Università di Pisa

Dept. of Information Engineering

Course Wireless Networks - 2021/2022

Virtualization (LAB)

Alessandra Fais – PhD Student

email: alessandra.fais@phd.unipi.it
web page: for.unipi.it/alessandra_fais/

LAB organization

- **PART I (theoretical)**
  - Introduction to SDN, NFV, MEC * concepts
  - Cloud computing and service-based architectures

* SDN = Software Defined Networking,
NFV = Network Function Virtualization,
MEC = Multi-access Edge Computing

- **PART II**
  - OpenStack cloud computing platform
  - OpenStack and NFV
  - Live session: OpenStack platform of the DII CrossLab project

LAB organization

* VM = Virtual Machine

- **PART III**
  - Virtualization overview and different approaches
    - VMs* on hypervisors, containers, alternative solutions

- **PART IV**
  - Containers -> Docker
  - Orchestrators -> Kubernetes
  - Hands-on session: Docker, docker-compose, Kubernetes

PART IV

Outline of Part IV

1) Containers characteristics
2) Docker
    - Objects
    - Architecture
    - Deployment modes
        - Single host VS Cluster
3) Kubernetes

Outline of Part IV

Hands-on session:

Docker

- ○ Installation
- ○ Execution flow
- ○ Commands
- ○ Dockerfile
- ○ Docker Compose
- ○ Persistent storage management (volumes)

Kubernetes

- ○ Installation
- ○ Complete application

Hands-on with Docker

Installing Docker

Installing Docker

- Let’s get started with **Docker Engine for Ubuntu**!

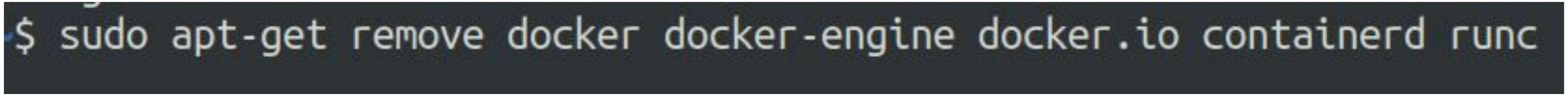
- What you need:
  - The 64-bit version of one of the Ubuntu versions among
    - Impish 21.10
    - Hirsute 21.04
    - Focal 20.04 (LTS)
    - Bionic 18.04 (LTS)

Docker guide section, here:
https://docs.docker.com/engine/install/ubuntu/

Installing Docker

- **First step:** uninstall old versions of Docker

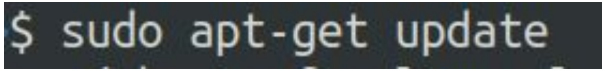
{ **docker, docker.io and docker-engine are the names of the older versions** }



```
$ sudo apt-get remove docker docker-engine docker.io containerd runc
```

- **Second step:** set up the **Docker repository**

1



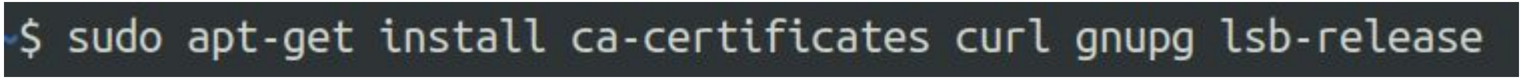
```
$ sudo apt-get update
```

{ **Update the** apt **package index** }

Installing Docker

2

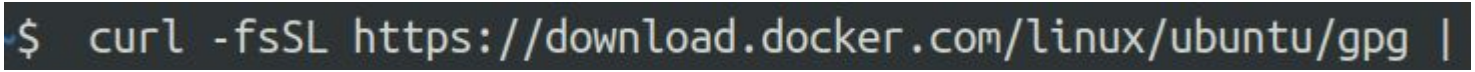
{ Install packages to allow apt to use a repository over HTTPS }



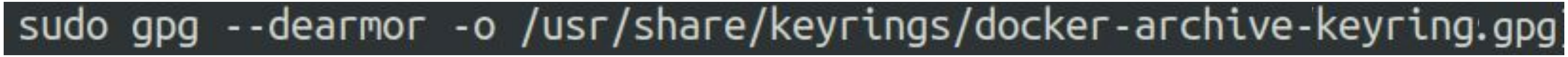
```
$ sudo apt-get install ca-certificates curl gnupg lsb-release
```

3

{ Add Docker's official GPG key }



```
$  curl -fsSL https://download.docker.com/linux/ubuntu/gpg |
```

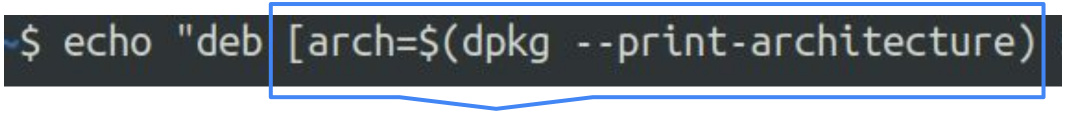


```
sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

Installing Docker

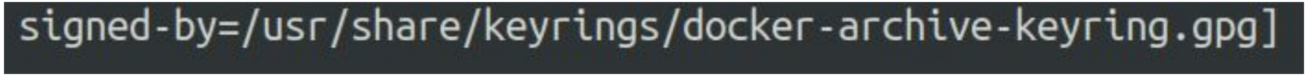
4

{ Set up the *stable* repository. }

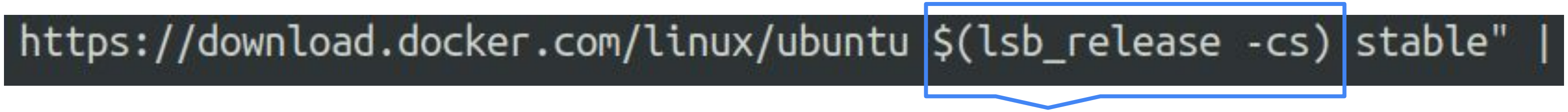


```
$ echo "deb [arch=$(dpkg --print-architecture)
```

in my case arch=amd64

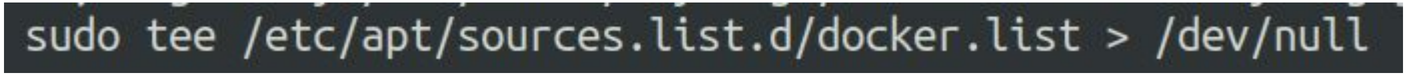


```
signed-by=/usr/share/keyrings/docker-archive-keyring.gpg]
```



```
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" |
```

in my case is focal



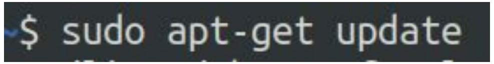
```
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

Note: write all in the same line!

Installing Docker

- **Third step:** install **Docker Engine**, **containerd** and **Docker Compose**

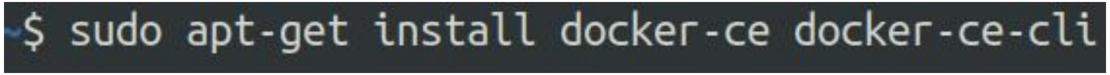
1



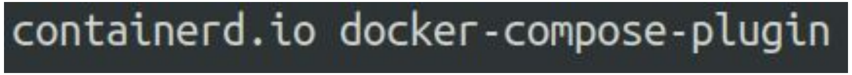
```
$ sudo apt-get update
```

{ **Update the** apt **package index** }

2



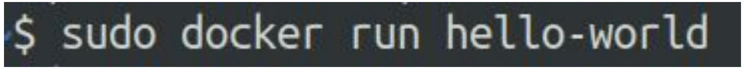
```
$ sudo apt-get install docker-ce docker-ce-cli
```



```
containerd.io docker-compose-plugin
```

{ **Install the latest version of** Docker Engine - Community **and** containerd }

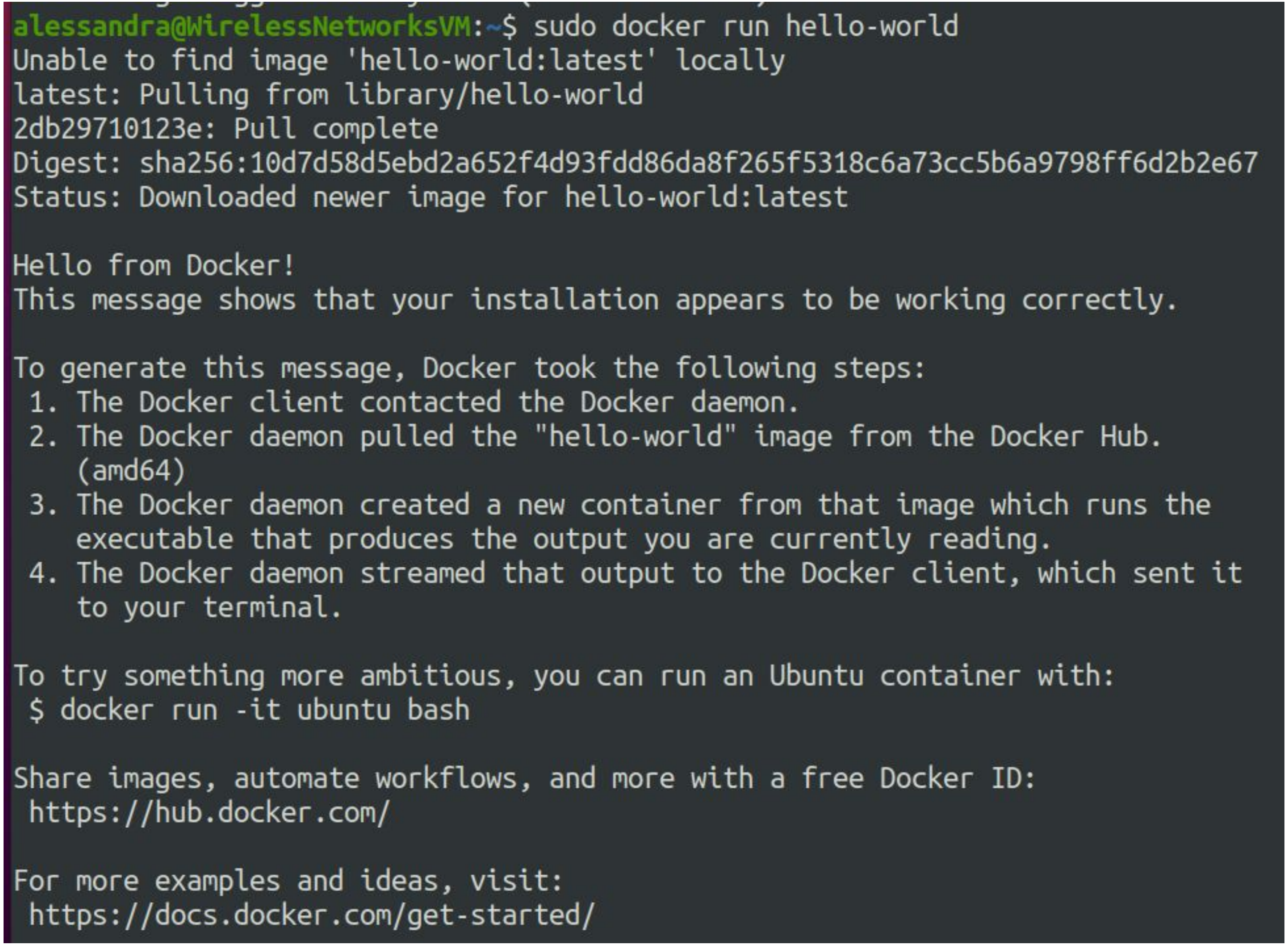
3



```
$ sudo docker run hello-world
```

