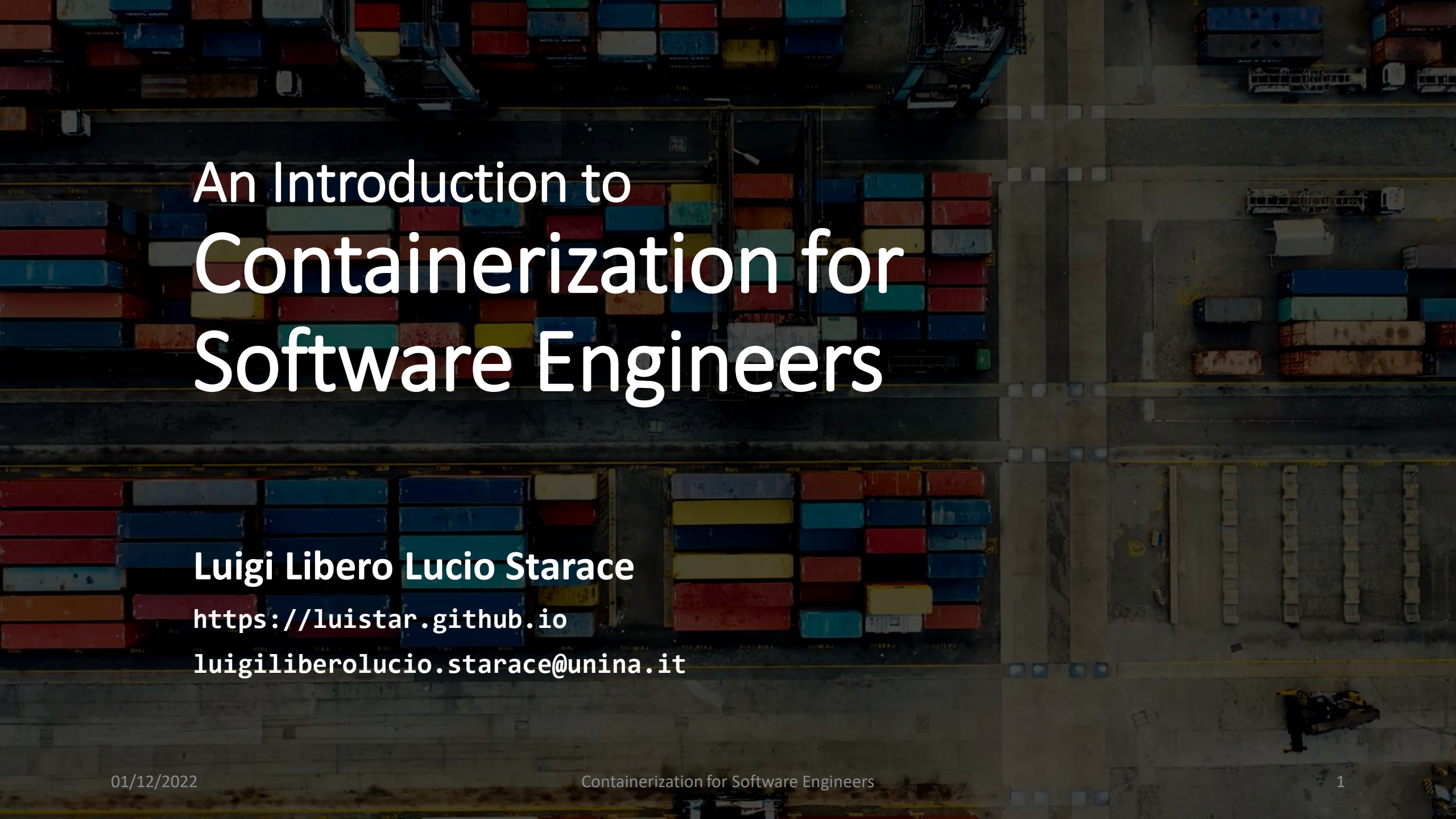


An Introduction to Containerization for Software Engineers

Luigi Libero Lucio Starace

https://luistar.github.io

luigiliberolucio.starace@unina.it

Trends in the Software Industry

- The software industry has changed

Before:

- Monolithic software
- Long development cycles
- Single target environment
- Slowly scale up

Now:

- Decoupled services
- Fast, iterative improvements
- Multiple target environments
- Must quickly scale up

