Università di Pisa

Dept. of Information Engineering

Course on Wireless Networks - 2020/2021

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
  - Hands-on session: VirtualBox + Ubuntu Linux VM creation

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

PART IV

Outline of Part IV

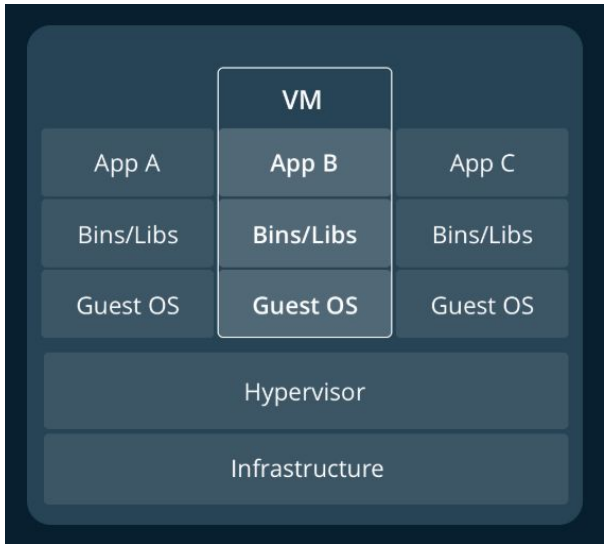
1) Containers characteristics
2) Docker
   - Objects
   - Architecture
   - Deployment modes
     - Single host VS Cluster
3) Hands-on session with Docker
   - Installation
   - Execution flow
   - Commands
   - Dockerfile
   - Persistent storage management (volumes)
   - Docker Compose
   - Kubernetes

Containers and Docker

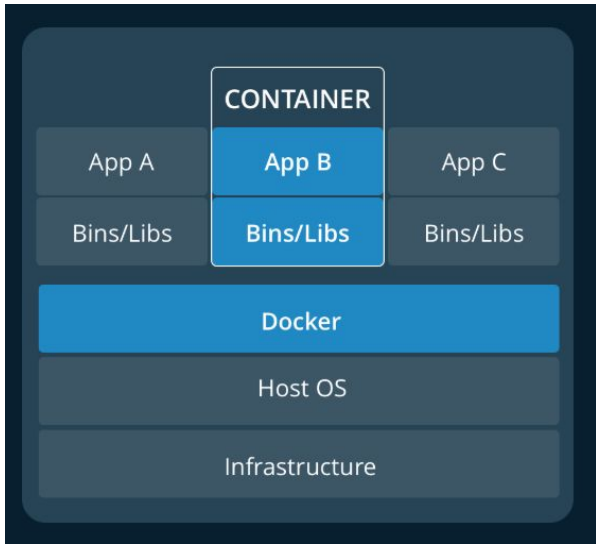
Container characteristics

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)

