Docker Training

VALENTINA GAGGERO

ISTITUTO ITALIANO
DI TECNOLOGIA

Docker Training - Contents

01 INTRODUCTION TO DOCKER

- What is Docker?
- Why Docker?
- Docker Basics
- Docker For Developers
- Docker Architecture
- Docker vs Virtual Machines

02 HOW TO DEVELOP WITH DOCKER

- Pulling of a Docker image
- Container creation
- Container status
- Files sharing
- Environment variables sharing

03 DOCKER ADVANCED COMMAND

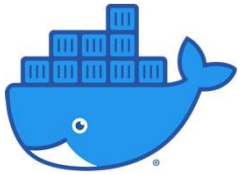
- Use the host network from a container
- Share yarp configuration
- Sharing devices
- Give full capabilities to a container

04 BUILD YOUR APPLICATION

- How to build an image
- Docker compose
- Docker logging
- Sharing devices
- Deploying a yarp basic application

01 - Introduction To Docker

What is Docker?



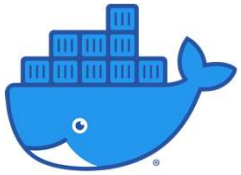
Wikipedia defines Docker as:

> *an open-source project that automates the deployment of software applications inside **containers** by providing an additional layer of abstraction and automation of **OS-level virtualization** on Linux.*

01 - Introduction To Docker

What is Docker?

Wikipedia defines Docker as:



> *an open-source project that automates the deployment of software applications inside **containers** by providing an additional layer of abstraction and automation of **OS-level virtualization** on Linux.*

Docker is a **tool** that allows developers to easily **deploy their applications in a sandbox** (called *containers*) to run on **the host operating system**

01 - Introduction To Docker

Why Docker?

Docker containers are process-isolated and **do not require** a hardware hypervisor. This means Docker containers are much smaller and require far fewer resources than a VM.

Containers can be shared across multiple team members, bringing much-needed portability across the development pipeline. This reduces '**works on my machine**' errors that plague developer teams.

Docker **is fast. Very fast**. While a VM can take an at least a few minutes to boot and be dev-ready, it takes anywhere from a few milliseconds to (at most) a few seconds to start a Docker container from a container image.

01 - Introduction To Docker

Docker Basics

Docker Image
The basis of a Docker container

Docker Container
The standard unit in which the application service resides. An instance of the image that can be run on a Docker host.

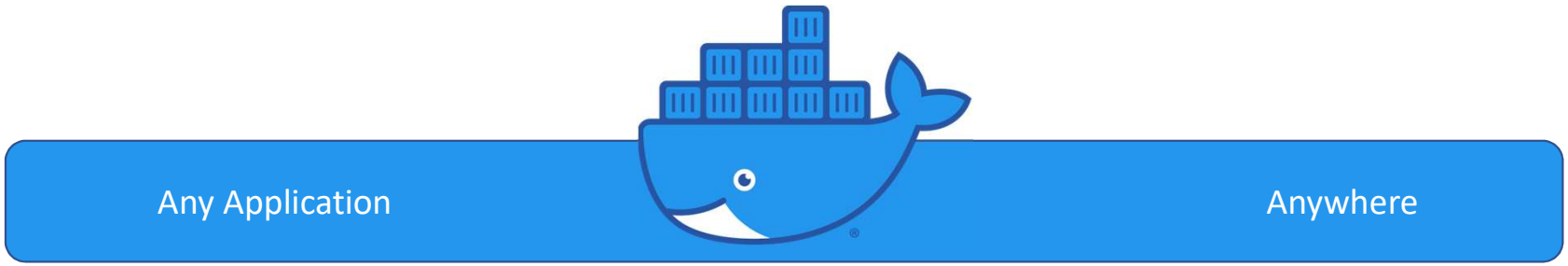
Docker Engine
Creates, ships and runs Docker containers deployable on physical or virtual host locally or cloud service provider

Registry
The registry can be a public registry, like **Docker hub**, or a private registry from an institute/ company. Service for finding and sharing container images with your team

Service
An application or part of an application, that provides a specific function

01 - Introduction To Docker

Docker For Developers



Build once ⟶ **Run anywhere**

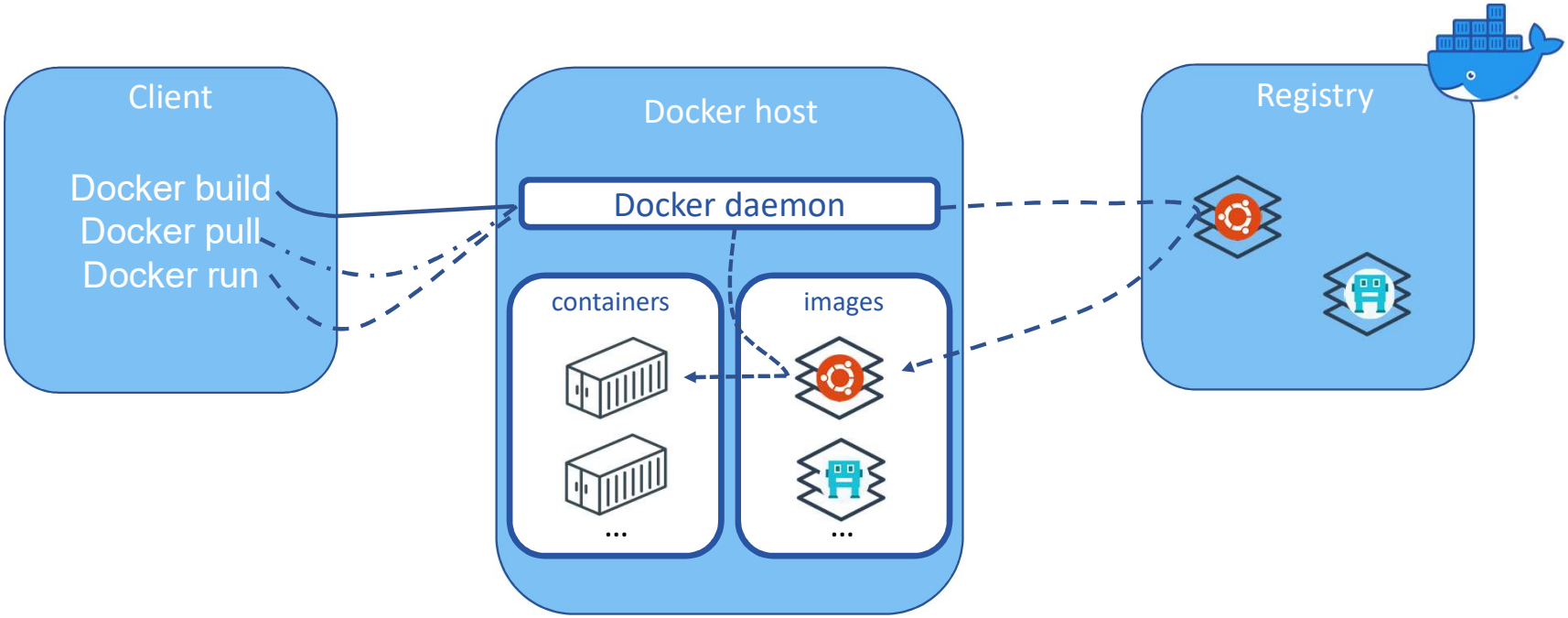
- Portable, isolated container
- Automate testing integration packaging
- Eliminate compatibility concerns

Configure once ⟶ **Run anything**

- Eliminate inconsistencies with versioning
- Improve speed and reliability
- Make lifecycle repeatable

01 - Introduction To Docker

Docker Architecture



01 - Introduction To Docker

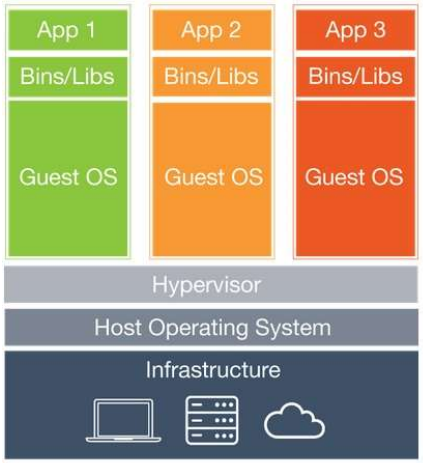
Docker vs Virtual Machine

Virtual Machines

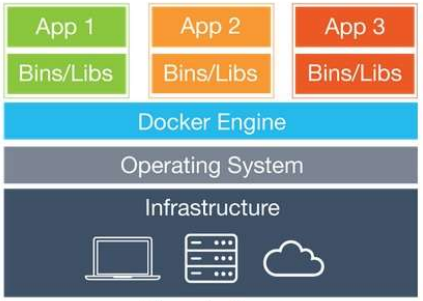
- relies on the system's physical hardware to emulate the exact same environment
- Needs an Hypervisor and a full OS inside
- Bigger Footprint (RAM and Storage
- Heavier
- Deployment/portability is tough

