DEPARTMENT OF INFORMATION ENGINEERING
UNIVERSITY OF PISA

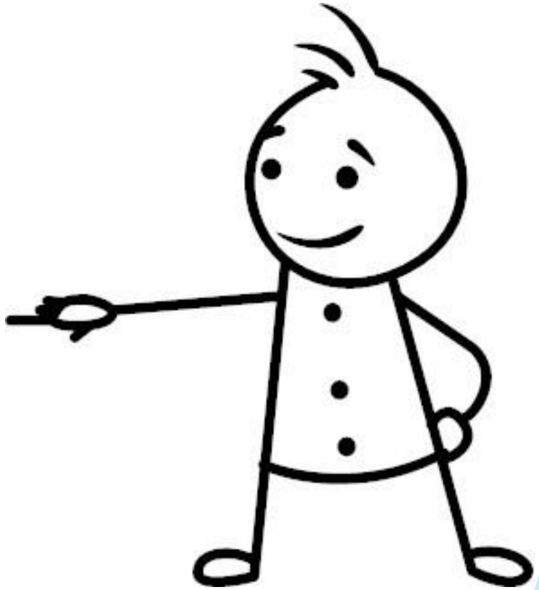
Virtualization (LAB)

Alessandra Fais – Ph.D. Student

alessandra.fais@phd.unipi.it

Where are we?

- Virtualization
- **Containerization and Docker**
- OpenStack



Outline of the lecture

1) Containers characteristics

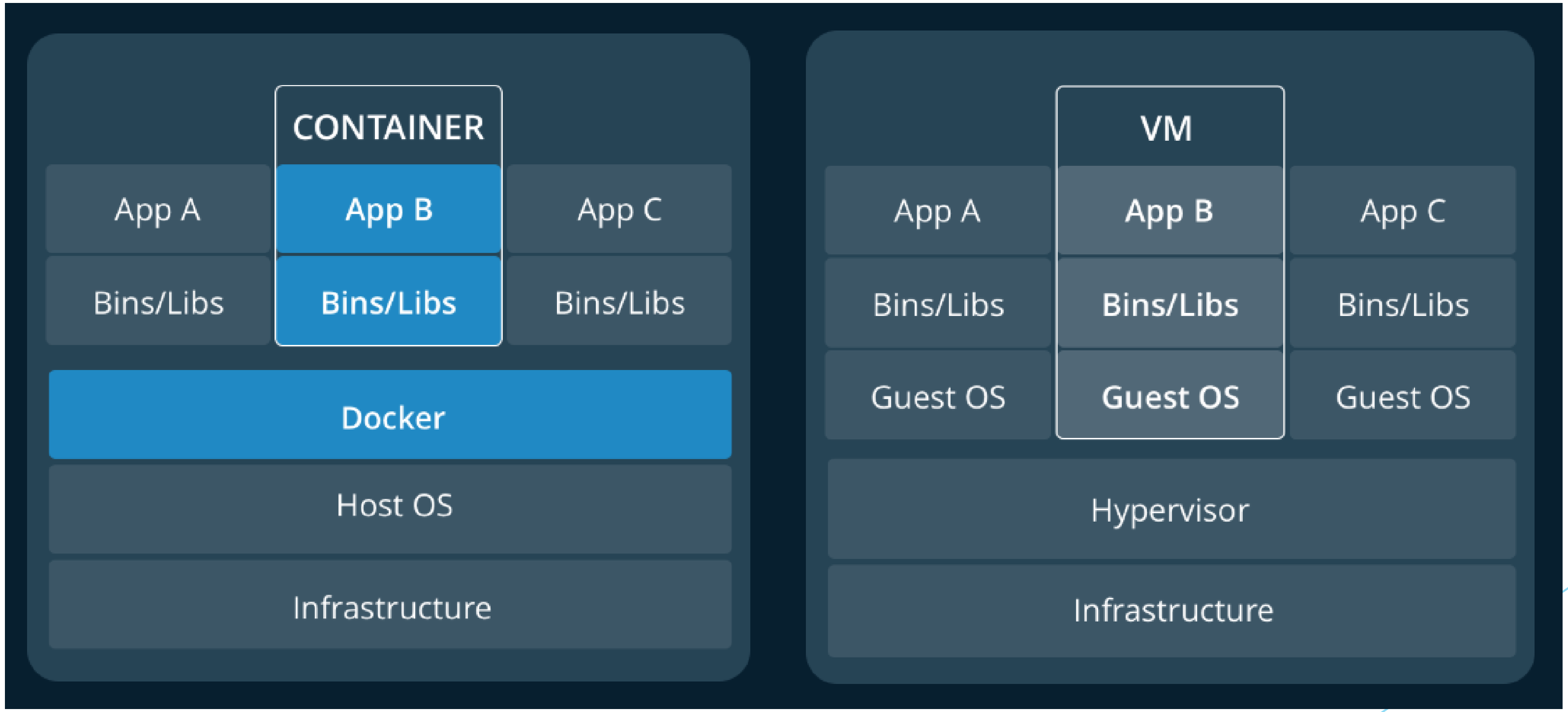
2) Docker
   - Objects
   - Architecture

3) Hands-on with Docker
   - Installation
   - Execution flow
   - Commands
   - Dockerfile
   - Docker Compose

Containers and Docker

Containers vs Virtual Machines

- A **VM hosts a whole Operating System** (*guest*), separated from the Host OS, over an **emulated hardware**
- A **container shares the OS kernel** with the host, avoiding hardware emulation (gain efficiency but loose isolation)

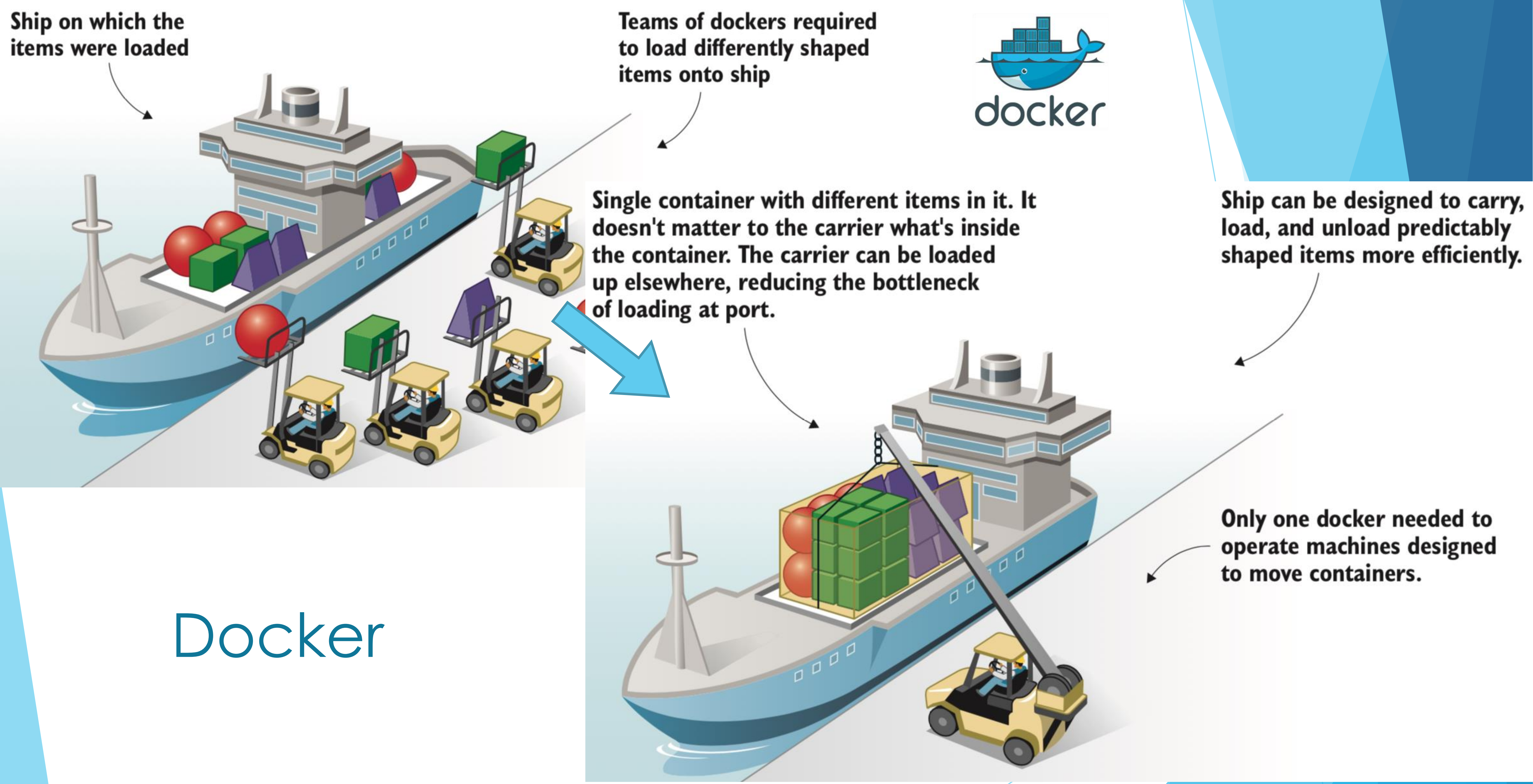


Containers: characteristics

- **Resources are shared** with the Host OS
  - Efficiency, overhead reduced
- **Portability**
  - **Build once, run anywhere!**

- Lightweight virtualization
  - Run dozens of instances at the same time (**high-density**)
- **Dependencies are embedded**
  - No need to configure and install

Docker



Ship on which the items were loaded

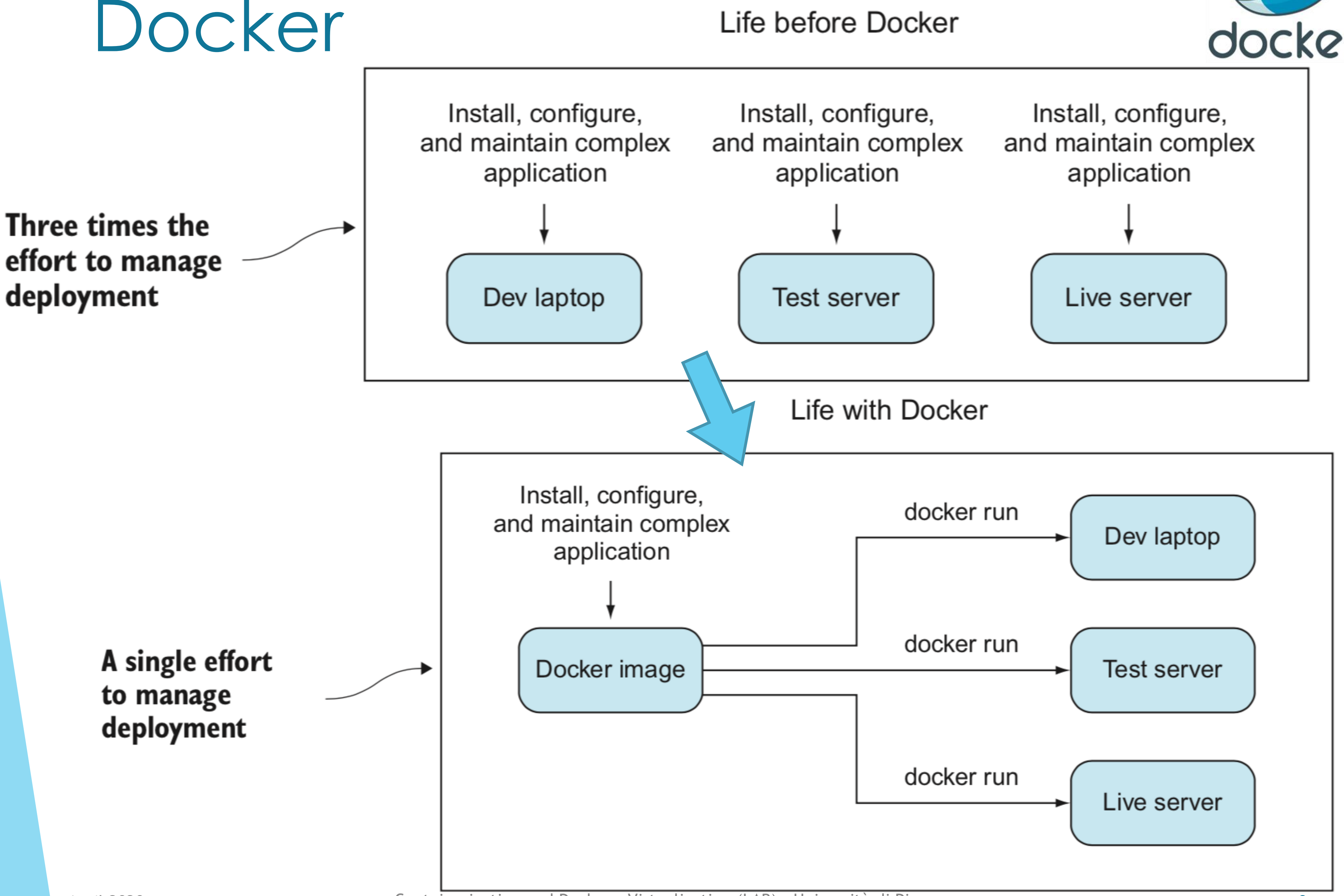
Teams of dockers required to load differently shaped items onto ship

Single container with different items in it. It doesn't matter to the carrier what's inside the container. The carrier can be loaded up elsewhere, reducing the bottleneck of loading at port.

Ship can be designed to carry, load, and unload predictably shaped items more efficiently.

Only one docker needed to operate machines designed to move containers.