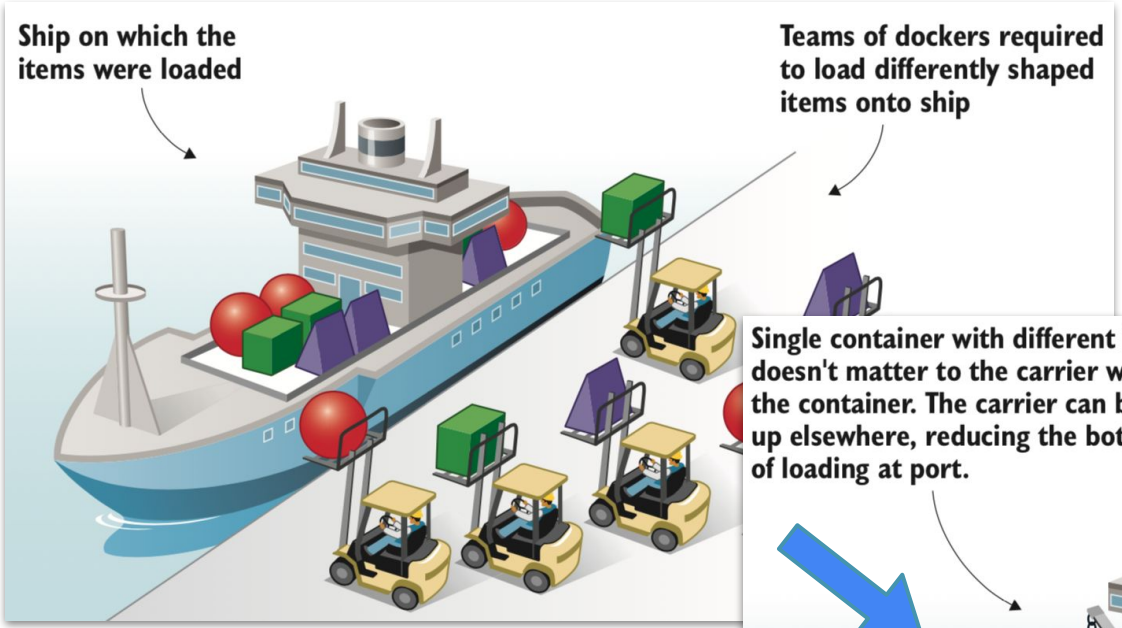


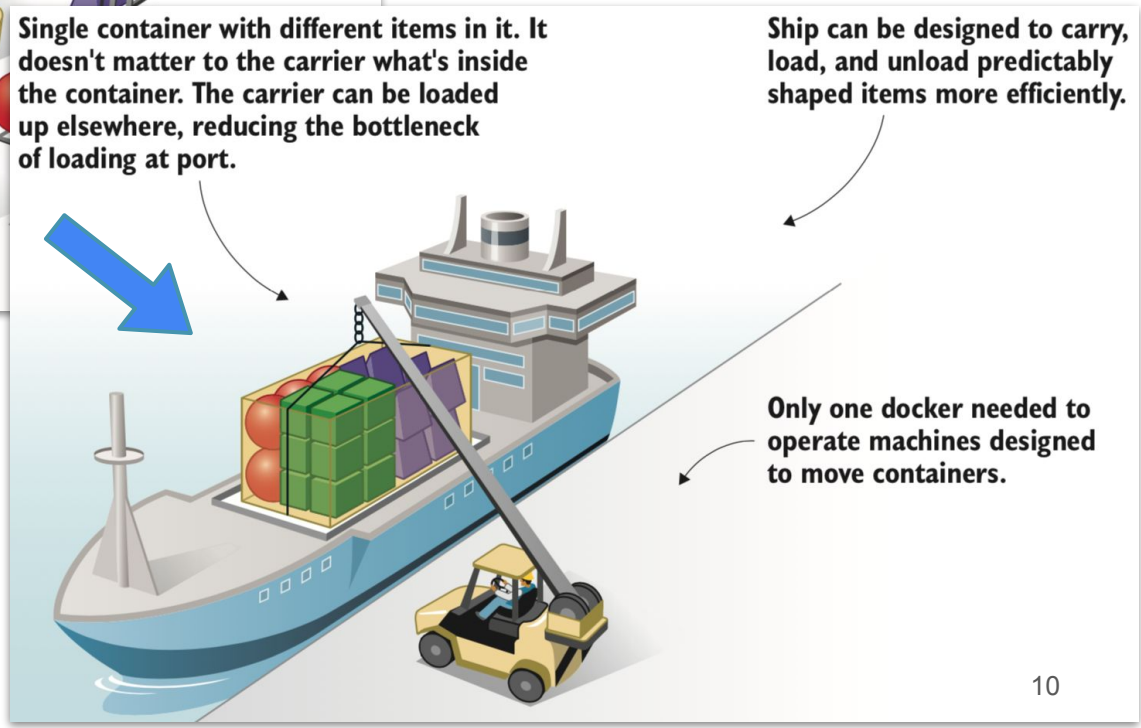
Containers: characteristics

- **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

Containers: Docker

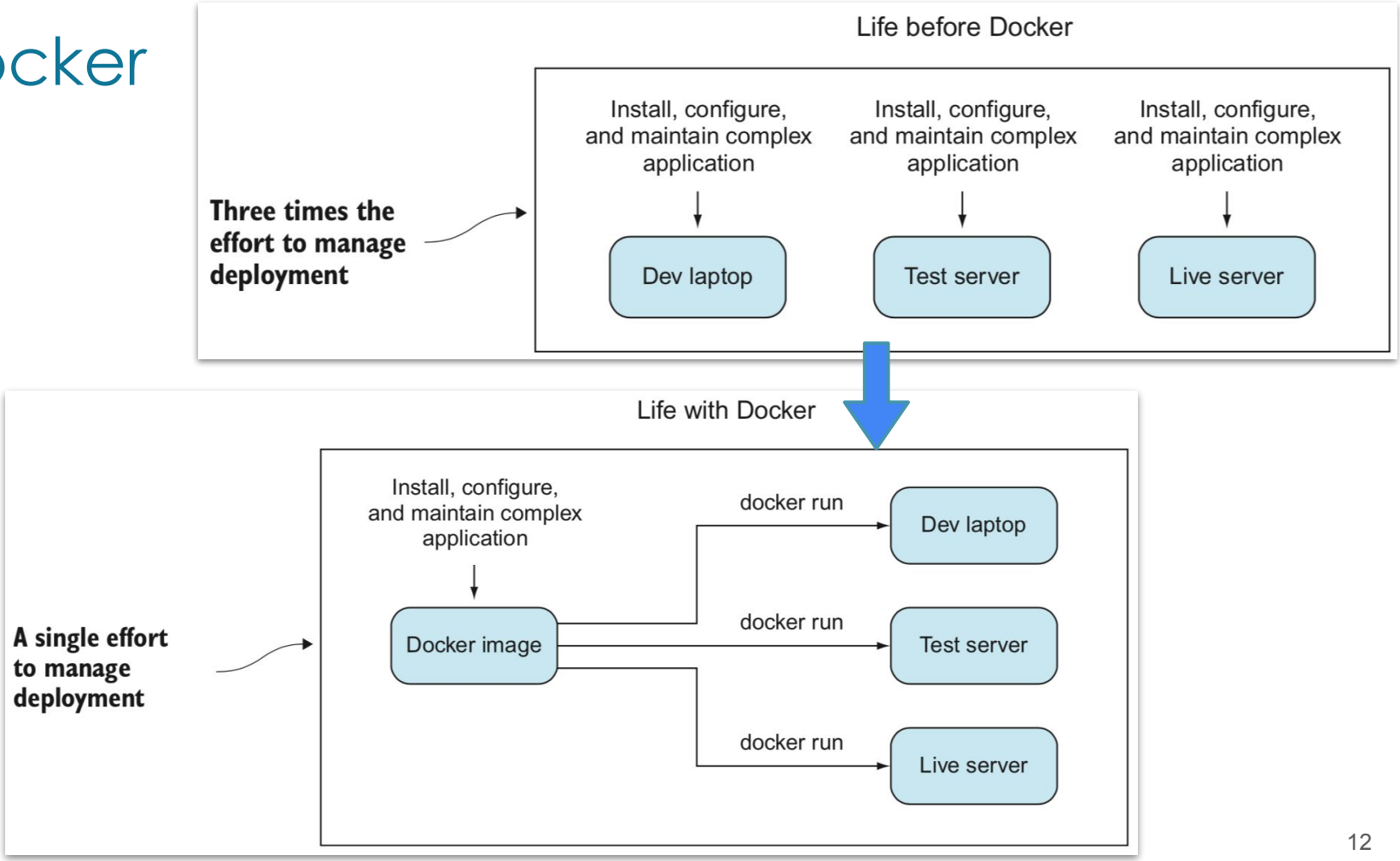




Docker

Introduction

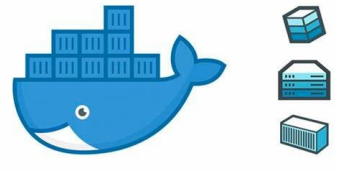
Docker


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

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



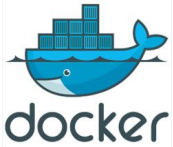
Docker

- 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**

Docker

Objects

Docker objects



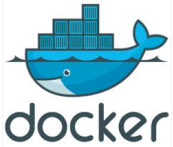
Image

- **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

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

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

Docker

Architecture

Docker ecosystem and deployment modes

Docker platform

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

Single host mode

Deploy containers on a single host machine

Cluster mode

Deploy containers of a Docker stack on all the nodes of a cluster (configuration with manager node + set of workers nodes)

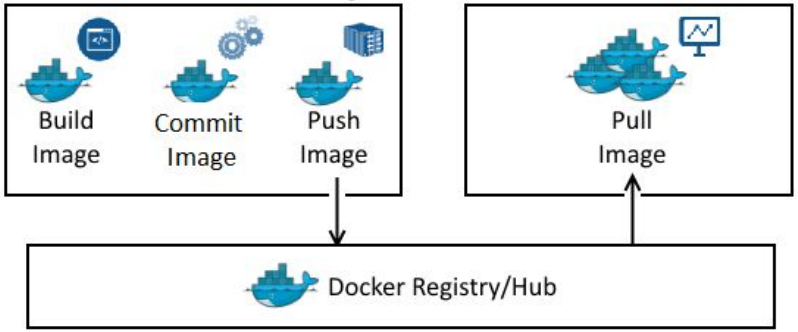
- Docker Swarm
- Mesos
- Kubernetes

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

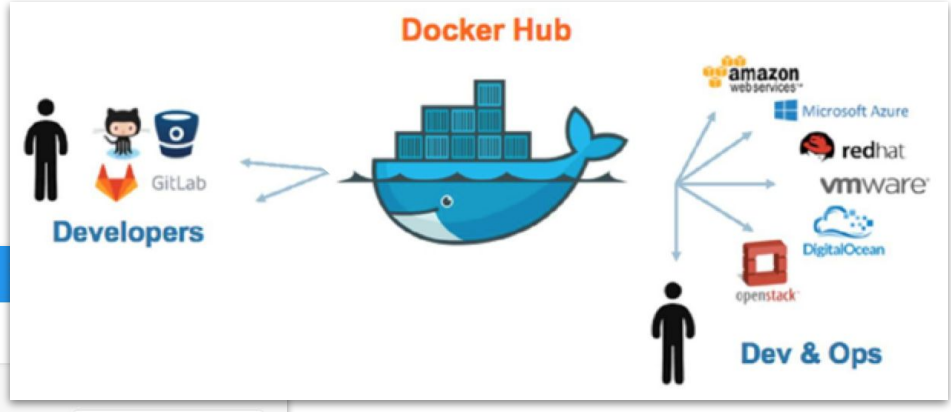
Docker Hub

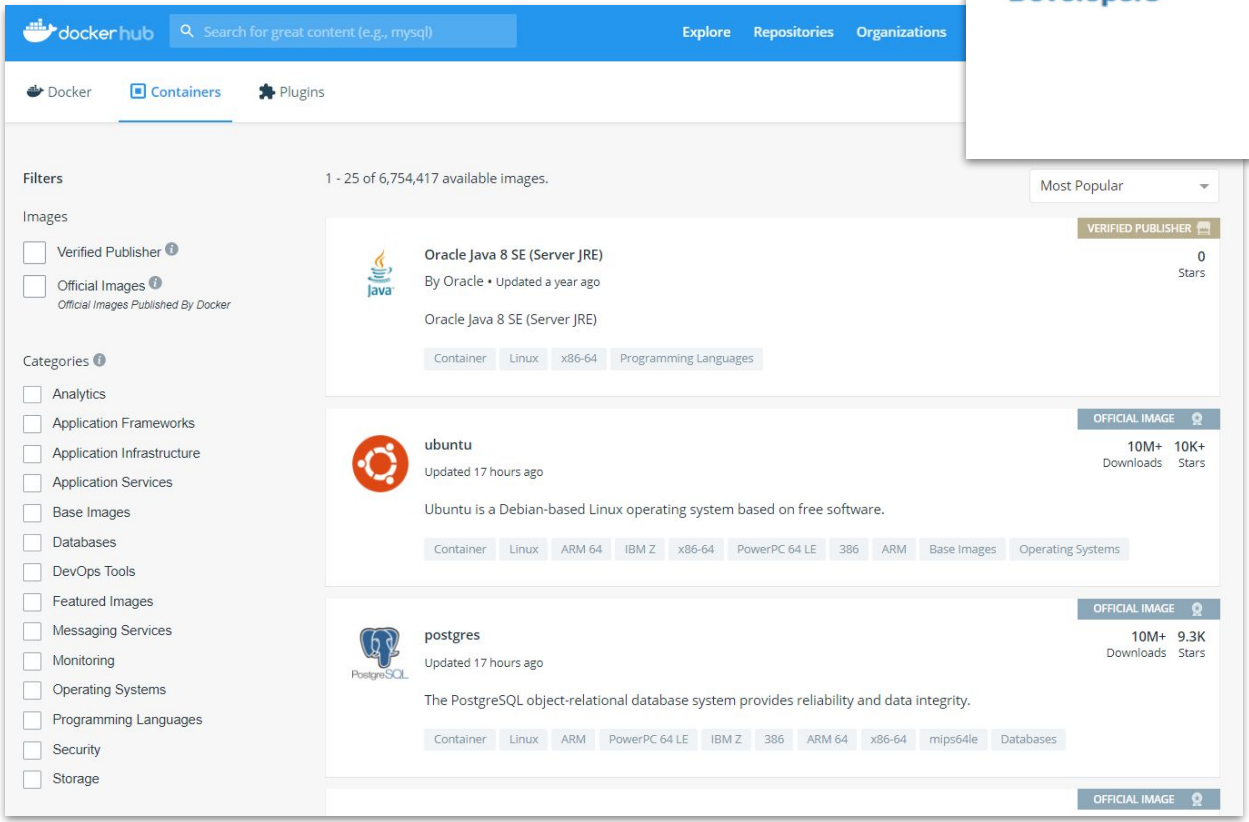


- Service provided by Docker for finding and sharing container images
- **Repositories** allow sharing container images with the Docker community
- Container images can be *pushed* to a repository or *pulled* from it
  - Official images (provided by Docker)
    - Clear documentation, best practices,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