Deployments are becoming more complex

- Each independent service/components uses many stacks
  - Languages
  - Frameworks
  - Databases
- Many different targets
  - Development enviroments
  - Pre-production, QA, staging…
  - Production: On premises, public cloud, hybrid solutions

The Challenge

Multiplicity of stacks

Preprocessing
Python 3.10, ffmpeg, libopencv, nodejs, vips, imagemagick

Analytics
OpenJDK 17, Elasticsearch, Grafana

Web App
Python 3.8, Django, psycopg, libcurl

App DB
PostgreSQL, Postgis, timescale

Queue
Redis, redis-sentinel

API Endpoint
Python 2.7, Flask, psycopg, gensim

Do services interact appropriately?

Multiplicity of Hardware Environments


Dev laptop


Production server


QA server

Public cloud


Disaster recovery

Is it possible to smoothly migrate between them?

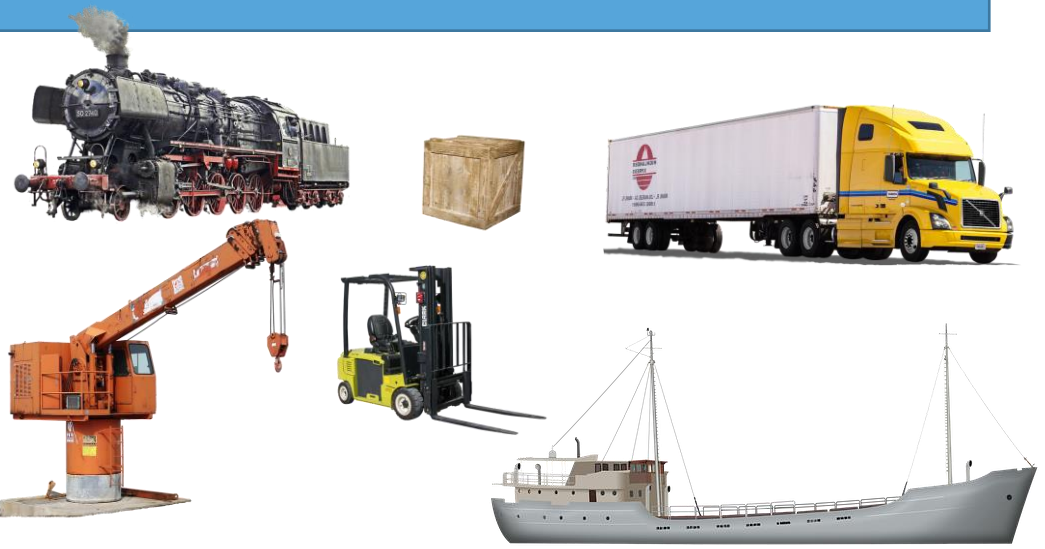
The «Matrix from Hell»

		Environments				
		Dev Laptop	Production Server	Distaster Recovery	Public Cloud	QA Server
Stacks	Web App	?	?	?	?	?
	API Endpoint	?	?	?	?	?
	Analytics	?	?	?	?	?
	App DB	?	?	?	?	?
	Queue	?	?	?	?	?
	Preprocessing	?	?	?	?	?