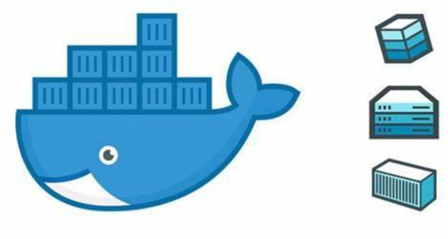
Docker



Life before Docker
Install, configure, and maintain complex application
Install, configure, and maintain complex application
Install, configure, and maintain complex application
Dev laptop
Test server
Live server
Three times the effort to manage deployment
Life with Docker
Install, configure, and maintain complex application
Docker image
docker run
docker run
docker run
Dev laptop
Test server
Live server
A single effort to manage deployment

Docker

- A definition:
  ‘‘Docker is an open source engine that automates the deployment of any application as a lightweight, portable, self-sufficient container that will run virtually anywhere’’

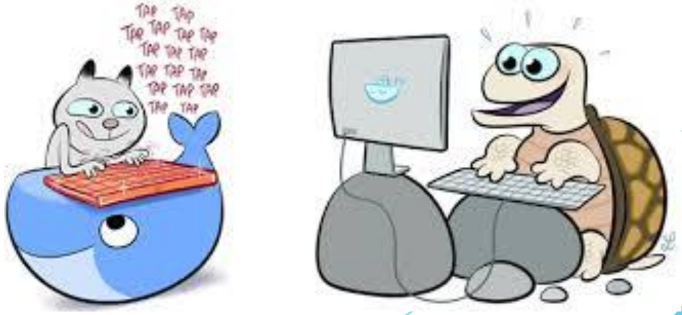


- What can you do with Docker?
  Docker allows to **create**, **manage** and **orchestrate** application containers
  - Each application component is packed in a separate container
  - Optimization of development, testing, delivery and deployment of applications

Docker

- Design oriented to **software development steps**
  - Local **development environment**
  - **Testing**
    - Containers isolate tests into their own environment

      → no need to clean up the environment after each test execution!
    - Parallelize tests across multiple machines
    - Create different system configurations to test against
  - **Delivery**
  - **Deployment**



First things first: Docker objects

- **Image**
  - **read-only template** with instructions for creating a Docker container
  - can be based on another image (extend the base image through a list of instructions defined in a *Dockerfile*)
  - example: build an image based on the **ubuntu** image which also installs our application and the configuration details required to run it
- **Container**
  - **runnable instance** of an image
  - defined by the image and the configuration options provided to it when created or started
  - unit for distributing and testing our application, along with its dependencies

First things first: Docker objects

- **Network**
  - **bridged network** : new containers on a single host are connected by default to it and can refer each other by IP address
  - **host network** : containers connected to this network share the host machine’s network (remove network isolation between containers and host)
  - **none network** : the container is not connected to any network
  - **overlay network** : allow connectivity among containers on different hosts
- **Volume**
  - **persistent storage** for containers
  - can be associated to one or more containers
  - can be shared among several containers
  - its lifespan is completely independent of the containers that use it

First things first: Docker objects

- **Service**
  - set of containers which are replicas of the same image
    - together they provide a load balanced service
    - scale up or down depending on the input load
  - **deploy containers in production**

- **Stack**
  - set of **interdependent services that interact to implement an application**
  - example: a voting application could be composed by (i) a service for the web interface which allows users to vote; (ii) a service to collect the votes of the users and store them in a Docker volume; (iii) a service for the web interface which shows the results of the voting in real time

The Docker Ecosystem

- Docker platform
  - **Docker Engine**
    - Create and run containers
  - **Docker Hub**
    - Cloud service (database) for storing and distributing images

- Cluster mode

Not covered in this course

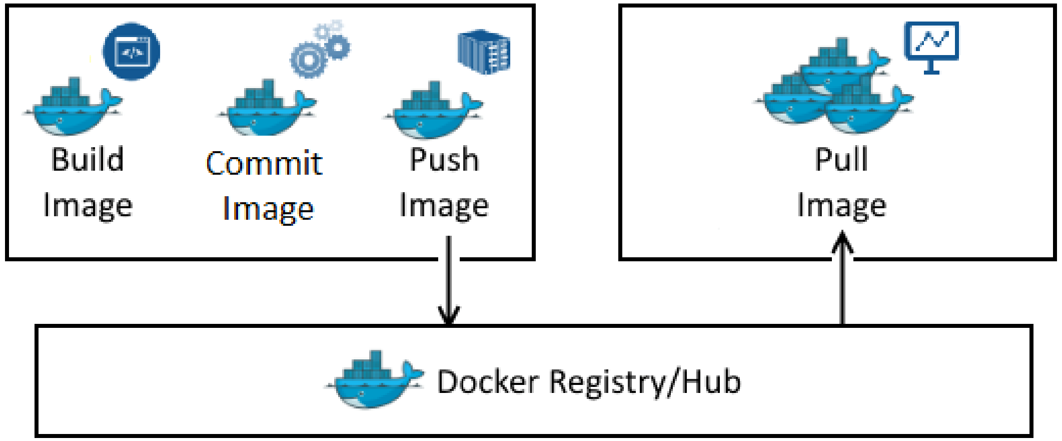
  - **Docker Swarm**
    - Deploy containers of a **Docker stack** on all the nodes of a cluster (swarm) instead of the single host
    - Swarm **manager** node and a set of swarm **workers** nodes
  - **Kubernetes**

Docker Engine

Docker guide sections, here:
https://docs.docker.com/install/
and here:
https://docs.docker.com/engine/docker-overview/

- Build and containerize applications!
- **client-server architecture**
  - Server runs the daemon process **dockerd**
    - **dockerd** creates and manages images, containers, networks, and volumes
  - API exposed to programs to instruct the **dockerd**
  - Command line interface (CLI) client **docker**
    - Uses Docker APIs to control or interact with the Docker daemon through scripting or direct CLI commands

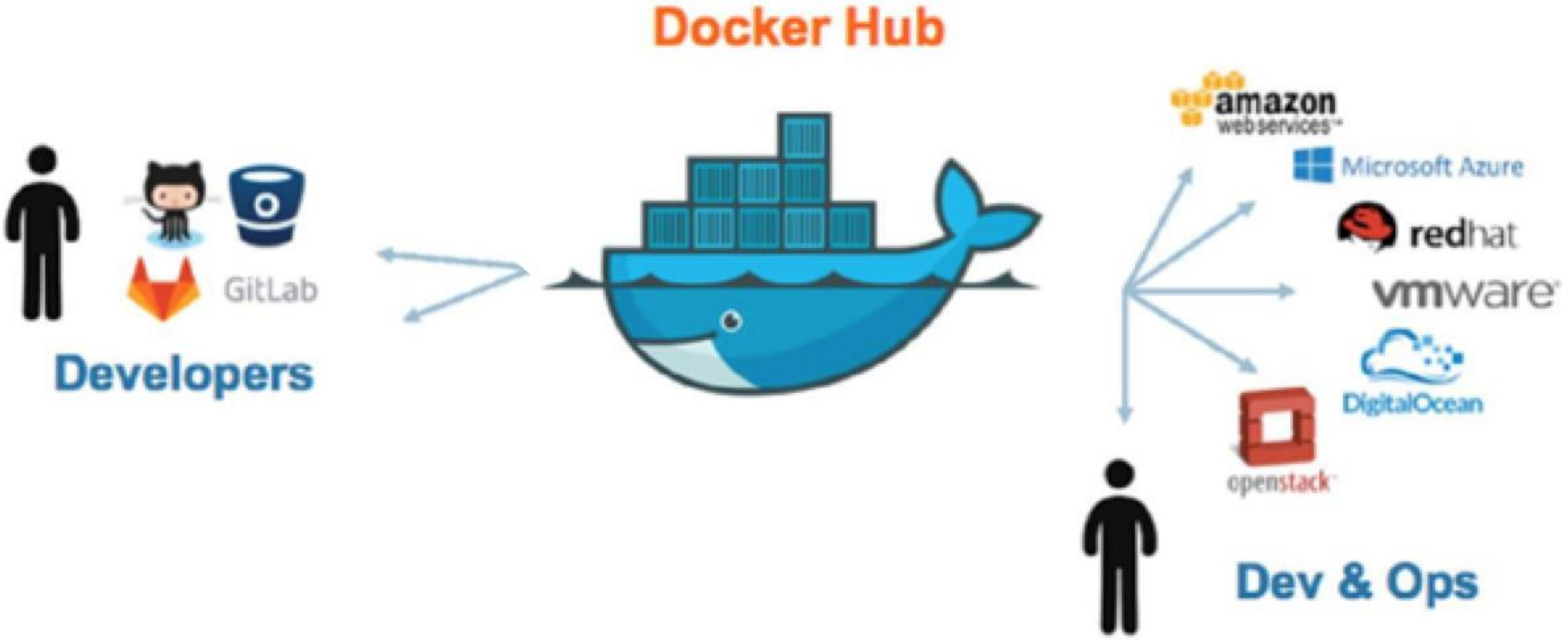
Docker Hub



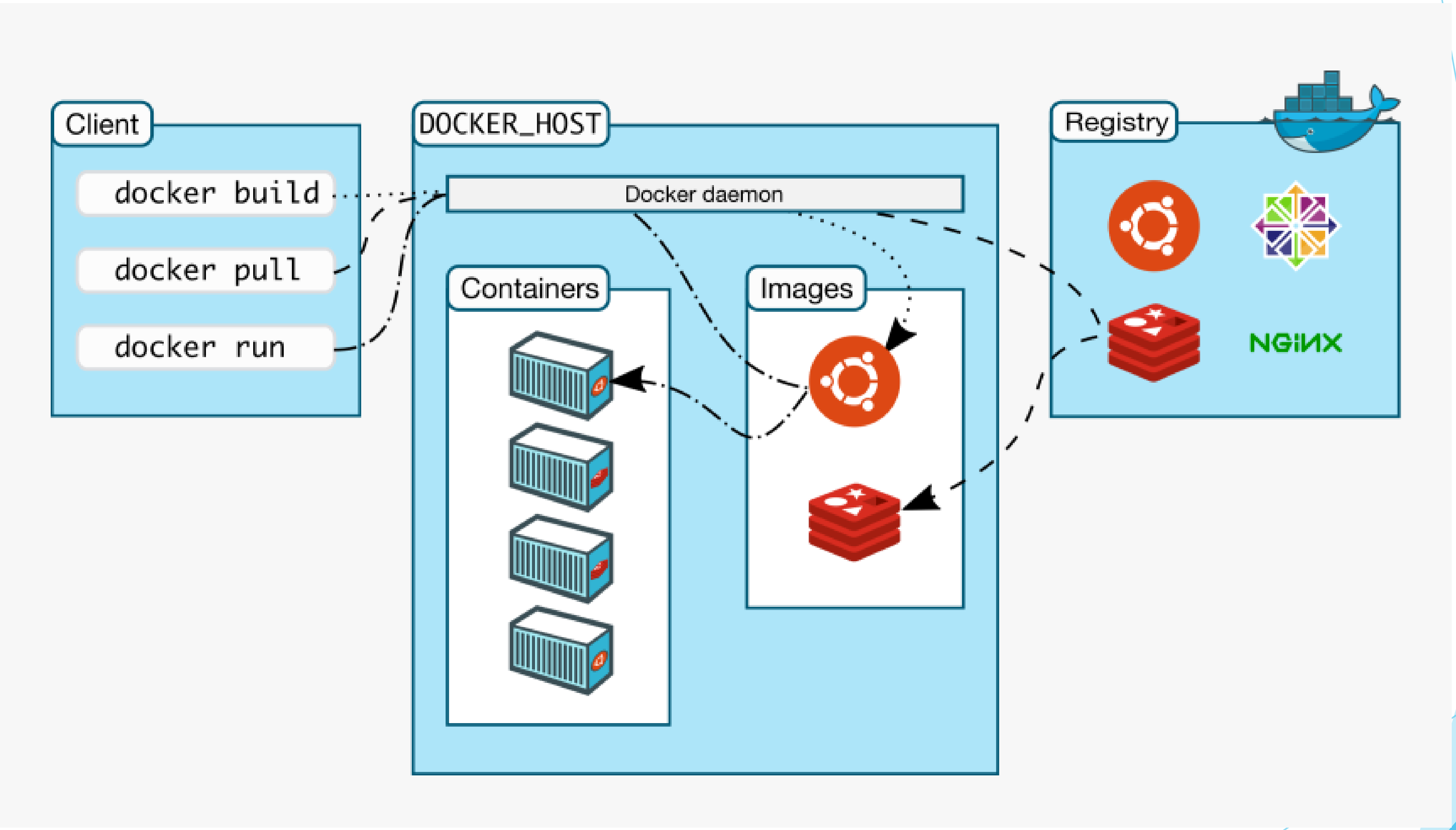
- Service provided by Docker for finding and sharing container images
- **Repositories** allow sharing container images with the Docker community
- Container images can be *pushed* to a repository or *pulled* from it
  - Official images (provided by Docker)
    - Clear documentation, best practices, design for most common use cases, scanned for security vulnerabilities
  - Publisher images (provided by external vendors)

Docker guide section, here: https://docs.docker.com/docker-hub/

Docker Hub


Docker Hub
GitLab
Developers
amazon web services
Microsoft Azure
redhat
vmware
DigitalOcean
openstack
Dev & Ops

Putting all together: Docker Architecture



Hands-on with Docker

Hands-on with Docker

Installation

Hands-on with Docker: **Installation**

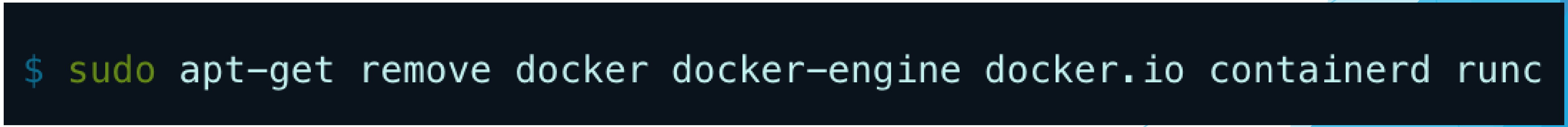
- Let's get started with **Docker Engine - Community for Ubuntu**!