Docker

- Smaller footprint (No RAM and differential storage)
- Lightweight
- Easy Deployment / portability with minimal requirements for running the application
- Faster



Virtual Machines



Containers

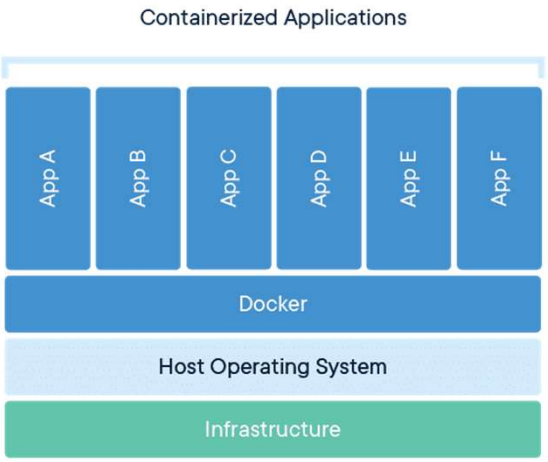
01 - Introduction To Docker

Docker vs Virtual machines

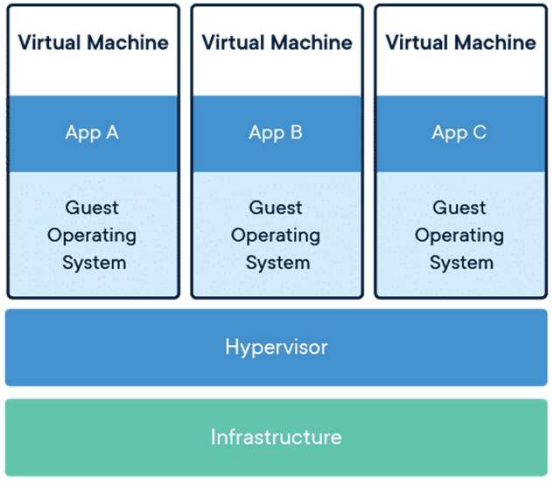
⚠ **Docker is NOT a virtual machine**

Docker

- Lightweight
- Easy Deployment / portability with minimal requirements
- Faster



Docker container



Virtual machine

Virtual machine

- Relies on the system's physical hardware
- Needs a Hypervisor and a full OS inside
- Heavier
- Deployment/portability tough

01 - Introduction To Docker

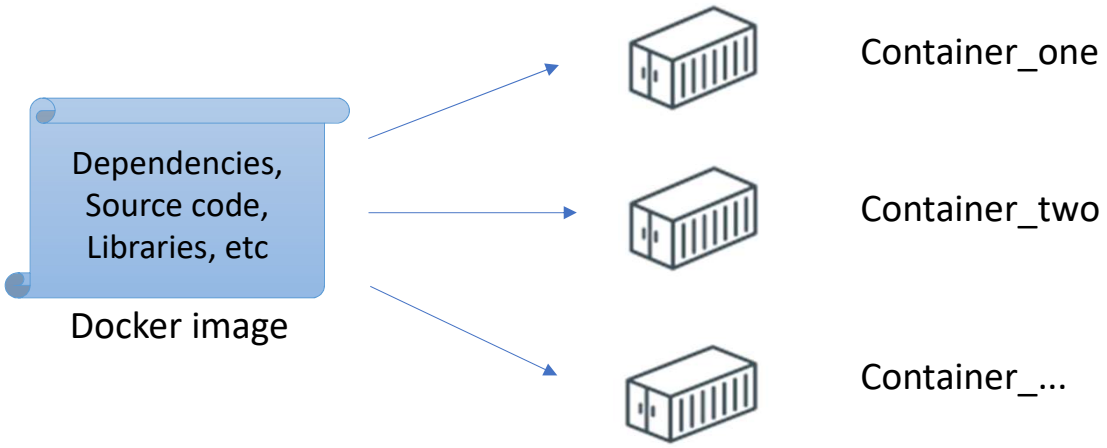
Docker Essentials

- **Docker Image:** is a read-only, inert template that comes with instructions for deploying containers. Images are stored on a Docker registry, such as Docker Hub, or on a local registry.

Images are read-only templates used to build containers.

- **Docker Container:** is an instance of a Docker image. It is a virtualized runtime environment that provides isolation capabilities for separating the execution of applications from the underpinning system.

Containers are deployed instances created from those templates.



02 – How To Develop With Docker

1. **Download the image from the registry**

   - `docker pull image_name`

     example: `docker image pull icubteamcode/superbuild:v2022.02.1_sources`

     image name | tag

   Related images are available on dockerhub/icubteamcode

2. **List all the images**

   - `docker image ls`
   - `docker images`

3. **Run the container**

   - `docker run image_name command`
   - `docker run --name containerOne image_name command`

    The `docker run` command runs a command in a new container

02 – How To Develop With Docker

3. List all the containers and check the container's name

- `docker container ls –a`
- `docker ps -a`

```
laura@laura-VirtualBox:~$ docker container ls -a
CONTAINER ID   IMAGE                                          NAMES
e8ce2a56f61e   icubteamcode/superbuild:v2022.02.1_sources     containerOne
8a976925ccde   icubteamcode/superbuild:v2022.02.1_sources     flamboyant_banach
```

Note the container's name!

The `docker container ls –a` and `docker ps –a` commands both show **all** containers

If you want to list **just running** containers, you have to use `docker container ls` or `docker ps`

02 – How To Develop With Docker

Exercise 1:

1.1 Run the `yarp server` command in a container named `yarpSrvContainer`

- `docker image pull icubteamcode/superbuild:v2022.02.1_sources`
- `docker run --name yarpSrvContainer icubteamcode/superbuild:v2022.02.1_sources yarp server`

```
laura@laura-VirtualBox:~$ docker container ls
CONTAINER ID   IMAGE                                      STATUS          PORTS                    NAMES
03e66c3b9601   icubteamcode/superbuild:v2022.02.1_sources  Up 12 seconds   10000/tcp, 10000/udp     yarpSrvContainer
```

Note: STATUS is up!

1.2 Run the `ls` command in a container named `cmdContainer`

- `docker run --name cmdContainer icubteamcode/superbuild:v2022.02.1_sources ls`

```
laura@laura-VirtualBox:~$ docker container ls -a
CONTAINER ID   IMAGE                                      STATUS                     PORTS     NAMES
4507893212d2   icubteamcode/superbuild:v2022.02.1_sources  Exited (0) 11 seconds ago            cmdContainer
```

Note: STATUS is exited!

If you want to fetch the log of a container → `docker logs container_name`

02 – How To Develop With Docker

Exercise 2: connect to the containers created in the previous slide

- `docker exec -it container_name bash`

The `docker exec` command runs a new command in a running container.

```
laura@laura-VirtualBox:~$ docker exec -it yarpSrvContainer bash
This images has release=master and had been building with superbuild_tag=Unstable. Metadata=24/04/2022
This images has release=master and had been building with superbuild_tag=Unstable. Metadata=24/04/2022
root@03e66c3b9601:/#
```

```
laura@laura-VirtualBox:~$ docker exec -it cmdContainer bash
Error response from daemon: Container 4507893212d2d12f062964c2fd4a3dede5b5efb646e5d4e7e49eda1c2a01e502 is not running
```

You can't connect to a container that is not running!

02 – How To Develop With Docker

How to create a container with an interactive bash shell

- `docker run -it --name interactiveContainer icubteamcode/superbuild:v2022.02.1_sources bash`

```
laura@laura-VirtualBox:~$ docker container ls
CONTAINER ID   IMAGE                                        STATUS              PORTS                    NAMES
13c60e7b5c0f   icubteamcode/superbuild:v2022.02.1_sources   Up About a minute   10000/tcp, 10000/udp   interactiveContainer
```

Note: STATUS is up!

The **-it** instructs Docker to allocate a pseudo-terminal connected to the container's stdin, creating an interactive bash shell in the container.

Exercise : close the container and check its status

- Type exit

```
laura@laura-VirtualBox:~$ docker container ls -a
CONTAINER ID   IMAGE                                        STATUS                      PORTS     NAMES
1bbd4871a025   icubteamcode/superbuild:v2022.02.1_sources   Exited (0) 5 seconds ago              interactiveContainer
```

Note: STATUS is exited!