Cargo Transportation before 1960s

Multiplicity of goods

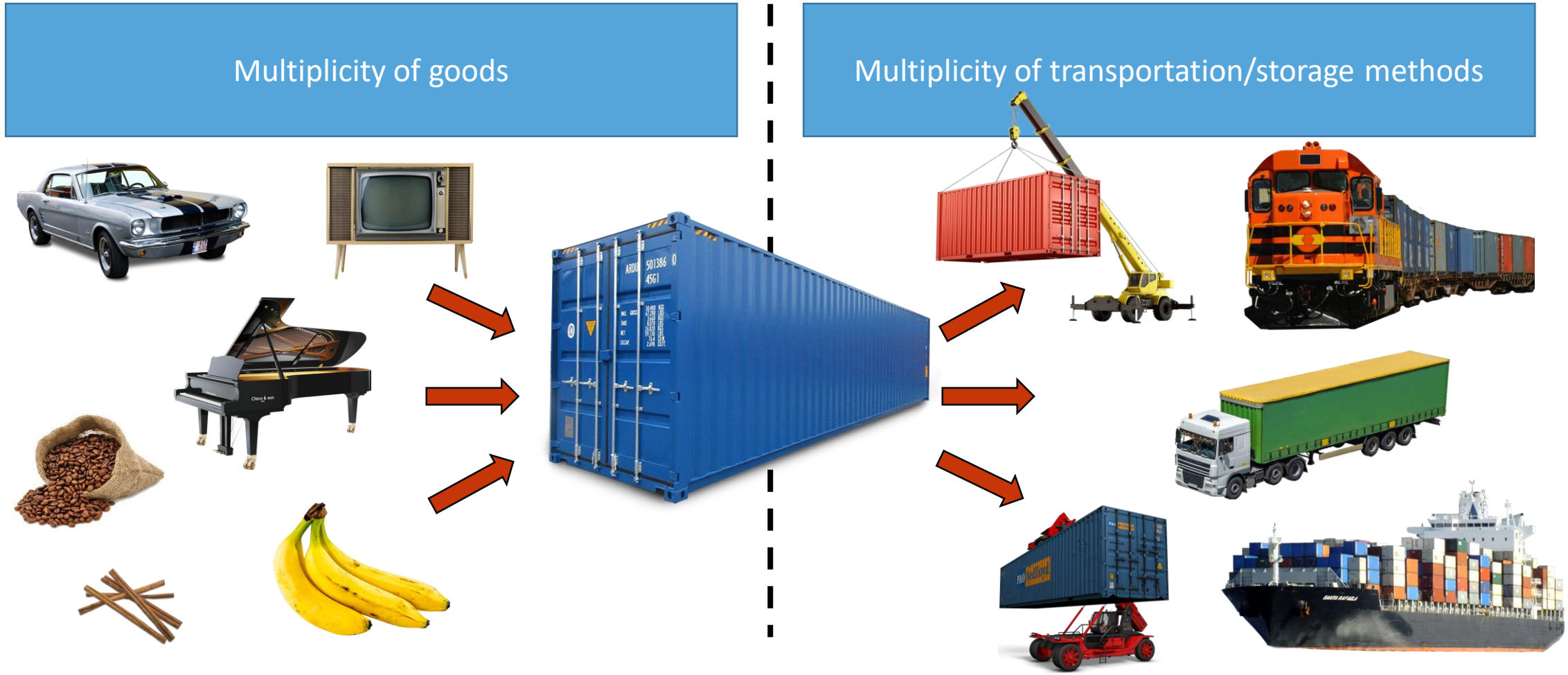
Multiplicity of transportation/storage methods



Do goods interact with each others?

Is it possible to smoothly change transport mode?