{ **Verify that** Docker Engine **has been installed correctly by running the** hello-world **image** }

The hello-world **image is a test image that is downloaded and run. When the container runs it, it prints an informational message and exits.**

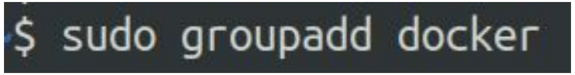


```
alessandra@WirelessNetworksVM:~$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:10d7d58d5ebd2a652f4d93fdd86da8f265f5318c6a73cc5b6a9798ff6d2b2e67
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

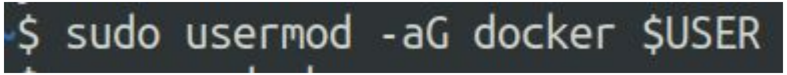
Our first running Docker image!

Use Docker as a non-root user

- Docker needs to be run by prefixing commands with **sudo**
- Execute the following commands to avoid prefacing the **docker** command with **sudo**

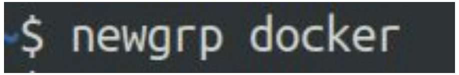


```
$ sudo groupadd docker
```



```
$ sudo usermod -aG docker $USER
```

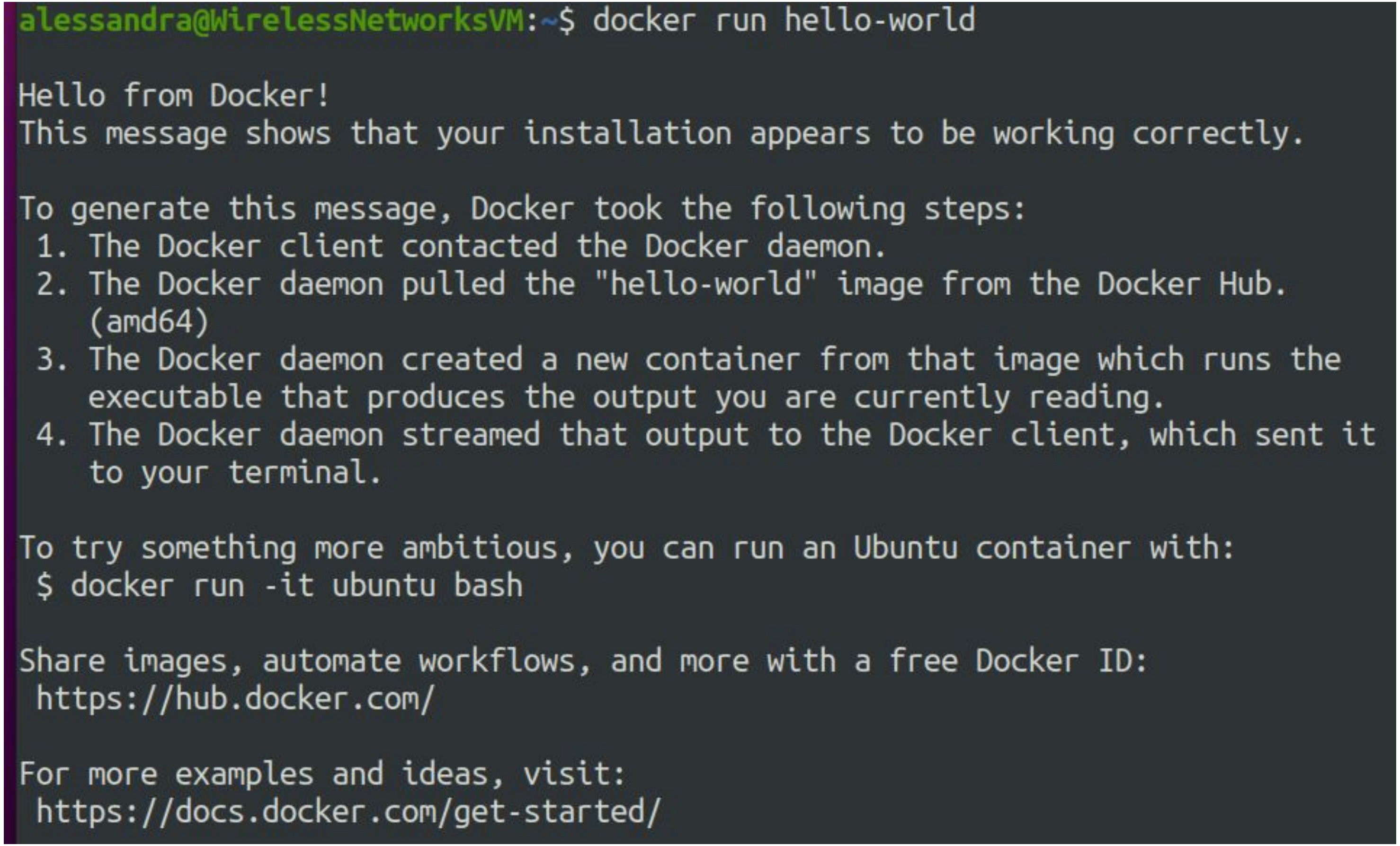
- Activate the changes to groups



```
$ newgrp docker
```

Docker guide section, here:
https://docs.docker.com/install/linux/linux-postinstall/

Use Docker as a non-root user



```
alessandra@WirelessNetworksVM:~$ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

- Verify that you can run **docker** command without **sudo**

Execution flow

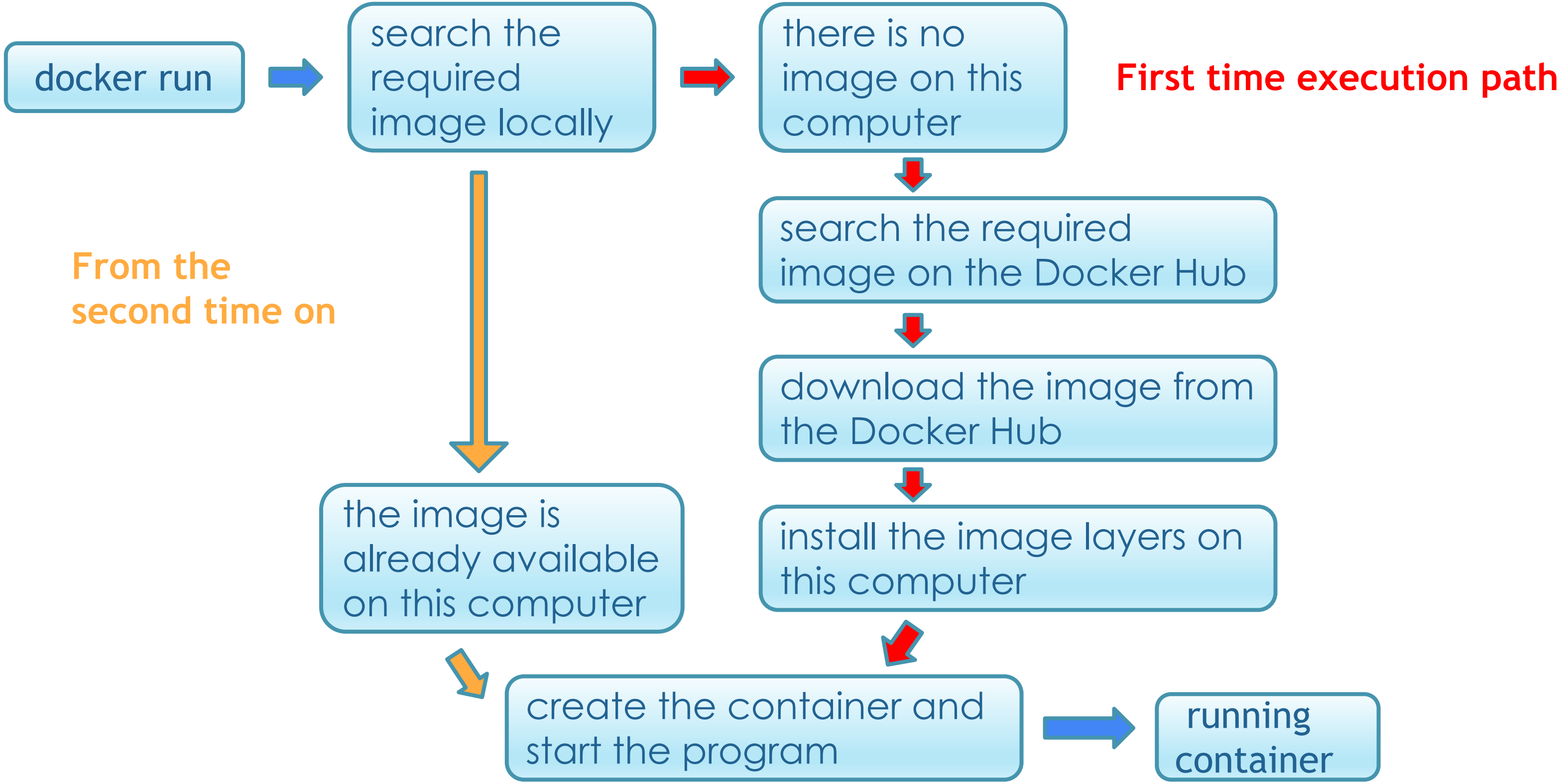
The execution flow

- **docker run** creates a running container starting from the **hello-world** image
- There is not a local copy of the required image: Docker downloads it from the Docker Hub then
- The image is used to create a running container
- The **echo** command executes
- The container is shut down