02 – How To Develop With Docker

How to start a container

- `docker start container_name`

```
laura@laura-VirtualBox:~$ docker container ls
CONTAINER ID   IMAGE                                      STATUS         PORTS                  NAMES
1bbd4871a025   icubteamcode/superbuild:v2022.02.1_sources  Up 3 seconds   10000/tcp, 10000/udp   interactiveContainer
```

Note: STATUS is up!

The `docker start` command starts one or more stopped containers

- The `-i` option allows you to attach container's STDIN
- The `-ai` option allows you to attach container's STDIN + STDout/STDerr

Which is the difference between `docker exec -it` and `docker start -i`?

- `docker exec -it` is specifically for running new things in a already **started** container, be it a shell or some other process.
- `docker start -i` is for running new things in a **exited** container

02 – How To Develop With Docker

How to attach to a running container

- `docker attach` *`container_name`*

  - `docker attach` attaches:
    - STDout/STDerr to a **running** container
    - STDin + STDout/STDerr to a **running interactive** container

Which is the difference between `docker attach` and `docker logs`?

  - `docker attach` streams the container's logs **from** the moment you launch **the attach command**
  - `docker logs` streams the container's logs **from the moment the process** on the container **started**, **until** the moment you run **the logs command**
  - `docker logs -f` does what attach + logs do: it streams the container's logs from the beginning to the end

02 – How To Develop With Docker

How to stop a container

- `docker stop container_name`

The `docker stop` command stops one or more running containers

How to remove one/more stopped container

- `docker rm container_name`

The `docker rm` command removes one or more **stopped** containers

How to remove all stopped container

- `docker container prune`

The `docker container prune` command removes all **stopped** containers

02 – How To Develop With Docker

Exercise 1 :

1.1 connect to `interativeContainer` and create a .txt file inside it

- `Docker exec –it interactiveContainer bash`
- `touch file.txt`

1.2 Open a new container and search for the .txt file

- `docker run -it --name newInteractiveContainer icubteamcode/superbuild:v2022.02.1_sources bash`

You will not find file.txt!

02 – How To Develop With Docker

Exercise 2 :

2.1 Commit your changes to a new image

- `docker commit container_name new_image_name`

```
laura@laura-VirtualBox:~$ docker commit interactiveContainer new_superbuild_img
sha256:fb45c3ae0b6715a11d61bd4edcdf073573b350fa0724be8d8f506911823713d9
```

The `docker commit` command create a new image from a container's changes

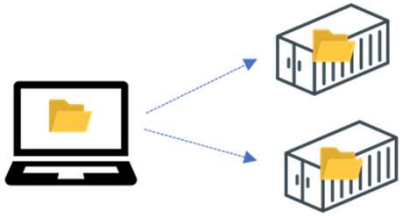
2.2 Check your images

```
laura@laura-VirtualBox:~$ docker image ls
REPOSITORY                  TAG                 IMAGE ID       CREATED         SIZE
new_superbuild_img          latest              fb45c3ae0b67   3 minutes ago   4.55GB
```

02 – How To Develop With Docker

How to share files between containers and host?

1. **Docker copy utility**



From host to container:

- `docker cp SRC_PATH <containerid>:DEST_PATH`

From container to host:

- `docker cp <containerid>:SRC_PATH DEST_PATH`

Exercise: copy a file from your host to the container `interactiveContainer`, then go inside it and check if the file is there

```
laura@laura-VirtualBox:~$ docker cp /home/laura/test.txt 1bbd4871a025:/root
laura@laura-VirtualBox:~$ docker exec -it interactiveContainer bash
Error response from daemon: Container 1bbd4871a025cf30e18e67334d8d3e6ba6e21a1542f92819731aedc80b50dfae is not running
laura@laura-VirtualBox:~$ docker start interactiveContainer
interactiveContainer
laura@laura-VirtualBox:~$ docker exec -it interactiveContainer bash
This images has release=v2022.02.1 and had been building with superbuild_tag=Custom. Metadata=01/04/2022
This images has release=v2022.02.1 and had been building with superbuild_tag=Custom. Metadata=01/04/2022
This images has release=v2022.02.1 and had been building with superbuild_tag=Custom. Metadata=01/04/2022
root@1bbd4871a025:/# ls
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  projects  root  run  sbin  srv  sys  test.txt  tmp  usr  var
root@1bbd4871a025:/#
```

02 – How To Develop With Docker

How to share files between containers and host?

2. Mount a folder

- `docker run –it` **`--mount type=bind,source=SRC_PATH,target=DEST_PATH`** *`image_name`* `bash`

Exercise:

1. Share a `.txt` file with a container of the image `icubteamcode/superbuild:v2022.02.1_sources`

```
laura@laura-VirtualBox:~$ docker run -it --mount type=bind,source=/home/laura/test.txt,target=/root/test.txt icubteamcode/superbuild:v2022.02.1_sources bash
This images has release=v2022.02.1 and had been building with superbuild_tag=Custom. Metadata=01/04/2022
root@d18886f94220:/# ls
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  projects  root  run  sbin  srv  sys  tmp  usr  var
root@d18886f94220:/# cd
root@d18886f94220:~# ls
test.txt
```

2. Edit your `.txt` file inside the container and then check the one on your host (you can use vim, nano)

Your changes will be present in the .txt file on your host!

02 – How To Develop With Docker

Sharing the environment variables

1. Sharing one variable

- `export MY_VAR="myvar"`
- `docker run -it --env MY_VAR image_name bash`

2. Sharing a file containing multiple environment variables

- ```
  cat env.list
    VAR1=value1
    VAR2=value2
    USER
  ```
- `docker run -it --env-file env.list  image_name bash`
  ```

02 – How To Develop With Docker

Sharing the environment variables

Exercise:

1. Create a variable on the host side, run a container sharing the same variable and check the value of the variable inside the container

```
laura@laura-VirtualBox:~$ docker run -it --env MY_VAR icubteamcode/superbuild:v2022.02.1_sources bash
This images has release=v2022.02.1 and had been building with superbuild_tag=Custom. Metadata=01/04/2022
root@970874edd176:/# echo $MY_VAR
my var
```

2. Change the value of MY_VAR inside the container and check the value on the host

The value doesn't change on the host!

3. Change the value of MY_VAR on the host and check the value inside the container

The value doesn't change inside the container!

02 – How To Develop With Docker

Exercise: open an `interactive` container called `guiContainer` and inside it open a `yarpview`

```
vvasco@iiticublap:~$ docker run -it --name guiContainer icubteamcode/superbuild:v2022.02.1_sources bash
This images has release=v2022.02.1 and had been building with superbuild_tag=Custom. Metadata=01/04/2022
root@a1071275445a:/# yarpview
qt.qpa.xcb: could not connect to display
qt.qpa.plugin: Could not load the Qt platform plugin "xcb" in "" even though it was found.
This application failed to start because no Qt platform plugin could be initialized. Reinstalling the application may fix this problem.

Available platform plugins are: eglfs, linuxfb, minimal, minimalegl, offscreen, vnc, xcb.

Aborted (core dumped)
root@a1071275445a:/#
```

Something is missing inside the container . . .

How to share graphical interfaces between host and container

- `docker run -it`
  `--env DISPLAY=${DISPLAY}`
  `--env XAUTHORITY=/root/.Xauthority`
  `--mount`
  `type=bind,source=${XAUTHORITY},target=/root/.Xauthority`
  `--mount type=bind,source=/tmp/.X11-unix,target=/tmp/.X11-unix`
  `image_name` `bash`

- `start XLaunch` (disable access control ☑)
- `docker run -it`
  `--env DISPLAY=YOUR_IP_ADDR:0`
  `image_name` `bash`

02 – How To Develop With Docker

Exercise: include the graphical interface in the previous command and open a `yarpview`

```
vvasco@iiticublap:~$ docker run -it --name guiContainerShare --env DISPLAY=${DISPLAY} --env XAUTHORITY=/root/.Xauthority --mount type=bind,source=
${XAUTHORITY},target=/root/.Xauthority --mount type=bind,source=/tmp/.X11-unix,target=/tmp/.X11-unix icubteamcode/superbuild:v2022.02.1_sources ba
sh
This images has release=v2022.02.1 and had been building with superbuild_tag=Custom. Metadata=01/04/2022
root@6e5213a700a9:/# yarp detect --write
[INFO] |yarp.os.impl.NameClient| No connection to nameserver
[INFO] |yarp.os.impl.NameClient| no connection to nameserver, scanning mcast
[INFO] |yarp.os.impl.FallbackNameClient| Polling for name server (using multicast), try 1 of max 3
[INFO] |yarp.os.impl.FallbackNameClient| Received address tcp://172.17.0.2:10000/
Checking for name server at ip 172.17.0.2 port 10000
If there is a long delay, try:
  yarp conf --clean

