Introduction to Docker

RES, Lecture 5

Olivier Liechti

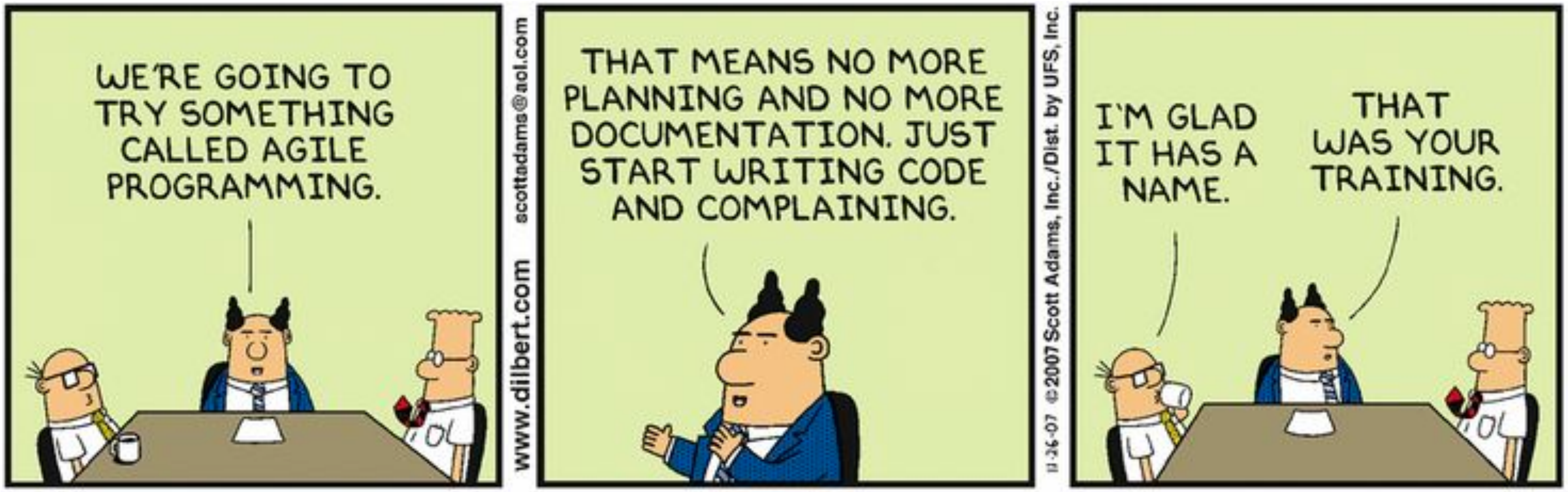
heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

Agenda

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- **Software engineering trends**
  - Agile software development, continuous delivery and DevOps
  - Automation and “Infrastructure as Code”
- **An introduction to Docker**
  - Why Docker in the RES course?
  - The engine, the images and the containers
  - Finding and using existing images
  - Creating your own images

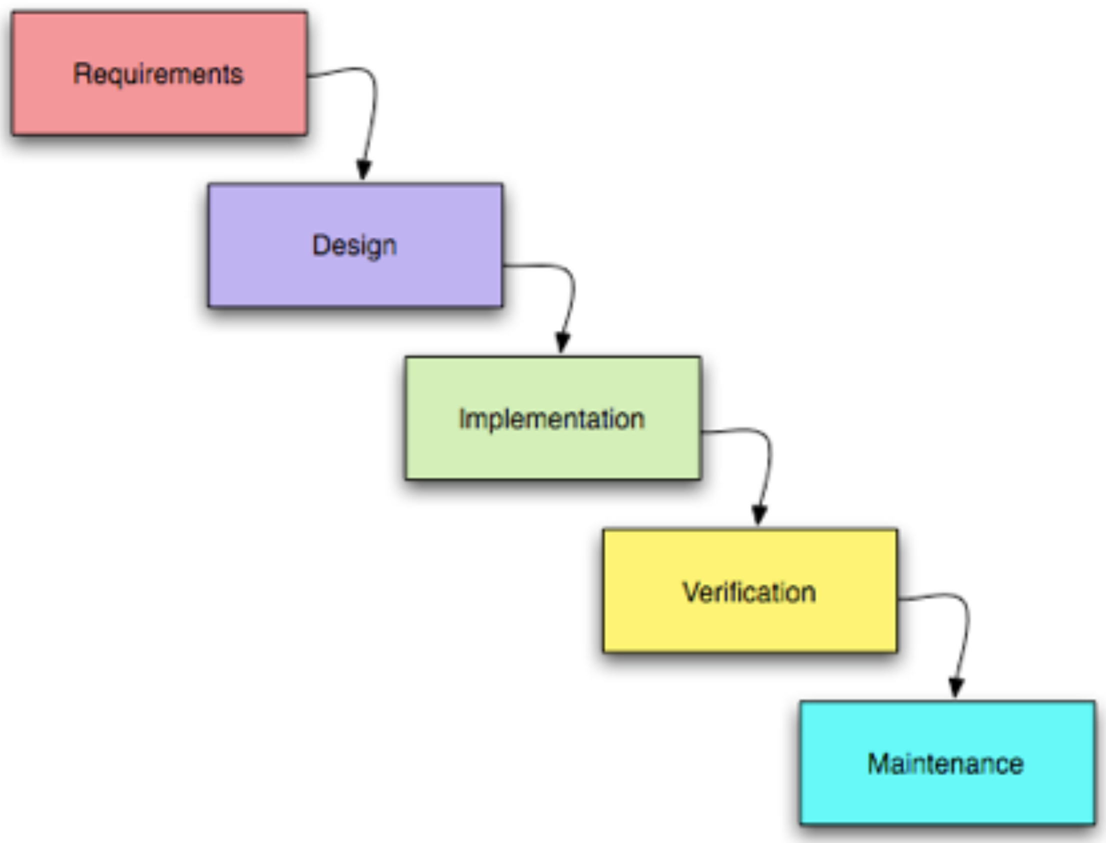
heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

