Fog and Cloud Computing
Lab

Daniele Santoro *(dsantoro@fbk.eu)* - Expert Research Engineers
Silvio Cretti *(scretti@fbk.eu)* – *Senior* Research Engineers

RiSING (Robust and Secure Distributed Computing)
Fondazione Bruno Kessler (FBK)

Trento, April 8th 2022

Lab Resources

- Shared Etherpad: https://annuel2.framapad.org/p/6s5u416vo7-9t4b
- White Board: https://tinyurl.com/2p8j7yra
- Interaction:
  - Etherpad
    - *Exercises check, Share Troubleshooting, Questions and Logs*
  - Zoom Chat (for those remotely connected)
    - *Discuss with your colleagues during exercises or directly/privately with me*
  - Rise your Hand (also via Zoom)
    - *If you need my attention or want to speak, don't be shy !!!*
  - Course Forum: https://tinyurl.com/27vmd9pj
    - *Questions and answers could be useful to others, be collaborative*

Lab Resources

- Slides
  - Uploaded before any lesson in Moodle
- Repositories of exercises
  - https://gitlab.fbk.eu/dsantoro/fcc-lab-2022
- Lab Virtual Machine:
  - **Lab VM on Azure (reference for exercises)**
  - Vagrant and VirtualBox on your laptop (possible choice)
    - https://www.virtualbox.org/, https://www.vagrantup.com/ and https://gitlab.fbk.eu/dsantoro/fcc-lab-2022