=======================================================
==
== FOUND
== /root is available at ip 172.17.0.2 port 10000
== /root can be browsed at http://172.17.0.2:10000/
==
== Address saved.
== YARP programs will now be able to use the name server.
==
=======================================================
root@6e5213a700a9:/# yarpview
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'
qrc:/YARPView/YARPViewStatusBar.qml:102:17: Unable to assign QQuickText to double
qrc:/YARPView/YARPViewStatusBar.qml:101:17: Unable to assign QQuickText to double
[INFO] |yarp.os.Port|/yarpview/img:i| Port /yarpview/img:i active at tcp://172.17.0.3:10002/
```

02– HANDS-ON

Exercise: re-play a recorded sequence from the robot using `yarpdataplayer`

Follow the instructions here:

1. Download the following compressed sequence:
   - `wget` http://www.icub.org/download/software/datasetplayer-demo/dataDisparity.zip
2. Use the `yarpSrvContainer` to launch `yarp server`
3. Create a new container called `dataPlayerContainer` that:
   - Can open GUI application
   - Shares the compressed example sequence
   - unzip `dataDisparity.zip`
   - Launch `yarpdataplayer` and upload the sequence
4. Open another container and launch inside a `yarpview`
5. Connect the yarpview port to `/icub/camcalib/left/out`

03 – DOCKER ADVANCED COMMANDS

How to use the host network from a container

- `docker run --network host image_name command`

The `--network host` option is used to make Docker containers use the same network of the host.

The `--network host` option is **not supported** on Windows/Mac!


If the `--network host` option is not used, Docker creates its own network

03 – DOCKER ADVANCED COMMANDS

Exercise 1:

1.1 On your host, run `yarpserver`

1.2 Create a container of `icubteamcode/superbuild:v2022.02.1_sources` **that shares yarp configuration and, once inside it, check the content of the yarp conf file**

```
laura@laura-VirtualBox:~$ docker run -it --mount type=bind,source=${HOME}/.config/yarp,target=/root/.config/yarp icubteamcode/superbuild:v2022.02.1_sources bash
This images has release=v2022.02.1 and had been building with superbuild_tag=Custom. Metadata=01/04/2022
root@eacf10f1f36f:/# yarp conf
/root/.config/yarp/_laura.conf
root@eacf10f1f36f:/# cat /root/.config/yarp/_laura.conf
192.168.111.128 10000 yarp
```

Note the ip address of your host!

1.3 On the host open a /tmp port and from the container write to that port

```
root@eacf10f1f36f:/# yarp write ... /tmp
[INFO] |yarp.os.Port|/tmp/port/1| Port /tmp/port/1 active at tcp://172.17.0.2:10003/
[INFO] |yarp.os.impl.PortCoreOutputUnit|/tmp/port/1| Sending output from /tmp/port/1 to /tmp using tcp
>>hello
```

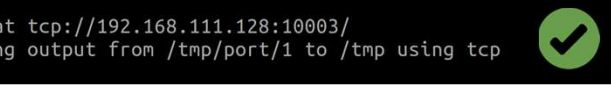
```
laura@laura-VirtualBox:~$ yarp read /tmp
yarp: Port /tmp active at tcp://192.168.111.128:10002/
yarp: Receiving input from /tmp/port/1 to /tmp using tcp
hello
```

Even if you shared the yarp configuration from the host, any yarp service inside the container uses the Docker network!

03 – DOCKER ADVANCED COMMANDS

Exercise 2: Repeat exercise 1, but including `--network host` when creating the container

```
laura@laura-VirtualBox:~$ docker run -it --network host --mount type=bind,source=${HOME}/.config/yarp,target=/root/.config/yarp icubteamcode/superbuild:v2022.02.1_sou
rces bash
This images has release=v2022.02.1 and had been building with superbuild_tag=Custom. Metadata=01/04/2022
root@laura-VirtualBox:/# yarp write ... /tmp
[INFO] |yarp.os.Port|/tmp/port/1| Port /tmp/port/1 active at tcp://192.168.111.128:10003/
[INFO] |yarp.os.impl.PortCoreOutputUnit|/tmp/port/1| Sending output from /tmp/port/1 to /tmp using tcp
>>hello
```



```
laura@laura-VirtualBox:~$ yarp read /tmp
yarp: Port /tmp active at tcp://192.168.111.128:10002/
yarp: Receiving input from /tmp/port/1 to /tmp using tcp
hello
```

Now, even the container uses the host network!

03 – DOCKER ADVANCED COMMANDS

Exercise:

1. **create a container `cont1` and launch `yarpserver` inside it**

```
(robotologyenv) PS C:\Users\lcavaliere> docker run -it --name cont1 icubteamcode/superbuild:v2022.02.1_sources bas
h
This images has release=v2022.02.1 and had been building with superbuild_tag=Custom. Metadata=01/04/2022
root@4c614d74d582:/# yarpserver

    ========================
   | __  __ ___ ____  ____  |
   | \ \/ //   ||  _ \ |  _ \ |
   |  \  // /| || |/ / | |/ / |
   |  / // ___ ||  _ \ |  _/  |
   | /_//_/  |_||_| \_\|_|    |
   |                         |
    ========================

Call with --help for information on available options
Using port database: :memory:
Using subscription database: :memory:
IP address: default
Port number: 10000
[INFO] |yarp.os.Port|/root| Port /root active at tcp://172.17.0.3:10000/
Registering name server with itself
 * register "/root" tcp "172.17.0.3" 10000
```

Note the ip address of Docker!

2. **create a container `cont2` and launch `yarp detect --write` inside it**

```
(robotologyenv) PS C:\Users\lcavaliere> docker run -it --name cont2 icubteamcode/superbuild:v2022.02.1_sources bas
h
This images has release=v2022.02.1 and had been building with superbuild_tag=Custom. Metadata=01/04/2022
root@b6327b6c7be6:/# yarp detect --write
[INFO] |yarp.os.impl.NameClient| No connection to nameserver
[INFO] |yarp.os.impl.NameClient| no connection to nameserver, scanning mcast
[INFO] |yarp.os.impl.FallbackNameClient| Polling for name server (using multicast), try 1 of max 3
[INFO] |yarp.os.impl.FallbackNameClient| Received address tcp://172.17.0.3:10000/
Checking for name server at ip 172.17.0.3 port 10000
If there is a long delay, try:
  yarp conf --clean

============================================================
==
== FOUND
== /root is available at ip 172.17.0.3 port 10000
== /root can be browsed at http://172.17.0.3:10000/
==
== Address saved.
== YARP programs will now be able to use the name server.
==
============================================================
root@b6327b6c7be6:/# yarp name list
registration name /root ip 172.17.0.3 port 10000 type tcp
registration name fallback ip 224.2.1.1 port 10000 type mcast
*** end of message
```

03 – DOCKER ADVANCED COMMANDS

Exercise:

3. inside `cont2`, create a `/tmp` port

```
root@b6327b6c7be6:/# yarp read /tmp
[INFO] |yarp.os.Port|    | Port /tmp active at tcp://172.17.0.2:10003/
[INFO] |yarp.os.impl.PortCoreInputUnit|    | Receiving input from /tmp/port/1 to /tmp using tcp
hello
```

4. create a container `cont3`, launch `yarp detect --write` inside it and then write to the `/tmp` port

```
root@fa90b7ac73d9:/# yarp write ... /tmp
[INFO] |yarp.os.Port|/tmp/port/1| Port /tmp/port/1 active at tcp://172.17.0.4:10002/
[INFO] |yarp.os.impl.PortCoreOutputUnit|/tmp/port/1| Sending output from /tmp/port/1 to /tmp using tcp
>>hello
```

Docker containers can always communicate each other !

03 – DOCKER ADVANCED COMMANDS

How to use a device of the host inside the container

- `docker run --device=SRC_PATH:DEST_PATH image_name command`

  ✏️ The `--device` option add a host device to the container.

  ⚠️ The `--device` option is not supported on Windows

Exercise1: create a container and try to record a message by using `arecord`