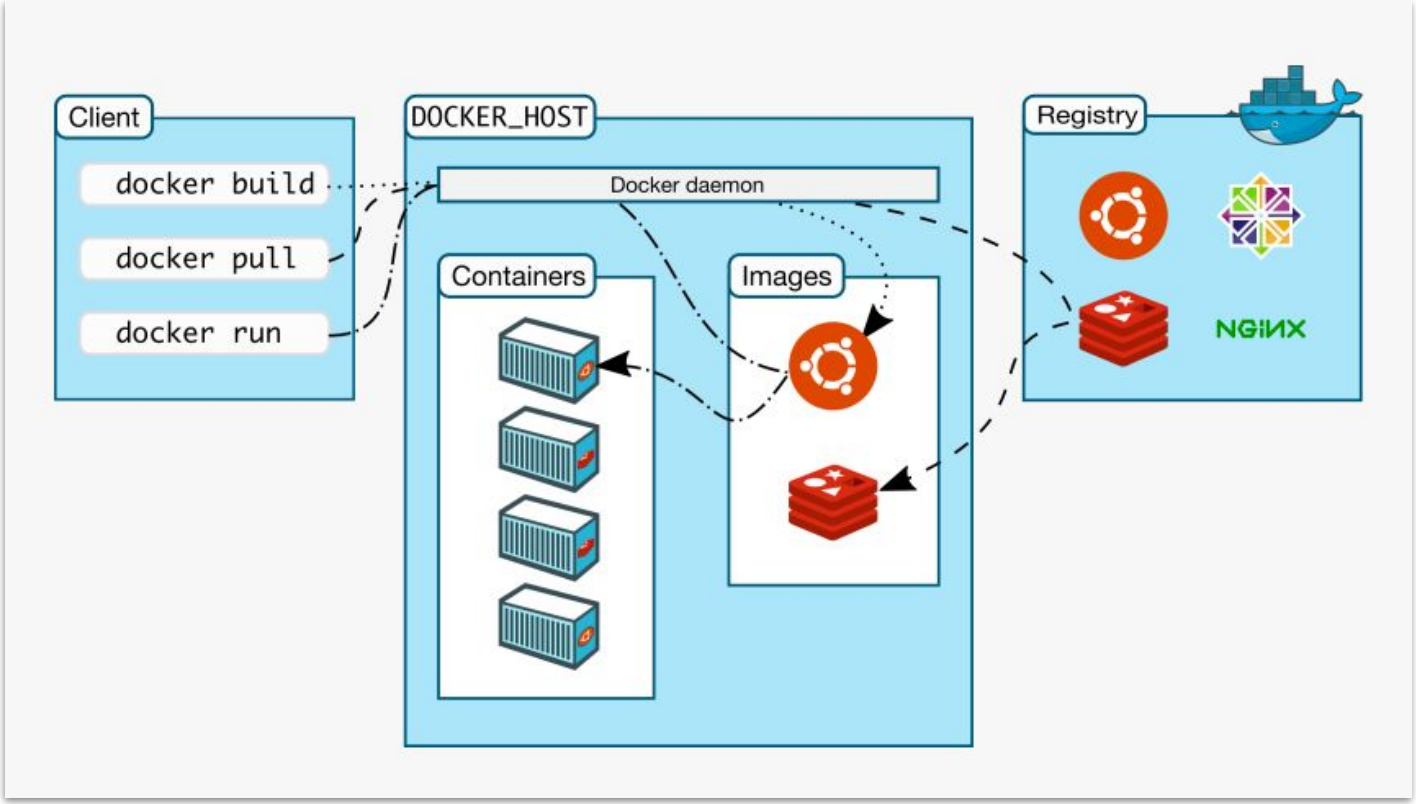
Docker guide section, here:
https://docs.docker.com/docker-hub/

Docker Hub


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


docker hub
Search for great content (e.g., mysql)
Explore
Repositories
Organizations
Docker
Containers
Plugins
Filters
1 - 25 of 6,754,417 available images.
Most Popular
Images
Verified Publisher
Official Images
Official Images Published By Docker
Categories
Analytics
Application Frameworks
Application Infrastructure
Application Services
Base Images
Databases
DevOps Tools
Featured Images
Messaging Services
Monitoring
Operating Systems
Programming Languages
Security
Storage
VERIFIED PUBLISHER
Oracle Java 8 SE (Server JRE)
By Oracle • Updated a year ago
0
Stars
Oracle Java 8 SE (Server JRE)
Container
Linux
x86-64
Programming Languages
OFFICIAL IMAGE
ubuntu
Updated 17 hours ago
10M+
Downloads
10K+
Stars
Ubuntu is a Debian-based Linux operating system based on free software.
Container
Linux
ARM 64
IBM Z
x86-64
PowerPC 64 LE
386
ARM
Base Images
Operating Systems
OFFICIAL IMAGE
postgres
Updated 17 hours ago
10M+
Downloads
9.3K
Stars
The PostgreSQL object-relational database system provides reliability and data integrity.
Container
Linux
ARM
PowerPC 64 LE
IBM Z
386
ARM 64
x86-64
mips64le
Databases
OFFICIAL IMAGE

Putting all together: Docker Architecture



Hands-on with Docker

Preliminary steps

Install VirtualBox Guest Additions (useful thing)

- VirtualBox Guest Additions
  - Set of drivers and applications to improve your VM usability
  - Useful functionalities for guest machines:
    - shared folders, shared clipboard, mouse pointer integration, better video support, and more

- From inside your Ubuntu VM guest terminal, run the commands

```
$ sudo apt update
$ sudo apt install build-essential dkms linux-headers-$(uname -r)
```

- From the VM menu click **Devices -> Insert Guest Additions CD Image**
- From inside your Ubuntu VM guest terminal, run (1) and then run (2)

(1)

```
$ sudo mkdir -p /mnt/cdrom
$ sudo mount /dev/cdrom /mnt/cdrom
```

(2)

```
$ cd /mnt/cdrom
$ sudo sh ./VBoxLinuxAdditions.run --nox11
```


https://linuxize.com/post/how-to-install-virtualbox-guest-additions-in-ubuntu/

Install VirtualBox Guest Additions (useful thing)

- Check if everything has been configured correctly
  (you should see a similar output)

```
Output
Verifying archive integrity... All good.
Uncompressing VirtualBox 5.2.32 Guest Additions for Linux........
...
VirtualBox Guest Additions: Starting.
```

- Reboot

```
$ sudo shutdown -r now
```

- Check if the installation has been successful
  - run (1) and you should see an output similar to (2)

(1)

```
$ lsmod | grep vboxguest
```

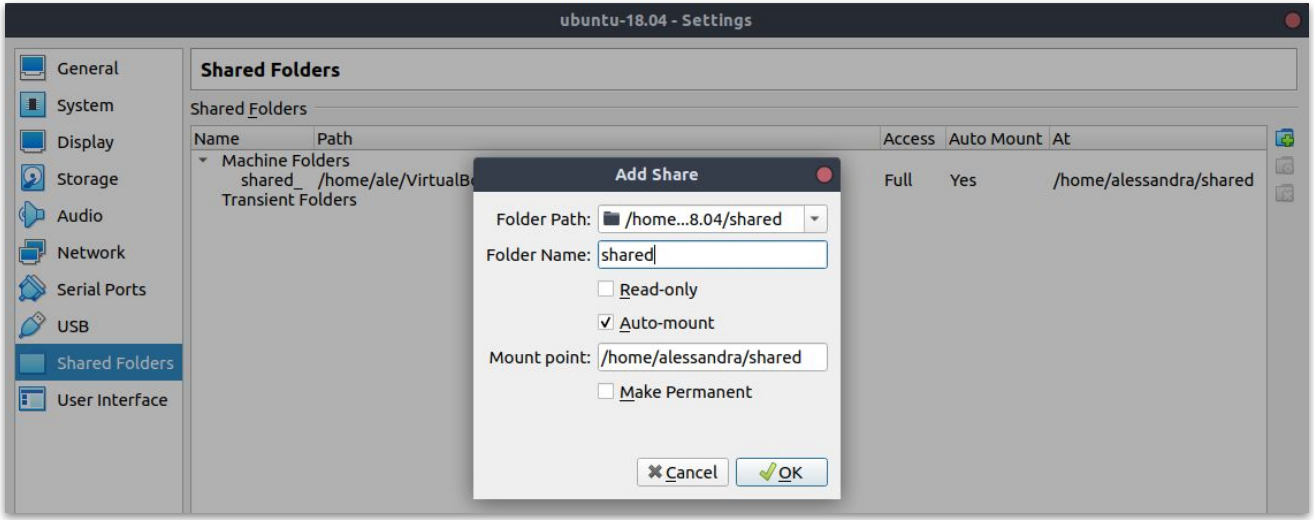
(2)

```
Output
vboxguest              303104  2 vboxsf
```

https://linuxize.com/post/how-to-install-virtualbox-guest-additions-in-ubuntu/

Let's create now a shared folder (useful thing)

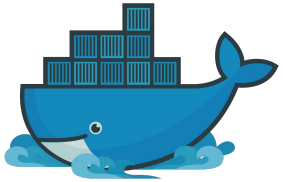
shared/ will be shared between your Host OS and the VM Guest OS



You can copy into this folder the code of the exercises!