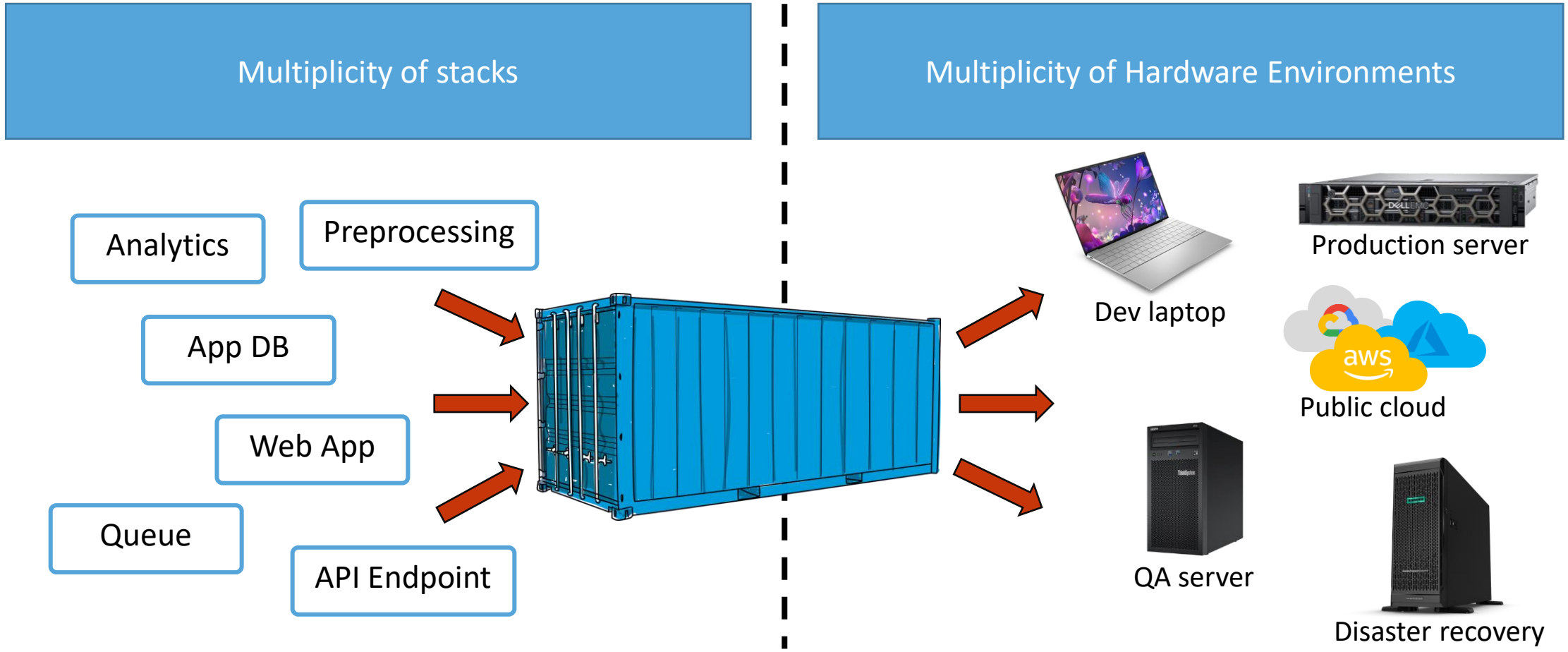
Solution: Containers



Containers

- **Standardized** (all have the same size)
- Can be loaded with virtually any good
- Prepared by the people in charge of shipping
  - Make sure that no unwanted interactions happen **inside**
- Sealed until final delivery
- During transport, all containers are the same
  - Easy to load, unload, stack, etc..

Containers for Code



Why should we bother?

Developers

- Only need to care about what's **inside** the container
- Simplify setup of dev env.
- No worries about library/dependencies conflicts
- **Build once, run anywhere***

***anywhere with the same architecture and a modern Linux kernel**

Operations

- Only need to care about what's **outside** the container
- Every container can be managed the same way
- Simplify lifecycle management
- **Configure once, run anything****

****anything built based on the same architecture and kernel**

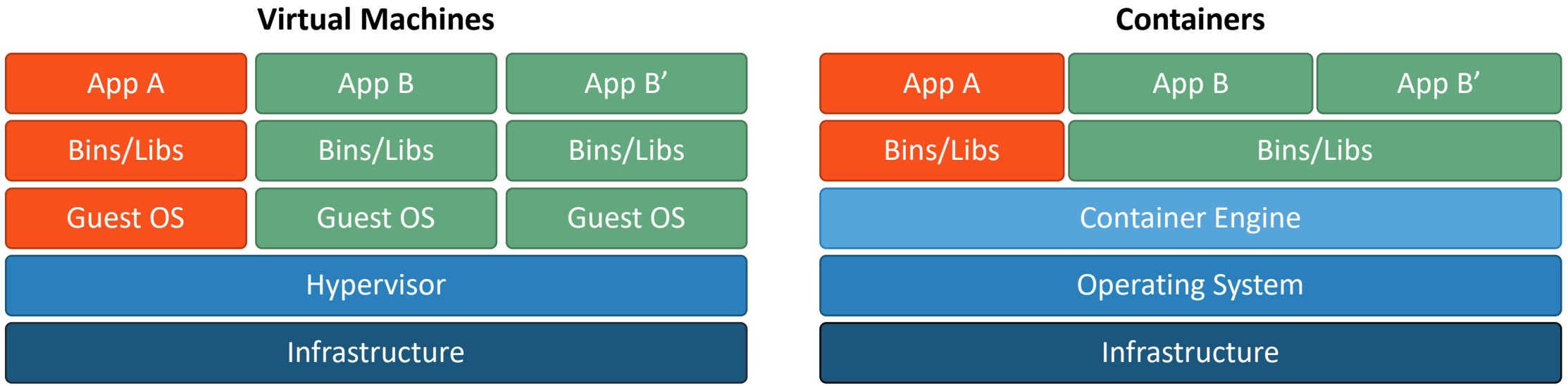
The «Matrix from Hell», solved

		Environments				
		Dev Laptop	Production Server	Distaster Recovery	Public Cloud	QA Server
Stacks	Web App					
	API Endpoint					
	Analytics					
	App DB					
	Queue					
	Preprocessing					