```
laura@laura-VirtualBox:~$ docker run -it icubteamcode/superbuild:v2022.02.1_sources bash
This images has release=v2022.02.1 and had been building with superbuild_tag=Custom. Metadata=01/04/2022
root@502b95a8fde5:/# arecord test.wav
ALSA lib confmisc.c:767:(parse_card) cannot find card '0'
ALSA lib conf.c:4732:(_snd_config_evaluate) function snd_func_card_driver returned error: No such file or directory
ALSA lib confmisc.c:392:(snd_func_concat) error evaluating strings
ALSA lib conf.c:4732:(_snd_config_evaluate) function snd_func_concat returned error: No such file or directory
ALSA lib confmisc.c:1246:(snd_func_refer) error evaluating name
ALSA lib conf.c:4732:(_snd_config_evaluate) function snd_func_refer returned error: No such file or directory
ALSA lib conf.c:5220:(snd_config_expand) Evaluate error: No such file or directory
ALSA lib pcm.c:2642:(snd_pcm_open_noupdate) Unknown PCM default
arecord: main:852: audio open error: No such file or directory
root@502b95a8fde5:/#
```

03 – DOCKER ADVANCED COMMANDS

Exercise2: repeat exercise 1, including the share of the sound device when creating the container

```
laura@laura-VirtualBox:~$ docker run -it --device=/dev/snd:/dev/snd icubteamcode/superbuild:v2022.02.1_sources bash
This images has release=v2022.02.1 and had been building with superbuild_tag=Custom. Metadata=01/04/2022
root@ff86724e80f6:/# arecord test.wav
Recording WAVE 'test.wav' : Signed 16 bit Little Endian, Rate 8000 Hz, Mono

                                    Recording...

^CAborted by signal Interrupt...
arecord: pcm_read:2178: read error: Interrupted system call
root@ff86724e80f6:/# aplay test.wav
Playing WAVE 'test.wav' : Signed 16 bit Little Endian, Rate 8000 Hz, Mono

                                    Playing...
```

03 – DOCKER ADVANCED COMMANDS

How to give high capabilities to a Docker container

- `docker run` **`--privileged`** *`image_name command`*

  The `--privileged` option allows to run containers in a privileged mode and give them the capabilities of its host machine, namely access to all its devices.

Exercise1: create a temporary file system (**tmpfs**), name it **none** and mount it to **/mnt**

```
laura@laura-VirtualBox:~$ docker run -it icubteamcode/superbuild:v2022.02.1_sources bash
This images has release=v2022.02.1 and had been building with superbuild_tag=Custom. Metadata=01/04/2022
root@b2c5d9efda77:/# mount -t tmpfs none /mnt
mount: /mnt: permission denied.
```

By default, most potentially dangerous kernel capabilities are dropped !

03 – DOCKER ADVANCED COMMANDS

Exercise2: repeat exercise 1, including the `--privileged` option. Then, list the disk space statistics (in **human readable** format) with the command `df -h`.

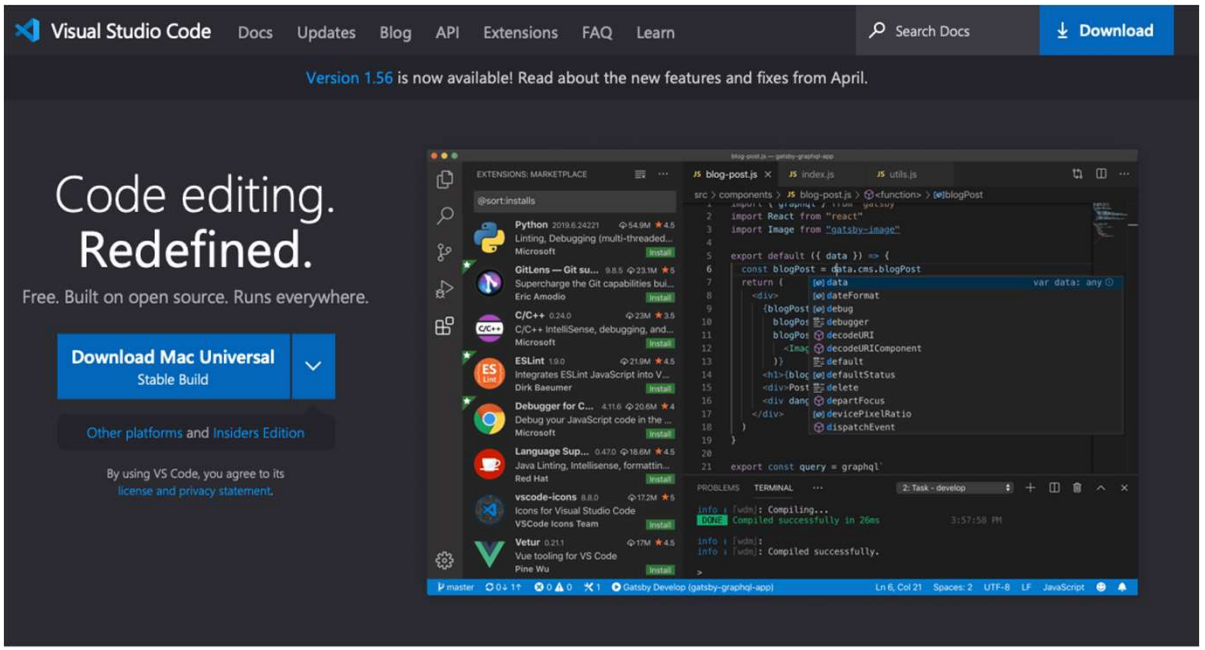
```
laura@laura-VirtualBox:~$ docker run -it --privileged icubteamcode/superbuild:v2022.02.1_sources bash
This images has release=v2022.02.1 and had been building with superbuild_tag=Custom. Metadata=01/04/2022
root@87a8527b4260:/# mount -t tmpfs none /mnt
root@87a8527b4260:/# df -h
Filesystem      Size  Used Avail Use% Mounted on
overlay         185G  167G  9.7G  95% /
tmpfs            64M     0   64M   0% /dev
tmpfs           5.0G     0  5.0G   0% /sys/fs/cgroup
shm              64M     0   64M   0% /dev/shm
/dev/sda5       185G  167G  9.7G  95% /etc/hosts
none            5.0G     0  5.0G   0% /mnt
```

Note the newly created file system!

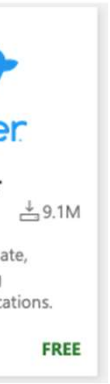
The same exercise can be done on Windows.

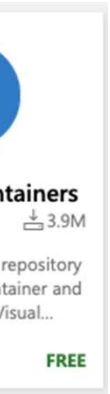
04 – BUILD YOUR APPLICATION

Vscode and Docker



IntelliSense | Run and Debug | Built-in Git | Extensions





04 – BUILD YOUR APPLICATION

How to build an image → Dockerfile

```
# our base image
FROM alpine:latest

# Install python and pip
RUN apk add --update py-pip

# upgrade pip
RUN pip install --upgrade pip

# install Python modules needed by the Python app
COPY requirements.txt /usr/src/app/
RUN pip install --no-cache-dir -r /usr/src/app/requirements.txt

# copy files required for the app to run
COPY app.py /usr/src/app/
COPY templates/index.html /usr/src/app/templates/

# tell the port number the container should expose
EXPOSE 5000

# run the application
CMD ["python", "/usr/src/app/app.py"]
```

04 – BUILD YOUR APPLICATION

Each DockerFile command creates a layer in the image

```
# our base image
FROM alpine:latest

# Install python and pip
RUN apk add --update py-pip

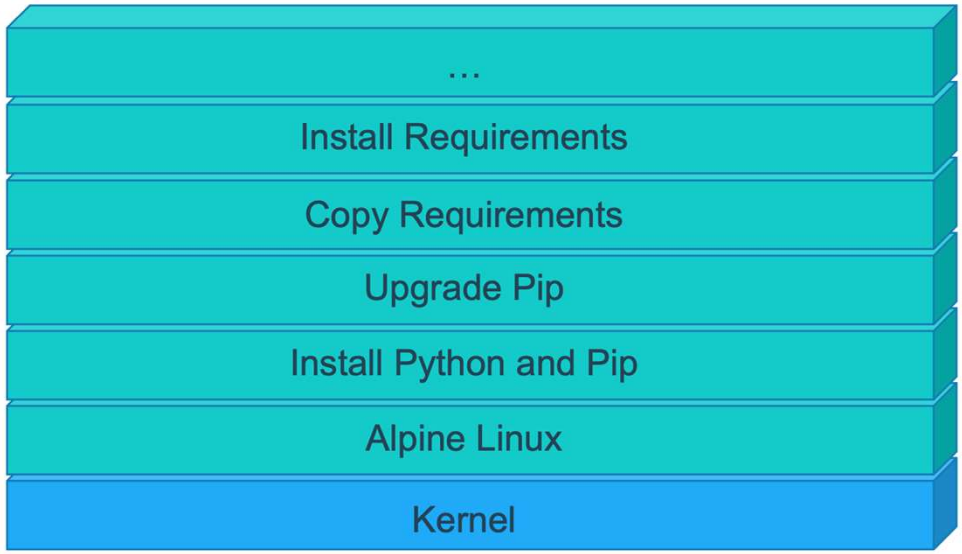
# upgrade pip
RUN pip install --upgrade pip