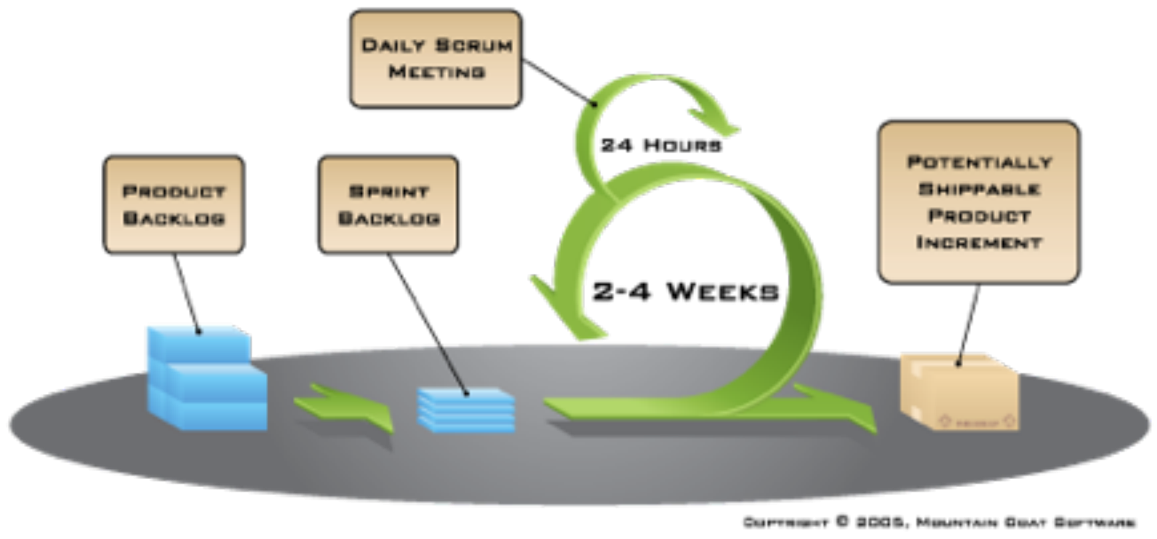

WE'RE GOING TO TRY SOMETHING CALLED AGILE PROGRAMMING.
THAT MEANS NO MORE PLANNING AND NO MORE DOCUMENTATION. JUST START WRITING CODE AND COMPLAINING.
I'M GLAD IT HAS A NAME.
THAT WAS YOUR TRAINING.
www.dilbert.com scottadams@aol.com
11-26-07 ©2007 Scott Adams, Inc./Dist. by UFS, Inc.

Agile Software Development

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- You are probably familiar with the “**waterfall model**”.
- Sequencing the development activities in these **successive phases** is not a problem.
- There are however important questions to ask yourself when you see this plan:
  - **How much time** do we spend in every phase?
  - During the project, **how many times** do we go through these phases?
  - **Who** is working in each phase?
- The answers to these questions define how “**agile**” you are, as opposed to an organisation that follows “traditional” development methodologies.




Daily Scrum Meeting
24 Hours
Product Backlog
Sprint Backlog
2-4 Weeks
Potentially Shippable Product Increment
Copyright © 2005, Mountain Goat Software

Agile Software Development

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

	Traditional	Agile
How much time do we spend in each phase?	**A lot of time**. In big projects, one could image to spend 6 months writing specs, 12 months designing the software, 24 months implementing it and 6 months testing it.	**A little time**. We only specify, design, implement and validate one feature at the time. Therefore, we don't need a lot of time to do each of these activities.
How many times do we go through these phases?	**Only once**. We *believe* that we can anticipate all user needs, all technical issues and that we don't need feedback until the end of the project.	**Many times (it is a cycle)**. We work in short iterations (1-2 weeks). In each iteration, we deliver an increment of functionality.
Who is working in each phase	**People from different parts of the company**. There is a "product" department, which hands over specs to an "architecture" department, which hands over design documents to a "development" department, which hands over to a "QA" department, which gives the green light to an "IT" department.	The **same small team works together** on all aspects of the product. If you have a large product, then you have multiple autonomous "feature teams" who work on different parts of the product.

Continuous Delivery

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- **Continuous delivery** is one of the “hot topics” in software engineering.
- It is an extension of the “**continuous integration**” practice that was made popular with eXtreme Programming (XP).
- The objective is to be able to move from a “feature idea” to a “feature available to end-users” ***very quickly*** and ***safely*** (i.e. without breaking things).
- Companies are now setting up “**continuous delivery pipelines**”, which **automate** all the **construction** and **validation** phases of the software.
- Simply stated: “You commit a code change. The modification enters the pipeline. The new version of the software is built and validated with unit tests, integration tests, functional tests, performance tests, etc.”. **At the end of the process, you know if you have a “deployable” new version**.
- A continuous delivery pipeline can be extended to support **continuous deployment** (Amazon deploys code to production many, many times a day).

Infrastructure as Code

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

- To build a continuous delivery pipeline, you need a **lot of different tools**.
- **Cloud** and **virtualization technologies** have changed many things:
  - In the past, many **system administration** activities required manual intervention. For software developers, many of them were happening "behind the scenes" and performed by **"strangers"**.
  - Today, **everything has an API** and can be programmed. Software developers can **"program infrastructure"**.
- **Docker** is one of that has gained a lot of popularity over the last 2-3 years.
  - In this context, it is used to create entire **multi-nodes environments** automatically and on the fly.
  - It **makes the validation of software more robust** (the test environment is not "too" different from production environment).

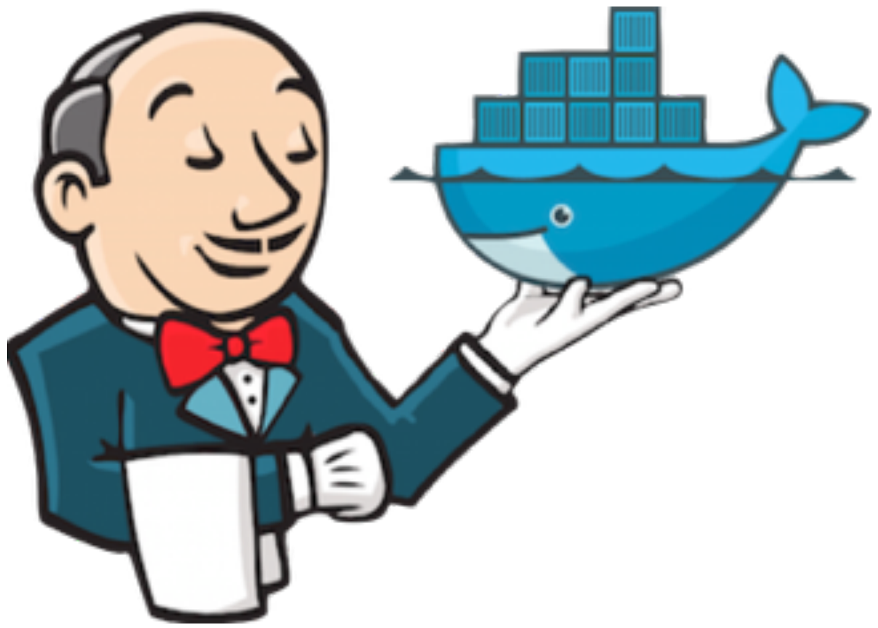
heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Why is it important for me? I am a (TR | TS | IE | IL) student...

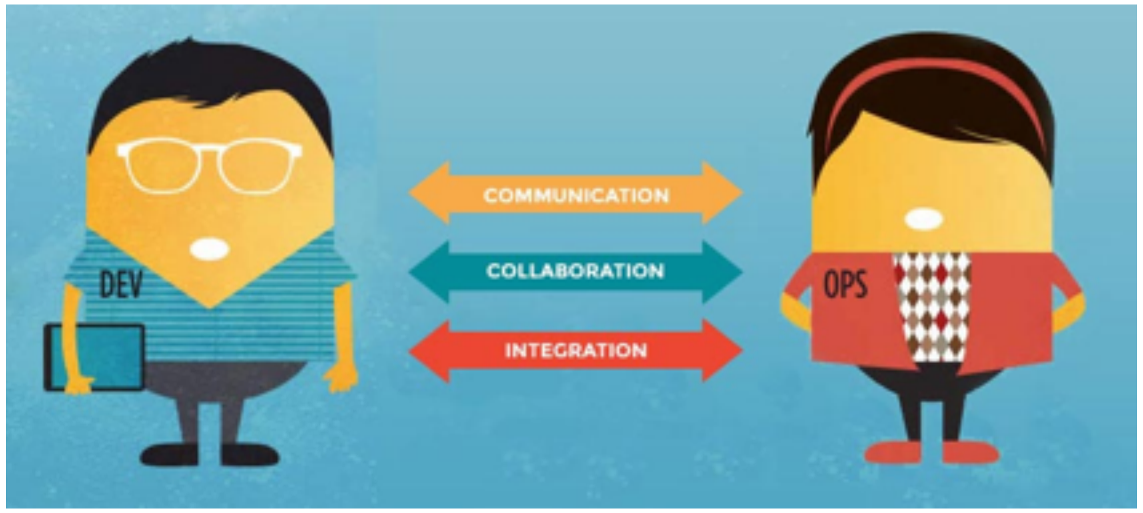
Because it means that many IT jobs are changing. New approaches are available. New skills are required (you have a ***big opportunity****!).*

Agile teams should be small. There should be no gap between "development" and "IT operations".