Installing Docker

Installing Docker



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

- 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)

- **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
```

Installing Docker

- **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

3

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

Add Docker's official GPG key

Installing Docker

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"
```

Installing Docker

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

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

Update the apt **package index**

1

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

2

```
$ 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.**

Use Docker as a non-root user

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

- 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
```

Our first running Docker 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`

Execution flow

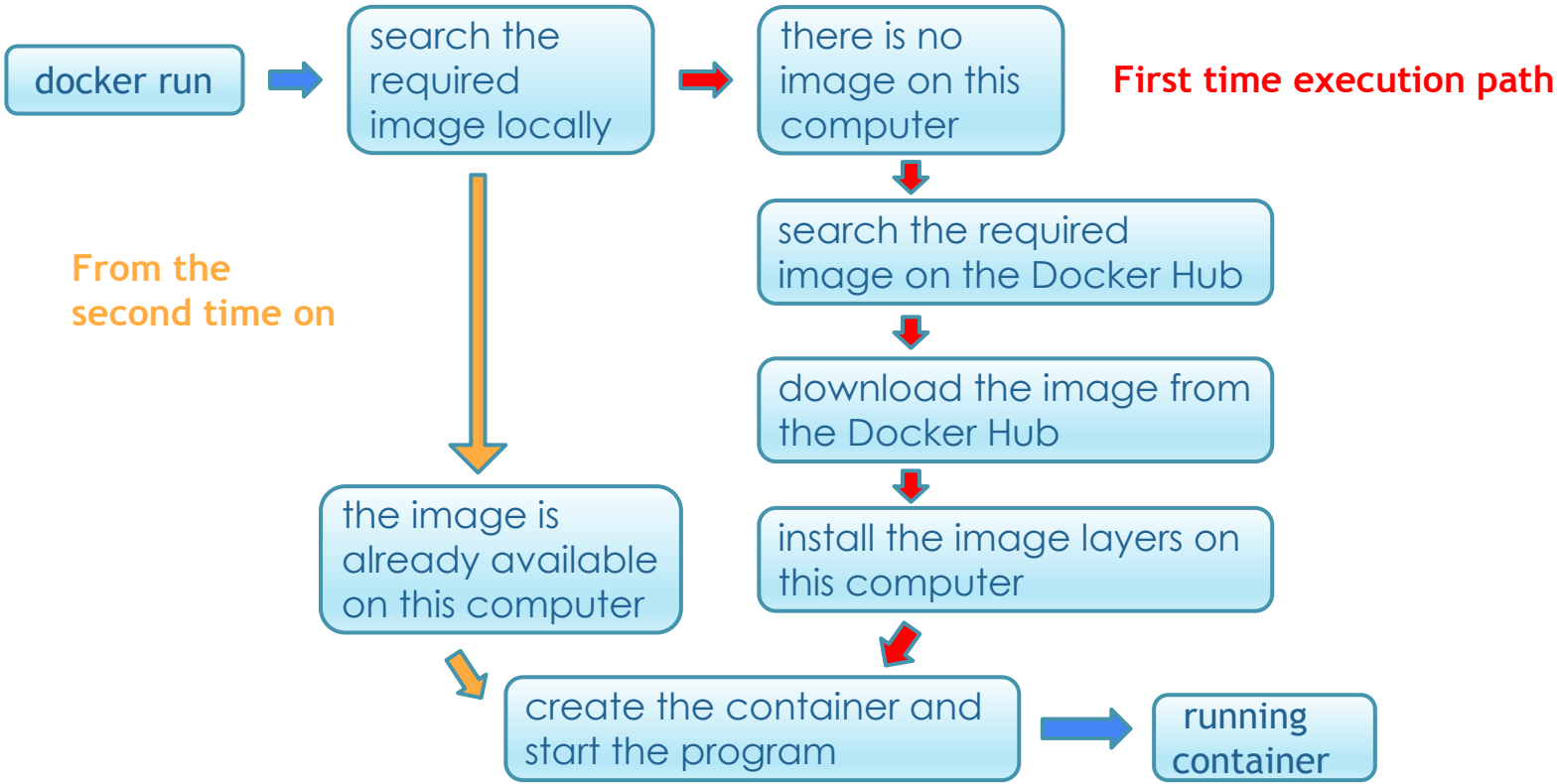
The execution flow

- **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



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

run command useful flags

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

- 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
```

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**

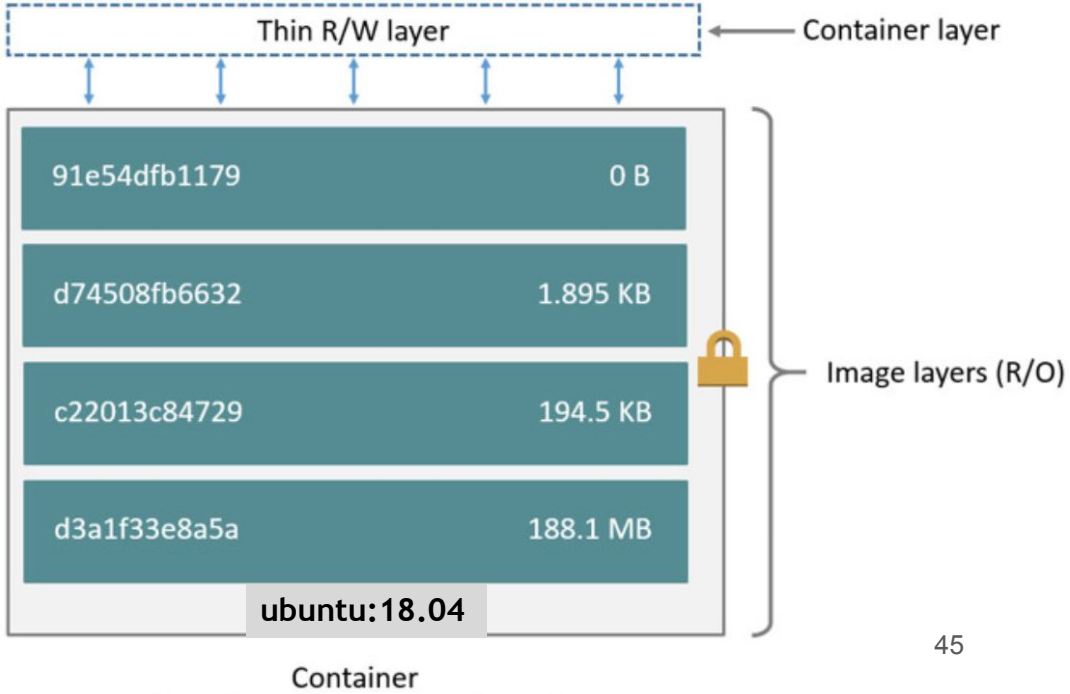
Dockerfile

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
```

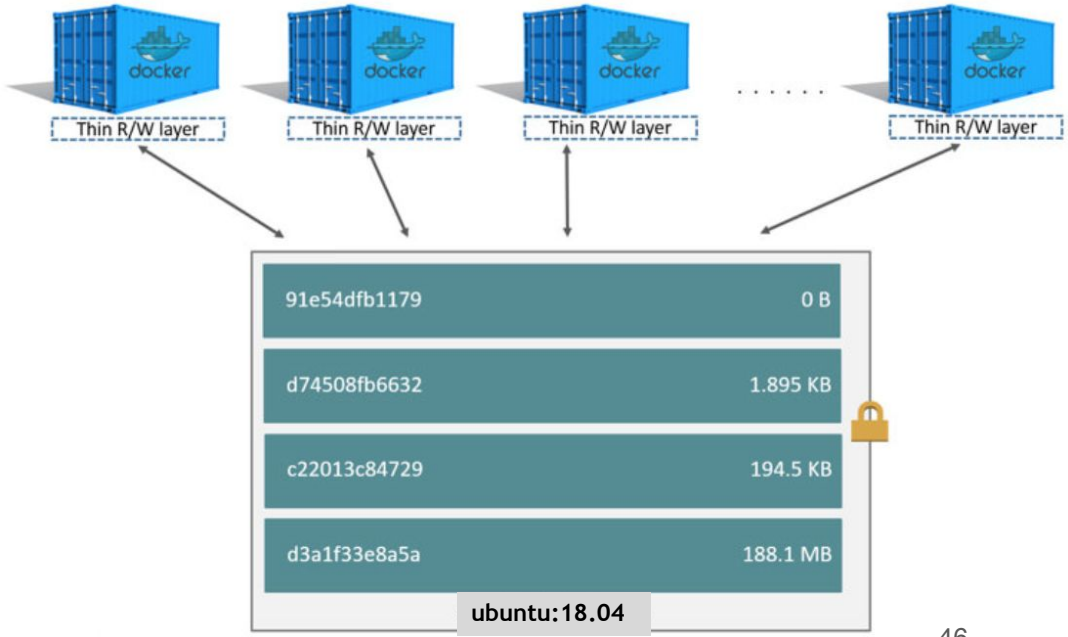


Dockerfile

- 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



Dockerfile instructions

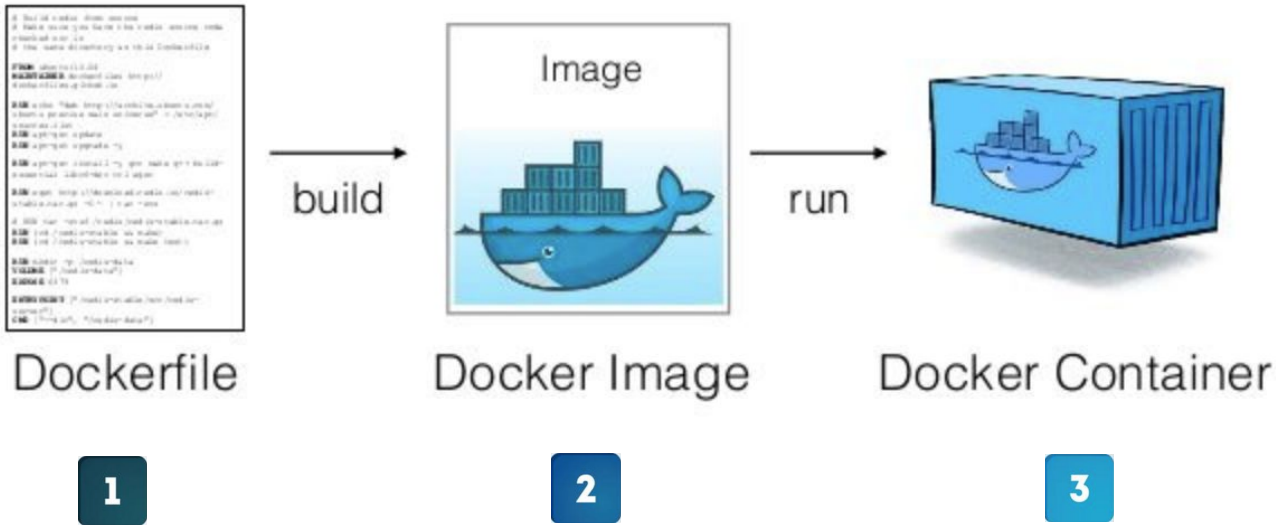
- FROM <image>[:tag] : start your image from a pre-existing parent image (e.g., official Docker image) called **base image**
- WORKDIR : working directory in the image private filesystem
- USER : specify the user used to run any subsequent RUN, CMD, ENTRYPOINT, … instructions
- COPY <src> <dst> : copy 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**