- **NOTE: running the same image for the second time will be faster!**

This is why a *local copy* of the image is kept, and there will be no need to download it from the Docker Hub after the first time

The execution flow



docker run

search the required image locally

there is no image on this computer

First time execution path

search the required image on the Docker Hub

download the image from the Docker Hub

install the image layers on this computer

From the second time on

the image is already available on this computer

create the container and start the program

running container

Commands

docker commands

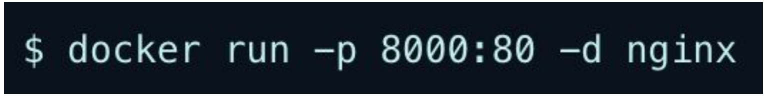
- ps | container ls : list the running containers in the system
  - docker container ls –a (--**all** flag lists all the containers, even if they are not running)
- images | image ls : list the images in the system
- image prune : remove unused images in the system
- run : create a running container starting from an image
- stop : stop a running container
- inspect : show information about a container
- logs : show the logs for a given container
- build : build a **dockerfile**
- search : search for an image in the Docker Hub
- pull : pull down a new image from the Docker Hub
- push : push a new image to the Docker Hub

Docker guide section, here: https://docs.docker.com/engine/reference/commandline/run/

run command useful flags

- A container can be run with several arguments set
  - -p HOST_PORT:CLIENT_PORT : port mapping
  - -d : detached mode (run container in background)
  - -i : interactive session (keep STDIN open)
  - --name CONTAINER_NAME
  - -t : attached text terminal

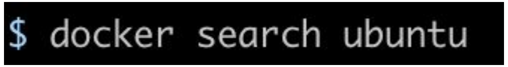
- Example:



```
$ docker run -p 8000:80 -d nginx
```

Whatever is running on port 80 in the nginx **container is available on port 8000 of** localhost

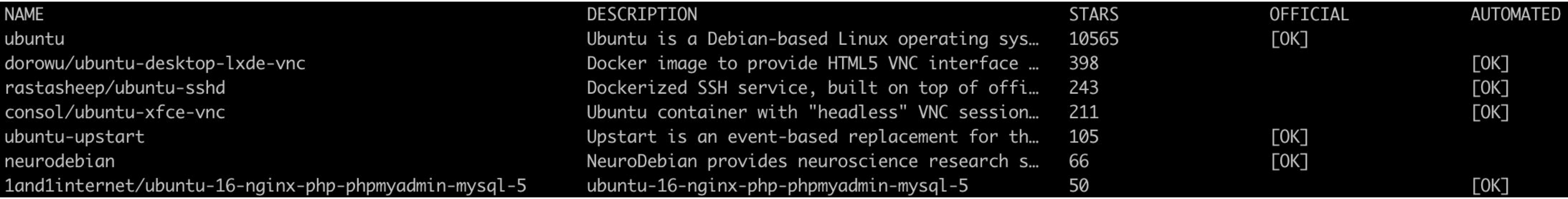
search command



```
$ docker search ubuntu
```

1

Search the Docker Hub registry for a ubuntu image

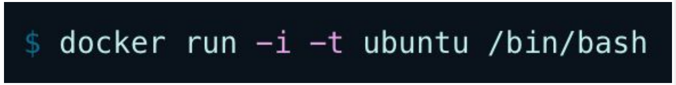


```
NAME                                                DESCRIPTION                                    STARS     OFFICIAL   AUTOMATED
ubuntu                                              Ubuntu is a Debian-based Linux operating sys…  10565     [OK]
dorowu/ubuntu-desktop-lxde-vnc                      Docker image to provide HTML5 VNC interface …  398                  [OK]
rastasheep/ubuntu-sshd                              Dockerized SSH service, built on top of offi…  243                  [OK]
consol/ubuntu-xfce-vnc                              Ubuntu container with "headless" VNC session…  211                  [OK]
ubuntu-upstart                                      Upstart is an event-based replacement for th…  105       [OK]
neurodebian                                         NeuroDebian provides neuroscience research s…  66        [OK]
1and1internet/ubuntu-16-nginx-php-phpmyadmin-mysql-5  ubuntu-16-nginx-php-phpmyadmin-mysql-5         50                   [OK]
```

2

Results can come from the top-level namespace for official image **or from the** public repository of a user

3



```
$ docker run -i -t ubuntu /bin/bash
```

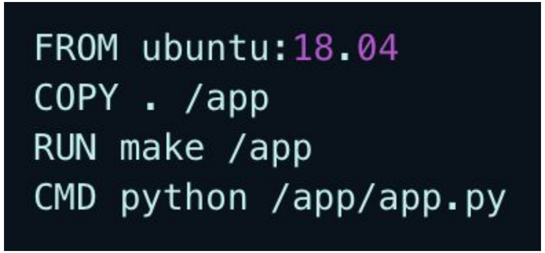
- **The** ubuntu **official image is pulled from the Docker Hub**
- **A new container is created and a** local read-write filesystem **is allocated to it**
- **A** network interface **is created to connect the container to the default network (an IP address is assigned to the container)**
- /bin/bash **is executed as the container starts: input can be provided using the keyboard and output is logged to our terminal**
- **Typing** exit **to terminate the** /bin/bash **cmd stops the container**

Dockerfile

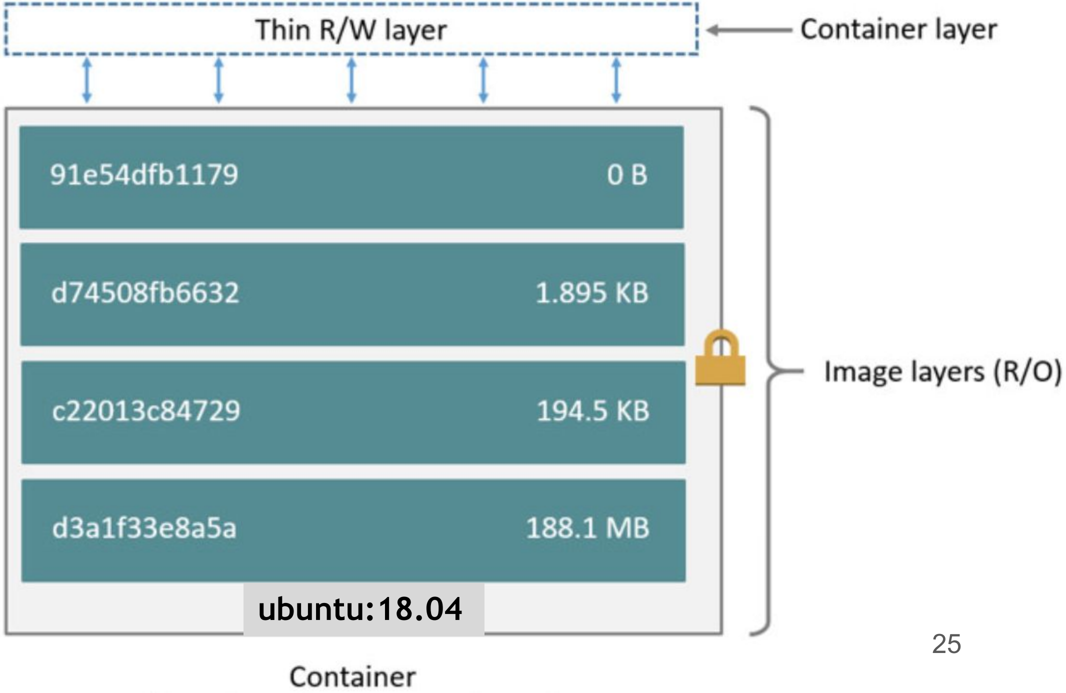
Dockerfile

- Defines the **steps** needed to create an image and run it
- Each **instruction** in the Dockerfile creates a **layer** in the image
  - readable/writeable layer on top of a bunch of read-only layers
- Only the layers which have been modified after a change in the Dockerfile are rebuilt
- Visualize all the layers by running **docker history** <image_name>

Dockerfile

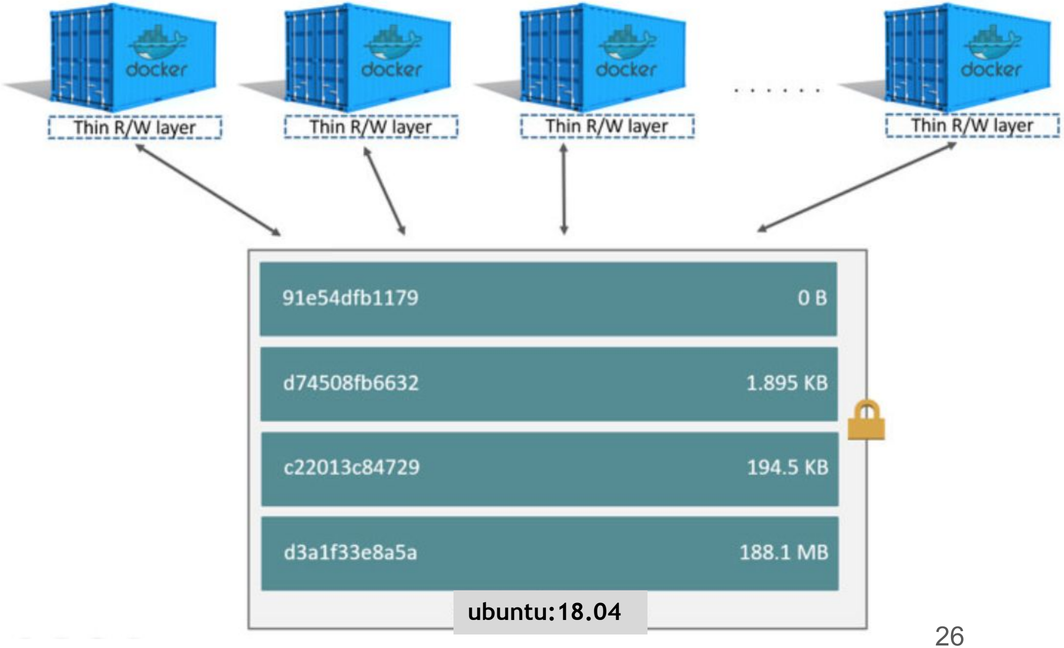


```
FROM ubuntu:18.04
COPY . /app
RUN make /app
CMD python /app/app.py
```