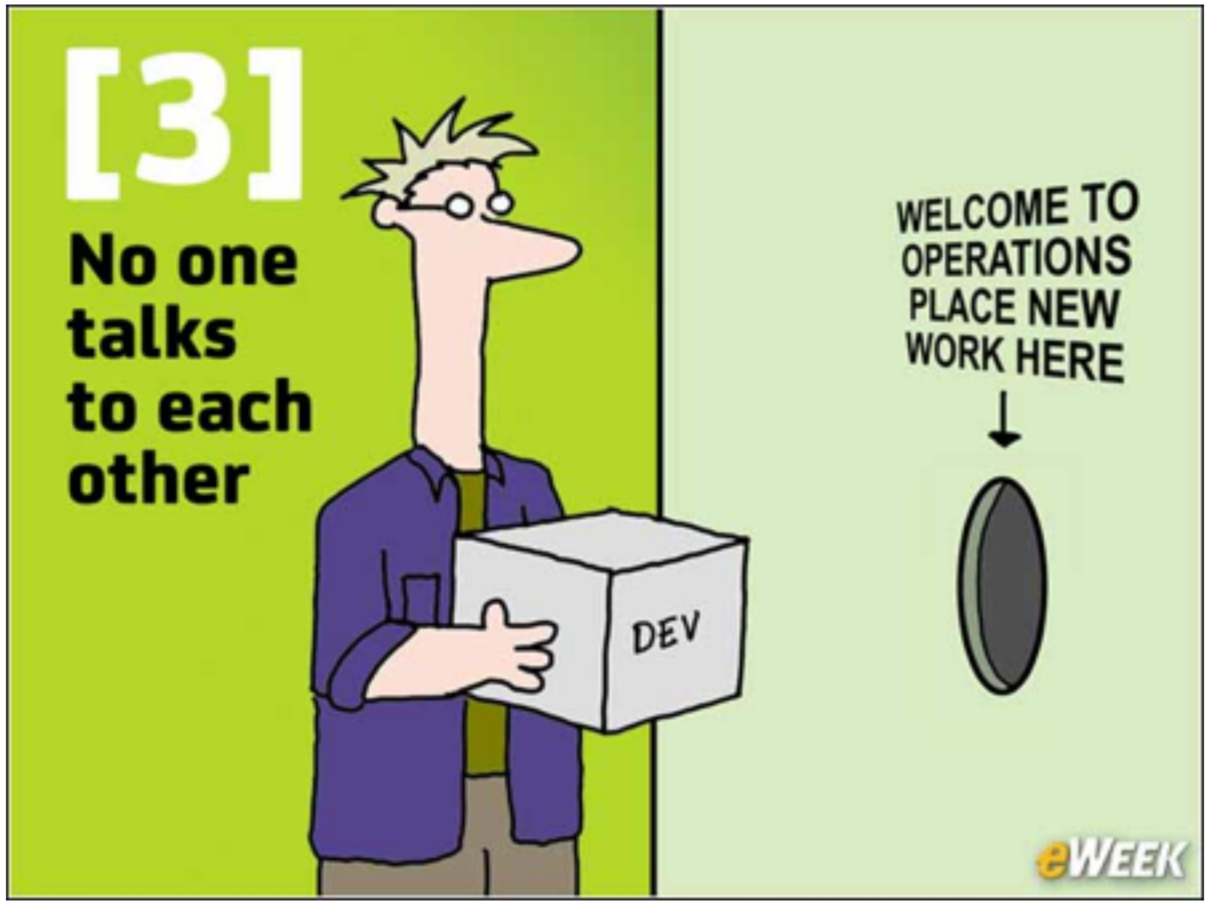
Some of the roles are blending.