- What you need:
  - The 64-bit version of one of the Ubuntu versions among
    - Eoan 19.10
    - Bionic 18.04 (LTS)
    - Xenial 16.04 (LTS)

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

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

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

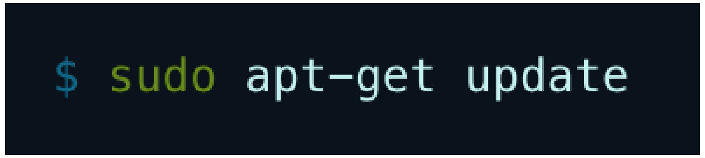


```
$ sudo apt-get remove docker docker-engine docker.io containerd runc
```

Hands-on with Docker: **Installation**

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

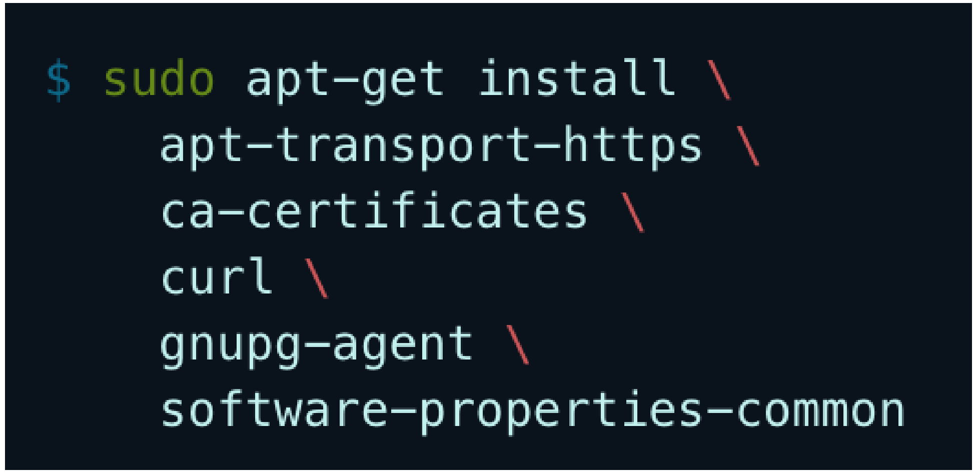
1



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

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

2

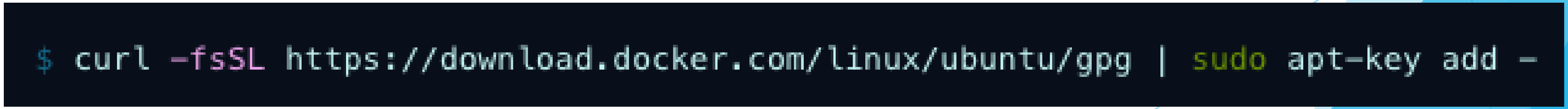


```
$ sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg-agent \
    software-properties-common
```

{ **Install packages to allow** apt **to use a repository over HTTP** }

{ **Add Docker's official GPG key** }

3

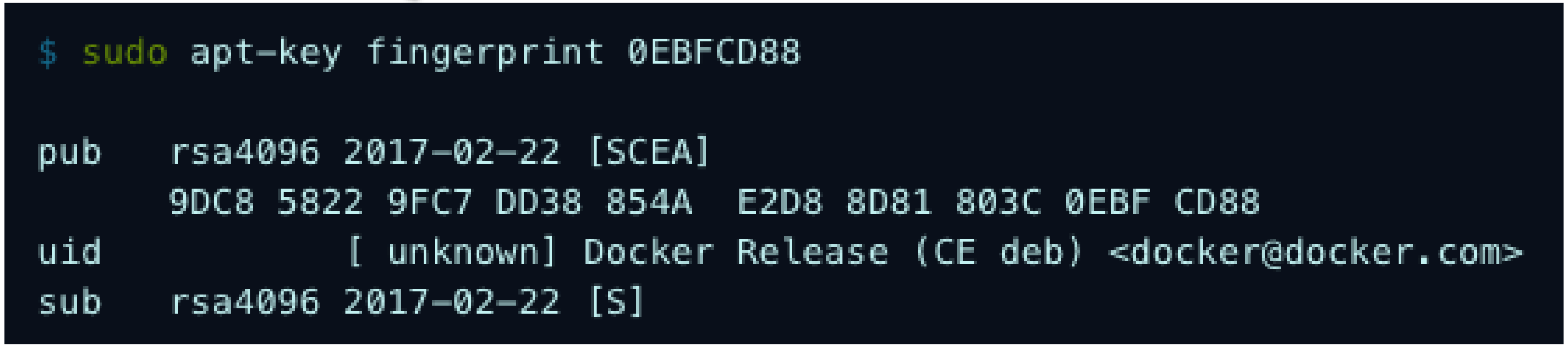


```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

Hands-on with Docker: **Installation**

4

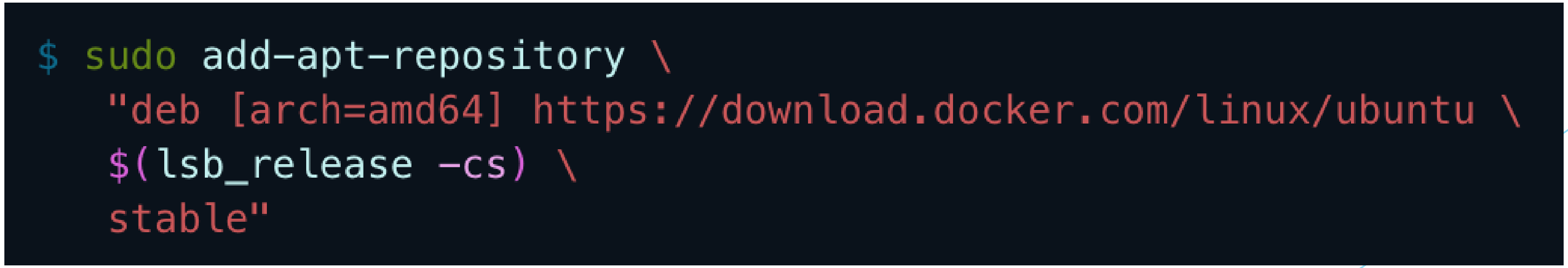
Verify that you now have the key with the fingerprint
9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88
by searching for the last 8 characters of the fingerprint



```
$ sudo apt-key fingerprint 0EBFCD88

pub   rsa4096 2017-02-22 [SCEA]
      9DC8 5822 9FC7 DD38 854A  E2D8 8D81 803C 0EBF CD88
uid           [ unknown] Docker Release (CE deb) <docker@docker.com>
sub   rsa4096 2017-02-22 [S]
```

Set up the ***stable*** **repository and return the name of your** Ubuntu **distribution.**

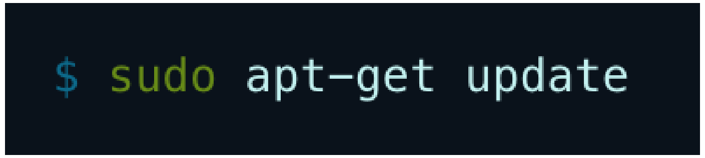
5



```
$ sudo add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
   $(lsb_release -cs) \
   stable"
```

Hands-on with Docker: **Installation**

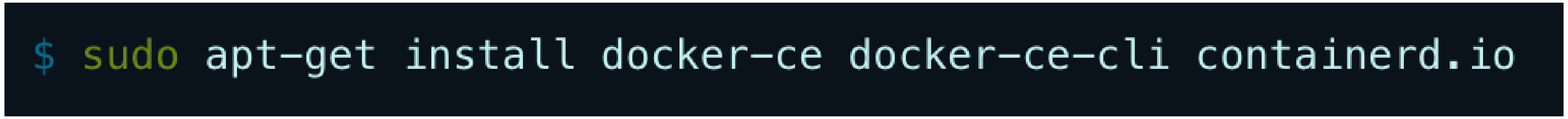
- **Third step:** install and update Docker from the repository



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

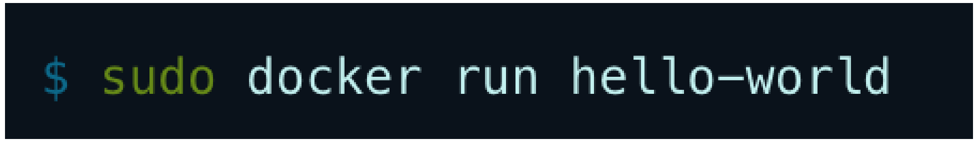
1 { **Update the** apt **package index** }

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



```
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

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

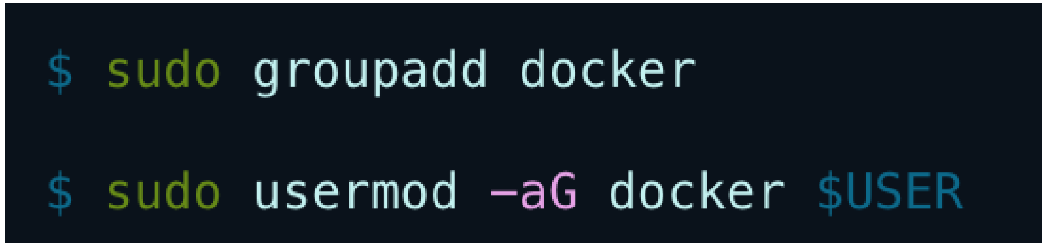


```
$ sudo docker run hello-world
```

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.**

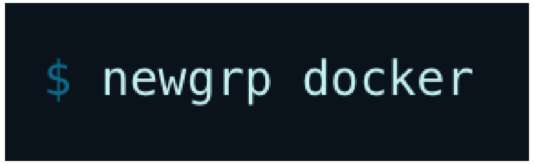
Hands-on with Docker: **use it as non-root user**

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



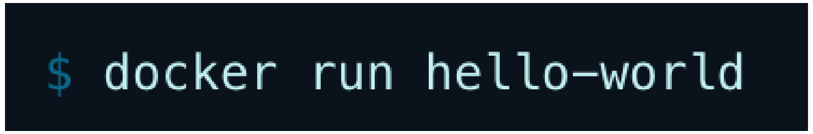
```
$ sudo groupadd docker

$ sudo usermod -aG docker $USER
```

- Activate the changes to groups



```
$ newgrp docker
```

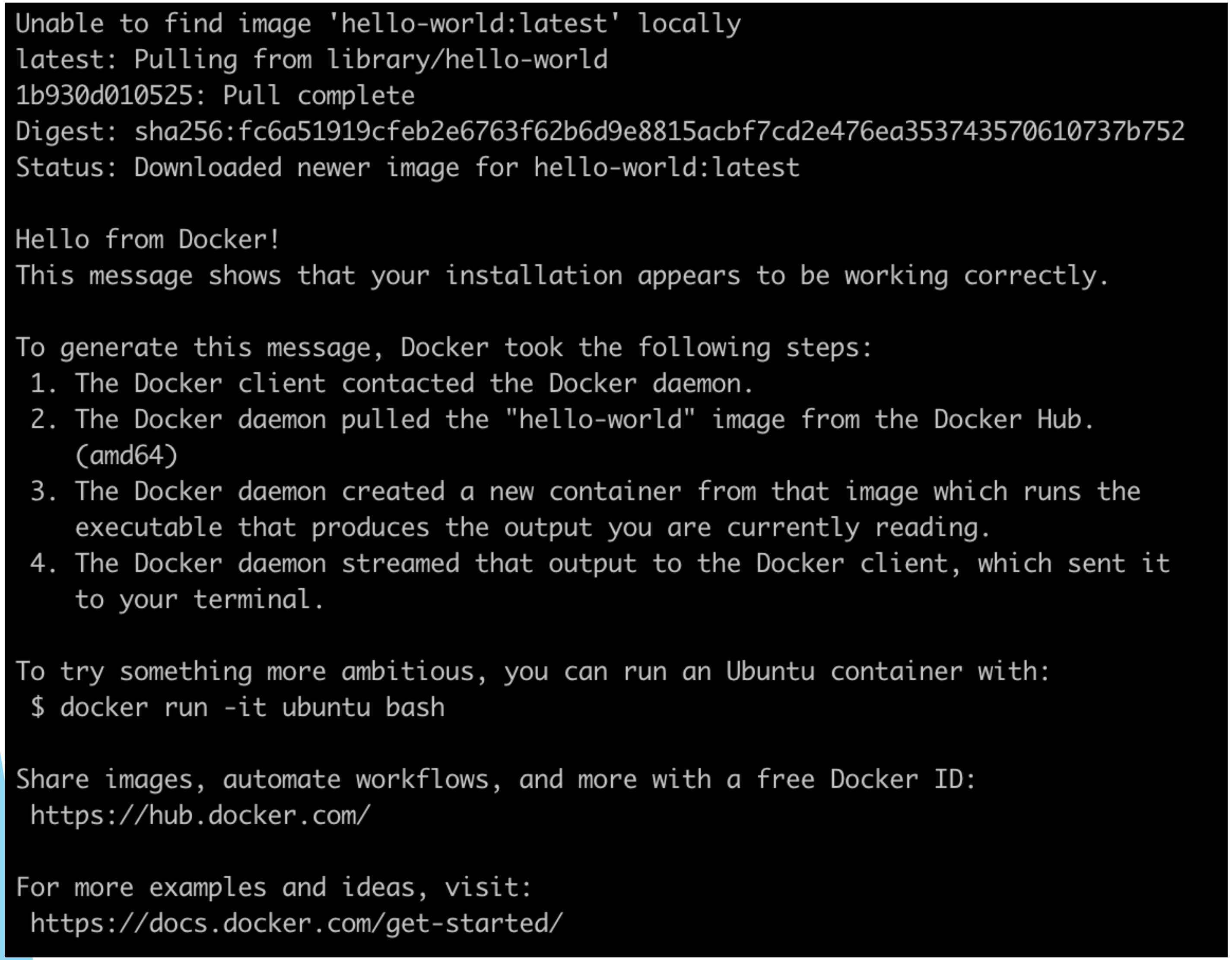
- Verify that you can run **docker** command without **sudo**