Virtual Machines vs Containers

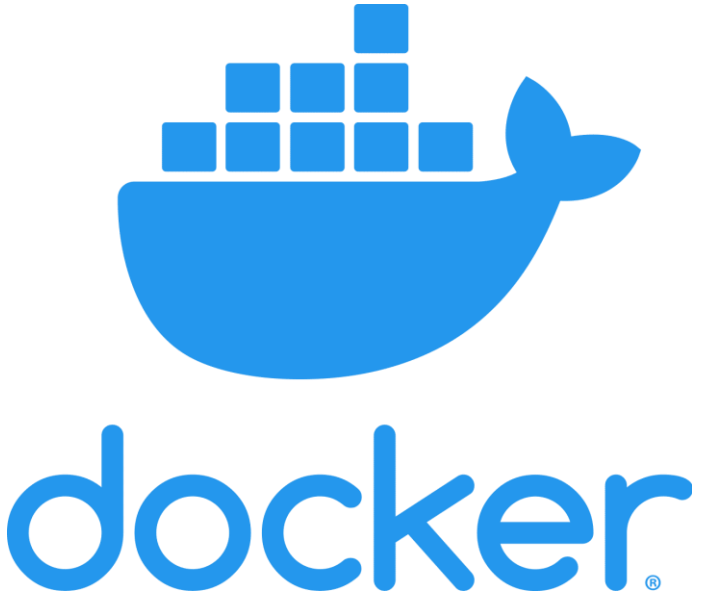
Don’t virtual machines solve the same problems as containers?

- Yes, but they’re **not as lightweight**
- Containers allow for **significantly faster** deployment, restart, etc…