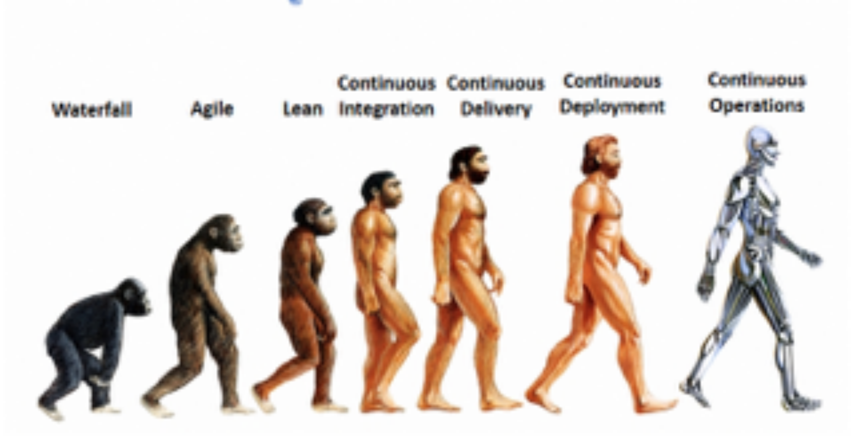


DevOps


DEV
COMMUNICATION
COLLABORATION
INTEGRATION
OPS

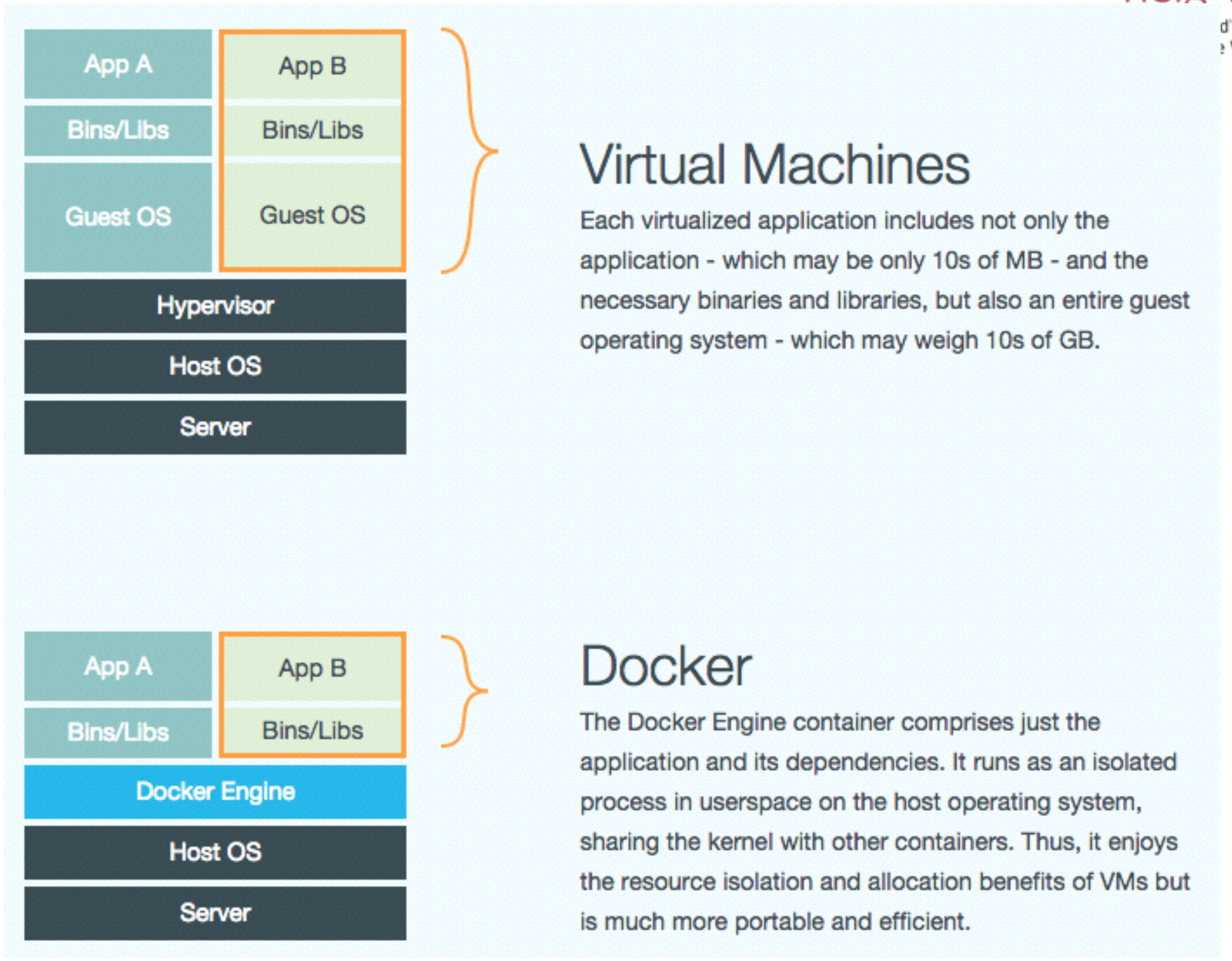
DevOps
DevOps Movement


[3]
No one talks to each other
DEV
WELCOME TO OPERATIONS PLACE NEW WORK HERE
eWEEK


Waterfall
Agile
Lean
Continuous Integration
Continuous Delivery
Continuous Deployment
Continuous Operations

I want change!
Wall of Confusion
I want stability!
Development
Operations

docker



Virtual Machines

Each virtualized application includes not only the application - which may be only 10s of MB - and the necessary binaries and libraries, but also an entire guest operating system - which may weigh 10s of GB.

Docker

The Docker Engine container comprises just the application and its dependencies. It runs as an isolated process in userspace on the host operating system, sharing the kernel with other containers. Thus, it enjoys the resource isolation and allocation benefits of VMs but is much more portable and efficient.

https://www.docker.com/whatisdocker/

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Docker image

Docker container

Why Docker in this course?

- Docker is a **lightweight virtualization technology**.
- It enables a new way for **packaging**, **distributing** and **running applications**.
- There are **different use cases** for the technology. Continuous delivery is one of the use cases.
- In this course, we will take advantage of Docker for these reasons:
  - It allows us to run multiple “very small” virtual machines (containers) on our machine. **Every container has its own IP address** and is connected to a **virtual network**.
  - This will allow us to create systems with **UDP clients and servers** and experiment with application protocols that rely on **multicast** or broadcast.
  - This will also allow us to create HTTP infrastructures, with proxies, load-balancers, static content servers, dynamic content servers, etc.

Docker on your machine

- You have two options for using Docker:
  - Firstly, you can use the Vagrant VM. Docker is installed (and a couple of images have be downloaded already). Since the VM was prepared at the beginning of the semester, it is not the latest version of Docker.
  - You can install “**Docker Machine**” on your host. This also works if you have a Windows or a Mac OS machine. In this setup, Docker Machine actually uses a VM similar to the Vagrant VM (it is smaller).

Docker containers (1)

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

- You can think of a container as a **lightweight** virtual machine. Each container is isolated from the others and has its own IP address.
- Each container is created from a docker image (one could say that a container is a **running instance of an image**).
- There are **commands** to start, list, stop and delete containers.

```
# Start a container (more on this later)
$ docker run

# List running containers
$ docker ps

# List all containers
$ docker ps -a

# Delete a container
$ docker rm

# Display logs produced by a container
$ docker logs
```

Docker containers (2)

- With Docker, the philosophy is to have **one application-level service per container** (it is not a strict rule).
- With Docker, the philosophy is also to **run several (many) containers on the same machine**.
- If you think of a typical **Web Application infrastructure**, you would have one or more containers for the apache web server, one container for the database, one container for the reverse proxy, etc.
- With Docker, containers tend also to be **short-lived**. Each container has an **entry point**, i.e. a command that is executed at startup. When this command returns, the container is stopped (and will typically be removed).
- **If a container dies, it should not be a big deal**. Instead of trying to fix it, one will create a new one (from the same image).

Docker images (1)