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

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

Hands-on with Docker: **our first running image!**



```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest: sha256:fc6a51919cfeb2e6763f62b6d9e8815acbf7cd2e476ea353743570610737b752
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/
```

{ **This is the output produced by**
docker run hello-world }

Hands-on with Docker

Execution Flow

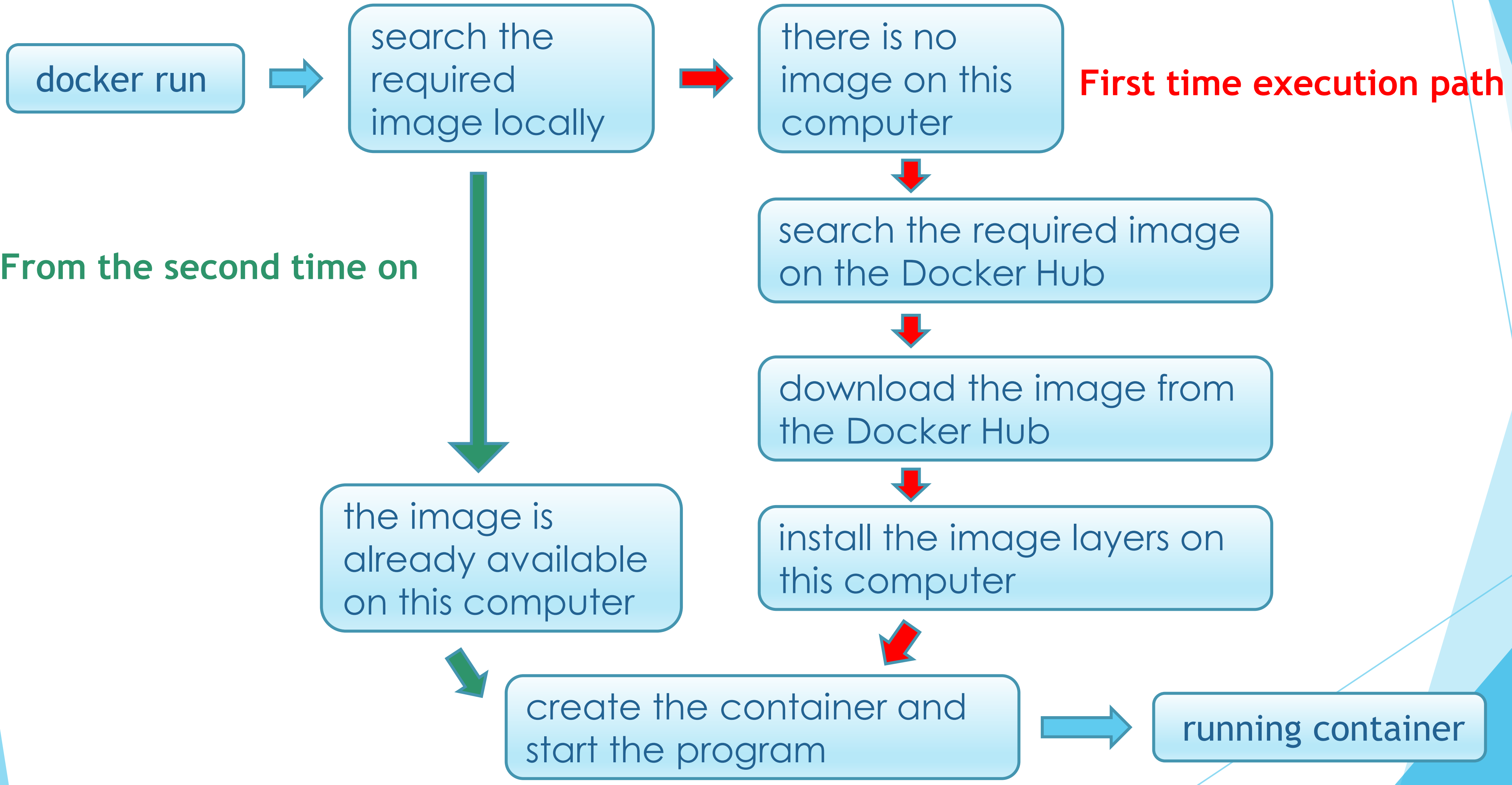
Hands-on with Docker: **the execution flow**

- **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

Hands-on with Docker: **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

Hands-on with Docker

Commands

Hands-on with Docker: docker **commands**

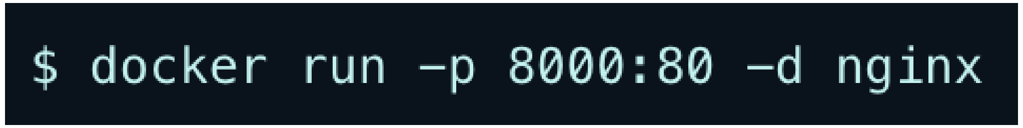
- ps | container ls : list the running containers in the system
  - docker container ls –a (**--all** flag lists all the containers, even if they are not running)
- images | image ls : list the images in the system
- image prune : remove unused images in the system
- run : create a running container 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

Hands-on with Docker: run **cmd 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

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

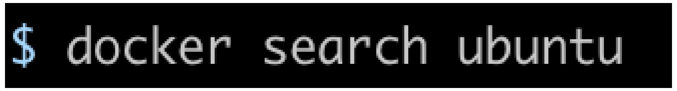
- Example:



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

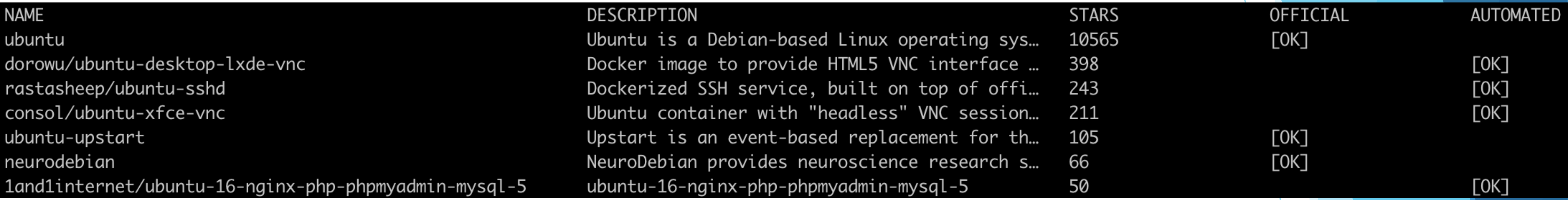
Whatever is running on port 80 in the nginx **container is available on port 8000 of** localhost

Hands-on with Docker: search **cmd**



```
$ docker search ubuntu
```

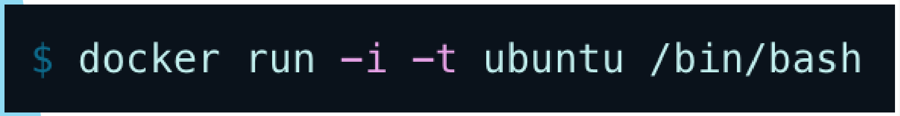
Search the Docker Hub registry for a ubuntu **image** 1



```
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]
```

Results can come from the top-level namespace for official image **or from the** public repository of a user 2

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 assignet 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**