Dockerfile

- All writes in a container are stored in the **top layer**
- When the container is deleted the writable layer is removed, while the **underlying image remains unchanged**
- Multiple containers can share the same underlying image and yet have their own data state (different top layer)
- Only the differences between a layer and the underlying one are stored



91e54dfb1179	0 B
d74508fb6632	1.895 KB
c22013c84729	194.5 KB
d3a1f33e8a5a	188.1 MB

ubuntu:18.04

Dockerfile instructions

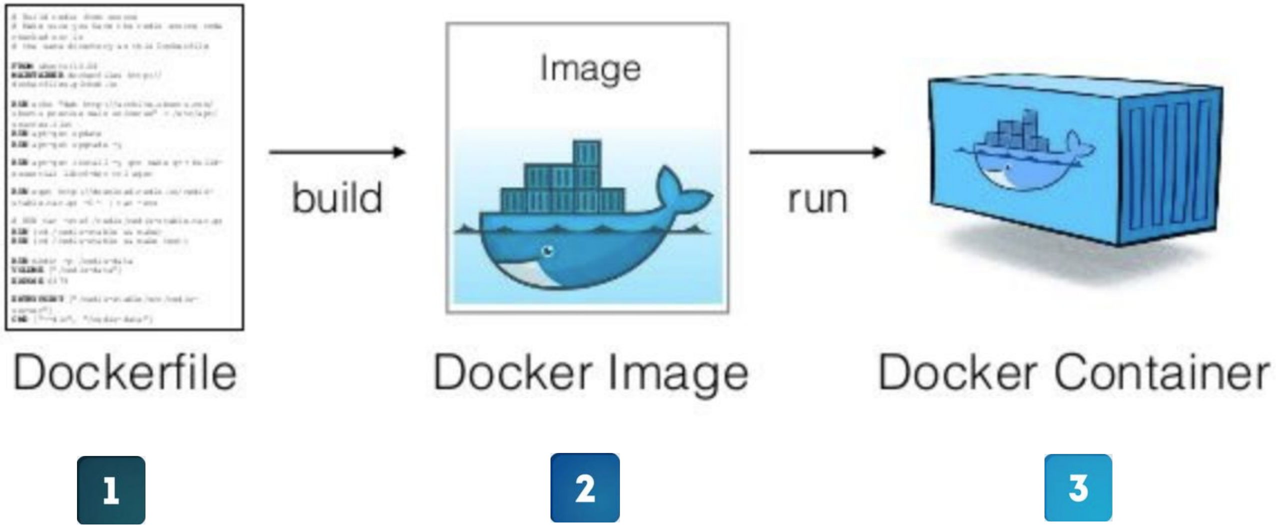
- FROM <image>[:tag] : start your image from a pre-existing parent image (e.g., official Docker image) called **base image**
- WORKDIR : working directory in the image private filesystem
- USER : specify the user used to run any subsequent RUN, CMD, ENTRYPOINT, … instructions
- COPY <src> <dst> : copy a file from the host to the image filesystem
- ENV : define environment variables (available in the container and in the subsequent instructions in the Dockerfile)
- RUN <cmd> | RUN ["exec", "param1", "param2"] : execute commands
- CMD ["exec", "param1", "param2", …] : specify the process to run inside the container when it starts up (a good CMD entry would be an interactive shell)
- EXPOSE : specify a port on which the container will listen at runtime (**note:** in order to publish the port you need to use **docker run -p <port>** when you start the container)

First example (single host)

A simple Docker application

Hands-on with Docker: simple application

- Let’s follow these steps to create a **single-component application**



Link to the code: https://github.com/alefais/virtualization-lab-unipi/tree/main/simple-docker-app

Hands-on with Docker: Dockerfile

```
# Specify a base image
FROM ubuntu:latest

# Set the author of the new image
LABEL maintainer=”alessandra.fais@phd.unipi.it”

# Specify a working directory
WORKDIR /usr/app

# Install needed packages
RUN apt-get update && apt-get install -y cowsay fortune

# Copy files
COPY ./entrypoint.sh ./

# Make the script executable
RUN chmod +x entrypoint.sh

# Configure the container in order to run as an executable
ENTRYPOINT ["/usr/app/entrypoint.sh"]
```

Base image from the Docker Hub

NB: each instruction creates a new layer on top of the current image

Start the container

Link to the code: https://github.com/alefais/virtualization-lab-unipi/tree/main/simple-docker-app

Hands-on with Docker: simple application

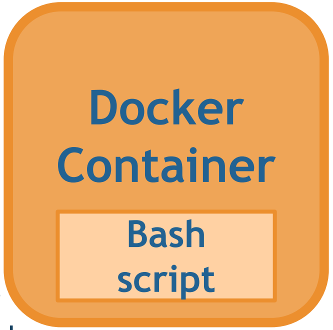
entrypoint.sh

```
#!/bin/bash
path="/usr/games"

$path/fortune | $path/cowthink
```

fortune displays a pseudo random message from a database of quotes

cowthink displays the image of a thinking cow in ASCII art saying the text in input

thanks to the **pipe** between the two commands, the quote generated by **fortune** is passed as input to **cowthink**



- Project structure
  - Root directory: **simple-docker-app**
  - .
    |__ entrypoint.sh
    |__ Dockerfile

Link to the code: https://github.com/alefais/virtualization-lab-unipi/tree/main/simple-docker-app

Hands-on with Docker: simple application

- Run the application

  From the base directory:

  - Run the Docker build process and tag the image
  - Run the container using the image tag

Docker Container
Bash script

```
$ docker build -t fais/cowsay-app .
```

Build context for the image (current directory)

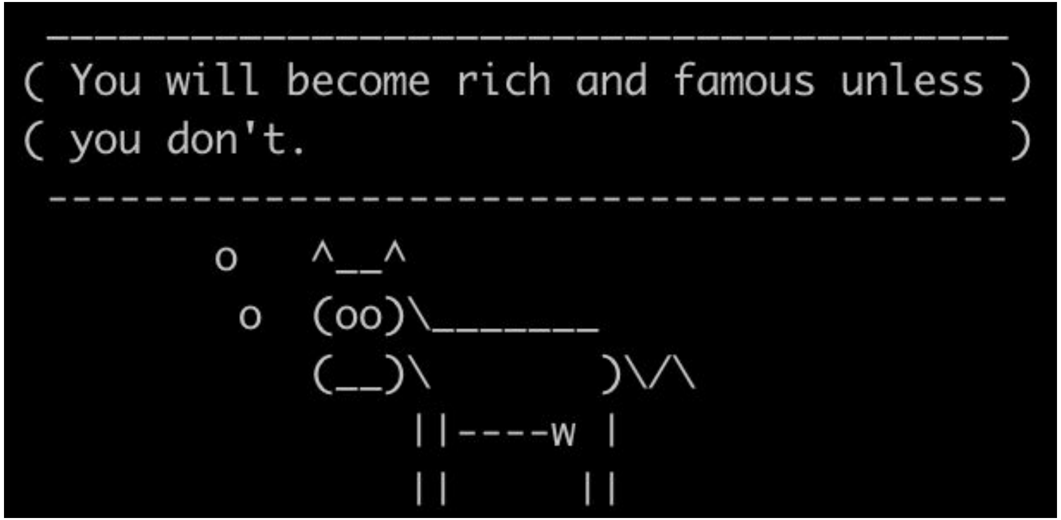
Tag to identify the image

```
$ docker run fais/cowsay-app
```

Link to the code: https://github.com/alefais/virtualization-lab-unipi/tree/main/simple-docker-app