# install Python modules needed by the Python app
COPY requirements.txt /usr/src/app/
RUN pip install --no-cache-dir -r /usr/src/app/requirements.txt

# copy files required for the app to run
COPY app.py /usr/src/app/
COPY templates/index.html /usr/src/app/templates/

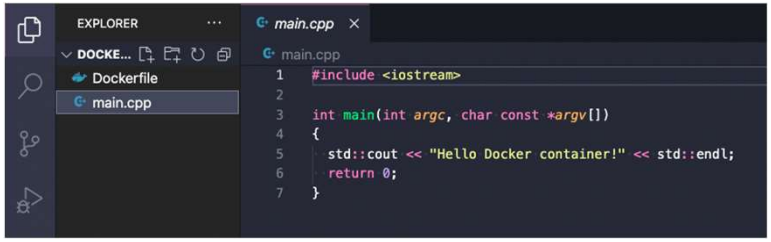
# tell the port number the container should expose
EXPOSE 5000

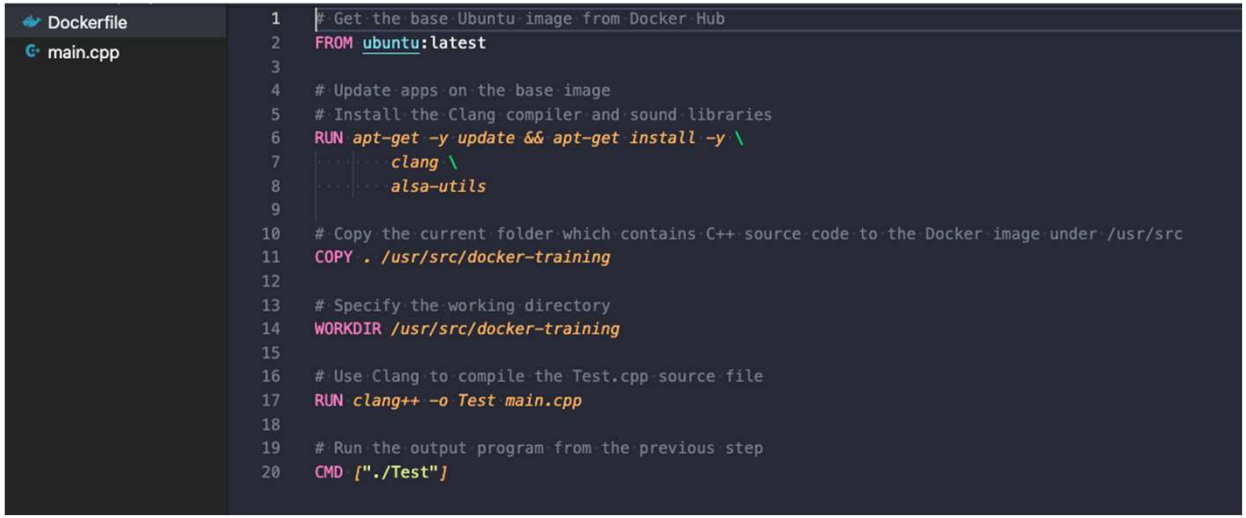
# run the application
CMD ["python", "/usr/src/app/app.py"]
```



04 – BUILD YOUR APPLICATION

Exercise: given the `main.cpp` and the `Dockerfile`, build your image





➢ `docker build -t image_name dockerfile_path`

✏️ The `docker build` command builds Docker images from a Dockerfile

04 – BUILD YOUR APPLICATION

Changing your start image

Dockerfile
main.cpp

```
# Get the base Ubuntu image from Docker Hub
FROM ubuntu:latest

# Update apps on the base image
# Install the Clang compiler and sound libraries
RUN apt-get -y update && apt-get install -y \
        clang \
        alsa-utils

# Copy the current folder which contains C++ source code to the Docker image under /usr/src
COPY . /usr/src/docker-training

# Specify the working directory
WORKDIR /usr/src/docker-training

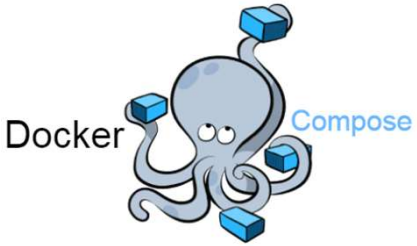
# Use Clang to compile the Test.cpp source file
RUN clang++ -o Test main.cpp

# Run the output program from the previous step
CMD ["./Test"]
```

`icubteamcode/superbuild:v2022.02.1_sources`

04 – Build Your Application

What is Docker Compose?



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

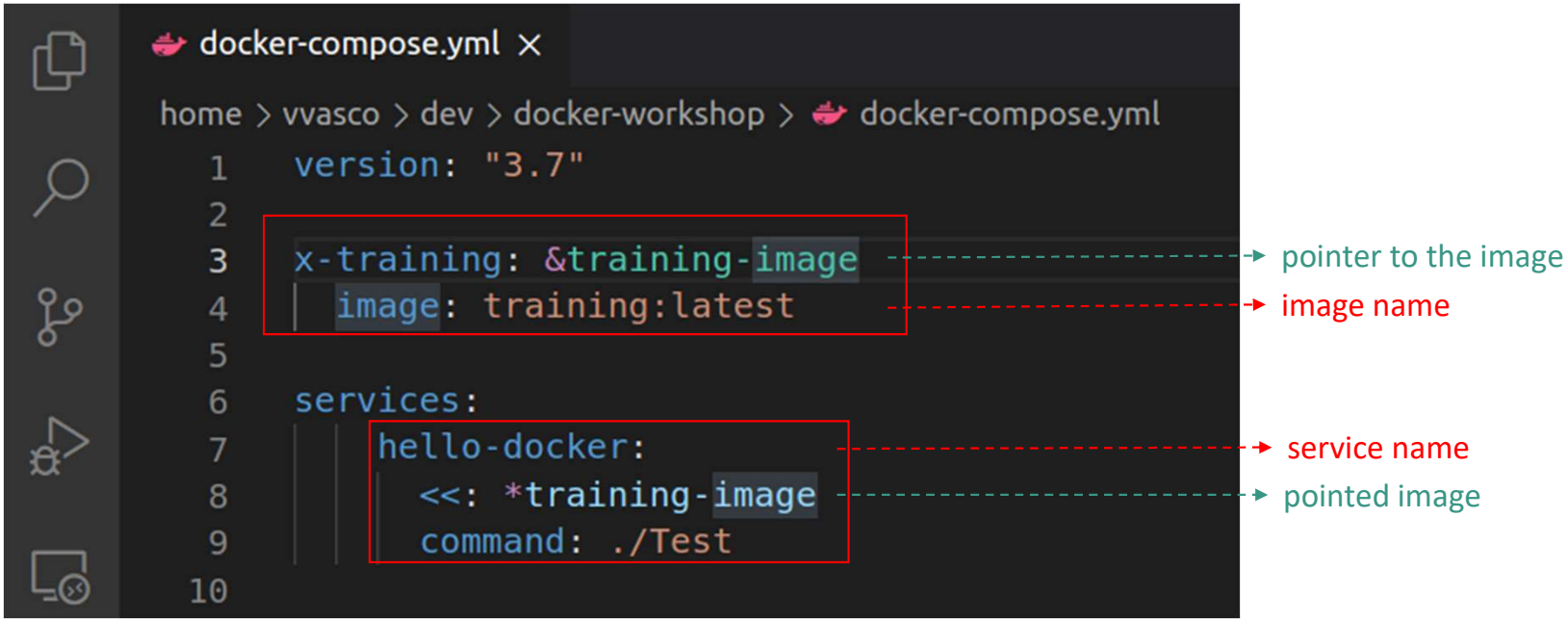
Why Docker Compose?

Configuring the application services through a **unique configuration file**

Create and start multiple services with a **single command**

02 – Build Your Application

How to create a YML configuration file



04 – Build Your Application

How to run the YML configuration file

- `docker-compose up`
- `docker-compose –f /path/to/file up`

Exercise: run your docker compose file and then check containers list

- `docker container ls -a`