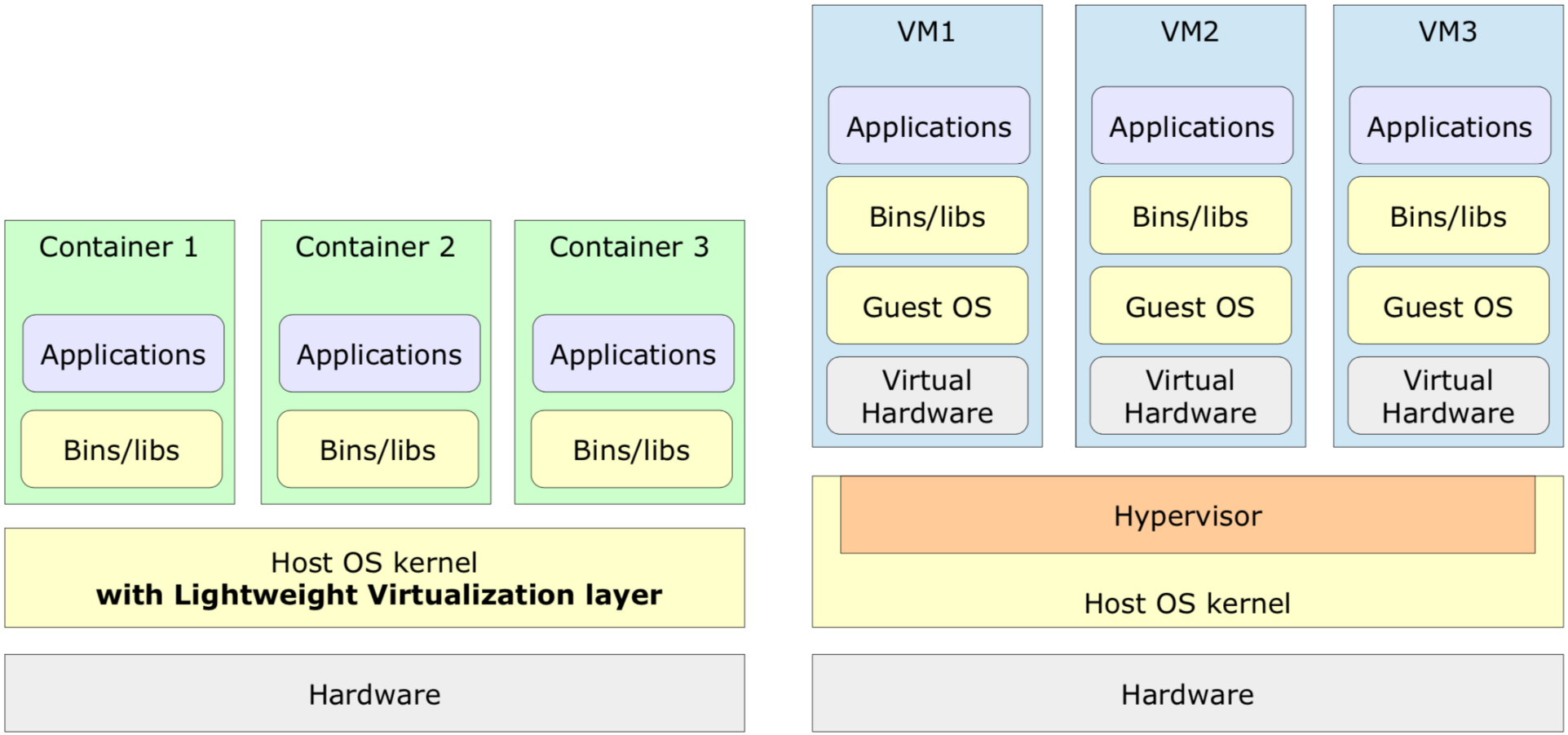
Today Lesson

- Correction of previous exercises (05, 06, 07)
- PaaS VS IaaS
- Docker
  - Why Docker
  - Install Docker
  - Hello World with Docker
  - Simple Docker image

Previous exercise correction

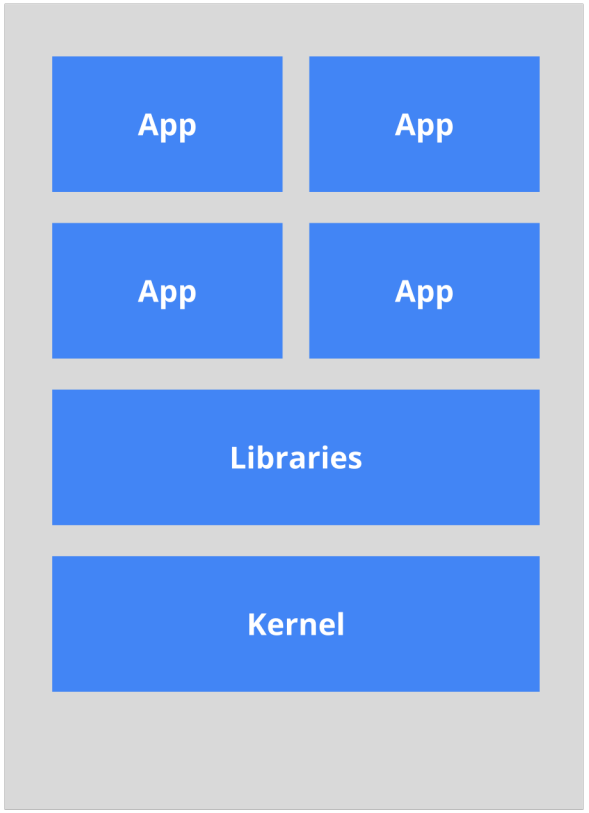
- No more than 20 minutes
  - E05, E06, E07
- Will show complete solutions
- 2 minutes for questions at the end of every exercise
- For those that still did not complete them yet
  - Follow me during correction
  - If doubts, ask
  - Try to complete them at home

Containers vs Hypervisors (Infrastructure)



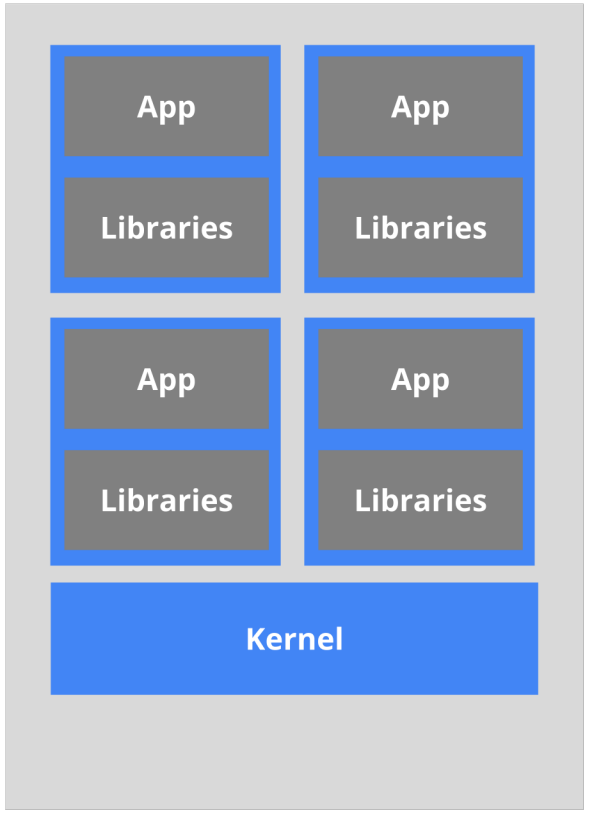
Containers VS Hypervisors (Software)