Docker Container

Bash script

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

Build context for the image (current directory)

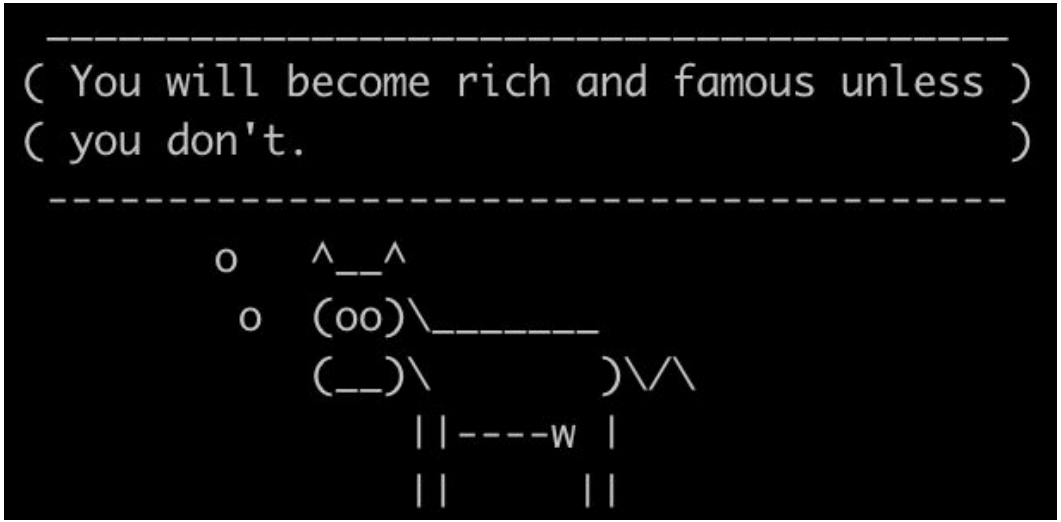
Tag to identify the image

```
$ 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:



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

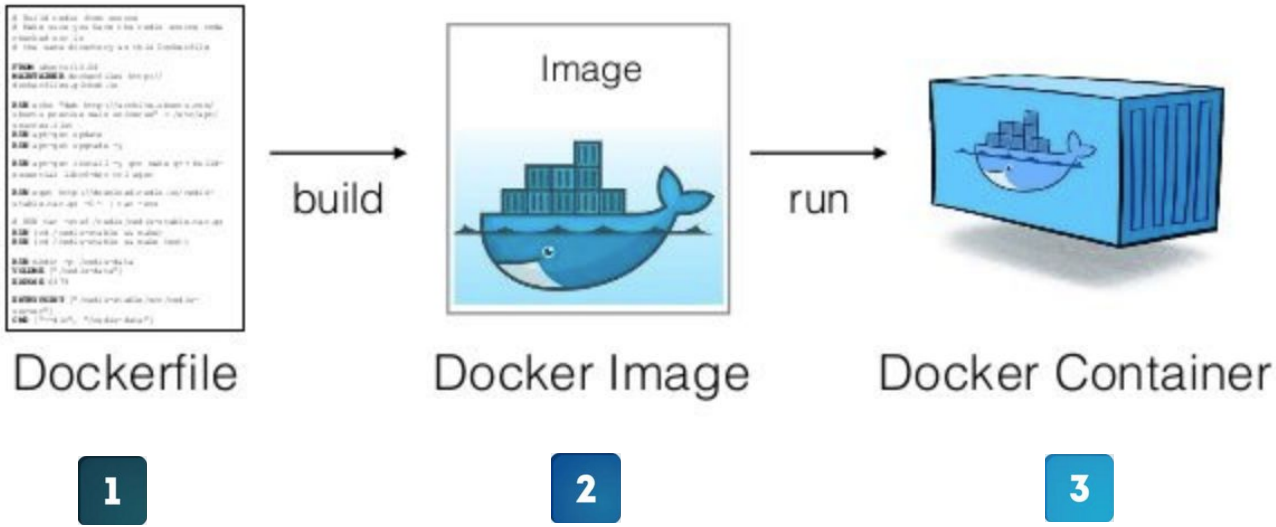
Second example (single host)

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**

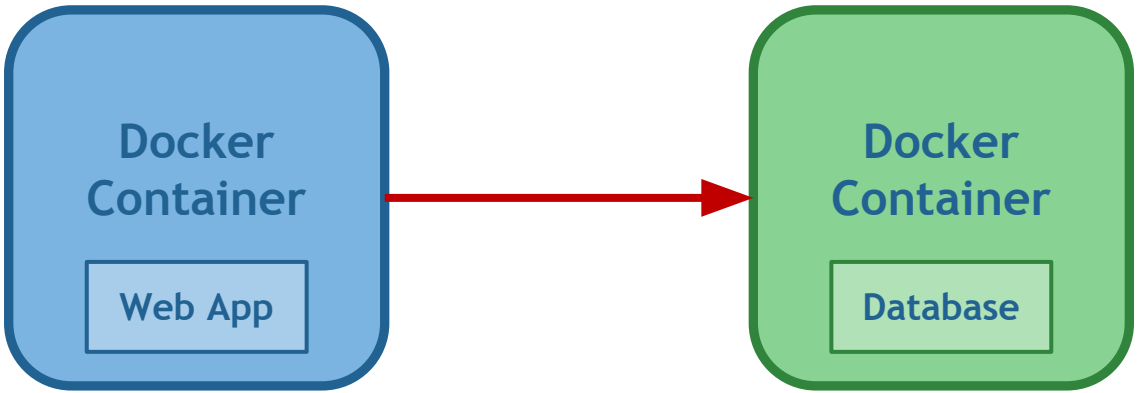


- Then set up networking and compose the services together

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

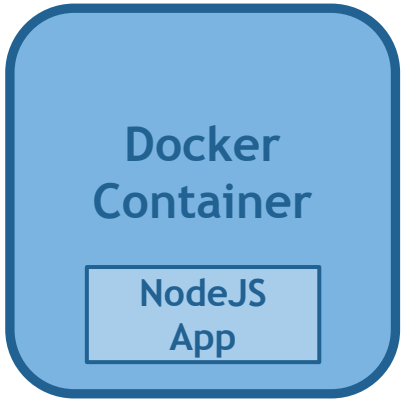
Hands-on with Docker: complex example

- 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') })
```

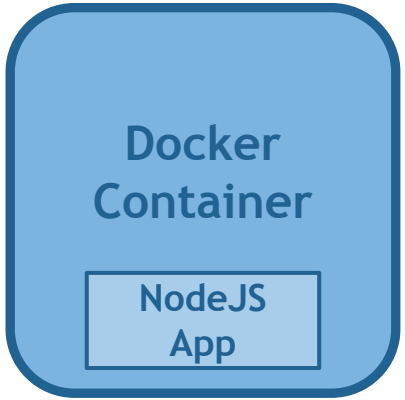
Use the Express framework

Docker Compose service name

Set initial visits

Define the root endpoint

Specify the listening port to 8081



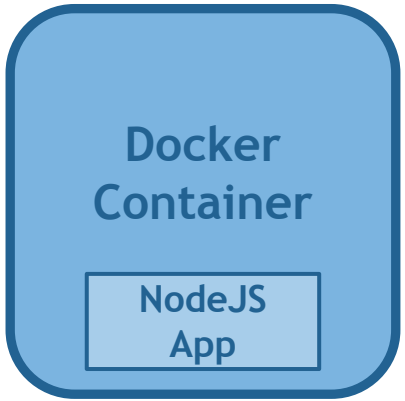
Link to the code: https://github.com/alefais/virtualization-lab-unipi/tree/main/composed-docker-app

Hands-on with Docker: package.json

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

Express framework
https://expressjs.com/

Redis 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

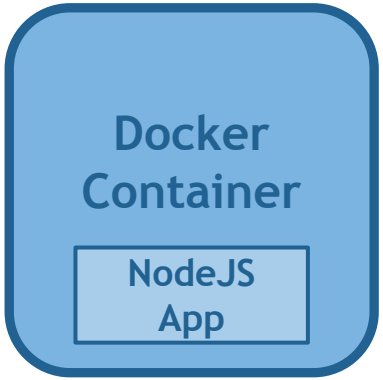
Hands-on with Docker: Dockerfile

```
# 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

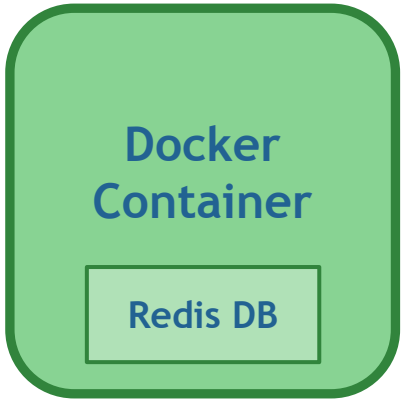
Start the container



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

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/
```