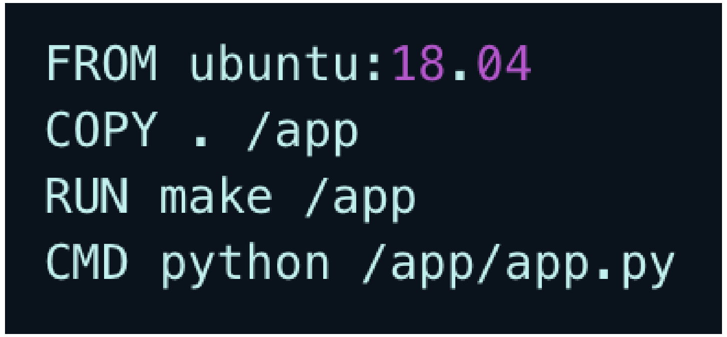
Hands-on with Docker

Dockerfile

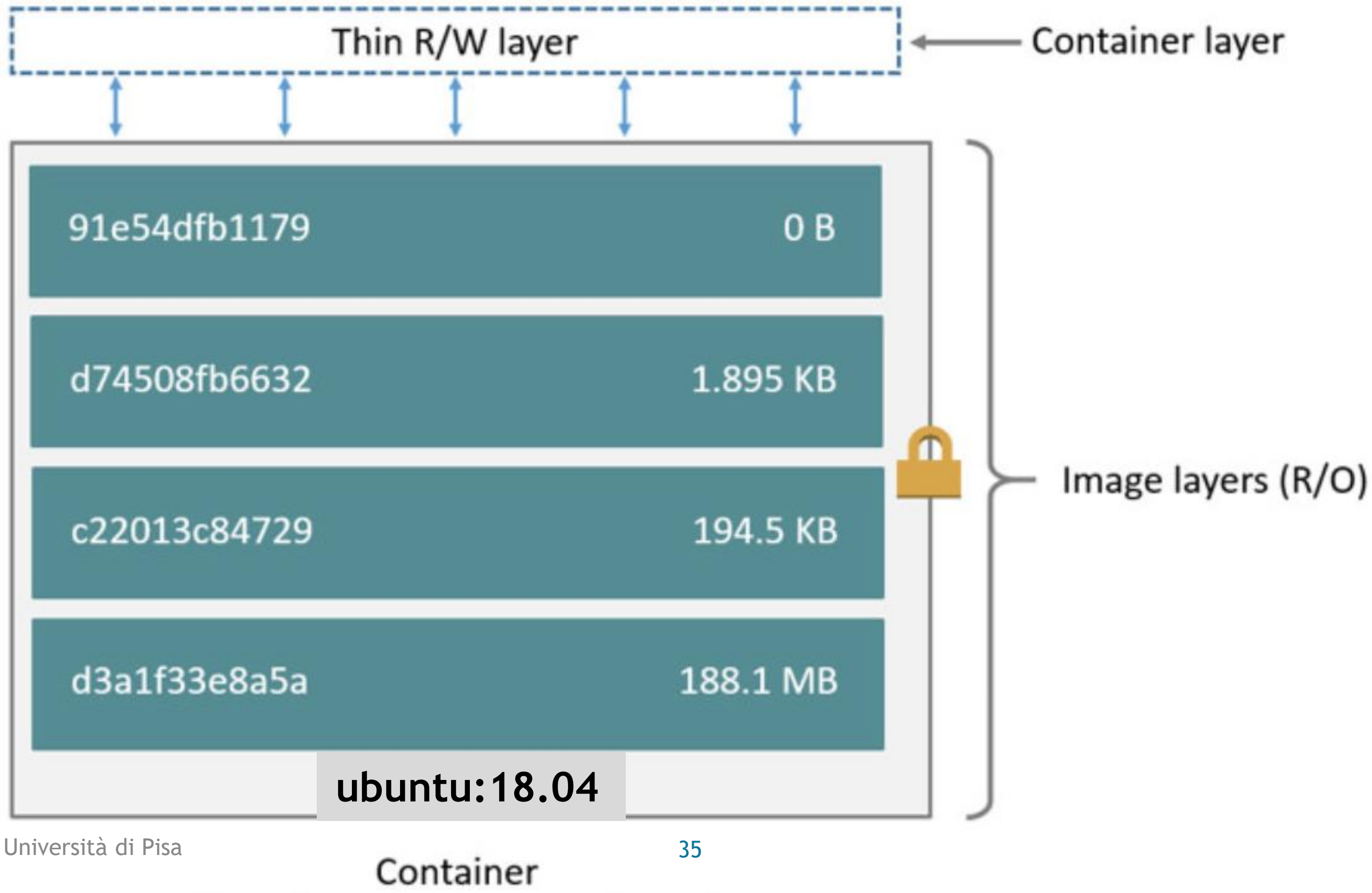
Dockerfile

- 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
```

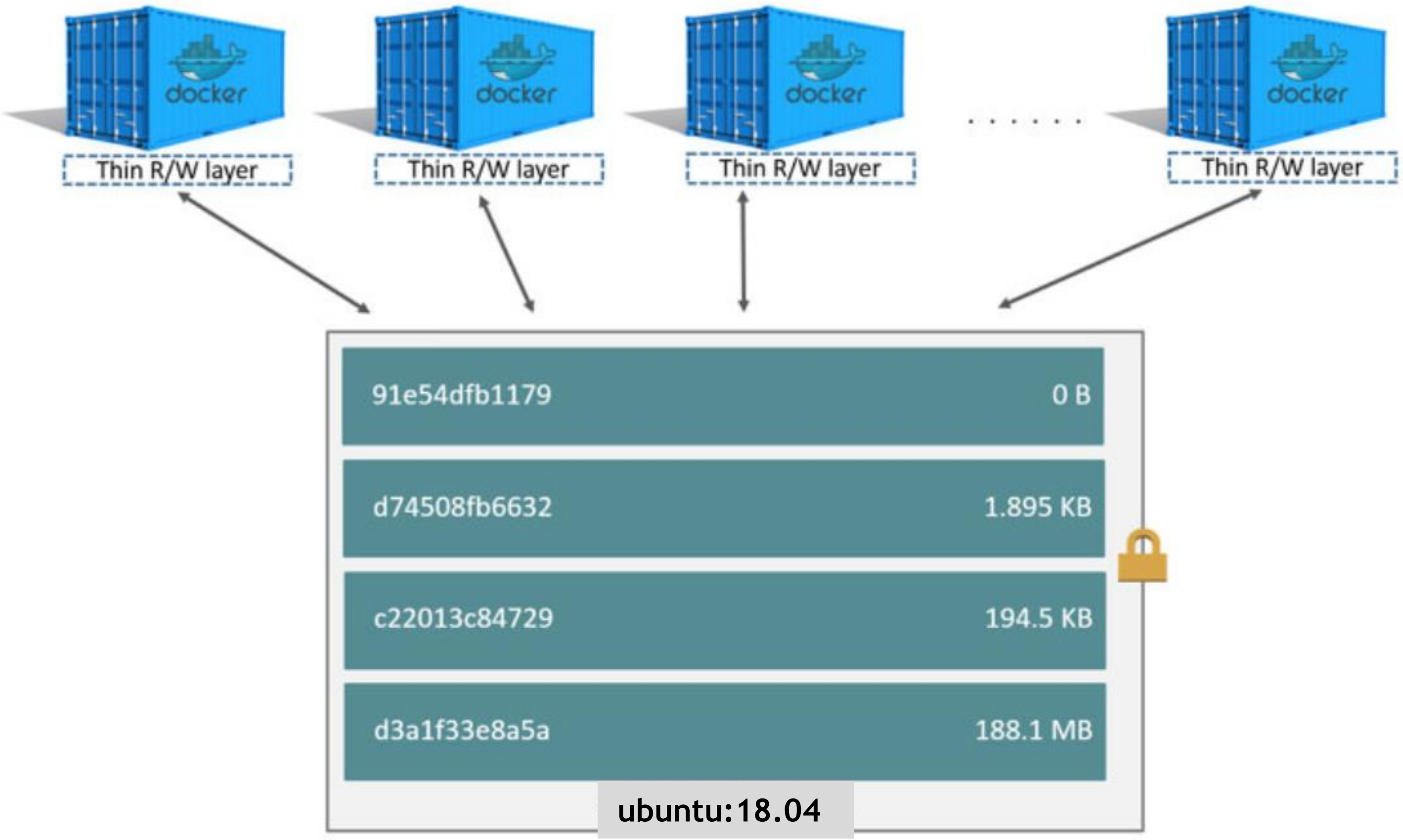


Dockerfile

- 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

Hands-on with Docker: **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)

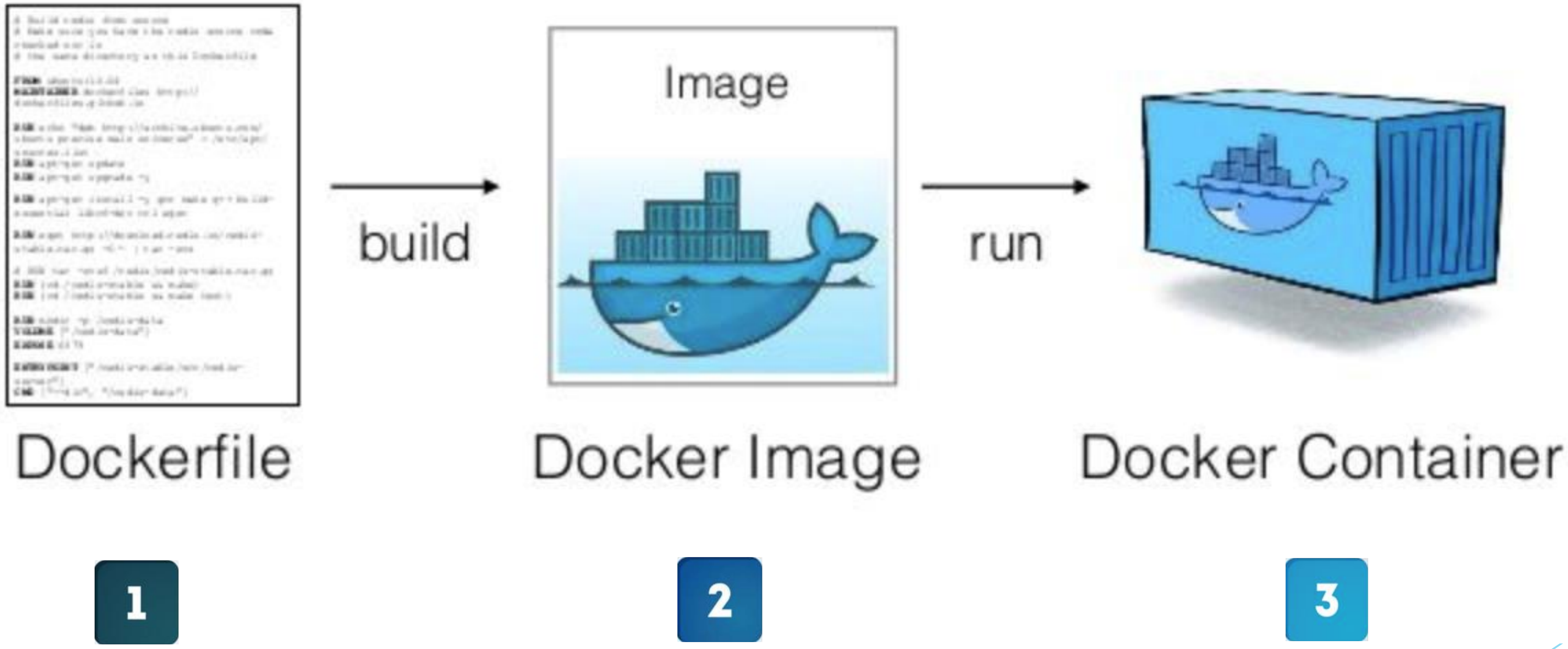
Hands-on with Docker

First Example:

a simple Docker application

Hands-on with Docker: simple application

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



Hands-on with Docker: Dockerfile

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

# Set the author of the new image
MAINTAINER Alessandra Fais

# 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 ./

# Give executable permissions
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

Create a new layer on top of the current image

Start the container

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**

Docker Container
Bash script

- Project structure
  - Root directory: **simple-docker-app**
  - .

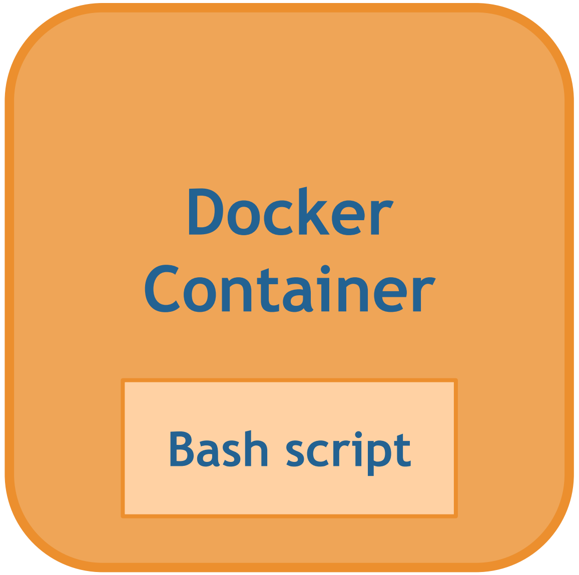
```
|__ entrypoint.sh
|__ Dockerfile
```

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



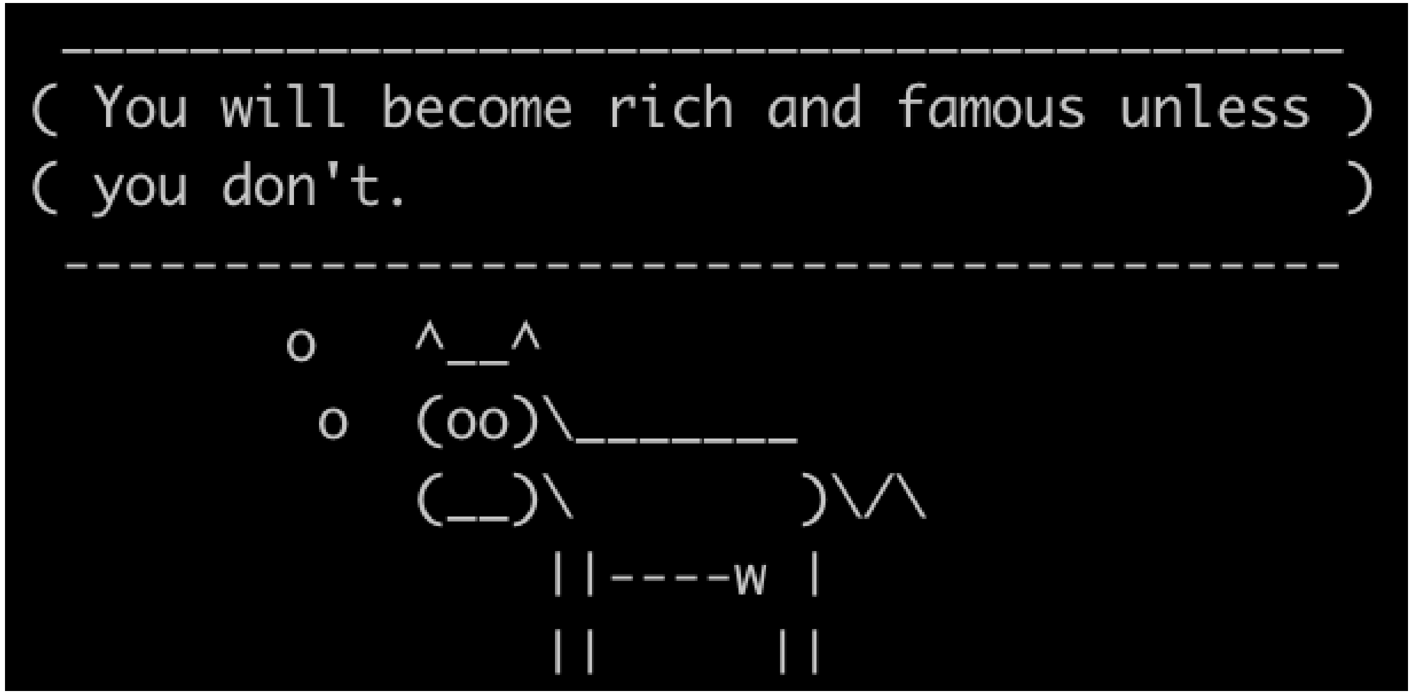
```
docker build -t fais/cowsay-app .