The old way: Applications on host



Heavyweight, non-portable
Relies on OS package manager

The new way: Deploy containers



Small and fast, portable
Uses OS-level virtualization

Containers are an application-centric way to deliver high-performing, scalable applications on the infrastructure of your choice.

Containers benefits 1/3

- **Agile application creation and deployment**: Increased ease and efficiency of container image creation and deployment compared to VM.
- **Continuous development, integration, and deployment**: Provides for reliable and frequent container image build and deployment with quick and easy rollbacks (due to image immutability).
- **Dev and Ops separation of concerns**: Create application container images at build/release time rather than deployment time, thereby decoupling applications from infrastructure.

Containers benefits 2/3

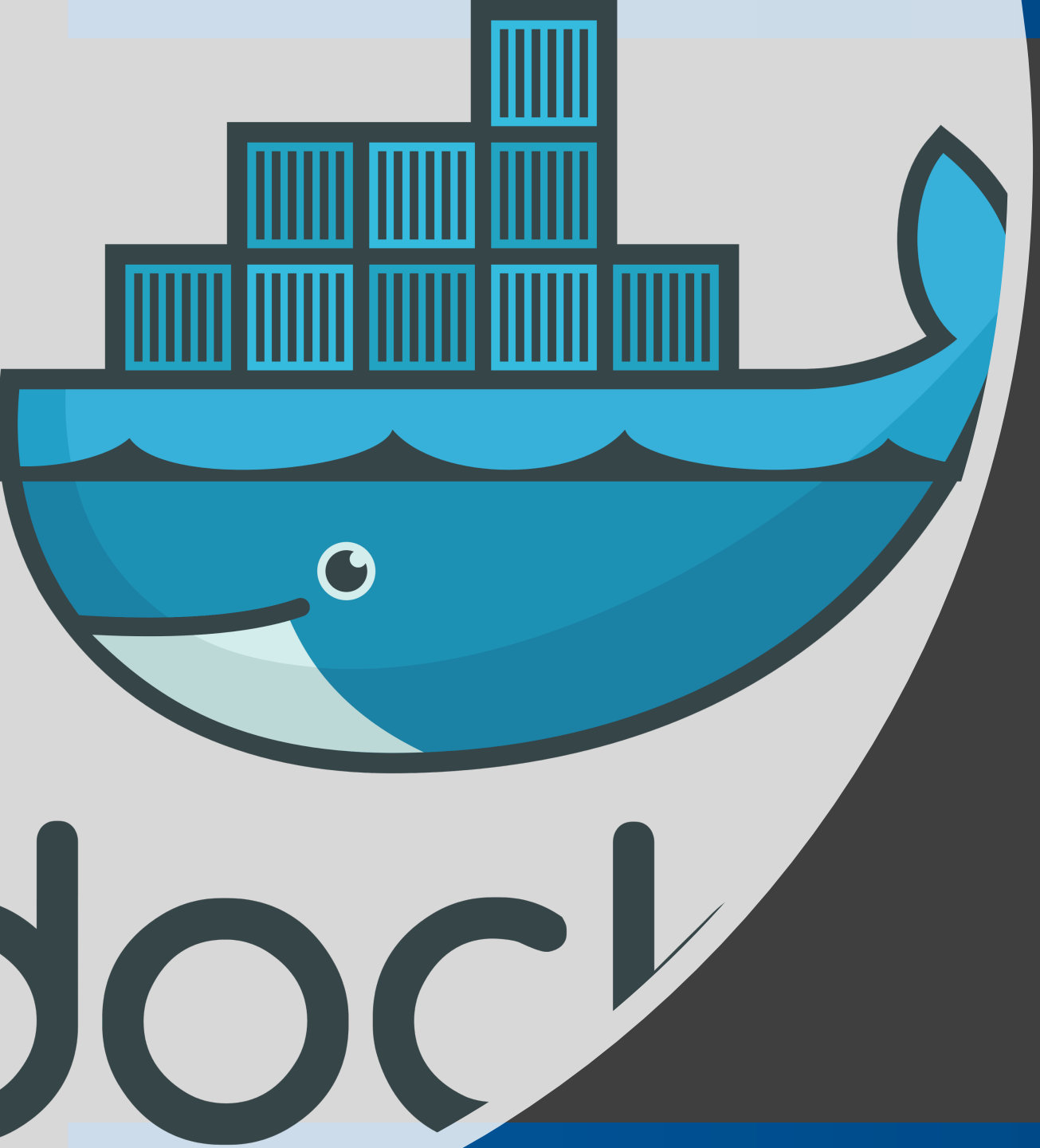
- **Observability** Not only surfaces OS-level information and metrics, but also application health and other signals.
- **Environmental consistency across development, testing, and production**: Runs the same on a laptop as it does in the cloud.
- **Cloud and OS distribution portability**: Runs on Ubuntu, RHEL, CoreOS, on-prem, Google Kubernetes Engine, and anywhere else.