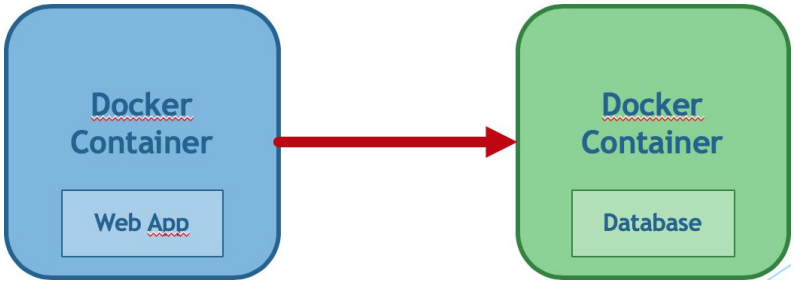
```
1.25.4/docker-compose-$(uname -s)-$(uname -m)" -o  /usr/local/bin/docker-compose
```

```
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



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"
```

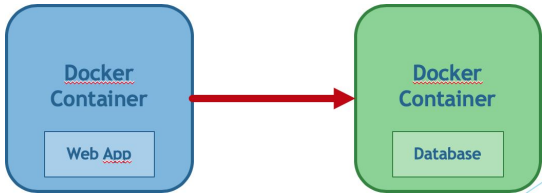
Docker Compose version

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

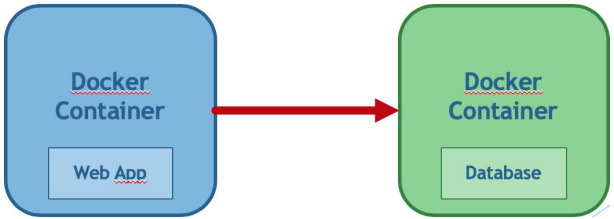
Map port **4001** on the local host to port **8081** on the Docker container