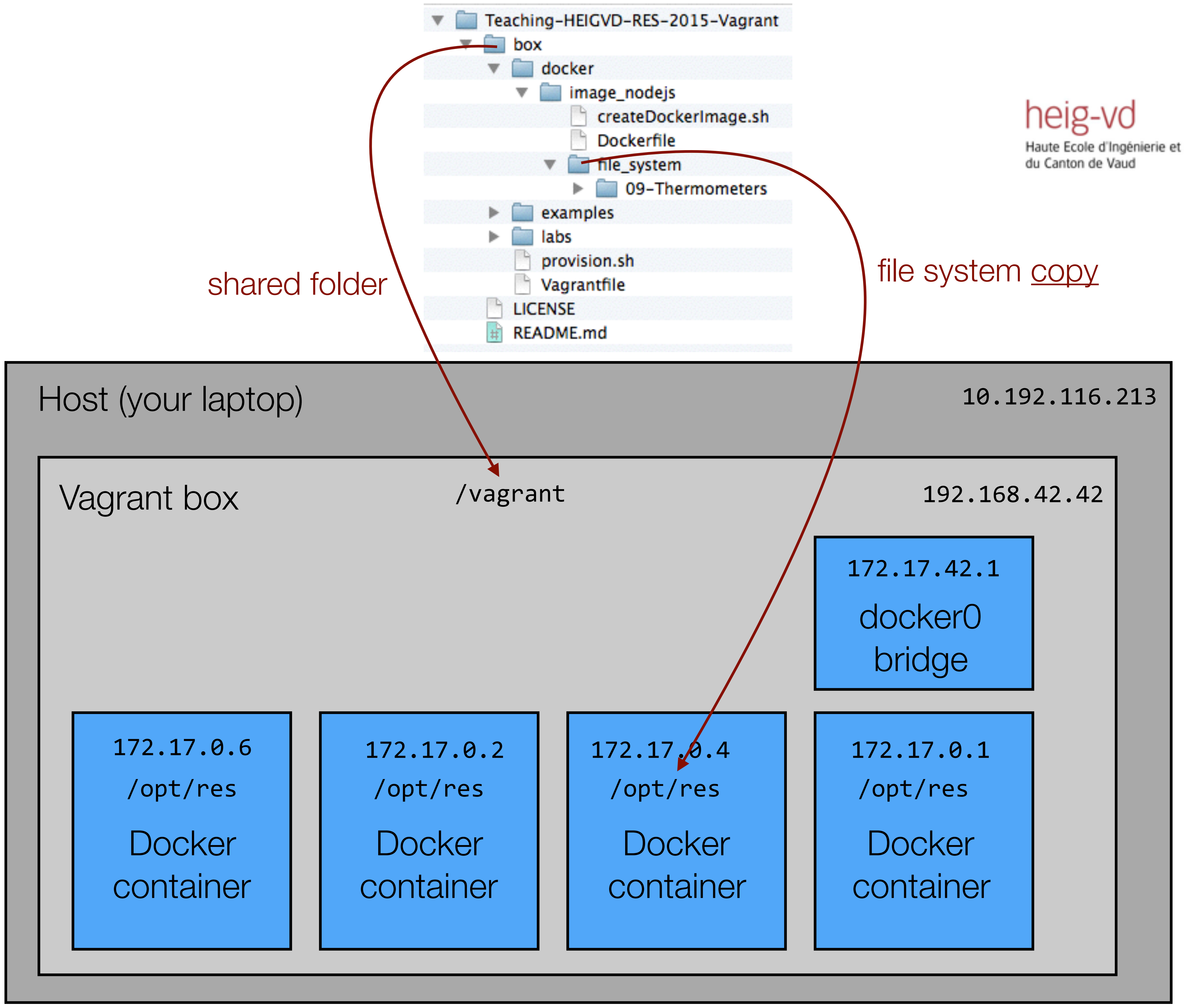
- **A Docker image is a template**, which is used to create containers.
- Every image is **built from a base image** and adds its own configuration and content.
- With Vagrant, we use a file named Vagrantfile to configure and provision a Vagrant box. With Docker, we use a file name **Dockerfile** to create an image. The file contains statements (FROM, RUN, COPY, ADD, EXPOSE, CMD, VOLUME, etc.)
- Just like the community is sharing Vagrant boxes, **the community is sharing Docker images**. This happens on the Docker Hub registry (https://registry.hub.docker.com/).
- You can build docker images locally (with the `docker build` command). You can also build them on the **Docker Hub** cloud service.

Docker images (2)

- Here is an example for a Dockerfile

```
# This image is based on another image

FROM dockerfile/nodejs:latest

# For information: who maintains this Dockerfile?

MAINTAINER Olivier Liechti

# When we create the image, we copy files from the host into
# the image file system. This is NOT a shared folder!

COPY file_system /opt/res/

# With RUN, we can execute commands when we create the image. Here,
# we install the PM2 process manager

RUN npm install -g pm2@0.12.9
```

```
# Create an image from this Dockerfile
$ docker build -t heigvd/res-demo .

# Execute /bin/bash in a new container, created from the image
$ docker run -i -t heigvd/res-demo /bin/bash
```



heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

Activity 1: use an existing Docker image (java server)

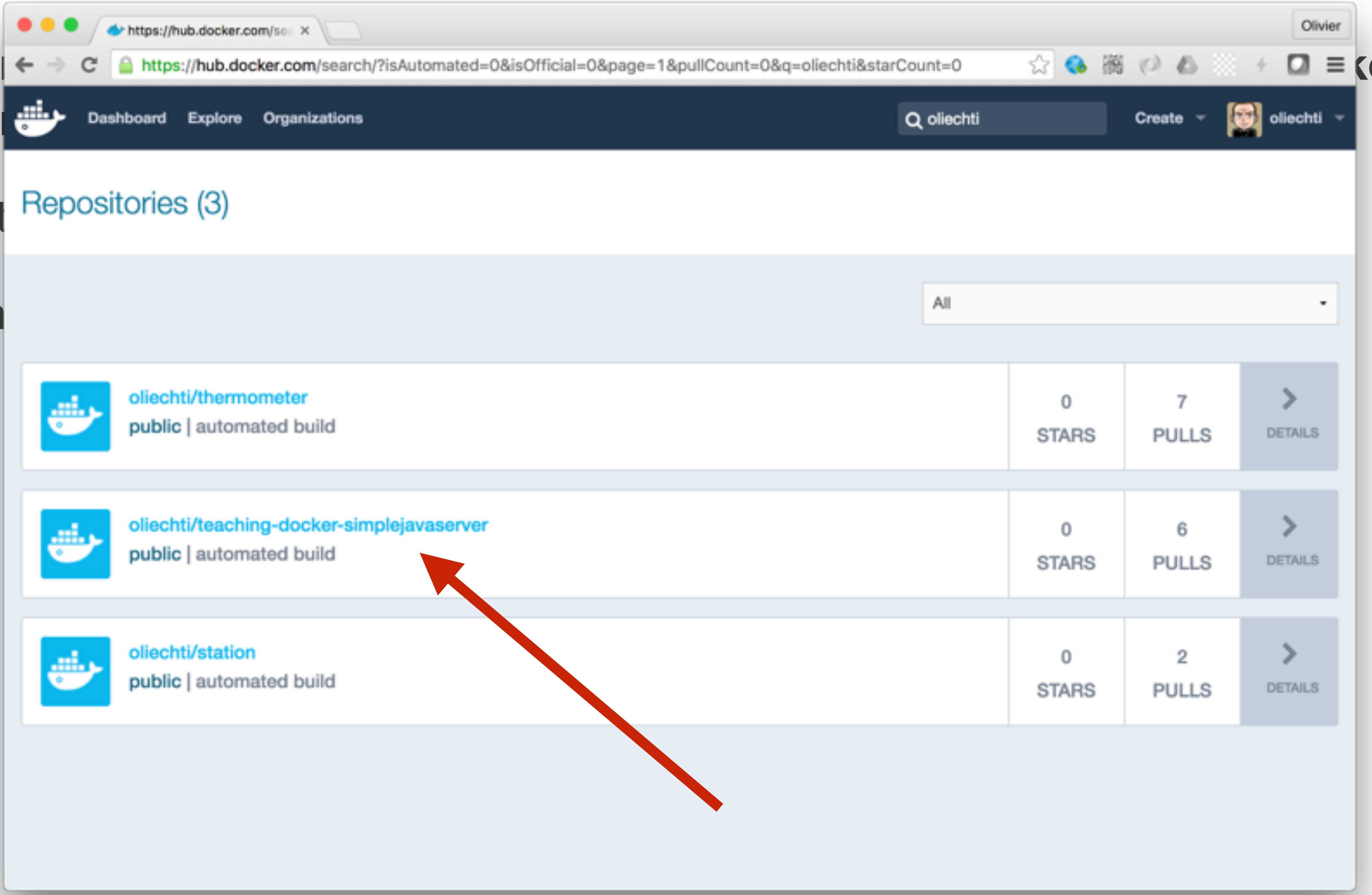
Use an existing Docker image

1. Find an image on **Docker Hub**
2. Understand that **the image packages an application** (read the **Dockerfile**). Understand that this application is a TCP server written in Java.
3. Start a **container** (map the **container port** to a VM port)
4. **Interact with the server** running in the container.