Hands-on with Docker: simple application

- One of the possible execution results:



```
 ________________________________________
( You will become rich and famous unless )
( you don't.                             )
 ----------------------------------------
        o   ^__^
         o  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Docker Container
Bash script

Link to the code: https://github.com/alefais/virtualization-lab-unipi/tree/main/simple-docker-app

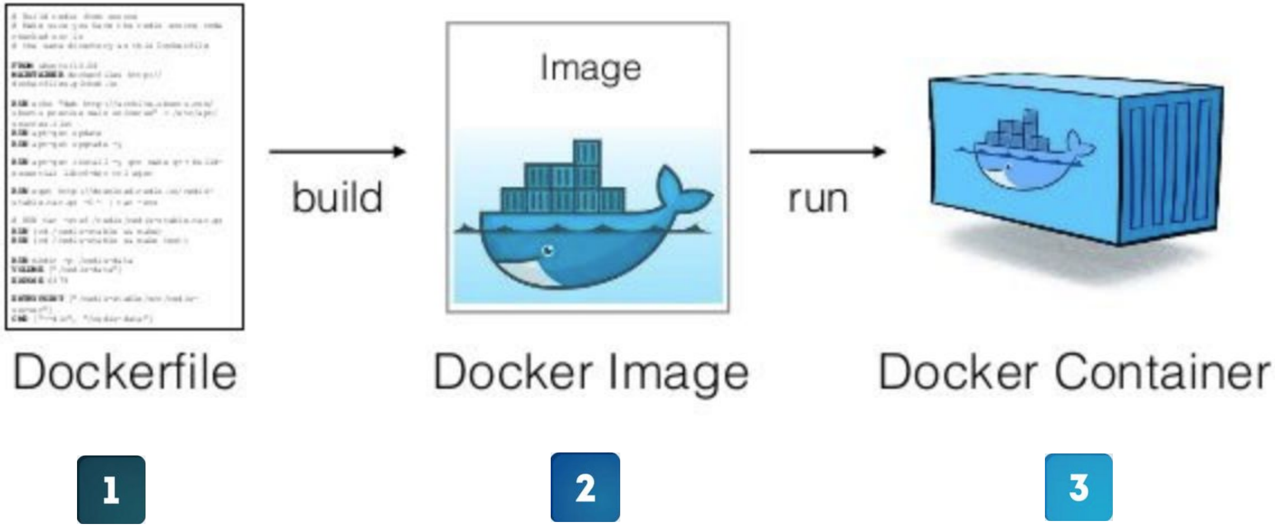
Second example (single host)

A composed Docker application

Introduction to docker-compose

Hands-on with Docker: complex example

- Let’s follow these steps to create each service in our **multi-component application**

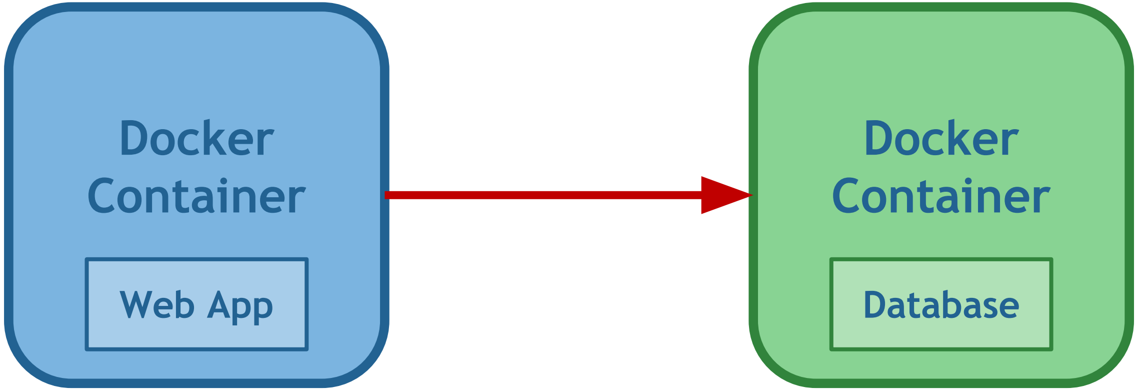


- Then set up networking and compose the services together

Link to the code: https://github.com/alefais/virtualization-lab-unipi/tree/main/composed-docker-app

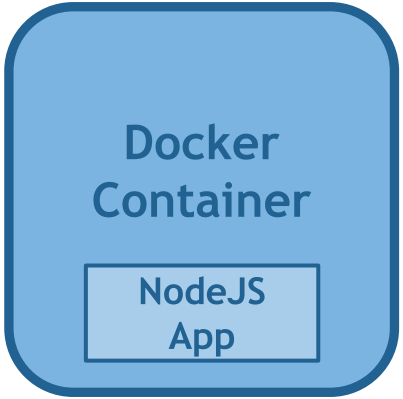
Hands-on with Docker: complex example

- Application implemented as two interacting services (Docker Stack)
- Idea : keep track of the number of visits to the web application
  - each time someone visits the app, the visitor counter is incremented



Link to the code: https://github.com/alefais/virtualization-lab-unipi/tree/main/composed-docker-app

Hands-on with Docker: the web service



- **Web Application component logic**
  - Receive an incoming request
  - Send back as response the number of received visits
- Steps
  - Create the project directory **composed-docker-app**
  - Create the file **index.js** containing the NodeJS app logic
  - Create the file **package.json** containing the NodeJS app dependencies
- How to run a NodeJS app (see Dockerfile)
  - Install dependencies for the NodeJS app with **npm install**
  - Start the app with **npm run start**

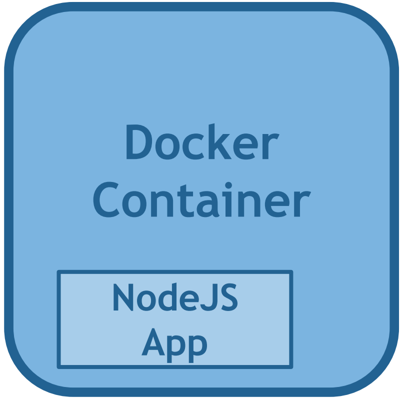
Link to the code: https://github.com/alefais/virtualization-lab-unipi/tree/main/composed-docker-app

Hands-on with Docker: index.js

```
const express = require('express')
const redis = require('redis')

const app = express()
const client = redis.createClient({
        host: 'redis-db',
        port: 6379
})

client.set('visits', 0);

app.get('/', (req, res) => {

        client.get('visits', (err, visits) => {

                res.send('Number of visits is: ' + visits)

                client.set('visits', parseInt(visits) + 1)
        })

})

app.listen(8081, ()=>{ console.log('Listening on port 8081') })
```

Use the Express framework (to easily create an HTTP server)

Docker Compose service name



Set initial visits value

Define the root endpoint (wait for an HTTP get request on the / endpoint)

When this happens:

1. Send response to the web page (updated visit counter)
2. Update DB content

Specify the listening port to 8081

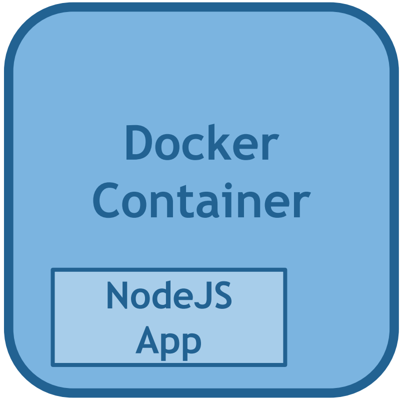
Link to the code: https://github.com/alefais/virtualization-lab-unipi/tree/main/composed-docker-app

Hands-on with Docker: package.json

```
{
        "dependencies": {
                "express": "*",
                "redis": "^3.1.1"
        },
        "scripts": {
                "start": "node index.js"
        }
}
```

Express framework
https://expressjs.com/

Latest Redis version compatible with version 3.1.1
https://hub.docker.com/_/redis/



Link to the code: https://github.com/alefais/virtualization-lab-unipi/tree/main/composed-docker-app

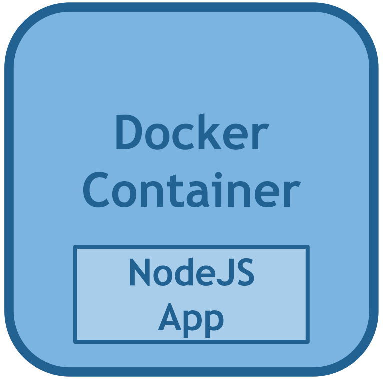
Hands-on with Docker: Dockerfile

```
# Specify a base image
FROM node:alpine

# Set the author of the new image
LABEL maintainer=”alessandra.fais@phd.unipi.it”

# Specify a working directory
WORKDIR /usr/app

# Copy the dependencies file
COPY ./package.json ./

# Install dependencies
RUN npm install

# Copy remaining files
COPY ./ ./

# Default command
CMD ["npm","start"]
```

Lightweight Node Docker image having node and npm already installed

Copy project files to the image

Start the container

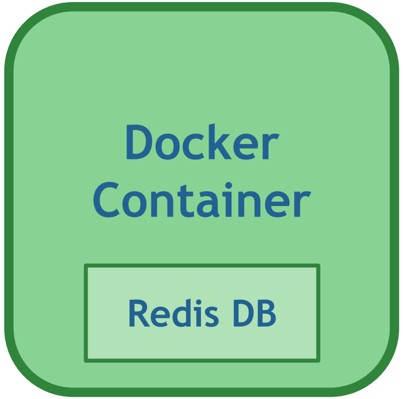


Link to the code: https://github.com/alefais/virtualization-lab-unipi/tree/main/composed-docker-app

Hands-on with Docker: the database

- **Database component logic**
  - Store the updated counter value
- Steps
  - Use the official **redis** image from the Docker Hub
- Project structure
  - Root directory: **composed-docker-app**

  -
    ```
    .
    |__ index.js
    |__ package.json
    |__ Dockerfile
    ```



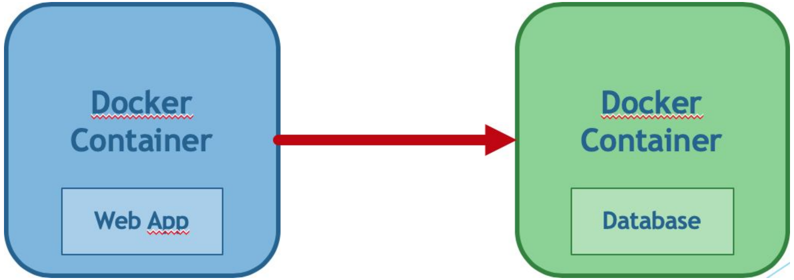
Link to the code: https://github.com/alefais/virtualization-lab-unipi/tree/main/composed-docker-app

Hands-on with Docker: compose the app

- Connect together the two Docker containers
  - Define a **Docker Compose YAML** file

- Steps
  - Create the file docker-compose.yml

- Project structure
  - Root directory: composed-docker-app
  - .
    |__ index.js
    |__ package.json
    |__ Dockerfile
    |__ docker-compose.yml



We use docker-compose instead of the approach of the previous simple example

Link to the code: https://github.com/alefais/virtualization-lab-unipi/tree/main/composed-docker-app

Hands-on with Docker: docker-compose.yml

```
version: "3"
services:
  redis-db:
    image: redis
    ports:
      - "6379:6379"
node-web-app:
   build: .
   ports:
     - "4001:8081"
```