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

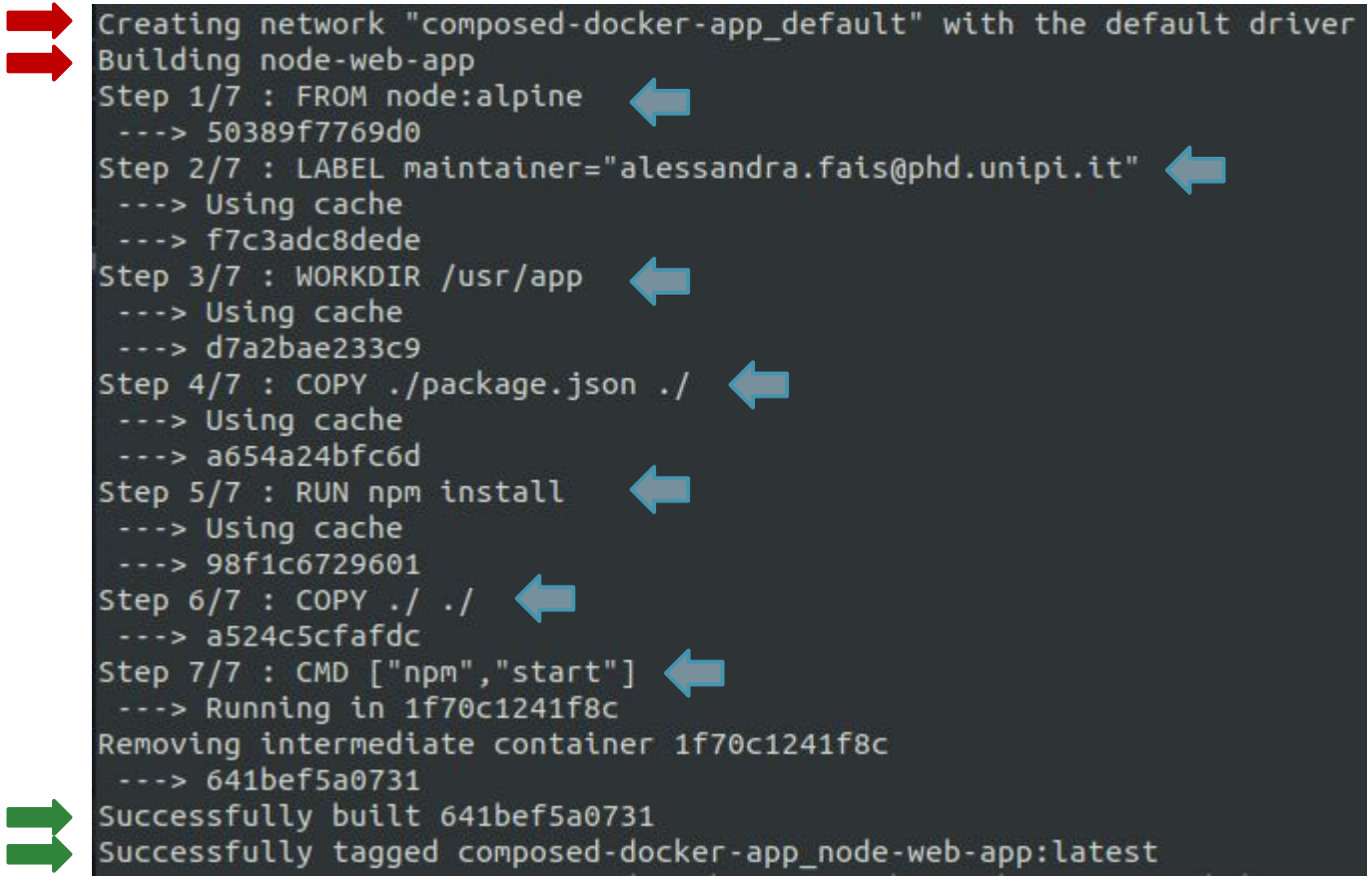
  docker-compose down

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



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

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

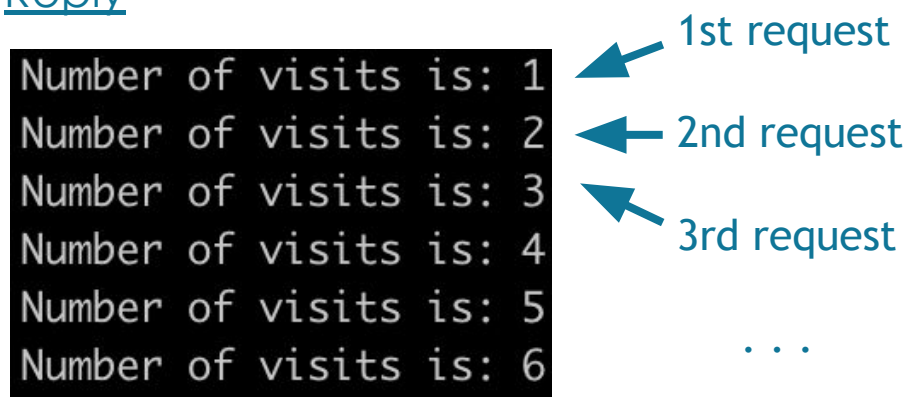
Hands-on with Docker: test with curl

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

- Request

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

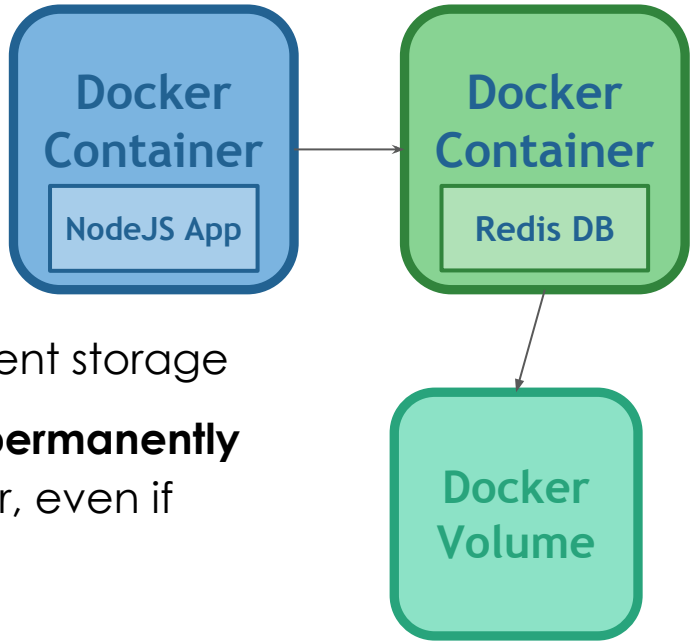
- Reply



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": ""
    }
],
```

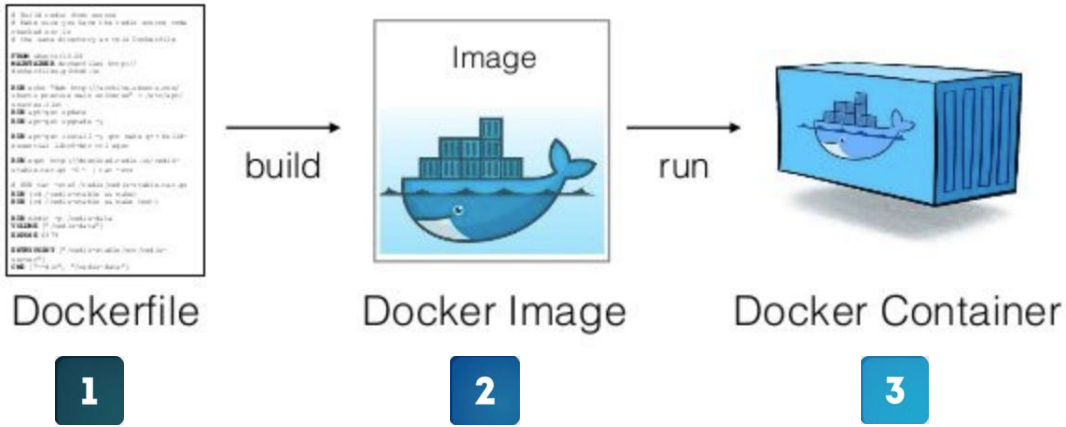
Third example (single host)

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**

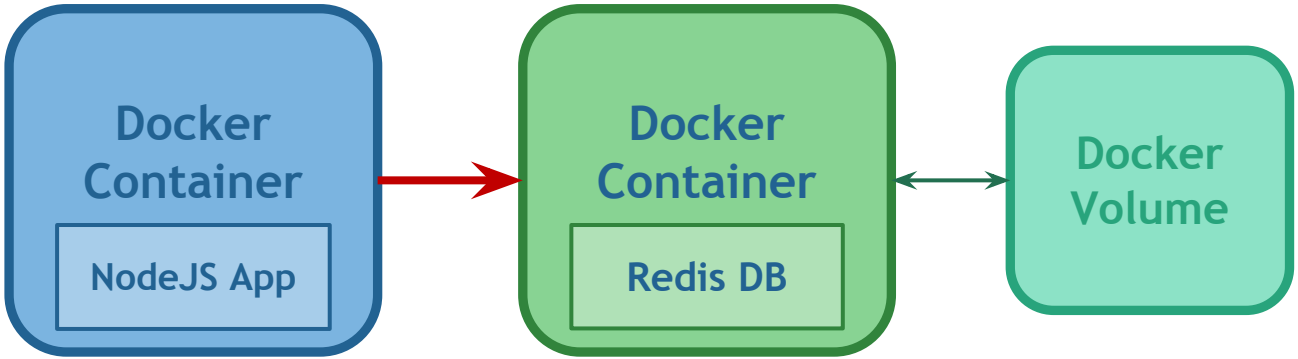


- Then create a volume, set up networking and compose the services together

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)

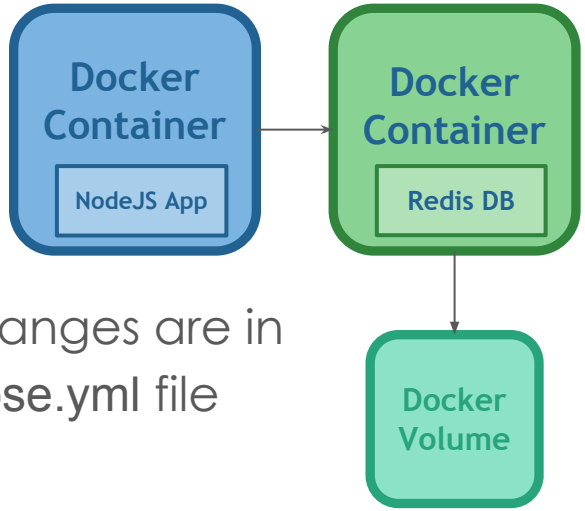


Link to the code: https://github.com/alefais/virtualization-lab-unipi/tree/main/composed-docker-app-volume

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
  ```

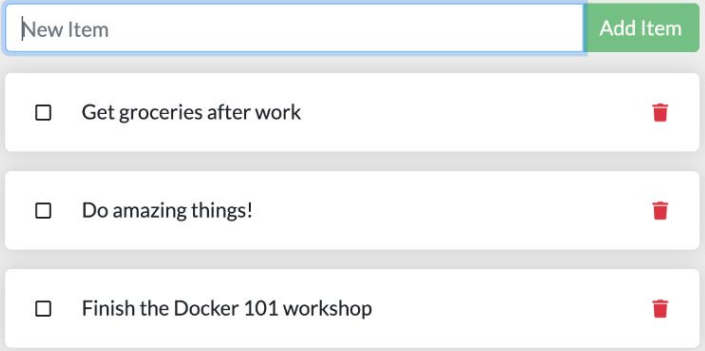
You can check your volume by running **docker volume ls**

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)



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

Cluster mode:
Docker and Kubernetes

Working on clusters with an orchestrator

- Many services cooperating together to implement a single application
  - Easy to scale and deploy single services (app components)
  - Model typically referred to as **microservice-based architecture**
- Orchestrators help managing these components running in containers
  - Run applications in a reliable way
  - Take advantage of existing Docker workloads and run them at scale

Kubernetes

Kubernetes: architectural components

- **Main idea**
  - deploy containerized applications to a cluster without tying them specifically to individual machines

- Distribution and scheduling of application containers is automated across a cluster in a more efficient way

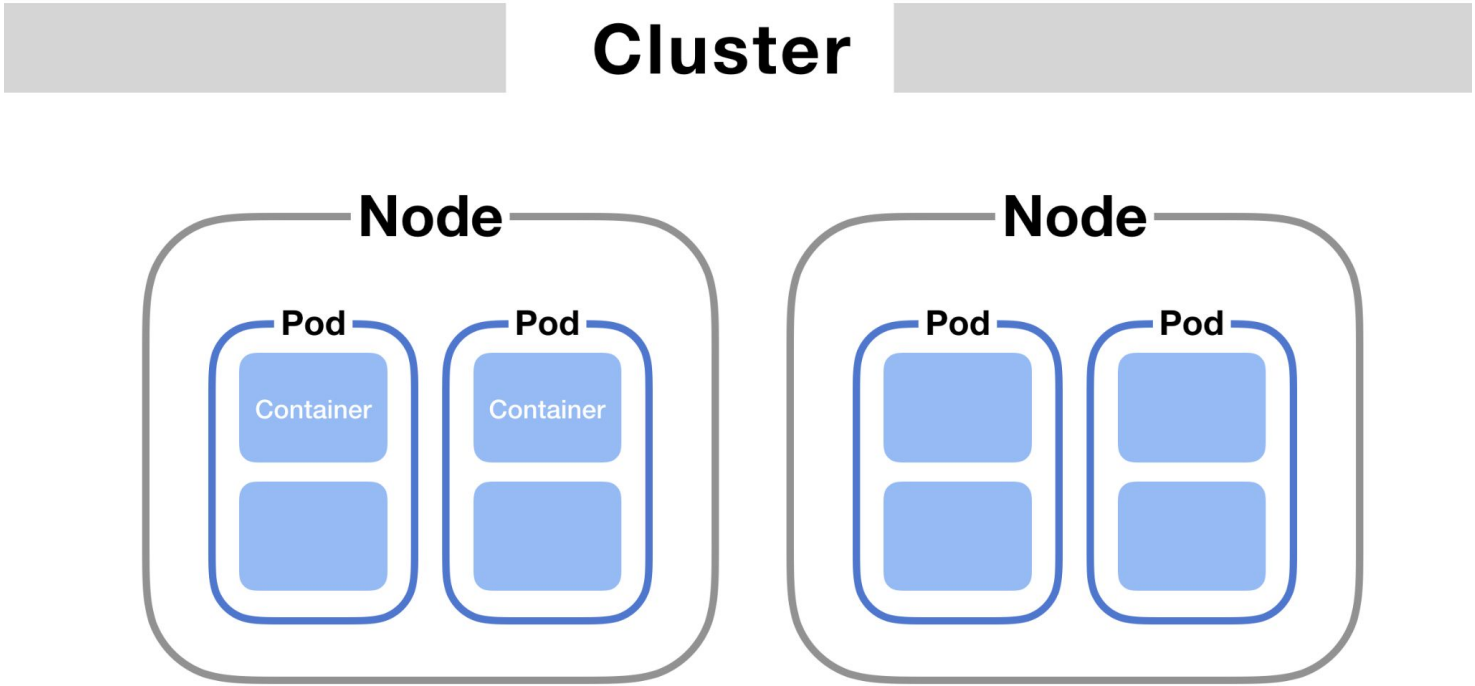
Kubernetes: architectural components

- **Kubernetes cluster service (control plane)**
  - Coordinate the cluster and expose an API
    - accept a **yaml** configuration file describing the desired app management on the infrastructure
  - Deploy app configuration on the infrastructure
    - check that the **yaml** configuration is running correctly at any point in time on the workers
- **Workers (nodes: VMs or physical machines)**
  - Basically container hosts (they run applications)
  - Communicate with the cluster service through the API