Use an existing Docker image

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

1. Find an image on **Docker Hub**
2. U **kerfile**).
   U
3. St
4. **In**


https://hub.docker.com/se
Olivier
https://hub.docker.com/search/?isAutomated=0&isOfficial=0&page=1&pullCount=0&q=oliechti&starCount=0
Dashboard
Explore
Organizations
oliechti
Create
oliechti
All
DETAILS

Repositories (3)

Repository	Stars	Pulls
oliechti/thermometer public | automated build	0 STARS	7 PULLS
oliechti/teaching-docker-simplejavaserver public | automated build	0 STARS	6 PULLS
oliechti/station public | automated build	0 STARS	2 PULLS

Use an existing Docker image

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

1. Find an image on **Docker Hub**

2. Understand that **the image packages an application** (read the **Dockerfile**). Understand that this application is a TCP server written in Java.

```
FROM java:8
MAINTAINER Olivier Liechti <olivier.liechti@heig-vd.ch>

#
# When we build the image, we copy the executable jar in the image file system.
#
COPY StreamingTimeServer-1.0-SNAPSHOT-standalone.jar /opt/app/server.jar

#
# Our application will accept TCP connections on port 2205. Note that the EXPOSE statement
# does not make the container accessible via the host. For the container to really be accessible,
# we must either use the -p or the -P flag when using the docker run command. With -p, we can
# specify an explicit port mapping (and EXPOSE is not even required). With -P, we let Docker
# assign random ports for each EXPOSEd port. We can then use the docker port command to know the port
# numbers that have been selected.
#
EXPOSE 2205

#
# This is the command that is executed when the Docker container starts
#
CMD ["java", "-jar", "/opt/app/server.jar"]
```

Use an existing Docker image

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

1. Find an image on **Docker Hub**

2. Understand that **the image packages an application** (read the **Dockerfile**). Understand that this application is a TCP server written in Java.

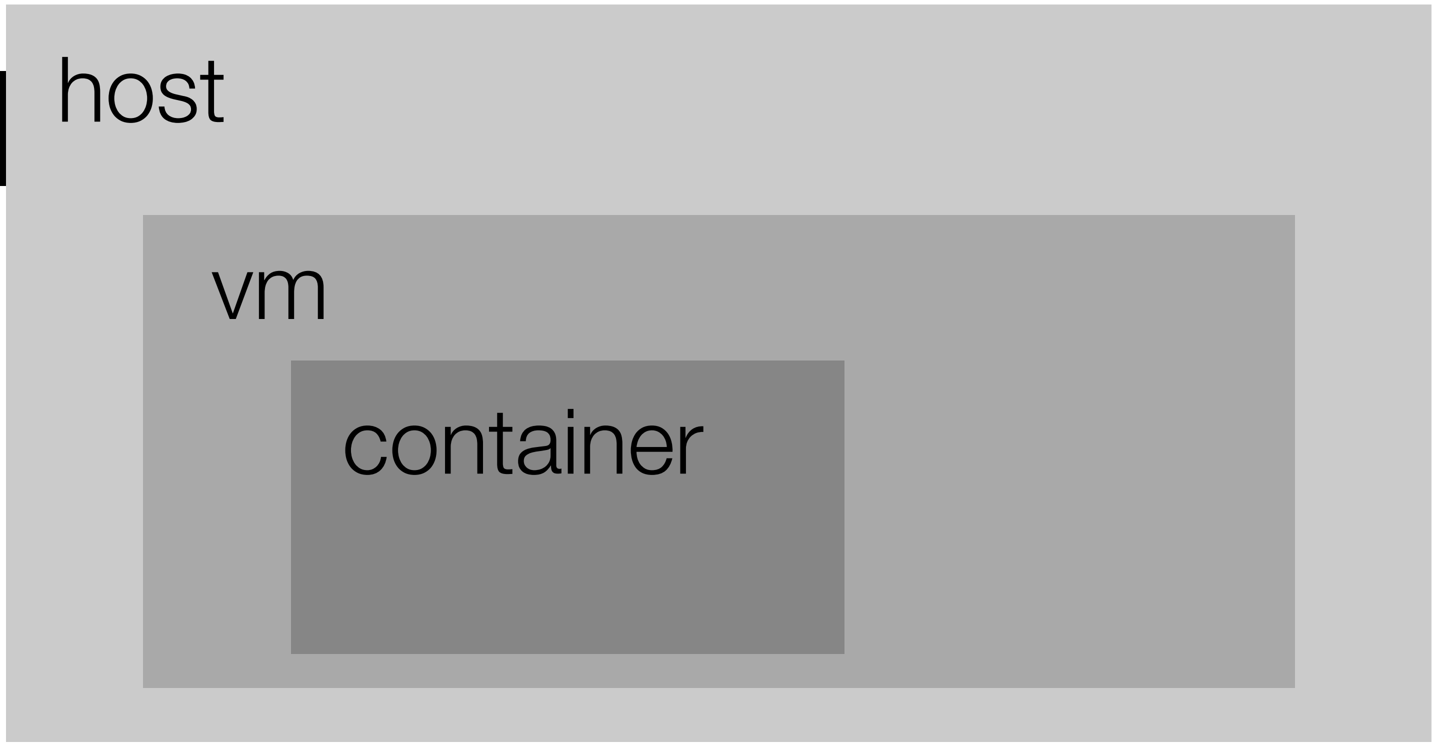
3. Start a **container** (map the **container port** to a VM port)

```
# Start a container (more on this later)
$ docker run -p 2205:2205 oliechti/teaching-docker-simplejavaserver

# List running containers
$ docker ps

# List all containers
$ docker ps -a
```

Use an existing Docker image

1. Find an image on **Docker Hub**
2. Understand that **the image packages an application** (read the **Dockerfile**). Understand that this application is a TCP server written in Java.
3. Start a **container** (map the **container port** to a VM port)
4. **Interact with the server** running in the container.

```
$ telnet ??? 2205
```



heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Activity 2: start several containers from 1 image