docker run fais/cowsay-app
```

Build context for the image

Tag to identify the image

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

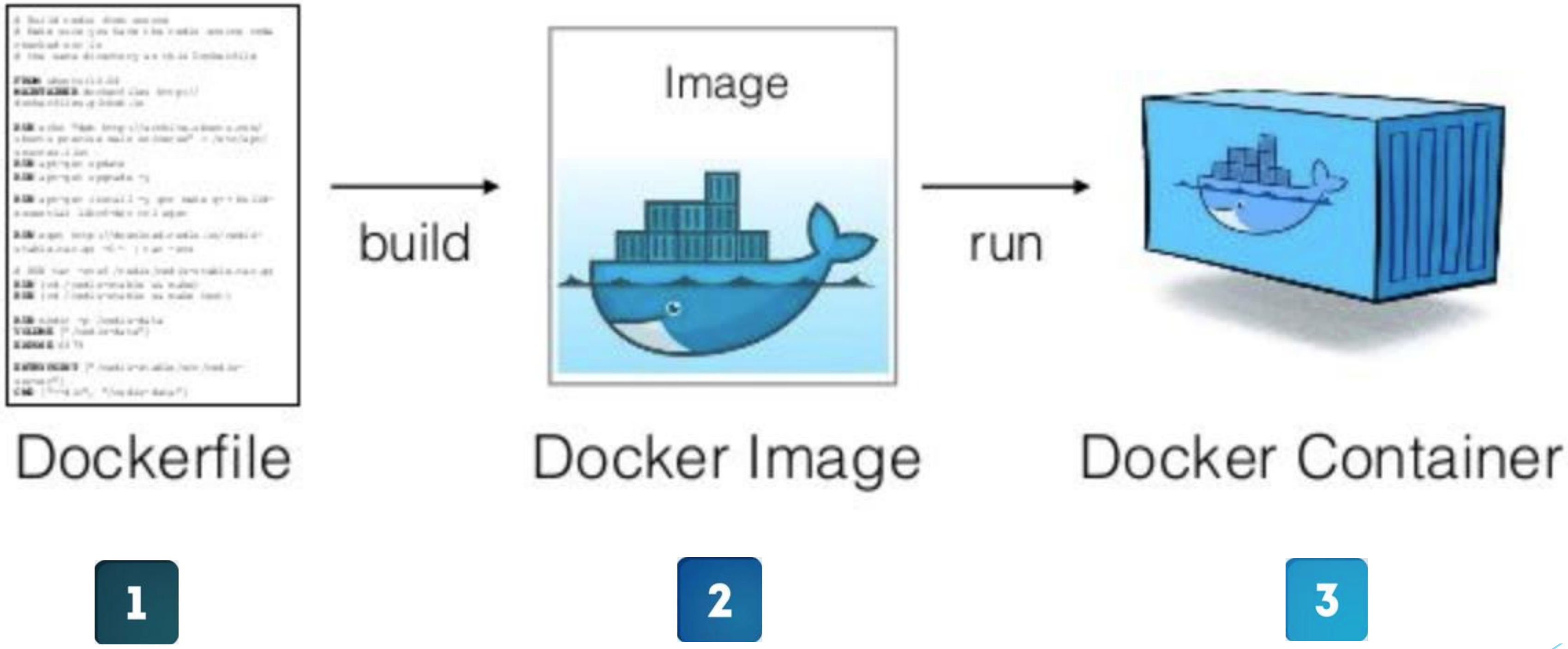
Hands-on with Docker

Second Example:
a composed Docker application

- Introduction to docker-compose

Hands-on with Docker: complex example

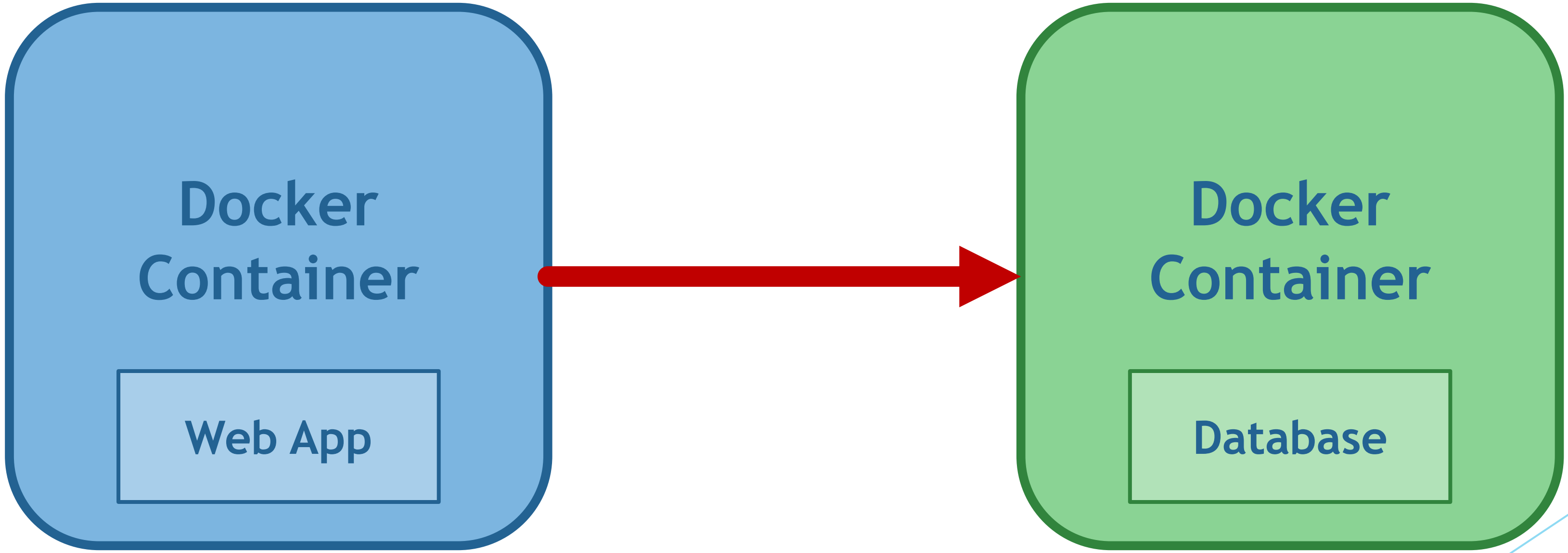
- Let’s follow these steps to create each service in our **multi-component application**



- Then set up networking and compose the services together

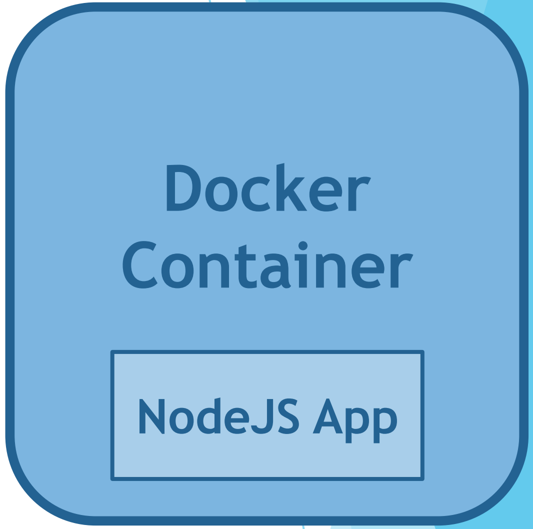
Hands-on with Docker: complex example

- 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



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**



Hands-on with Docker: index.js

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

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

dbClient.set('visits', 0);

app.get('/', (req, res) => {
    dbClient.get('visits', (err, visits) => {
    var currentVisits = parseInt(visits);
    res.send('Number of visits is: ' + (currentVisits + 1))
        dbClient.set('visits', (currentVisits + 1))
    })
})

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

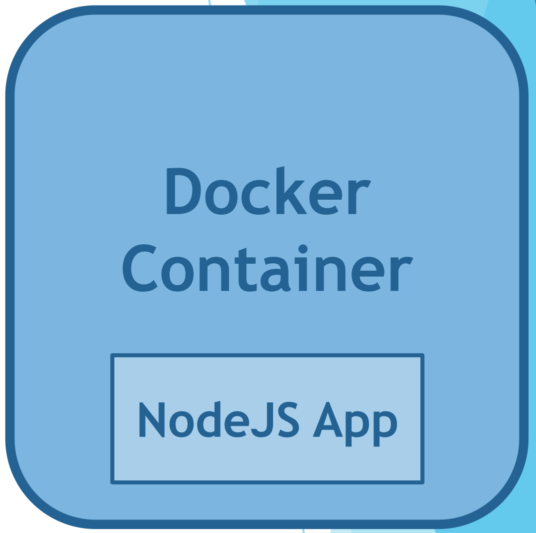
Use the Express framework

Docker Compose service name

Set inital visits

Define the root endpoint

Specify the listening port to 8081

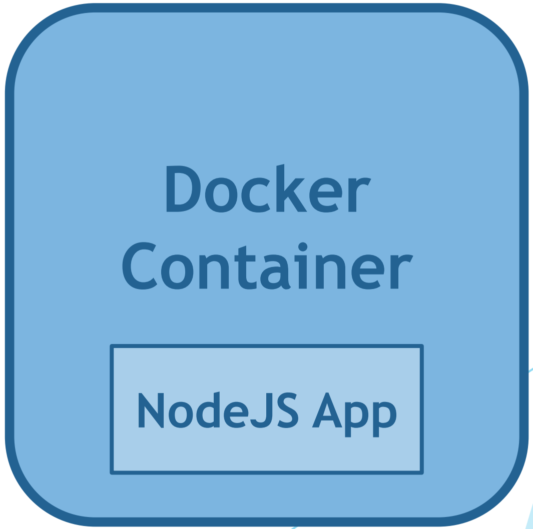


Hands-on with Docker: package.json

```
{
    "dependencies": {
        "express": "*",
        "redis": "2.8.0"
    },
    "scripts": {
        "start": "node index.js"
    }
}
```

Express framework
https://expressjs.com/

Redis v2.8.0
https://hub.docker.com/_/redis/



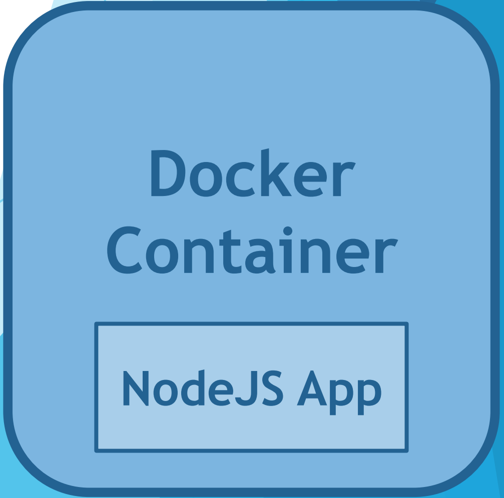
Hands-on with Docker: **Dockerfile**

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

# Specify a working directory
WORKDIR /usr/app

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

# Install dependencies

RUN npm set progress=false
RUN npm config set registry https://registry.npmjs.org/
RUN npm install

# Copy remaining files
COPY ./ ./

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

Lightweight Node Docker image having node and npm already installed

Copy project artifacts to the image

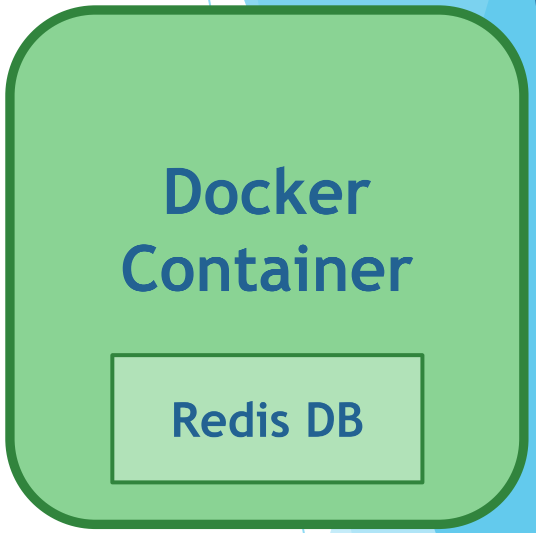
Start the container



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
    ```



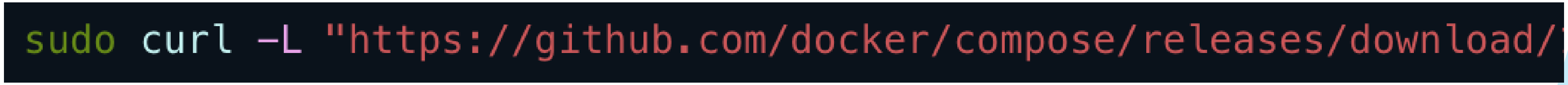
Hands-on with Docker: compose the app

- Connect together the two Docker containers

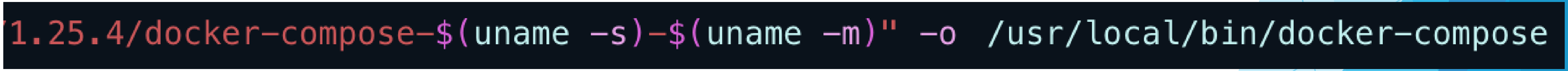
We will use docker-compose instead of the approach of the previous simple example

- Install docker-compose

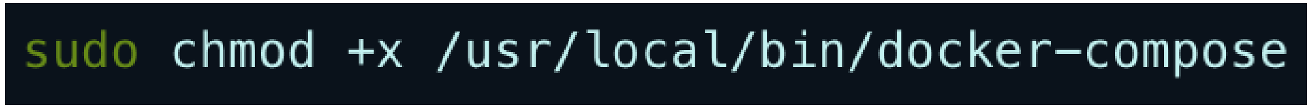
Download the current stable release of docker-compose



```
sudo curl -L "https://github.com/docker/compose/releases/download/
```



```
1.25.4/docker-compose-$(uname -s)-$(uname -m)" -o  /usr/local/bin/docker-compose
```

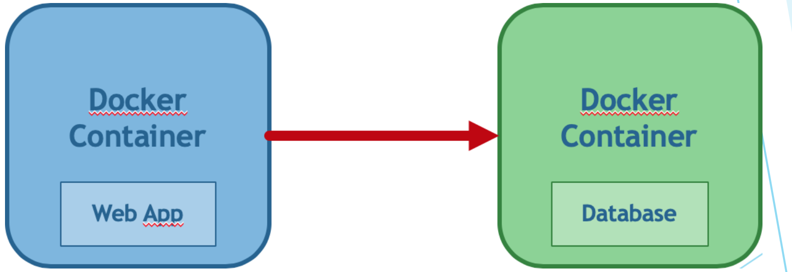


```
sudo chmod +x /usr/local/bin/docker-compose
```

Apply executable permissions to the binary

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
```

Hands-on with Docker: **docker-compose.yml**

```yaml
version: "2.4"
services:
  redis-server:
    image: redis
  node-app:
    build: .
    ports:
      - "4001:8081"
```

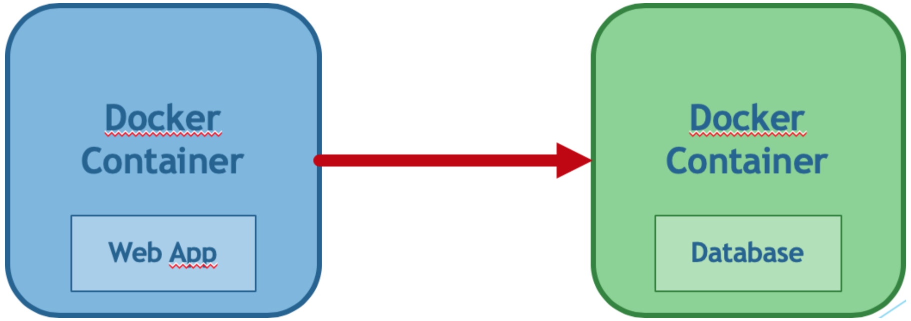
Docker Compose version

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

Service based on the **redis:latest** image from Docker Hub

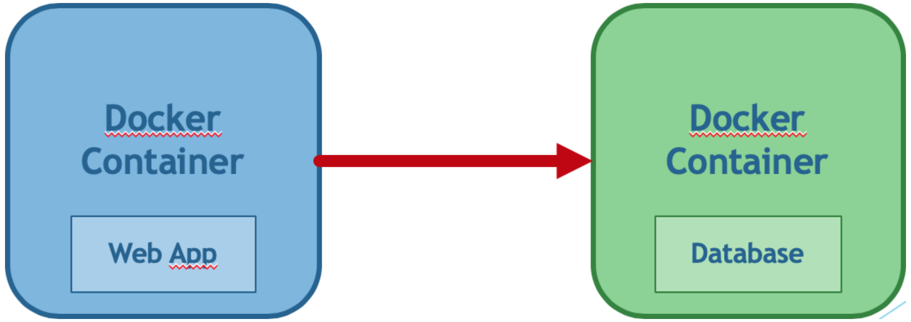
Service built using the Dockerfile in the root directory

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



Hands-on with Docker: run the application

From the root directory:



- Start up the containers

```
docker-compose up
```

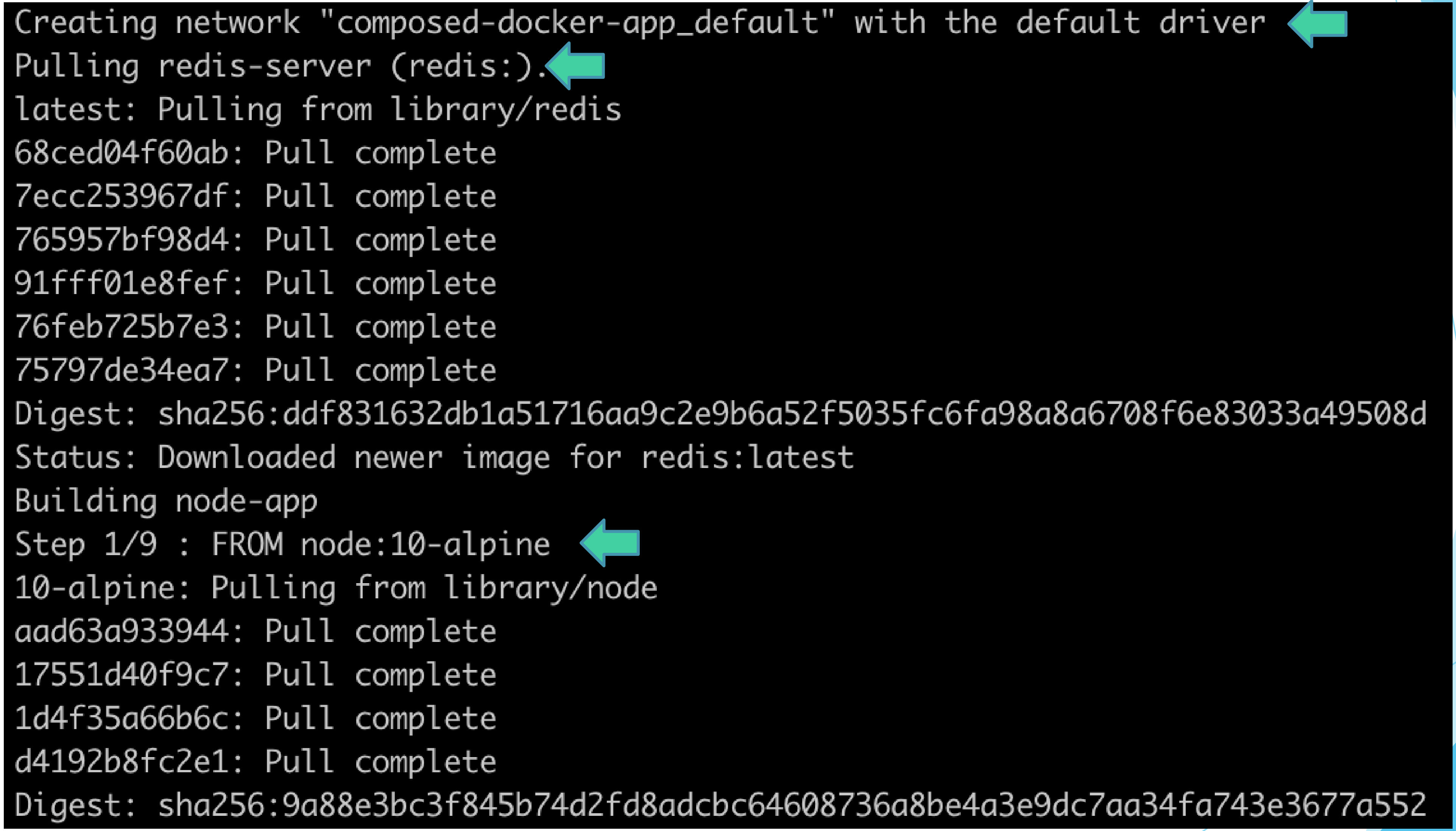
Now access the application at http://localhost:4001

- Stop the containers

```
docker-compose down
```

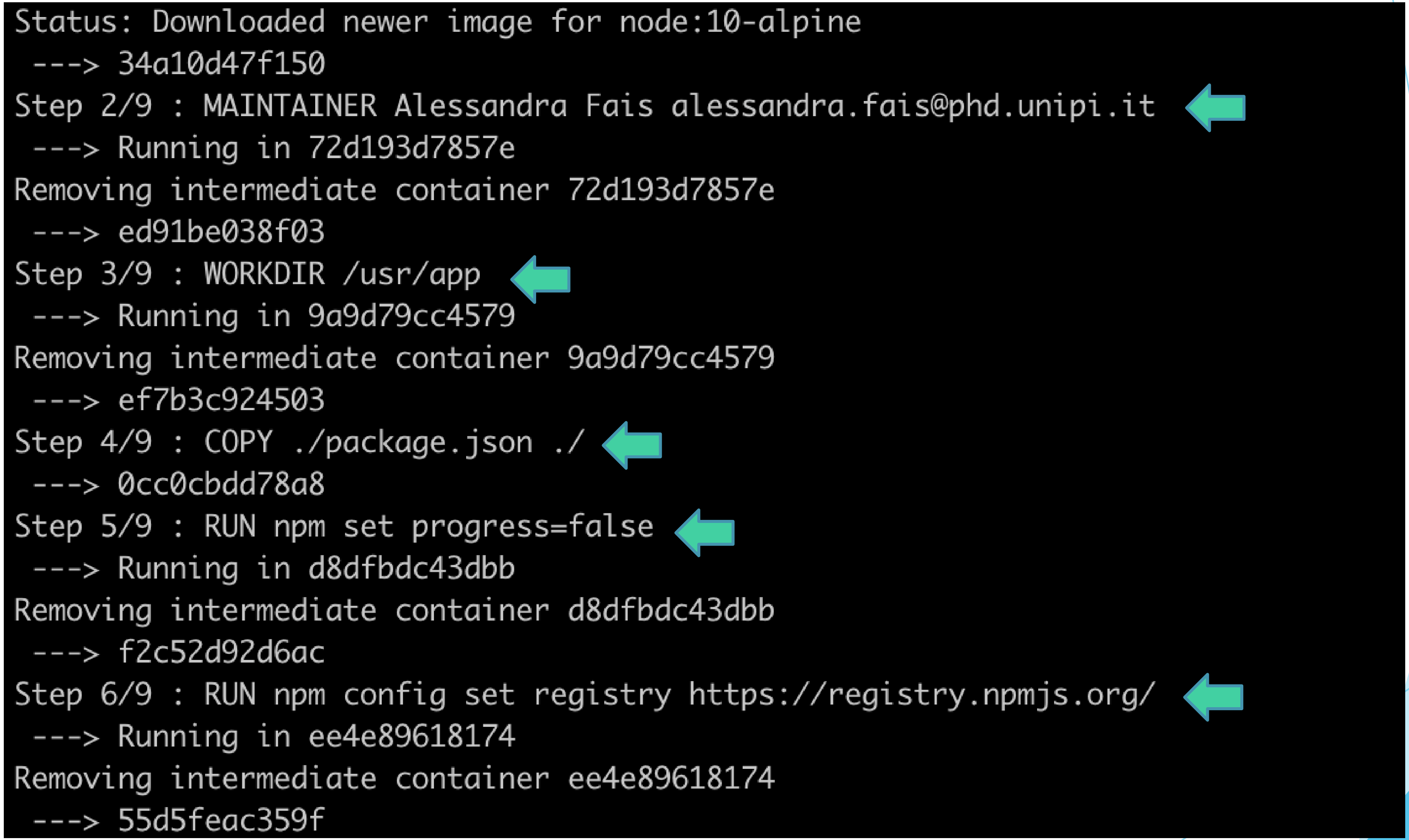
- Stop the NodeJS Docker container
- Stop the Redis database container
- Remove the NodeJS Docker container
- Remove the Redis database container
- Remove the network

Hands-on with Docker: execution output



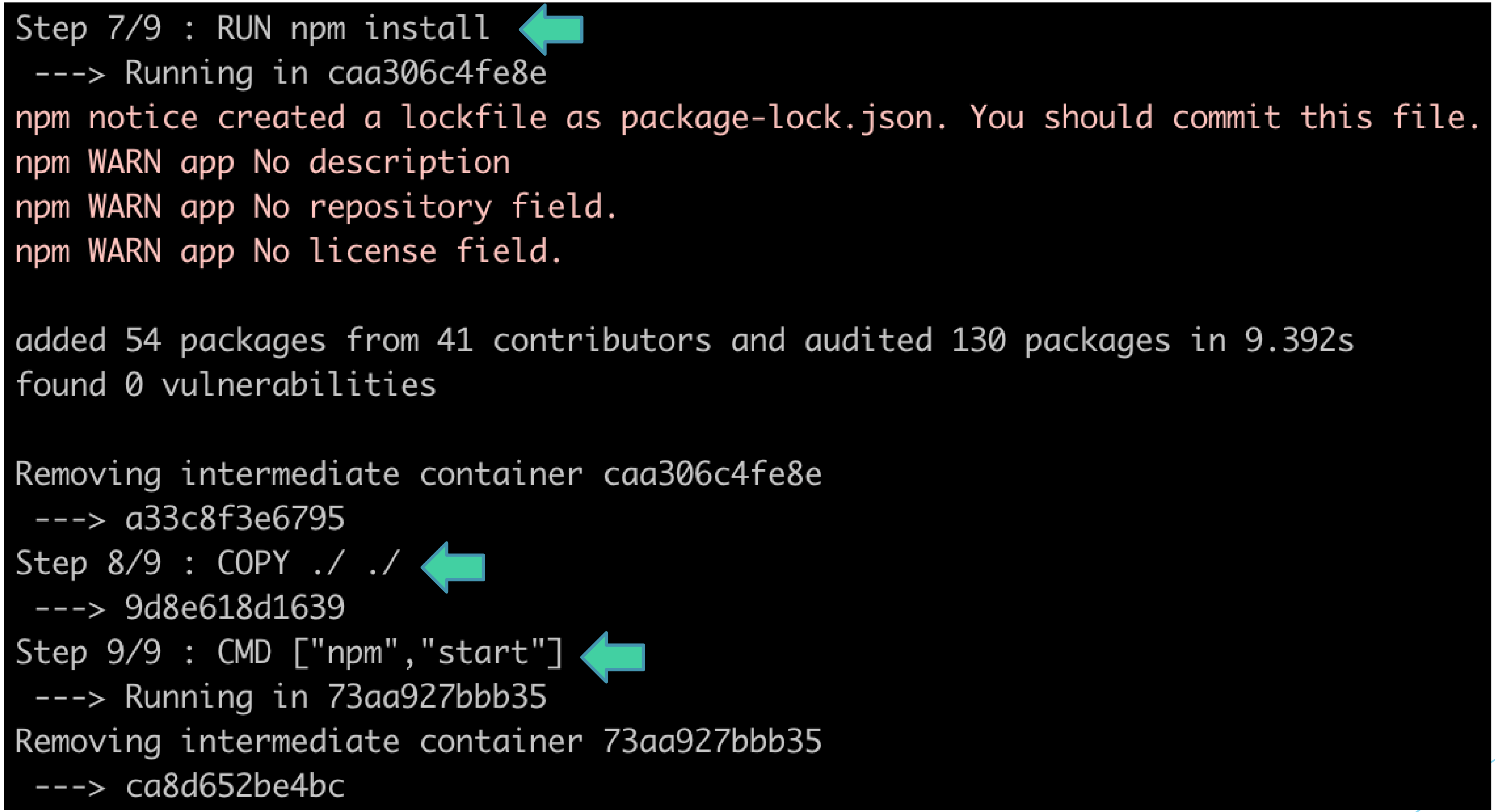
```
Creating network "composed-docker-app_default" with the default driver
Pulling redis-server (redis:).
latest: Pulling from library/redis
68ced04f60ab: Pull complete
7ecc253967df: Pull complete
765957bf98d4: Pull complete
91fff01e8fef: Pull complete
76feb725b7e3: Pull complete
75797de34ea7: Pull complete
Digest: sha256:ddf831632db1a51716aa9c2e9b6a52f5035fc6fa98a8a6708f6e83033a49508d
Status: Downloaded newer image for redis:latest
Building node-app
Step 1/9 : FROM node:10-alpine
10-alpine: Pulling from library/node
aad63a933944: Pull complete
17551d40f9c7: Pull complete
1d4f35a66b6c: Pull complete
d4192b8fc2e1: Pull complete
Digest: sha256:9a88e3bc3f845b74d2fd8adcbc64608736a8be4a3e9dc7aa34fa743e3677a552
```

Hands-on with Docker: execution output



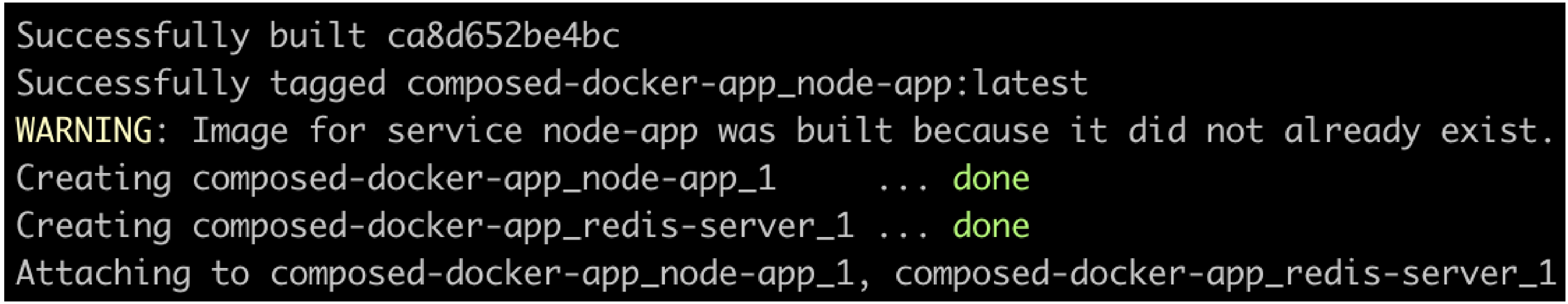
```
Status: Downloaded newer image for node:10-alpine
 ---> 34a10d47f150
Step 2/9 : MAINTAINER Alessandra Fais alessandra.fais@phd.unipi.it
 ---> Running in 72d193d7857e
Removing intermediate container 72d193d7857e
 ---> ed91be038f03
Step 3/9 : WORKDIR /usr/app
 ---> Running in 9a9d79cc4579
Removing intermediate container 9a9d79cc4579
 ---> ef7b3c924503
Step 4/9 : COPY ./package.json ./
 ---> 0cc0cbdd78a8
Step 5/9 : RUN npm set progress=false
 ---> Running in d8dfbdc43dbb
Removing intermediate container d8dfbdc43dbb
 ---> f2c52d92d6ac
Step 6/9 : RUN npm config set registry https://registry.npmjs.org/
 ---> Running in ee4e89618174
Removing intermediate container ee4e89618174
 ---> 55d5feac359f
```