Kubernetes: architectural components

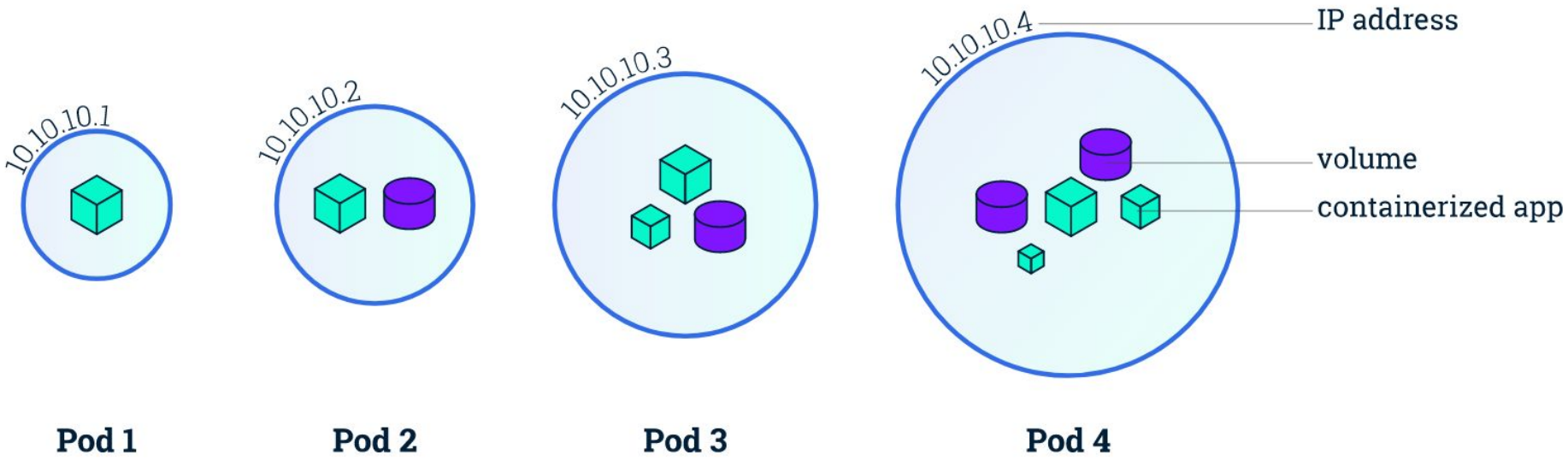
- **yaml** file contains configuration information useful for the application deployment on the infrastructure
- **Pod**
  - Run on the nodes/workers
    - Scheduled by the **master** across the nodes in the cluster
  - Main entity and smallest unit of deployment
    - can run one or more containers (from images)
  - **Replicas**: how many of these pods I want to run?

Kubernetes: architecture overview



https://matthewpalmer.net/kubernetes-app-developer/articles/kubernetes-networking-guide-beginners.html

Kubernetes: architecture overview

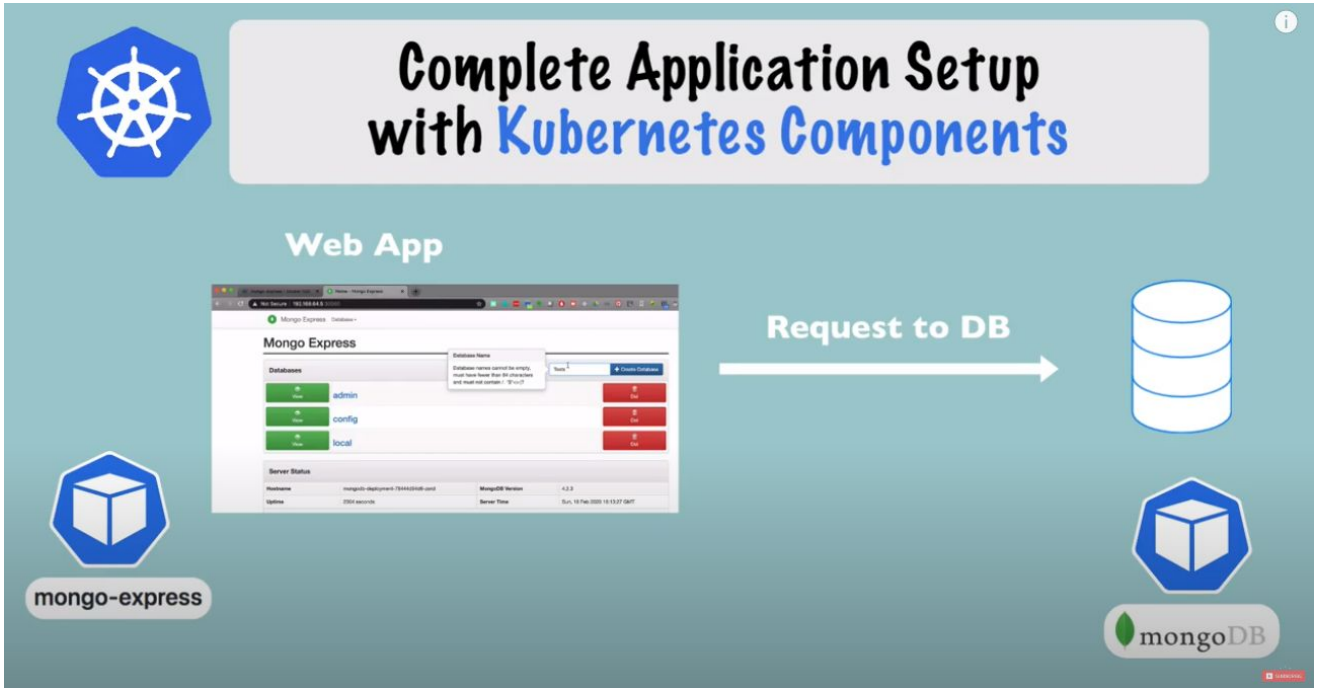


https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/

Exercise

A complete application deployment using Kubernetes

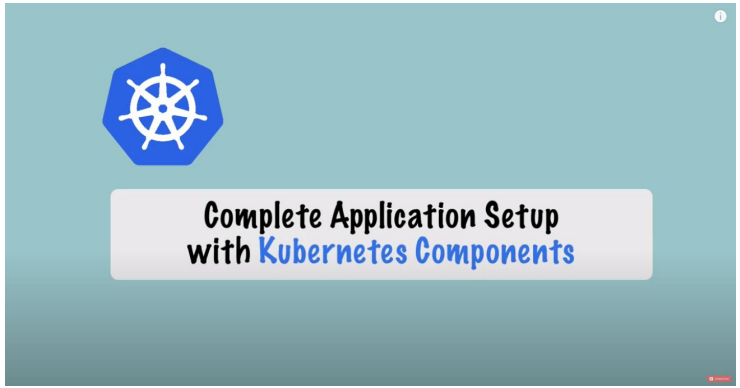
Hands-on with Kubernetes



Link to the video lesson: https://www.youtube.com/watch?v=EQNO_kM96Mo&t=555s

Hands-on with Kubernetes

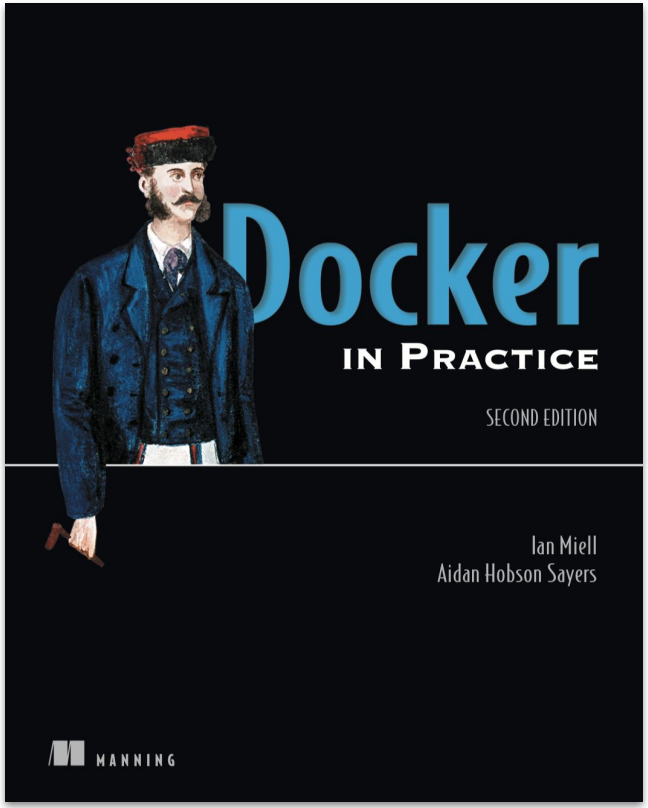
0:25 - Overview diagram of Kubernetes components that will be created (deployment pods, services, …)
1:42 - Browser Request Flow through all the Kubernetes components
2:17 - MongoDB deployment
6:22 - Secret
12:34 - Internal service for MongoDB
17:09 - MongoExpress deployment
19:53 - ConfigMap
24:00 - MongoExpress external service
29:27 - Setup finished - review



Link to the video lesson: https://www.youtube.com/watch?v=EQNO_kM96Mo&t=555s

Useful references

Useful reference book



Link

Link

Useful references on Docker and Dockerfiles

- 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/

Useful 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/

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/

Useful references on Kubernetes

- Starting with clusters management using Kubernetes
  - https://docs.docker.com/get-started/kube-deploy/
- Learn Kubernetes basics: using Minikube to create a Cluster
  - https://kubernetes.io/docs/tutorials/kubernetes-basics/create-cluster/cluster-intro/
- Learn Kubernetes basics: small tutorial
  - https://kubernetes.io/docs/tutorials/hello-minikube/
- Learn Kubernetes basics: Pods and Nodes
  - https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/
- Kubernetes guide for beginners
  - https://matthewpalmer.net/kubernetes-app-developer/articles/kubernetes-networking-guide-beginners.html