```
CONTAINER ID   IMAGE             COMMAND    CREATED          STATUS                       PORTS     NAMES
9a59d1d4ce92   training:latest   "./Test"   5 seconds ago    Exited (0) 3 seconds ago             docker-workshop_hello-docker_1
```

Note the container's name!

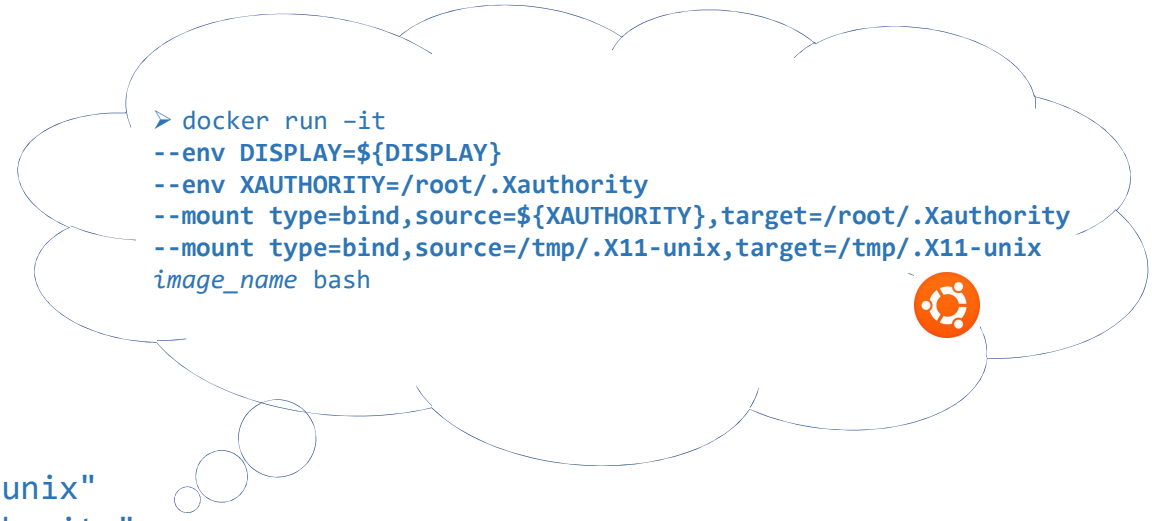
How to stop the YML configuration file

- `docker-compose down`
- `docker-compose -f /path/to/file down`

The containers names are a combination of a project name and the service name. You can specify the project name by setting `--project-name "name"`, or by setting an environment variable: `COMPOSE_PROJECT_NAME`

04 – Build Your Application

Sharing environment and volumes



Example:

```
x-training: &training-image
  image: training:latest

  volumes:
    - "/tmp/.X11-unix:/tmp/.X11-unix"
    - "${XAUTHORITY}:/root/.Xauthority"
    - "${HOME}/.config/yarp:/root/.config/yarp"

  environment:
    - DISPLAY=${DISPLAY}
    - QT_X11_NO_MITSHM=1
    - XAUTHORITY=/root/.Xauthority
```

04 – Build Your Application

Sharing environment and volumes

Example:

```
x-training: &training-image
  image: training:latest

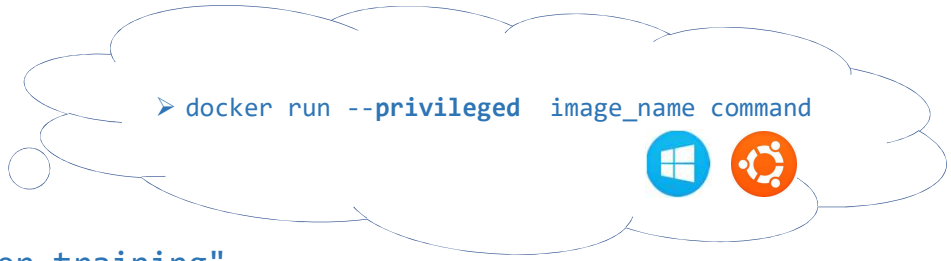
  environment:
    - DISPLAY=YOUR_IP_ADDR:0

  privileged: true
```

04 – Build Your Application

Sharing devices

```
x-training: &training-image
  image: training:latest
  privileged: true
  volumes:
    - "${HOME}:/usr/src/docker-training"
```




```
services:
  share-dev:
    <<: *training-image
    devices:
      - "/dev/snd:/dev/snd"
    command: sh -c "arecord -f S16_LE -d 10 -r 16000 /usr/src/docker-training/test-mic.wav"
```

04 – Build Your Application

Adding as many services as you want → the depends-on option

```
services:
    yarp-server:
      <<: *training-image
      command: sh -c "yarpserver --write"

    yarp-dev:
      <<: *training-image
      command: sh -c "yarpdev --device fakeFrameGrabber --width 640 --height 480"

    yarp-view:
      <<: *training-image
      command: sh -c "yarpview --name /view --x 0 --y 0 --p 50"

    yarp-connect:
      <<: *training-image
      command: sh -c "yarp connect /grabber /view tcp"
```

04 – Build Your Application

Adding as many services as you want → the depends-on option

```
services:
    yarp-server:
      <<: *training-image
      command: sh -c "yarpserver --write"

    yarp-dev:
      <<: *training-image
      command: sh -c "yarp detect --write; yarpdev --device fakeFrameGrabber --width 640 --height 480"

    yarp-view:
      <<: *training-image
      command: sh -c "yarp detect --write; yarpview --name /view --x 0 --y 0 --p 50"

    yarp-connect:
      <<: *training-image
      command: sh -c "yarp detect --write; yarp connect /grabber /view tcp"
```

04 – Build Your Application

Exercise: execute this `docker-compose.yml` by running `docker-compose up` from the right folder

```
yarp-server_1    | Using port database: :memory:
yarp-server_1    | Using subscription database: :memory:
yarp-server_1    | IP address: default
yarp-server_1    | Port number: 10000
yarp-server_1    | [INFO] |yarp.os.Port| Port /laura active at tcp://172.26.0.3:10000/
yarp-connect_1   | This images has release=master and had been building with superbuild_tag=Stable. Metadata=16/09/2021
yarp-connect_1   | [ERROR] |yarp.os.Network| Failure: could not find source port /grabber
basic_yarp-connect_1 exited with code 1
yarp-dev_1       | [WARNING] |yarp.device.grabberDual| The 'grabberDual' device is deprecated in favour of 'frameGrabber_nws_yarp' (and eventually 'frameGrabberCropper
').
yarp-dev_1       | [WARNING] |yarp.device.grabberDual| The old device is no longer supported, and it will be deprecated in YARP 3.6 and removed in YARP 4.
yarp-dev_1       | [WARNING] |yarp.device.grabberDual| Please update your scripts.
yarp-dev_1       | [INFO] |yarp.device.grabberDual| Period parameter not found, using default of 0.03 s
yarp-dev_1       | [WARNING] |yarp.device.grabberDual| 'capabilities' parameter not found or misspelled, the option available are COLOR(default) and RAW, using default
yarp-dev_1       | [INFO] |yarp.os.Port| Port /grabber/rpc active at tcp://172.26.0.5:10003/
yarp-dev_1       | [INFO] |yarp.os.Port| Port /grabber active at tcp://172.26.0.5:10004/
yarp-dev_1       | [INFO] |yarp.device.fakeFrameGrabber| Test grabber period 0.0333333 / freq 30 , mode [line]
yarp-dev_1       | [INFO] |yarp.os.Port| Port /fakeFrameGrabber/rpc active at tcp://172.26.0.5:10005/
yarp-dev_1       | [INFO] |yarp.dev.PolyDriver| Created device <fakeFrameGrabber>. See C++ class FakeFrameGrabber for documentation.
yarp-dev_1       | [INFO] |yarp.dev.PolyDriver| Created wrapper <grabberDual>. See C++ class ServerGrabber for documentation.
```

Note how to debug a container!

