qa)`
- Execute a command inside a container (not recommended):
  `docker exec <options> <container> <command>`
- Open a terminal inside a container (debug only!!!):
  `docker exec -ti <container> /bin/bash`

# Images Manipulation I

- Download an image:
  `docker pull <image>`
- See the list of “top-level” images:
  `docker images`
  - -a to show all images
  - -q to show only the hash of each image
- Remove an image:
  `docker rmi <image>`
- Remove all images:
  `docker rmi $(docker images -qa)`
- Get the stack of an image:
  `docker history <image>`

# Images Manipulation II

## Creating Images

- In order to build your image, run this command:

```
docker build -t <image_name>[:<tag_name>] <location>
```

- For instance, when inside your project at the level of your *Dockerfile*:

```
docker build -t docker-node-example:latest .
```

- You can then push your image to a registry (e.g. *Dockerhub*):

```
docker login
docker tag <image>:<tag> <username>/<repo>[:<tag>]
docker push <username>/<repo>[:<tag>]
```

# Outline

1. Introduction
2. Docker - Concepts
3. Docker - Usage
4. **Example**

# Step 1

## Setup

- In this example we will:
  - Build an image
  - Push it to Dockerhub
  - Create an automated build for it
- By now you should have:
  - Docker installed on your machine
  - An account on *Github*
  - An account on *Dockerhub*
  - Pulled the node:7.0.0 image

# Step 2

## Creating a Node App

- Go to https://github.com/polchky/docker-node-example and fork the repository.
- Clone your forked repository on your machine and cd into it.
- We will not build our Nodejs app locally but directly create an image.

# Step 3

## Building an Image

- Study the *Dockerfile*.
- To build your image, open a terminal and run

```
docker build -t docker-node-example .
```

- Your image should now appear in docker.
- Run your image by executing

```
docker run -d -p 9000:8080 docker-node-example
```

  - You can now access your app at http://localhost:9000
  - On OSX, you need to replace localhost by the IP of your docker machine.
    - Get it by executing docker-machine config default
- Play around with your container(s), exec into it.

# Step 4

## Pushing to *Dockerhub*

- Login to *Dockerhub*:

```
docker login
```

- Tag your image:

```
docker tag docker-node-example \
<username>/docker-node-example
```

- Push your image to *Dockerhub*:

```
docker push <username>/docker-node-example
```

# Step 5

## Creating an Automated Build

- On *Dockerhub*, go to your account settings, and then *Linked Accounts*.
  - Link your Github account, with read-write access.
- Go to your pushed docker image, *Builds*.
- Create a new automated build using Github.
- Select your github username and your docker-node-example.
- Take a look at the behavior customization.
- Save the automated build.
- Each time you push to the master branch (default behavior) on Github, a build of your image is triggered on *Dockerhub*.
  - You can also trigger a build manually.

# Conclusion

- Docker and its tools suite can be a noticeable improvement to standard software delivery and orchestration methods.
- Docker may be still young, but is in active development and has a consequent community.

- You may consider deploying and delivering your project using Docker and Docker-Compose.
  - This will be considered as an extra "feature" of your project.
  - However, be careful to include the "dockerization" time in your planning to avoid any surprises.

# External Resources

## Some Useful Links

- The official docker documentation:
  https://docs.docker.com
- Docker commands cheat sheet:
  https://github.com/wsargent/docker-cheat-sheet
- A good explanation and visual representation of the concepts of layers and processes:
  http://merrigrove.blogspot.ch/2015/10/visualizing-docker-containers-and-images.html