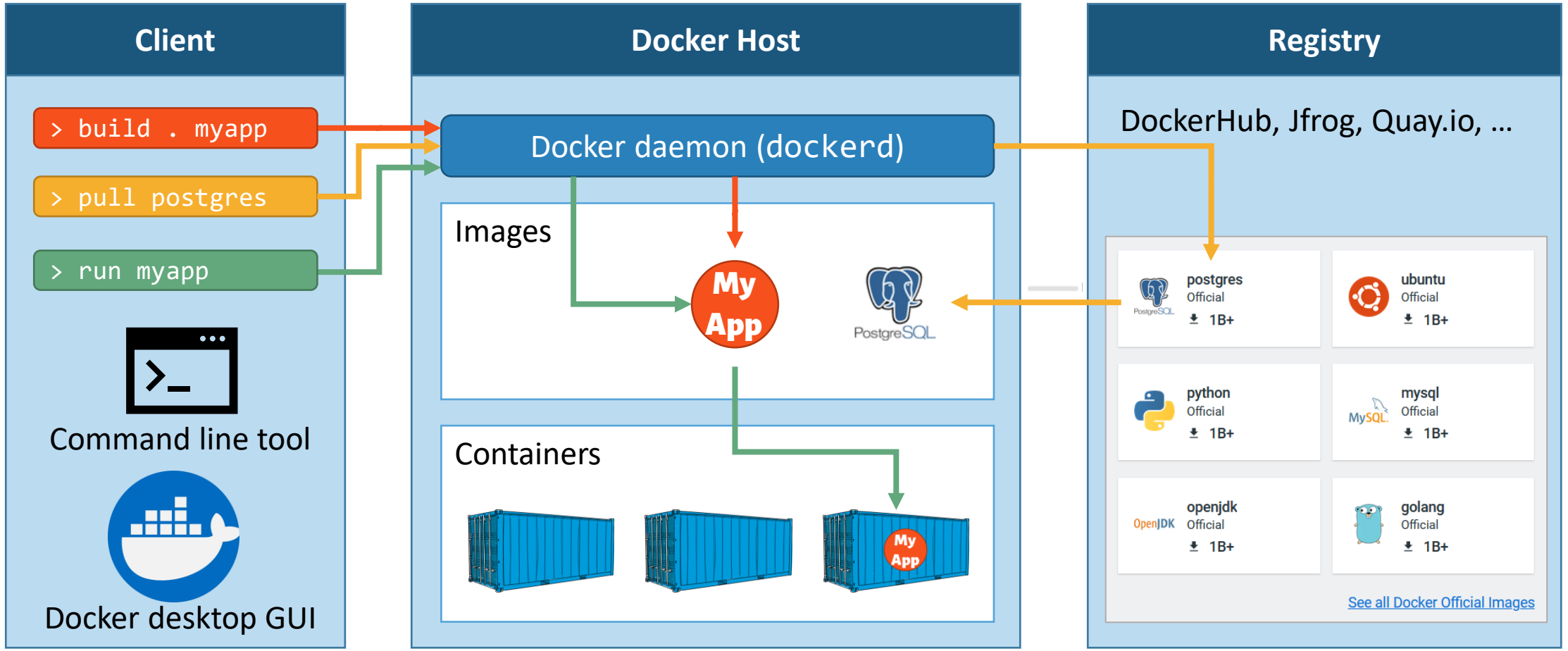


Docker: The Container Engine

- https://www.docker.com/
- Project started in 2013
- Used by more than **13 million devs**
- More than **9 million «dockerized» applications**
- **De facto standard** for containerizing software
- Alternatives exist:
  - LXD, BuildKit, Buildah, Podman, …