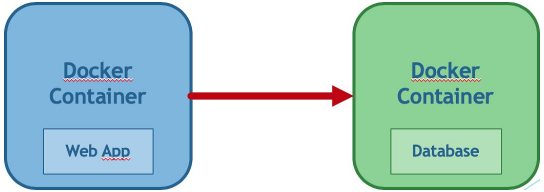
Docker Compose version

Services (containers) which are part of the Docker Compose build

Service based on the **redis:latest** image from Docker Hub (use standard port)

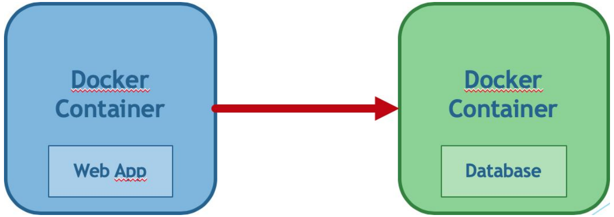
Service built using the Dockerfile in the root directory

Map port **4001** on the local host to port **8081** on the Docker container



Link to the code: https://github.com/alefais/virtualization-lab-unipi/tree/main/composed-docker-app

Hands-on with Docker: run the application



From the root directory:

- Start up the containers

  docker-compose up

You can now access the application at **http://localhost:4001**

- Stop/start the containers

  docker-compose stop
  docker-compose start

- Stop the app and remove containers and networks

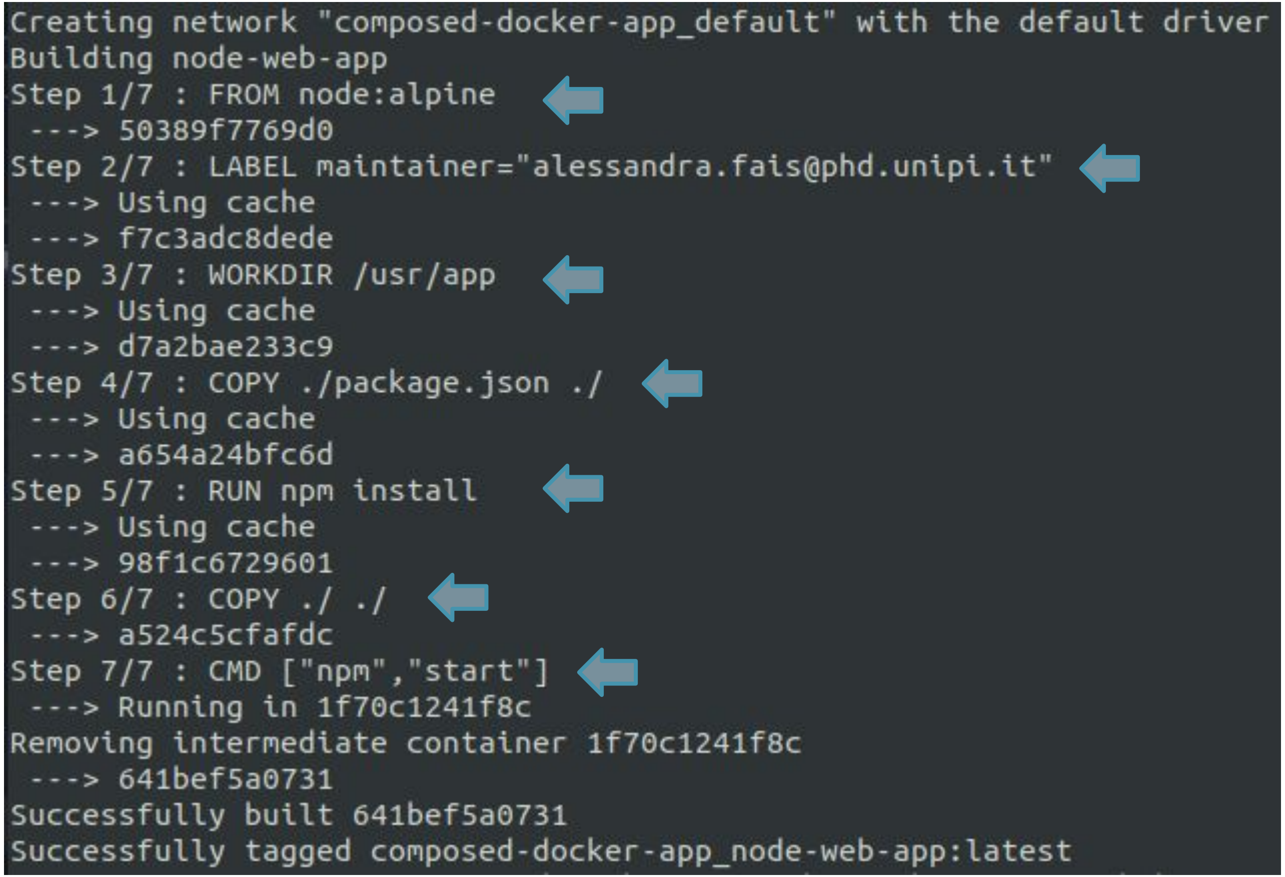
  docker-compose down

Here what happens under the hood:

1. Stop node-web-app container
2. Stop redis-db container
3. Remove node-web-app container
4. Remove redis-db container
5. Remove the network

Link to the code: https://github.com/alefais/virtualization-lab-unipi/tree/main/composed-docker-app

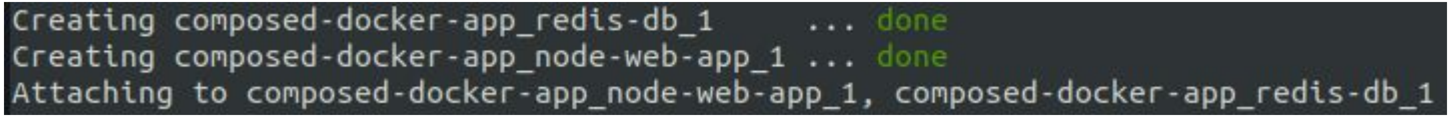
Hands-on with Docker: execution output



```
Creating network "composed-docker-app_default" with the default driver
Building node-web-app
Step 1/7 : FROM node:alpine
 ---> 50389f7769d0
Step 2/7 : LABEL maintainer="alessandra.fais@phd.unipi.it"
 ---> Using cache
 ---> f7c3adc8dede
Step 3/7 : WORKDIR /usr/app
 ---> Using cache
 ---> d7a2bae233c9
Step 4/7 : COPY ./package.json ./
 ---> Using cache
 ---> a654a24bfc6d
Step 5/7 : RUN npm install
 ---> Using cache
 ---> 98f1c6729601
Step 6/7 : COPY ./ ./
 ---> a524c5cfafdc
Step 7/7 : CMD ["npm","start"]
 ---> Running in 1f70c1241f8c
Removing intermediate container 1f70c1241f8c
 ---> 641bef5a0731
Successfully built 641bef5a0731
Successfully tagged composed-docker-app_node-web-app:latest
```

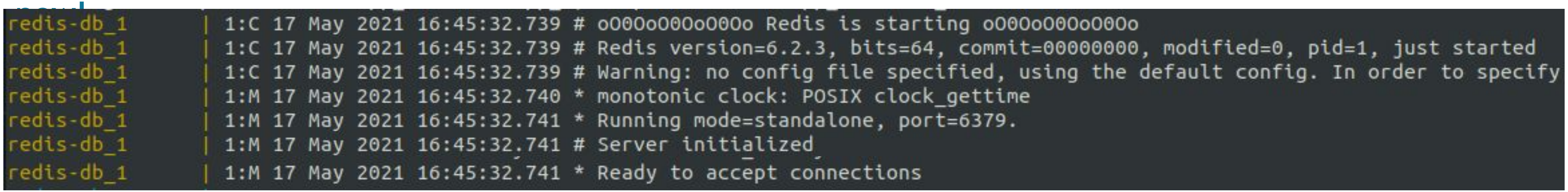
Link to the code: https://github.com/alefais/virtualization-lab-unipi/tree/main/composed-docker-app

Hands-on with Docker: execution output



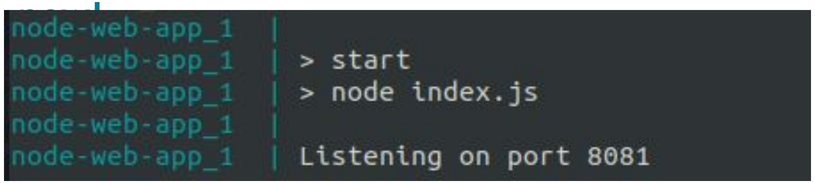
```
Creating composed-docker-app_redis-db_1     ... done
Creating composed-docker-app_node-web-app_1 ... done
Attaching to composed-docker-app_node-web-app_1, composed-docker-app_redis-db_1
```

redis-db is up and running



```
redis-db_1      | 1:C 17 May 2021 16:45:32.739 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
redis-db_1      | 1:C 17 May 2021 16:45:32.739 # Redis version=6.2.3, bits=64, commit=00000000, modified=0, pid=1, just started
redis-db_1      | 1:C 17 May 2021 16:45:32.739 # Warning: no config file specified, using the default config. In order to specify
redis-db_1      | 1:M 17 May 2021 16:45:32.740 * monotonic clock: POSIX clock_gettime
redis-db_1      | 1:M 17 May 2021 16:45:32.741 * Running mode=standalone, port=6379.
redis-db_1      | 1:M 17 May 2021 16:45:32.741 # Server initialized
redis-db_1      | 1:M 17 May 2021 16:45:32.741 * Ready to accept connections
```