```
laura@laura-VirtualBox:~/docker-training/basic$ docker logs basic_yarp-connect_1
This images has release=master and had been building with superbuild_tag=Stable. Metadata=16/09/2021
[ERROR] |yarp.os.Network| Failure: could not find source port /grabber
```

04 – Build Your Application

Adding as many services as you want → the depends-on option

```
services:
    yarp-server:
      <<: *training-image
      command: sh -c "yarpserver --write"

    yarp-dev:
      <<: *training-image
      depends_on:
        - yarp-server
      command: sh -c "yarpdev --device fakeFrameGrabber --width 640 --height 480"

    yarp-view:
      <<: *training-image
      depends_on:
        - yarp-dev
      command: sh -c "yarpview --name /view --x 0 --y 0 --p 50"

    yarp-connect:
      <<: *training-image
      depends_on:
        - yarp-view
        - yarp-dev
      command: sh -c "yarp connect /grabber /view tcp"
```

The **depends-on** option expresses dependency between services.

04 – Build Your Application

Adding as many services as you want → the depends-on option

```
services:
    yarp-server:
      <<: *training-image
      command: sh -c "yarpserver --write"

    yarp-dev:
      <<: *training-image
      depends_on:
        - yarp-server
      command: sh -c “sleep 10; yarpdev --device fakeFrameGrabber --width 640 --height 480"

    yarp-view:
      <<: *training-image
      depends_on:
        - yarp-dev
      command: sh -c "yarpview --name /view --x 0 --y 0 --p 50"

    yarp-connect:
      <<: *training-image
      depends_on:
        - yarp-view
        - yarp-dev
      command: sh -c "yarp connect /grabber /view tcp"
```

04 – Build Your Application

Exercise: execute this `docker-compose.yml` by running `docker-compose up` from the right folder

```
yarp-view_1      | /view
yarp-view_1      | [INFO] |yarp.os.Port|/view| Port /view active at tcp://10.245.74.24:10002/
yarp-dev_1       | This images has release=v2022.02.0 and had been building with superbuild_tag=Custom. Metadata=08/03/2022
yarp-connect_1   | This images has release=v2022.02.0 and had been building with superbuild_tag=Custom. Metadata=08/03/2022
yarp-connect_1   | [ERROR] |yarp.os.Network| Failure: could not find source port /grabber
compose_yarp-connect_1 exited with code 1
yarp-dev_1       | [WARNING] |yarp.device.grabberDual|fakeFrameGrabber| (device grabberDual) (height 480) (single_threaded 1) (subdevice fakeFrameGrabber
) (width 640)
yarp-dev_1       | [WARNING] |yarp.device.grabberDual|fakeFrameGrabber| The 'grabberDual' device is deprecated in favour of 'frameGrabber_nws_yarp' (and
eventually 'frameGrabberCropper').
yarp-dev_1       | [WARNING] |yarp.device.grabberDual|fakeFrameGrabber| The old device is no longer supported, and it will be deprecated in YARP 3.7 and
removed in YARP 4.
yarp-dev_1       | [WARNING] |yarp.device.grabberDual|fakeFrameGrabber| Please update your scripts.
yarp-dev_1       | [DEBUG] |yarp.dev.PolyDriver|fakeFrameGrabber| Parameters are (device grabberDual) (height 480) (single_threaded 1) (subdevice fakeFra
meGrabber) (width 640)
yarp-dev_1       | [INFO] |yarp.device.grabberDual|fakeFrameGrabber| Period parameter not found, using default of 0.03 s
yarp-dev_1       | [WARNING] |yarp.device.grabberDual|fakeFrameGrabber| 'capabilities' parameter not found or misspelled, the option available are COLOR(
default) and RAW, using default
yarp-dev_1       | [INFO] |yarp.os.Port|/grabber/rpc| Port /grabber/rpc active at tcp://10.245.74.24:10003/
yarp-dev_1       | [INFO] |yarp.os.Port|/grabber| Port /grabber active at tcp://10.245.74.24:10004/
```

Note how to debug a container!

```
icub@iiticublap266:~/Downloads/docker-training/compose$ docker logs compose_yarp-connect_1
This images has release=v2022.02.0 and had been building with superbuild_tag=Custom. Metadata=08/03/2022
[ERROR] |yarp.os.Network| Failure: could not find source port /grabber
```

The **depends-on** option is not the solution because…
… it waits for a service to be up but not for the module to be up!

04 – Build Your Application

Additional options: restart policy

```
services:
    yarp-server:
      <<: *training-image
      command: sh -c "yarpserver --write"

    yarp-dev:
      <<: *training-image
      depends_on:
        - yarp-server
      command: sh -c “sleep 10; yarpdev --device fakeFrameGrabber --width 640 --height 480"

    yarp-view:
      <<: *training-image
      depends_on:
        - yarp-dev
      command: sh -c "yarpview --name /view --x 0 --y 0 --p 50"

    yarp-connect:
      <<: *training-image
      depends_on:
        - yarp-view
        - yarp-dev
      command: sh -c "yarp connect /grabber /view tcp“
      restart: on-failure
```

With the `restart: on-failure` option, the container will get restarted if it stops because of an error.

04 – BUILD YOUR APPLICATION

Exercise: execute this `docker-compose.yml` by running `docker-compose up` from the right folder

```
yarp-server_1    | Call with --help for information on available options
yarp-server_1    | Using port database: :memory:
yarp-server_1    | Using subscription database: :memory:
yarp-server_1    | IP address: default
yarp-server_1    | Port number: 10000
yarp-server_1    | [INFO] |yarp.os.Port|/root| Port /root active at tcp://172.18.0.1:10000/
yarp-dev_1       | This images has release=v2022.02.0 and had been building with superbuild_tag=Custom. Metadata=08/03/2022
yarp-connect_1   | This images has release=v2022.02.0 and had been building with superbuild_tag=Custom. Metadata=08/03/2022
yarp-connect_1   | [ERROR] |yarp.os.Network| Failure: could not find source port /grabber
yarp-view_1      | This images has release=v2022.02.0 and had been building with superbuild_tag=Custom. Metadata=08/03/2022
yarp-view_1      | QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'
yarp-view_1      | qrc:/YARPView/YARPViewStatusBar.qml:102:17: Unable to assign QQuickText to double
yarp-view_1      | qrc:/YARPView/YARPViewStatusBar.qml:101:17: Unable to assign QQuickText to double
compose_yarp-connect_1 exited with code 1
yarp-view_1      | /view
```

```
icub@iiticublap266:~/Downloads/docker-training/compose$ docker logs compose_yarp-connect_1
This images has release=v2022.02.0 and had been building with superbuild_tag=Custom. Metadata=08/03/2022
[ERROR] |yarp.os.Network| Failure: could not find source port /grabber
This images has release=v2022.02.0 and had been building with superbuild_tag=Custom. Metadata=08/03/2022
[ERROR] |yarp.os.Network| Failure: could not find source port /grabber
This images has release=v2022.02.0 and had been building with superbuild_tag=Custom. Metadata=08/03/2022
[ERROR] |yarp.os.Network| Failure: could not find source port /grabber
This images has release=v2022.02.0 and had been building with superbuild_tag=Custom. Metadata=08/03/2022
[ERROR] |yarp.os.Network| Failure: could not find source port /grabber
This images has release=v2022.02.0 and had been building with superbuild_tag=Custom. Metadata=08/03/2022
[ERROR] |yarp.os.Network| Failure: could not find source port /grabber
This images has release=v2022.02.0 and had been building with superbuild_tag=Custom. Metadata=08/03/2022
[ERROR] |yarp.os.Network| Failure: could not find source port /grabber
This images has release=v2022.02.0 and had been building with superbuild_tag=Custom. Metadata=08/03/2022
[ERROR] |yarp.os.Network| Failure: could not find source port /grabber
This images has release=v2022.02.0 and had been building with superbuild_tag=Custom. Metadata=08/03/2022
[INFO] |yarp.os.Network| Success: Added connection from /grabber to tcp://view
```

Note how to debug a container!

04 – HANDS-ON

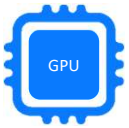
Exercise: Create image and application to launch `yarpdataplayer`

Follow the instructions here:

1. Create an image containing the needed applications (`yarp, yarpdataplayer`) and the data to be played:
    1. Select the correct image to start from;
    2. Download the following compressed sequence:
        - `wget http://www.icub.org/download/software/datasetplayer-demo/dataDisparity.zip`
        - `Move the file to /root`
    3. Unzip `dataDisparity.zip` -> you will have to install:
        - `apt update && apt install –y unzip`
    4. Change the working directory to where you unzip the data
2. Create a compose file to launch the application:
    1. Create services to launch yarp server, yarpdataplayer, yarpview and the connection between `/icub/camcalib/left/out` and yarpview;
3. After launching, select the sequence in `yarpdataplayer`

Useful Links

Docker documentation



GPU support:

- docker
- docker-compose

Docker hub

Basic Docker deployment