Hands-on with Docker: execution output

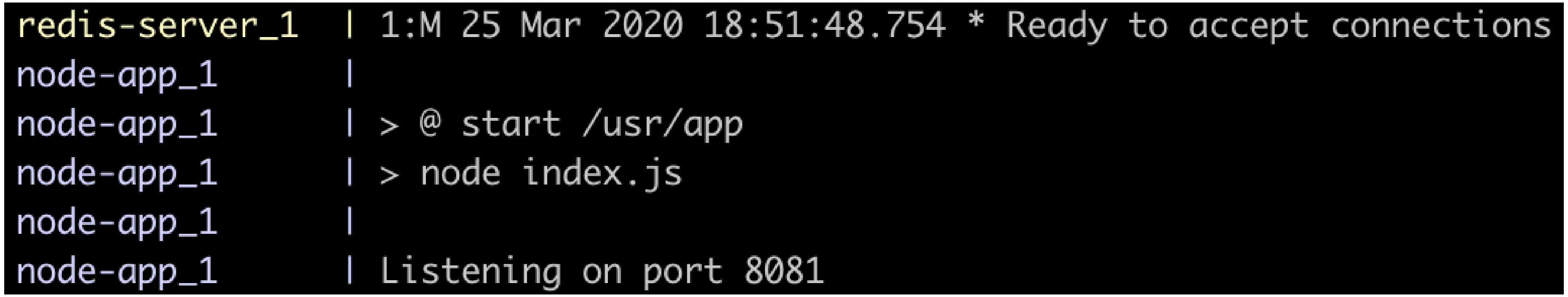


```
Step 7/9 : RUN npm install
 ---> Running in caa306c4fe8e
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN app No description
npm WARN app No repository field.
npm WARN app No license field.

added 54 packages from 41 contributors and audited 130 packages in 9.392s
found 0 vulnerabilities

Removing intermediate container caa306c4fe8e
 ---> a33c8f3e6795
Step 8/9 : COPY ./ ./
 ---> 9d8e618d1639
Step 9/9 : CMD ["npm","start"]
 ---> Running in 73aa927bbb35
Removing intermediate container 73aa927bbb35
 ---> ca8d652be4bc
```

Hands-on with Docker: execution output



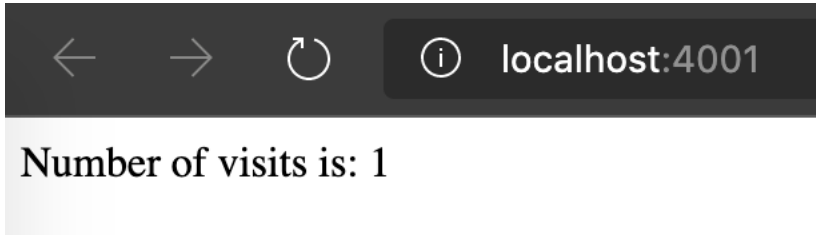
```
Successfully built ca8d652be4bc
Successfully tagged composed-docker-app_node-app:latest
WARNING: Image for service node-app was built because it did not already exist.
Creating composed-docker-app_node-app_1      ... done
Creating composed-docker-app_redis-server_1 ... done
Attaching to composed-docker-app_node-app_1, composed-docker-app_redis-server_1
```



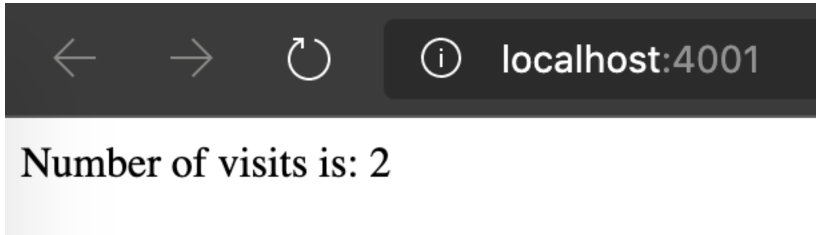
```
redis-server_1  | 1:M 25 Mar 2020 18:51:48.754 * Ready to accept connections
node-app_1      |
node-app_1      | > @ start /usr/app
node-app_1      | > node index.js
node-app_1      |
node-app_1      | Listening on port 8081
```

NOW THE APP IS UP AND RUNNING!!

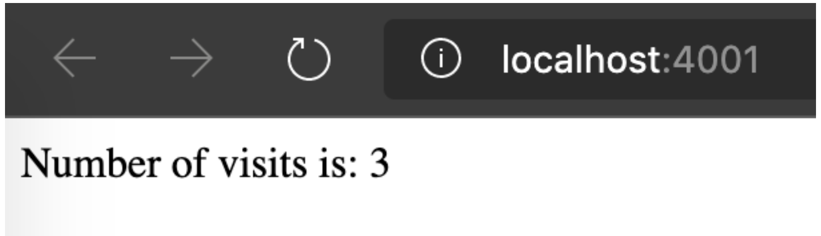
Hands-on with Docker: test with a browser



Number of visits is: 1



Number of visits is: 2



Number of visits is: 3

1st request

2nd request

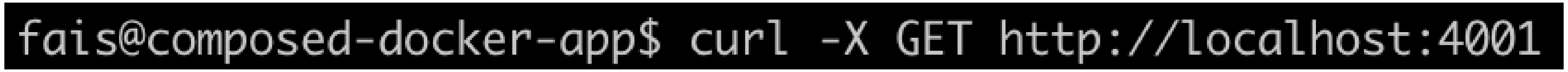
3rd request

. . .

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

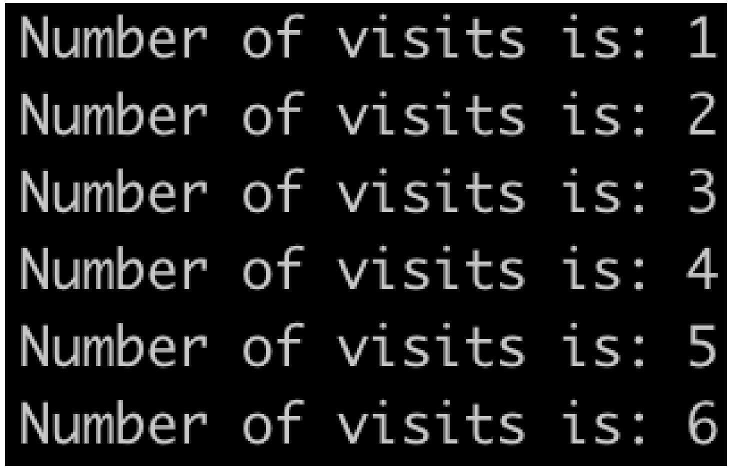
We can send some **HTTP GET** requests by using **curl**

- 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

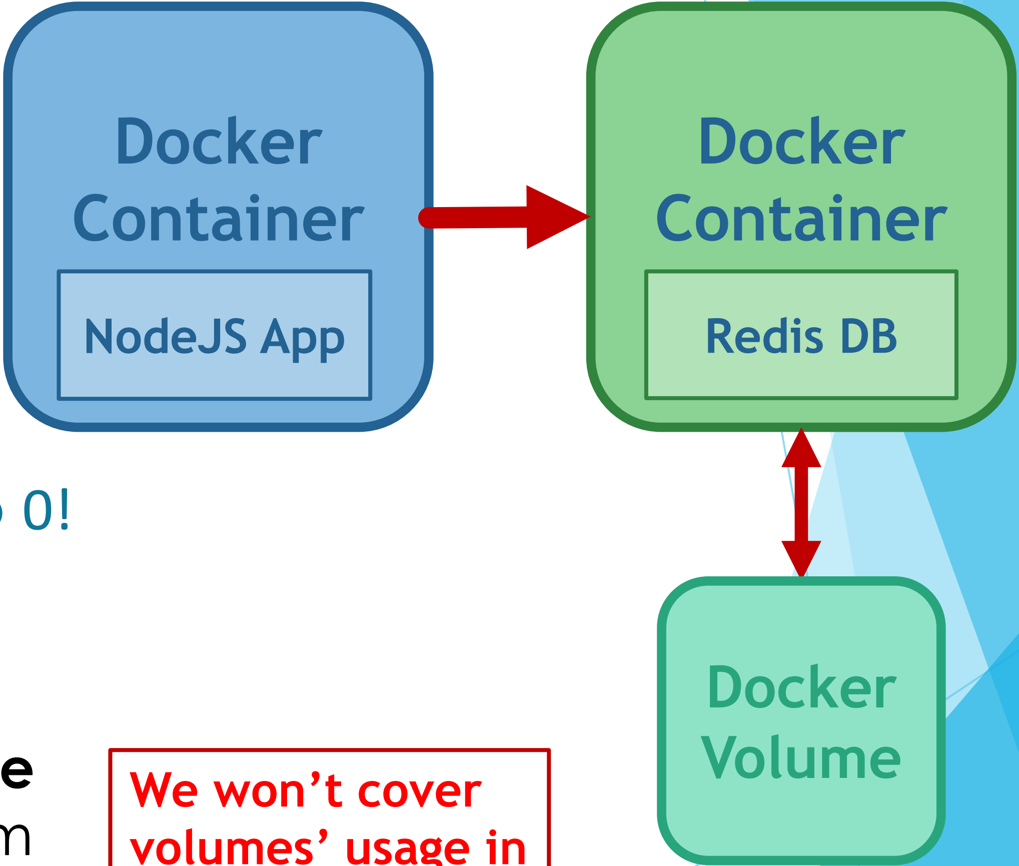
2nd request

3rd request

. . .

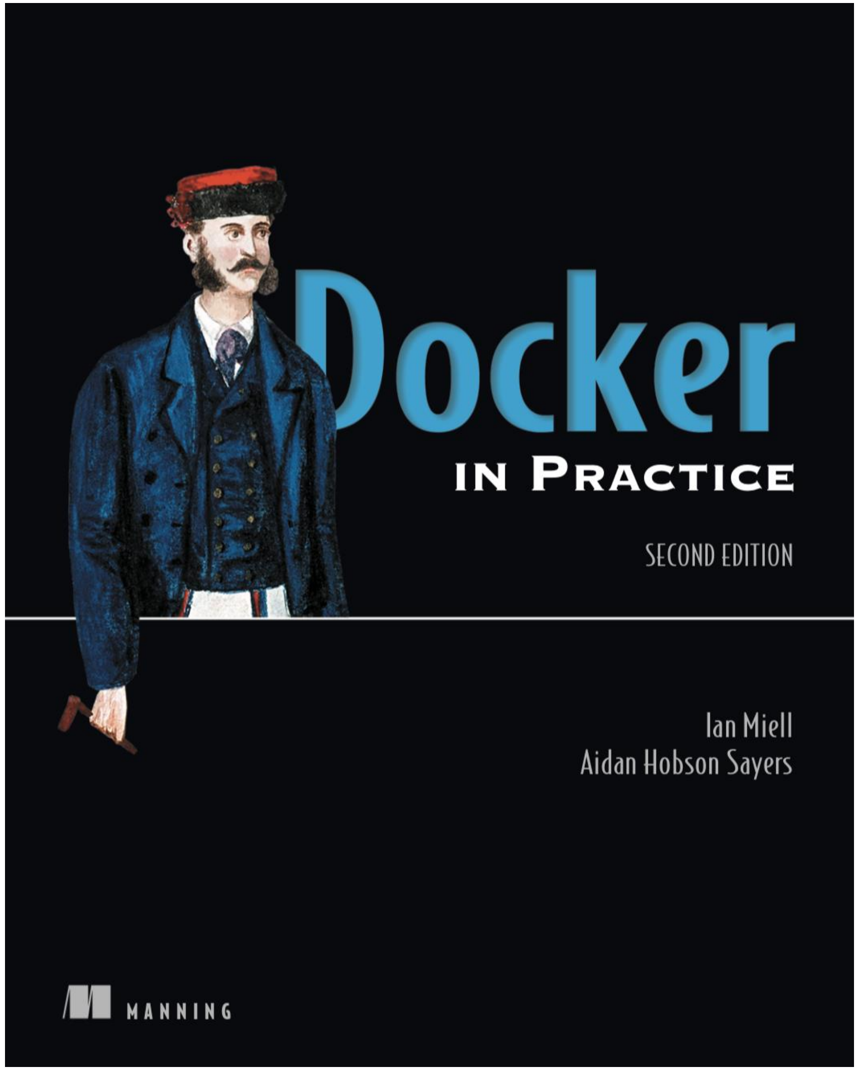
Hands-on with Docker: note on storage

- What happens if we run again our complex example?
  - Run `docker-compose down`
  - Run `docker-compose up`
- The visit counter has been resetted to 0!
  - Redis container does not provide permanent storage
  - Some applications could need to **store data permanently** (maintain data from one execution to another)
    - Solution: use **volumes**!



We won't cover volumes' usage in this course, but it's important to know they exist!

Useful reference book



Other useful references on Docker

- Docker quickstart
  - https://docs.docker.com/get-started/
- Dockerfile reference
  - https://docs.docker.com/engine/reference/builder/
- Best practices for writing Dockerfiles
  - https://docs.docker.com/develop/develop-images/dockerfile_best-practices/
- A Dockerfile tutorial by examples
  - https://takacsmark.com/dockerfile-tutorial-by-example-dockerfile-best-practices-2018/
- Interesting point of view on containers and VMs
  - https://www.docker.com/blog/containers-are-not-vms/

Other references on Docker networking

- Docker networking overview
  - https://docs.docker.com/network/
- Container networking
  - https://docs.docker.com/config/containers/container-networking/
- Bridge network description and tutorial
  - https://docs.docker.com/network/bridge/
  - https://docs.docker.com/network/network-tutorial-standalone/
- Host network description and tutorial
  - https://docs.docker.com/network/host/
  - https://docs.docker.com/network/network-tutorial-host/

Other references on Docker Compose

- 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/

And if you want to go ahead on your own…

- Managing data in Docker and how to use Volumes
  - https://docs.docker.com/storage/
  - https://docs.docker.com/storage/volumes/
- Starting with clusters management (Swarm, Kubernetes)
  - https://docs.docker.com/get-started/swarm-deploy/
  - https://docs.docker.com/get-started/kube-deploy/