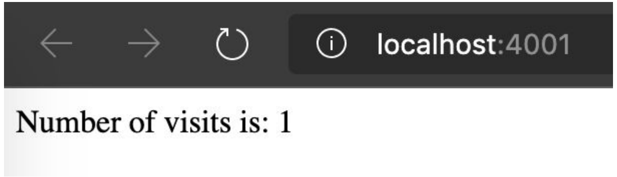
node-web-app is up and running



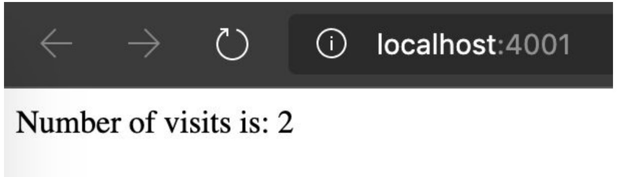
```
node-web-app_1  |
node-web-app_1  | > start
node-web-app_1  | > node index.js
node-web-app_1  |
node-web-app_1  | Listening on port 8081
```

Link to the code: https://github.com/alefais/virtualization-lab-unipi/tree/main/composed-docker-app

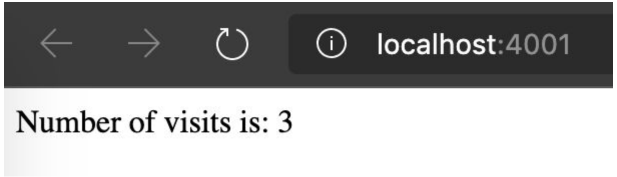
Hands-on with Docker: test with a browser



Number of visits is: 1



Number of visits is: 2



Number of visits is: 3

1st request

2nd request

3rd request

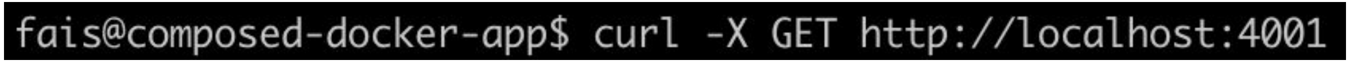
. . .

Link to the code: https://github.com/alefais/virtualization-lab-unipi/tree/main/composed-docker-app

Hands-on with Docker: test with **curl**

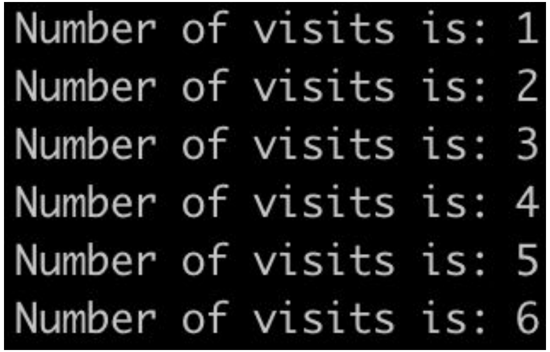
We can send some **HTTP GET** requests by using **curl**

- Request



```
fais@composed-docker-app$ curl -X GET http://localhost:4001
```

- Reply



```
Number of visits is: 1
Number of visits is: 2
Number of visits is: 3
Number of visits is: 4
Number of visits is: 5
Number of visits is: 6
```

1st request

2nd request

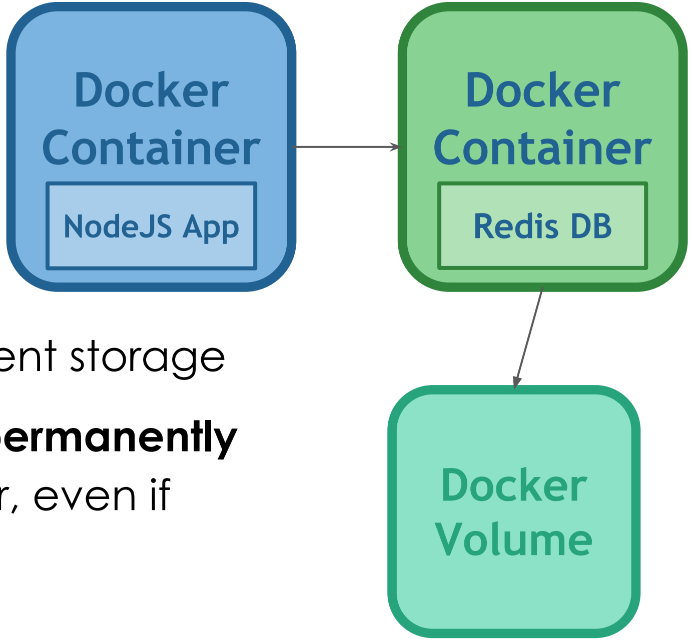
3rd request

. . .

Link to the code: https://github.com/alefais/virtualization-lab-unipi/tree/main/composed-docker-app

Hands-on with Docker: note on storage

- What happens if we run again our application?
  - Run **docker-compose down**
  - Run **docker-compose up**



- The visit counter has been resetted to **0**
  - The current solution does not provide permanent storage
  - Some applications could need to **store data permanently** (maintain data from one execution to another, even if containers are removed/deleted)
    - Solution: use **volumes**!

Docker volumes

Working with volumes: main concepts

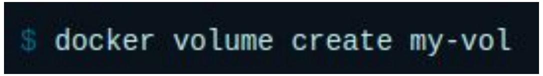
Preferred mechanism for persisting data generated by, and used by, Docker containers

- Completely managed by Docker
  - Manage them using Docker CLI commands or Docker API
- Portability and security
  - Work on both Linux and Windows containers
  - Can be stored on remote hosts / cloud providers
    - New drivers to encrypt their content
  - Can be shared safely among multiple containers
  - Can be pre-populated by a container

Create and manage volumes

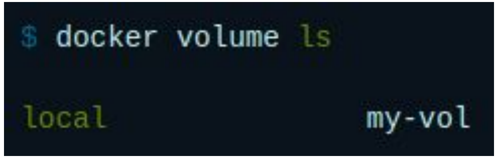
- Volume management is independent on the lifecycle of a given container using it
- A volume can be created and managed outside of the scope of any container

{ Create a volume }



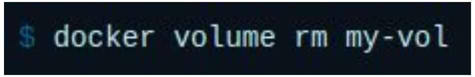
```
$ docker volume create my-vol
```

{ List volumes }



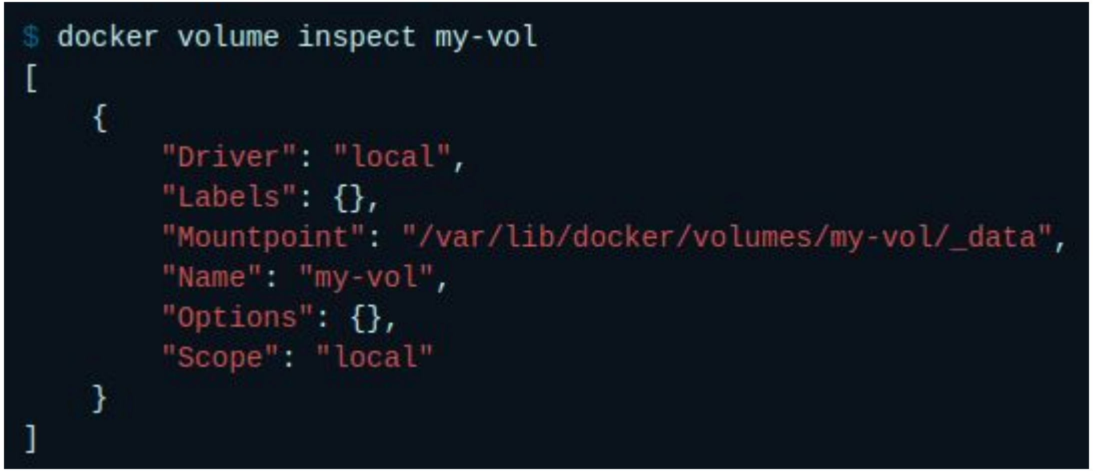
```
$ docker volume ls

local               my-vol
```

{ Remove a volume }



```
$ docker volume rm my-vol
```

{ Inspect a volume }

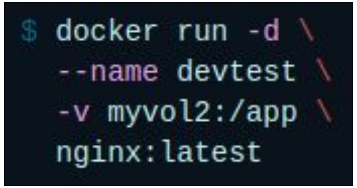


```
$ docker volume inspect my-vol
[
    {
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/my-vol/_data",
        "Name": "my-vol",
        "Options": {},
        "Scope": "local"
    }
]
```

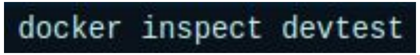
Start a container with a volume

- Start a container with a volume that doesn’t exist yet
- Docker creates the volume

Create a container named devtest from the image nginx:latest; attach it to a new volume named myvol2 that will be mounted into /app in the container



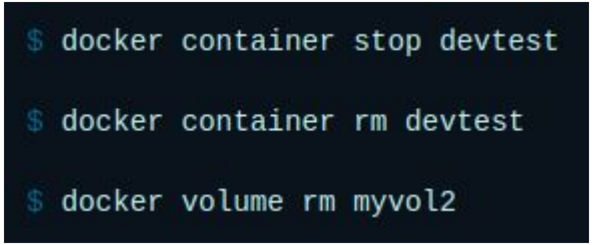
```
$ docker run -d \
  --name devtest \
  -v myvol2:/app \
  nginx:latest
```



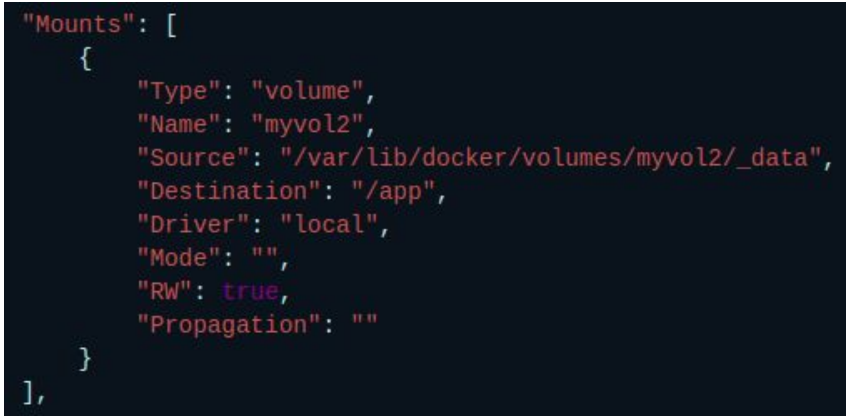
```
docker inspect devtest
```

Inspect the volume to check if it has been created and mounted correctly

Stop the container and remove it together with the volume



```
$ docker container stop devtest

$ docker container rm devtest

$ docker volume rm myvol2
```



```
"Mounts": [
    {
        "Type": "volume",
        "Name": "myvol2",
        "Source": "/var/lib/docker/volumes/myvol2/_data",
        "Destination": "/app",
        "Driver": "local",
        "Mode": "",
        "RW": true,
        "Propagation": ""
    }
],
```