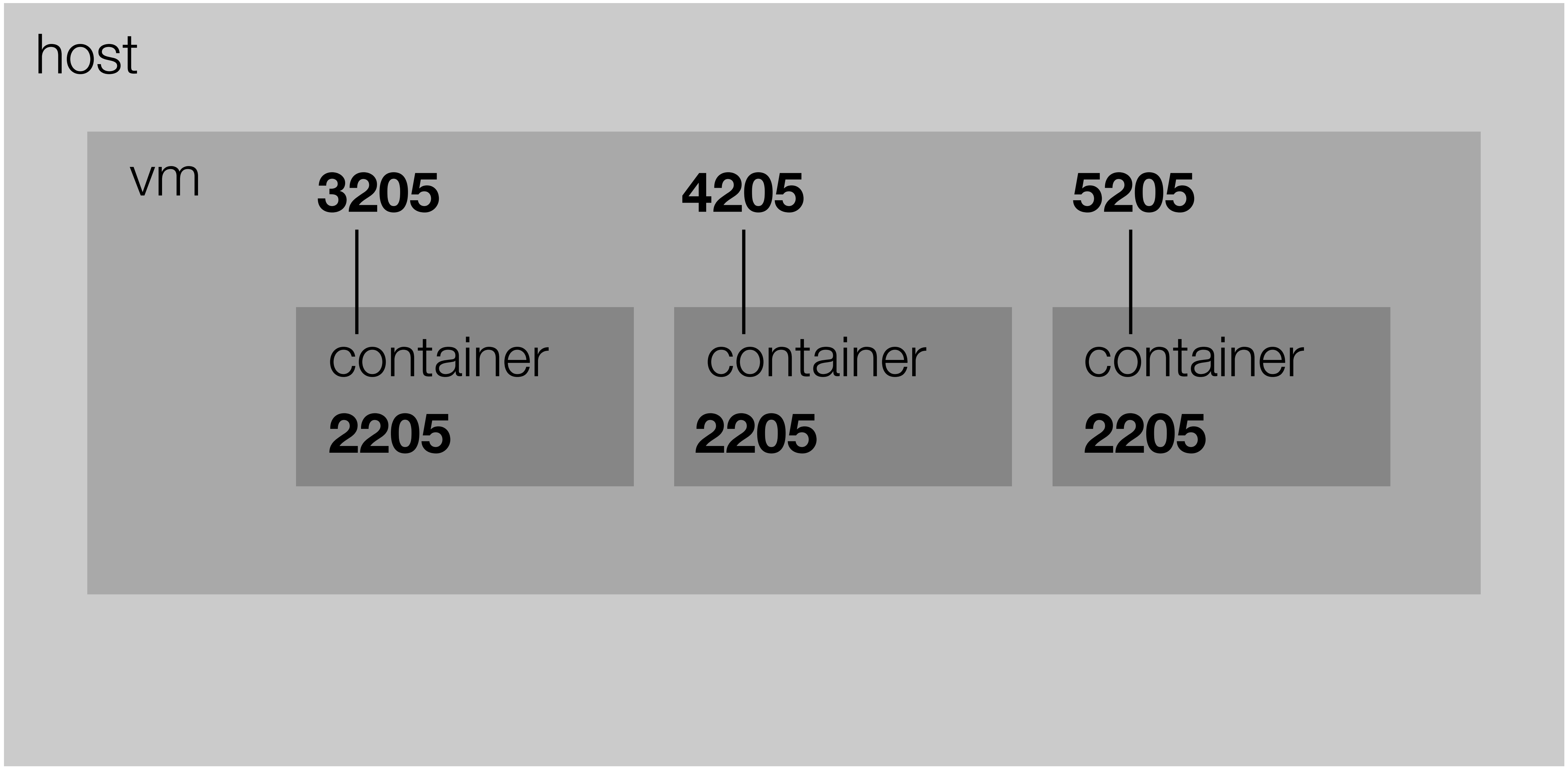
Multiple containers from 1 image

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

1. Understand that **we need multiple ports** on the VM.
2. Start container 1, with port mapping 1
3. Start container 2, with port mapping 2
4. Start container 3, with port mapping 3
5. Interact with the 3 servers

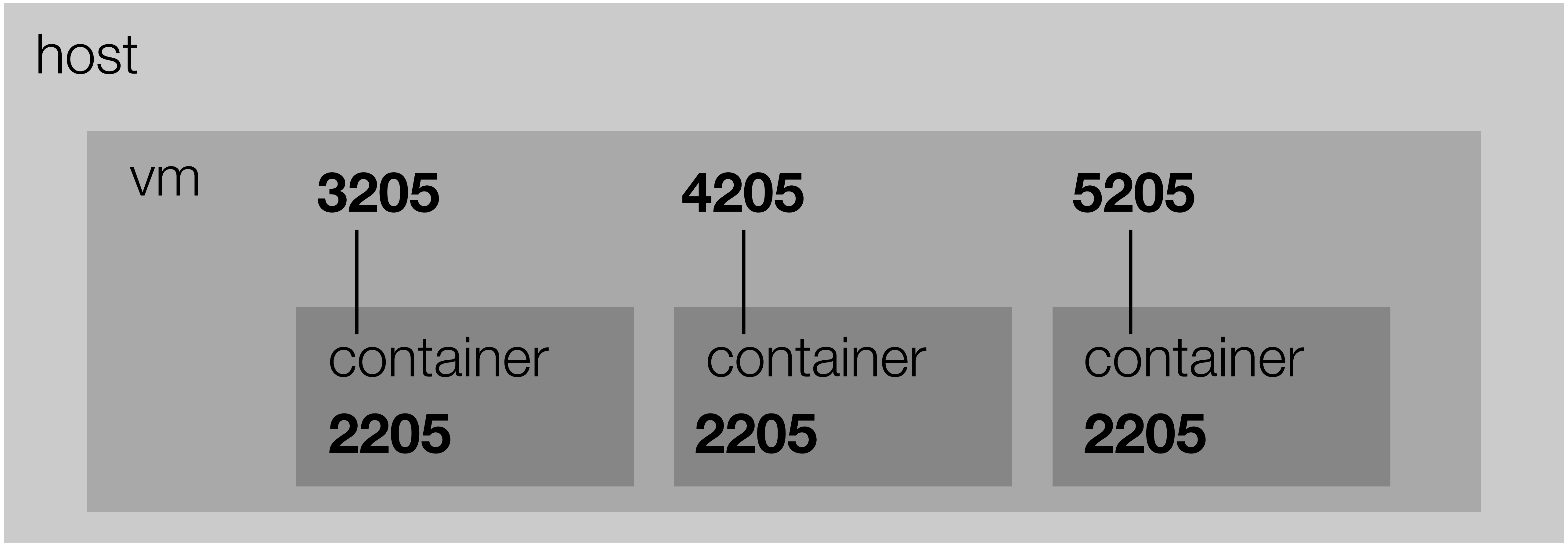
Multiple containers from 1 image

1. Understand that **we need multiple ports** on the VM.



Multiple containers from 1 image

1. Understand that **we need multiple ports** on the VM.



```
$ docker run -p 3205:2205 oliechti/teaching-docker-simplejavaserver
$ docker run -p 4205:2205 oliechti/teaching-docker-simplejavaserver
$ docker run -p 5205:2205 oliechti/teaching-docker-simplejavaserver
```

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Activity 3: use the UDP images