Containers benefits 3/3

- **Application-centric management**: Raises the level of abstraction from running an OS on virtual hardware to running an application on an OS using logical resources.
- **Loosely coupled, distributed, elastic, liberated micro-services**: Applications are broken into smaller, independent pieces and can be deployed and managed dynamically – not a monolithic stack running on one big single-purpose machine.
- **Resource utilization**: High efficiency and density.

Containers disadvantages

- Less isolation/security
  - Others weaknesses: Images distribution, Image scanning
- A solution
  - Combine Virtual Machines and Containers
- Kata containers
  - Kata Containers is an open source community working to build a secure container runtime with lightweight virtual machines that feel and perform like containers, but provide stronger workload isolation using hardware virtualization technology as a second layer of defense. [https://katacontainers.io/]



Docker

Docker

- OS level virtualization (*lightweight*)
- Relies on Linux kernel features: **cgroups** and **namespaces**
- Layered filesystem (similar as git commit)
  - Images as packaged containers derived incrementally from a pre-existing one
- Enable:
  - DevOps
  - Microservice architecture
  - Portability
- https://www.docker.com/ (docs: https://docs.docker.com/)

Docker: Images VS Containers

- **Docker Images [ref]**:
  - A read-only template with instructions for creating a Docker container.
    - Often, an image is *based on* another image, with some additional customization. *For example, you may build an image which is based on the ubuntu image, but installs the Apache web server and your application.*
  - You might create your own images or you might only use those created by others and published in a registry.
  - To build your own image, you create a *Dockerfile* with a simple syntax for defining the steps needed to create the image and run it. Each instruction in a Dockerfile creates a layer in the image.
  - It is the object that makes your application portable.

Docker: Images VS Containers

- **Docker Containers [ref]**:
  - A runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.
  - By default, a container is relatively well isolated from other containers and its host machine. You can control how isolated a container's network, storage, or other underlying subsystems are from other containers or from the host machine.
  - A container is defined by its image as well as any configuration options you provide to it when you create or start it. When a container is removed, any changes to its state that are not stored in persistent storage disappear.