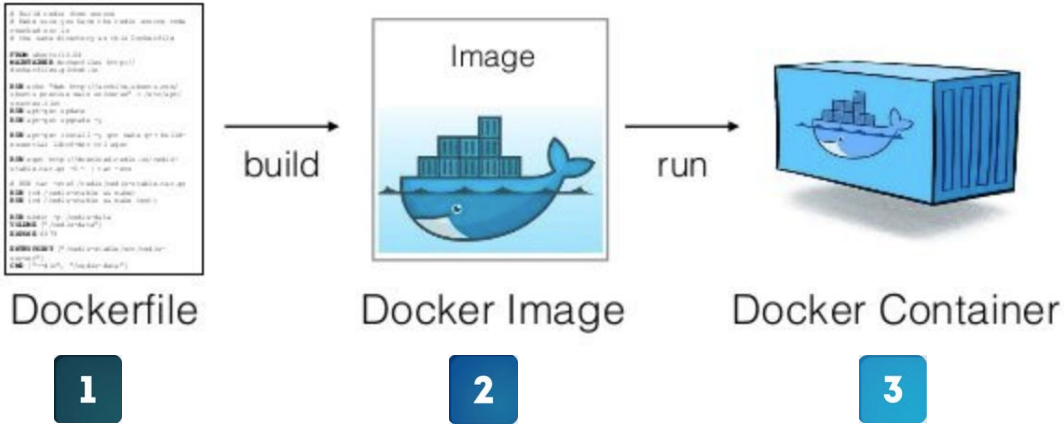
Third example (single host)

A composed Docker application with persistent storage

Introduction to docker-compose and volumes

Hands-on with Docker: complex example with volumes

- Let’s follow these steps to create each service in our **multi-component application**

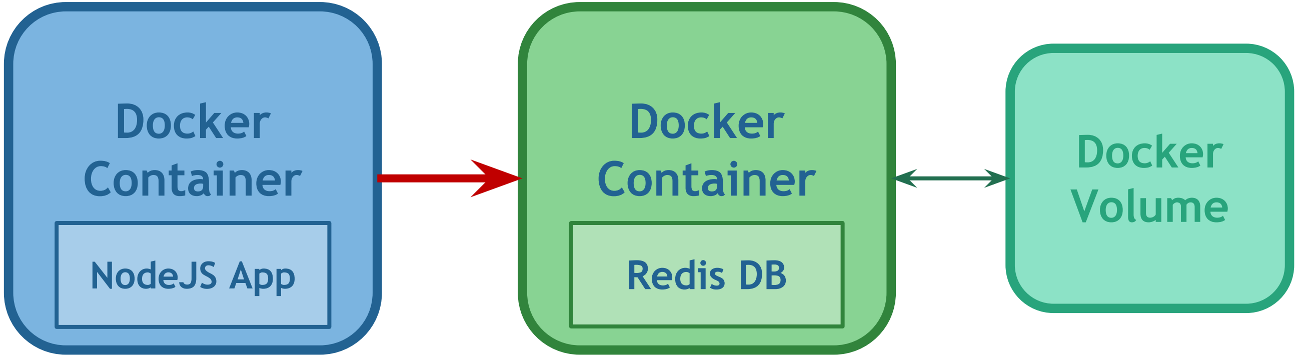


- Then create a volume, set up networking and compose the services together

Link to the code: https://github.com/alefais/virtualization-lab-unipi/tree/main/composed-docker-app-volume

Hands-on with Docker: complex example with volumes

- Application implemented as two interacting services (Docker Stack)
- Idea : keep track of the number of visits to the web application
  - each time someone visits the app, the visitor counter is incremented
  - the counter is backed-up in a volume (permanent storage)

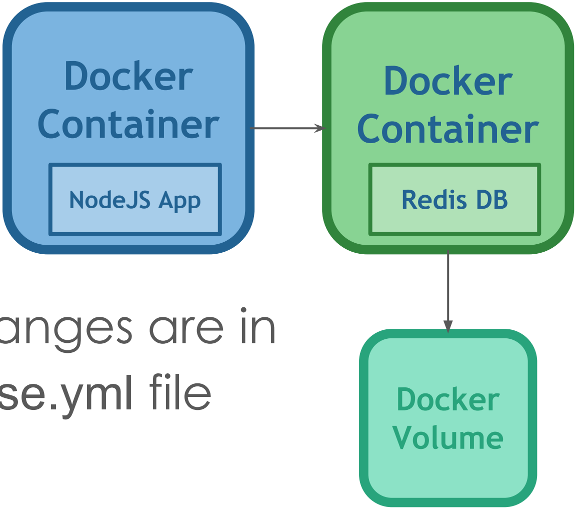


Link to the code: https://github.com/alefais/virtualization-lab-unipi/tree/main/composed-docker-app-volume

Use a volume with docker-compose

```
version: "3"
services:
  redis-db:
    image: redis
    ports:
      - "6379:6379"
    command: --appendonly yes
    volumes:
      - redis-db-data:/data
node-web-app:
   build: .
   ports:
     - "4001:8081"
```

Let’s add a volume to our application!



Most relevant changes are in the docker-compose.yml file

```
volumes:
   redis-db-data:
    external:
         name: composed-docker-app-volume_redis-db-data
```

Link to the code: https://github.com/alefais/virtualization-lab-unipi/tree/main/composed-docker-app-volume

Hands-on with Docker: run the application

From the root directory:

- Create the volume

```
docker volume create --name=composed-docker-app-volume_redis-db-data
```

- Start up the containers

```
docker-compose up
```

- Stop/start the containers

```
docker-compose stop
docker-compose start
```

- Stop the app and remove containers and networks

```
docker-compose down
```

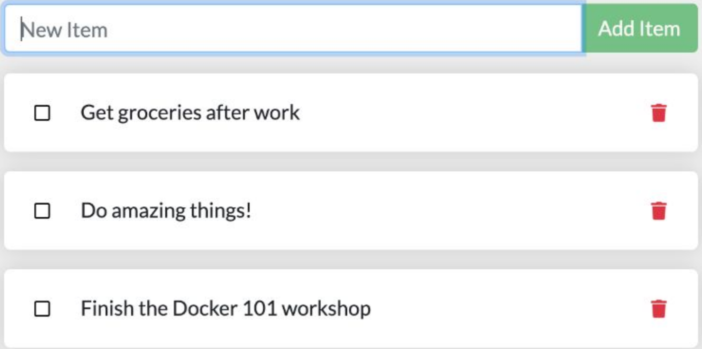
You can check your volume by running **docker volume ls**

You can now access the application at **http://localhost:4001**

The visits counter value is stored in the volume, and can be retrieved from it during successive runs of the app!

Link to the code: https://github.com/alefais/virtualization-lab-unipi/tree/main/composed-docker-app

Exercise (optional)



New Item

Add Item

□ Get groceries after work

□ Do amazing things!

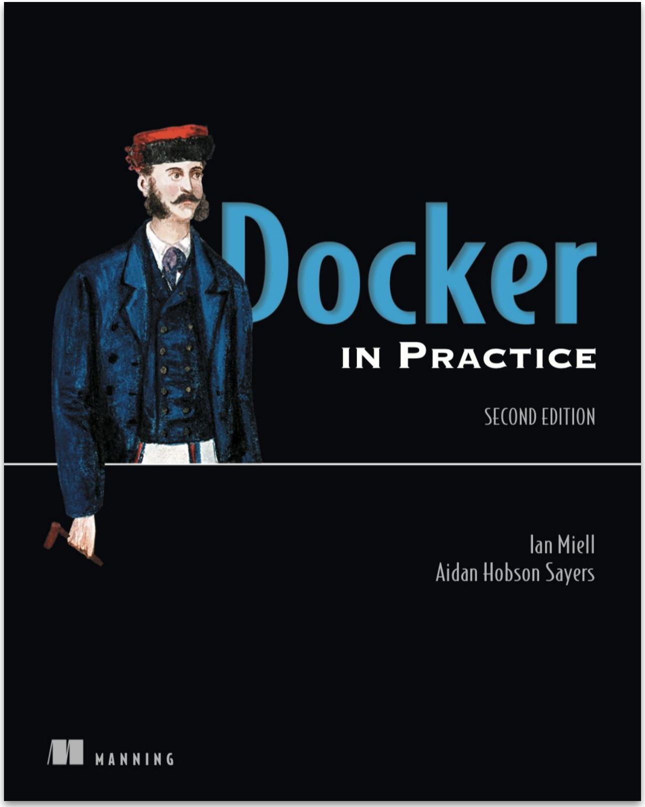
□ Finish the Docker 101 workshop

If you want to practice more, you can try to follow this tutorial to build and run a **Todo sample application**.

Here is the link to the tutorial: https://docs.microsoft.com/en-us/visualstudio/docker/tutorials/docker-tutorial

Useful references

Useful reference book



Docker
IN PRACTICE
SECOND EDITION

Ian Miell
Aidan Hobson Sayers

MANNING

Link

Kubernetes in practice

Brian Storti

Link

Useful references on Docker Compose and data management

- Get started with Docker Compose
  - https://docs.docker.com/compose/gettingstarted/
- A Docker Compose tutorial by examples
  - https://takacsmark.com/docker-compose-tutorial-beginners-by-example-basics/
- More on data management in Docker: volumes and other approaches
  - https://docs.docker.com/storage/
  - https://docs.docker.com/storage/volumes/
  - https://docs.docker.com/compose/compose-file/compose-file-v3/#volume-configuration-reference
  - https://docs.docker.com/compose
  - https://thenewstack.io/methods-dealing-container-storage/