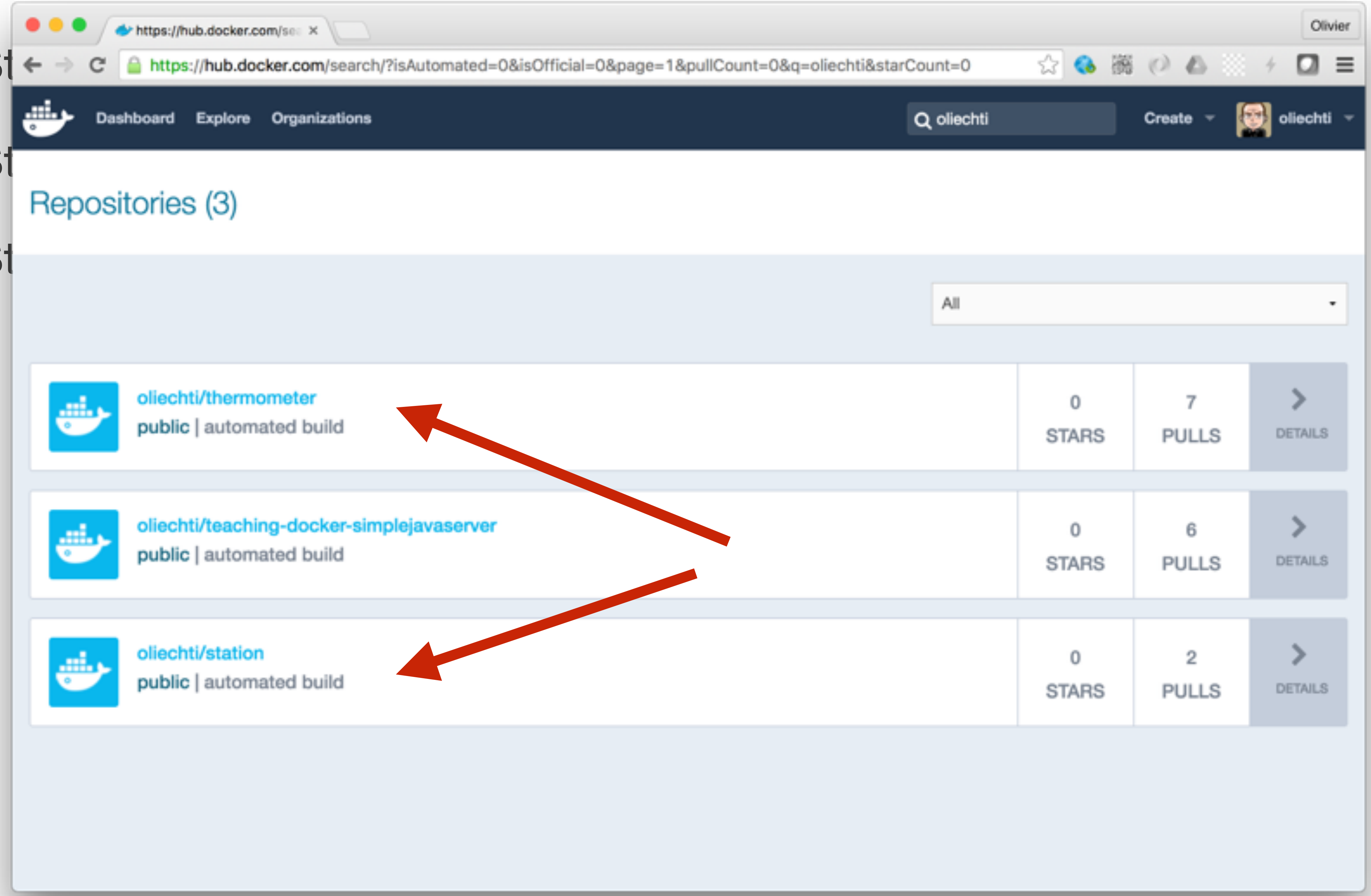
Multiple containers from 1 image

1. Find the images on Docker Hub
2. Start a “thermometer” container and observe.
3. Start a “station” thermometer and observe.
4. Start a second “thermometer” container and observe.

Multiple containers from 1 image

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

1. Find the images on Docker Hub
2. St
3. St
4. St


https://hub.docker.com/se
Olivier
https://hub.docker.com/search/?isAutomated=0&isOfficial=0&page=1&pullCount=0&q=oliechti&starCount=0
Dashboard
Explore
Organizations
oliechti
Create
oliechti
All
DETAILS

Repositories (3)

oliechti/thermometer public | automated build	0 STARS	7 PULLS	DETAILS
oliechti/teaching-docker-simplejavaserver public | automated build	0 STARS	6 PULLS	DETAILS
oliechti/station public | automated build	0 STARS	2 PULLS	DETAILS

Multiple containers from 1 image

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

1. Find the images on Docker Hub
2. Start a “thermometer” container and observe.
3. Start a “station” thermometer and observe.
4. Start a second “thermometer” container and observe.

```
$ docker run -p 2205:2205 oliechti/thermometer

$ docker run -p 2205:2205 oliechti/station

$ docker run -p 2205:2205 oliechti/thermometer
```

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Activity 4: analyze the JavaScript code

```
/*
 This program simulates a "smart" thermometer, which publishes the measured temperature
 on a multicast group. Other programs can join the group and receive the measures. The
 measures are transported in json payloads with the following format:

   {"timestamp":1394656712850,"location":"kitchen","temperature":22.5}

 Usage: to start a thermometer, type the following command in a terminal
        (of course, you can run several thermometers in parallel and observe that all
        measures are transmitted via the multicast group):

   node thermometer.js location temperature variation

*/

var protocol = require('./sensor-protocol');

/*
 * We use a standard Node.js module to work with UDP
 */
var dgram = require('dgram');

/*
 * Let's create a datagram socket. We will use it to send our UDP datagrams
 */
var s = dgram.createSocket('udp4');

process.on('SIGINT', function() {
    process.exit();
});
```

```
/*
 * Let's define a javascript class for our thermometer. The constructor accepts
 * a location, an initial temperature and the amplitude of temperature variation
 * at every iteration
 */
function Thermometer(location, temperature, variation) {

       this.location = location;
       this.temperature = temperature;
       this.variation = variation;

/*
   * We will simulate temperature changes on a regular basis. That is something that
   * we implement in a class method (via the prototype)
   */
       Thermometer.prototype.update = function() {
              var delta = this.variation - (Math.random() * this.variation * 2);
              this.temperature = (this.temperature + delta);

/*
         * Let's create the measure as a dynamic javascript object,
         * add the 3 properties (timestamp, location and temperature)
         * and serialize the object to a JSON string
         */
              var measure = {
                     timestamp: Date.now(),
                     location: this.location,
                     temperature: this.temperature
              };
              var payload = JSON.stringify(measure);

/*
          * Finally, let's encapsulate the payload in a UDP datagram, which we publish on
          * the multicast address. All subscribers to this address will receive the message.
          */
              message = new Buffer(payload);
              s.send(message, 0, message.length, protocol.PROTOCOL_PORT, protocol.PROTOCOL_MULTICAST_ADDRESS, function(err, bytes) {
                     console.log("Sending payload: " + payload + " via port " + s.address().port);
              });

       }

/*
        * Let's take and send a measure every 500 ms
        */
       setInterval(this.update.bind(this), 500);

}
```

estion

```
/*
 * Let's get the thermometer properties from the command line attributes
 * Some error handling wouln't hurt here...
 */
var location = process.argv[2];
var temp = parseInt(process.argv[3]);
var variation = parseInt(process.argv[4]);

/*
 * Let's create a new thermoter - the regular publication of measures will
 * be initiated within the constructor
 */
var t1 = new Thermometer(location, temp, variation);
```

```
/*
 This program simulates a "data collection station", which joins a multicast
 group in order to receive measures published by thermometers (or other sensors).
 The measures are transported in json payloads with the following format:

   {"timestamp":1394656712850,"location":"kitchen","temperature":22.5}

 Usage: to start the station, use the following command in a terminal

   node station.js

*/

/*
 * We have defined the multicast address and port in a file, that can be imported both by
 * thermometer.js and station.js. The address and the port are part of our simple
 * application-level protocol
 */
var protocol = require('./sensor-protocol');

/*
 * We use a standard Node.js module to work with UDP
 */
var dgram = require('dgram');

/*
 * Let's create a datagram socket. We will use it to listen for datagrams published in the
 * multicast group by thermometers and containing measures
 */
var s = dgram.createSocket('udp4');
s.bind(protocol.PROTOCOL_PORT, function() {
  console.log("Joining multicast group");
  s.addMembership(protocol.PROTOCOL_MULTICAST_ADDRESS);
});

/*
 * This call back is invoked when a new datagram has arrived.
 */
s.on('message', function(msg, source) {
       console.log("Data has arrived: " + msg + ". Source port: " + source.port);
});
```