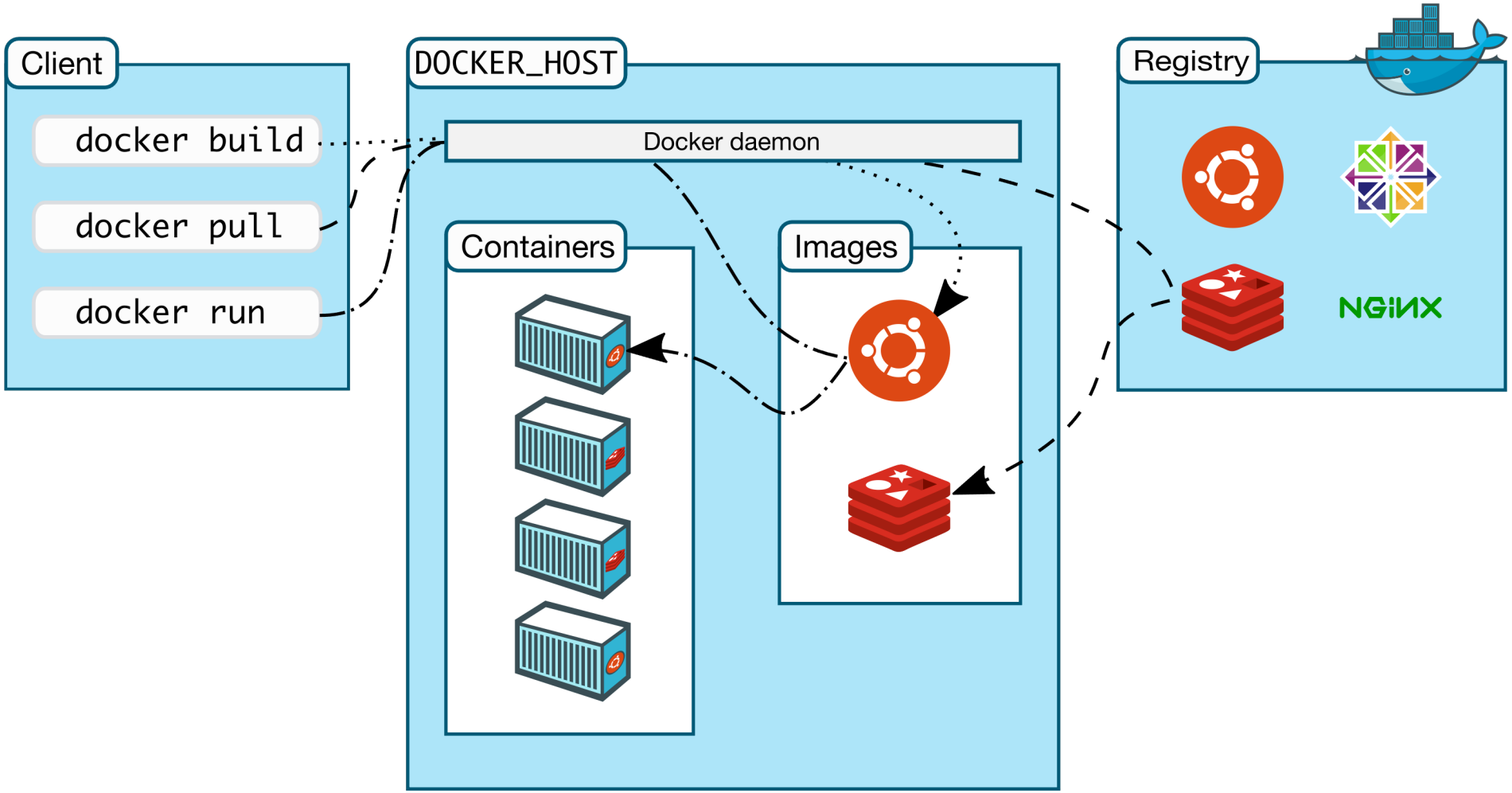
Docker: Images VS Containers

To use a computer science metaphor, if an image is a class, then a container is an instance of a class, in other words a runtime object.

Docker main components

- **Docker daemon** - the main process that manages containers
- **Docker Host** – the host (physical or virtual) where Docker daemon runs
- **Docker client** - for communicating with the Docker daemon
- **Docker registry** - image repository
  - https://hub.docker.com/

Docker Host Overview



Docker related Tools

- **Docker compose** - for deploying multi-container apps
  - https://docs.docker.com/compose/
- **Docker machine** - for setting up remote Docker Hosts
  - https://docs.docker.com/machine/overview/
- **Docker swarm** - container orchestrator
  - https://docs.docker.com/engine/swarm/

Exercise 8 – Create a Vagrant VM for Docker

- **Time:** 10 minutes
  - 6 minutes: *Try by yourself and ask for support*
    - Give an ack when completed succesfully
  - 4 minutes: *Cross check and Verify*

- **Description:**
  - Create a dedicated virtual machine to use for as docker host. This VM can be used for multiple exercises.

- **Instructions:**
  - https://gitlab.fbk.eu/dsantoro/fcc-lab-2022/-/tree/master/e09

Exercise 9 – Install and Verify Docker

- **Time:** 8 minutes
  - 4 minutes: *Try by yourself and ask for support*
    - Give an ack when completed succesfully
  - 4 mintues: *Cross check and Verify*

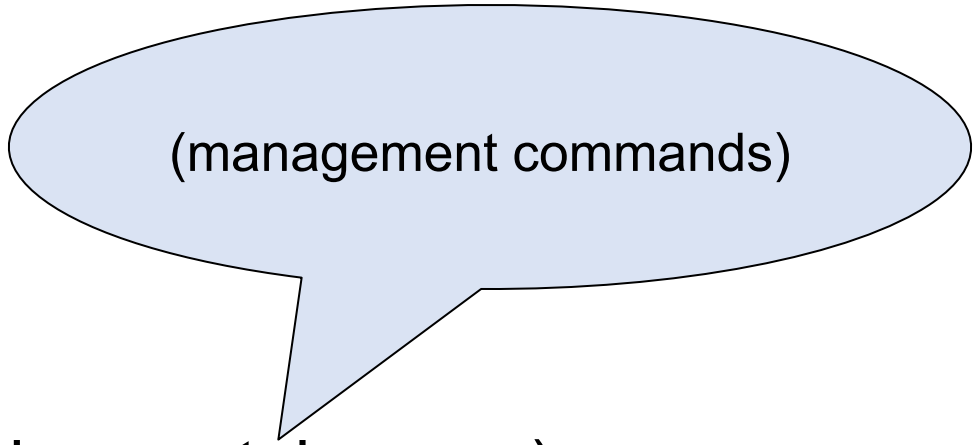
- **Description:**
  - Install the Docker engine and test it is working.

- **Instructions:**
  - https://gitlab.fbk.eu/dsantoro/fcc-lab-2022/-/tree/master/e09

Container Lifecycle

- Create - **docker create** <image>
- Start - **docker start** <container id>
- Stop - **docker kill** , **docker stop** <container id>
- Restart - **docker restart** <container id>
- Remove - **docker rm** <container id>

Some Basic Commands



$ **docker version**

$ **docker <command> --help**

$ **docker run** [-it -d] <image> (docker container run)

$ **docker ps** [-a] (docker container ls)

$ **docker images** (docker image ls)

$ **docker rm** <container id> (docker container rm)

$ **docker rmi** <image name> (docker image rm)

Debugging and Logging

How to debug and check what's happening in Docker?

- `docker inspect` (info about a container)
  - Tip: `docker inspect <container_id> | jq -C .[] | less -RN`
- `docker stats` (statistics on ram, cpu etc)
- `docker events` (Docker Host related events)
- `docker logs` (get logs of apps inside a container)

Run in Interactive or Detached Mode

- **Interactive**

  $ docker run **-it** <image> <args>

- **Detached**

  $ docker run **-d -p <port host>:<port guest>** <image> <args>

Exercise 10 – Hello World with Docker

- **Time:** ~13 minutes
  - 5 minutes: *Try by yourself*
  - 8 minutes: *Check, Verify, Ask*

- **Description:** Start a generic container and practice with the Docker commands we have seen so far.
  1. Run a `jpetazzo/clock` container
  2. Try to understand how the startup happens
  3. Check if the container is in execution (*tip: you may need another shell*)
  4. List container images
  5. Practice with *docker logs* and *docker inspect*
  6. Stop the container
  7. Show stopped containers
  8. Remove the container
  9. Remove the image
  10. Verify that the container has been removed together with its image

- **Instructions:**
  https://gitlab.fbk.eu/dsantoro/fcc-lab-2022/-/tree/master/e10

Exercise 11 – Build a custom Docker image

- **Time:** ~17 minutes
  - 7 minutes: *Try by yourself*
  - 10 minutes: *Check, Verify, Ask*

- **Description:** Create a personalised image starting from a generic one. Understand how and why your image is different from the initial one. Play with layers and start a container based on the custom image.

- **Instructions:**
  https://gitlab.fbk.eu/dsantoro/fcc-lab-2022/-/tree/master/e11

Build a Docker image with a Dockerfile

- DockerFile [ref]:

```
FROM <original image>
ADD <filename> <destination path>
RUN <command>
…
EXPOSE 80
```

- Run the new image

```
$ docker build -t <image namespace>/<image name>
$ docker images
$ docker run -ti <image namespace>/<image name>
```