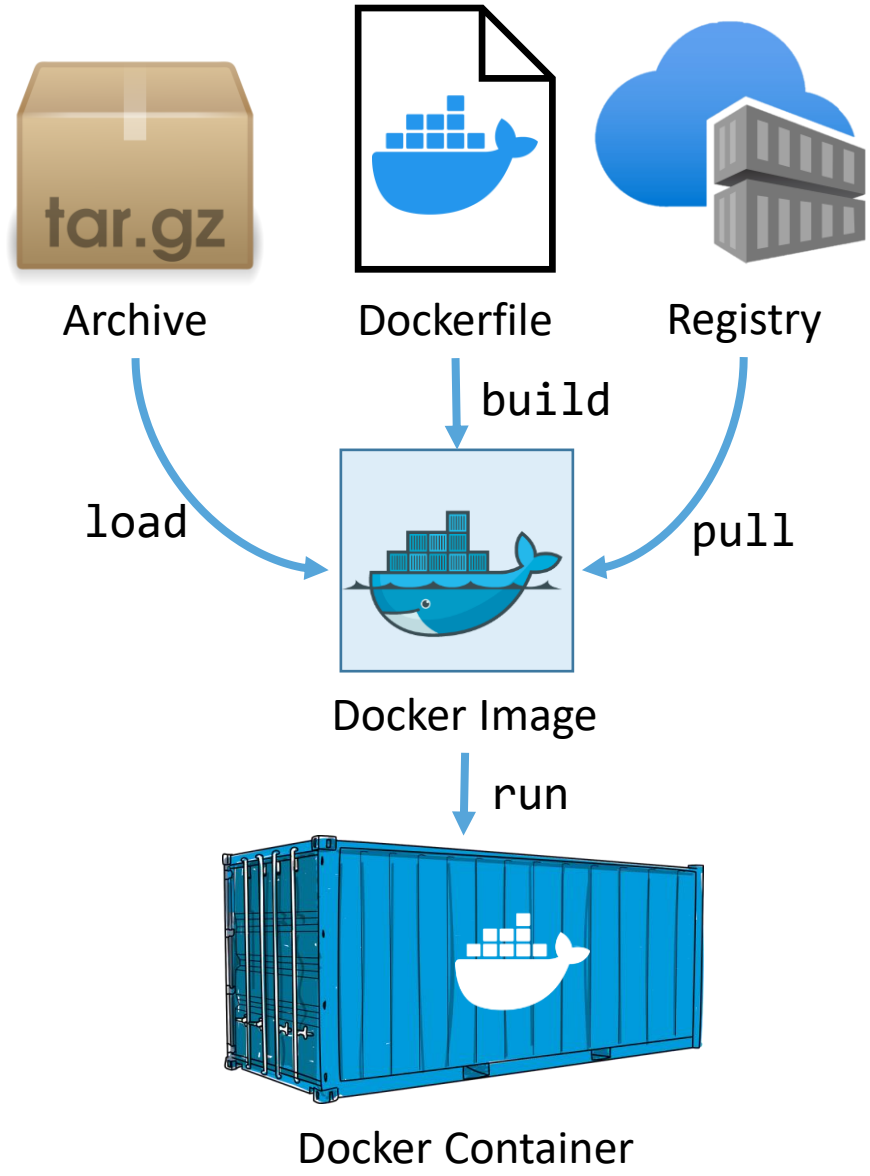


Docker Architecture



Docker Images

- Portable, read-only templates
- Contain all the instruction to create a container
- Can be **loaded** from a tar archive file
- Can be **downloaded** from a registry
- Can be built by **extending** an **existing image** with a list of instruction specified in a text file (**Dockerfile**)



Getting to know Docker

Hands on session with Docker basics

Running our first container: pulling the image

```
$> docker pull ubuntu:20.04
20.04: Pulling from library/ubuntu
675920708c8b: Pull complete
Digest: sha256:35ab2bf57814e9ff49e365efd5a5935b6915eede5c7f8581e9e1b85e0eecbe16
Status: Downloaded newer image for ubuntu:20.04
docker.io/library/ubuntu:20.04

$> docker image list
REPOSITORY          TAG                 IMAGE ID       CREATED         SIZE
ubuntu              20.04               a0ce5a295b63   3 weeks ago     72.8MB
```

Running our first container

```
$> docker run -it --name my-first-container ubuntu:20.04
root@f2ce5afe0cba:/# ls
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root
run  sbin  srv  sys  tmp  usr  var
root@f2ce5afe0cba:/# apt update -qq && apt install -y cowsay fortune
root@f2ce5afe0cba:/# /usr/games/fortune | /usr/games/cowsay
 _____________________________________
/ Never look up when dragons fly \
\ overhead.                         /
 -------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
root@f2ce5afe0cba:/# exit
```

Running our first container: start and attach

```
$> docker container list --all
CONTAINER ID   IMAGE         COMMAND  CREATED      STATUS      NAMES
f2ce5afe0cba   ubuntu:20.04  "bash"   11 mins ago  Exited (0)  my-first-container

$> docker start my-first-container

$> docker container list --all
CONTAINER ID   IMAGE         COMMAND  CREATED      STATUS      NAMES
f2ce5afe0cba   ubuntu:20.04  "bash"   11 mins ago  Up 10 secs  my-first-container

$> docker attach my-first-container
root@f2ce5afe0cba:/# /usr/games/fortune
Never laugh at live dragons.
                -- Bilbo Baggins [J.R.R. Tolkien, "The Hobbit"]
```

Running our first container: transfer files

```
$> echo "Hello" > file.txt

$> docker cp ./file.txt my-first-container:/home/file.txt

$> docker attach my-first-container
root@b43ea0a68502:/# ls /home/
file.txt
root@b43ea0a68502:/# cat /home/file.txt
"Hello"
root@b43ea0a68502:/# echo "Hello UniNA!" > /home/file.txt
root@b43ea0a68502:/# read escape sequence

$> docker cp my-first-container:/home/file.txt ./file.txt

$> type file.txt
Hello UniNA!
```

Running our first container: detach and kill

- To detach from the interactive terminal, press the hotkeys CTRL+P followed by CTRL+Q

```
$> docker attach my-first-container
root@f2ce5afe0cba:/# /usr/games/fortune
Never laugh at live dragons.
                -- Bilbo Baggins [J.R.R. Tolkien, "The Hobbit"]
root@f2ce5afe0cba:/# read escape sequence

$> docker container list --all
CONTAINER ID   IMAGE          COMMAND   CREATED       STATUS       NAMES
f2ce5afe0cba   ubuntu:20.04   "bash"    11 mins ago   Up 59 secs   my-first-container

$> docker kill my-first-container
my-first-container
```

Running our first container: exec and rm

```
$> docker start my-first-container

$> docker exec -ti my-first-container bash -c /usr/games/fortune
You never hesitate to tackle the most difficult problems.

$> docker container list --all
CONTAINER ID   IMAGE          COMMAND   CREATED       STATUS       NAMES
f2ce5afe0cba   ubuntu:20.04   "bash"    11 mins ago   Up 59 secs   my-first-container

$> docker kill my-first-container

$> docker rm my-first-container

$> docker container list --all
CONTAINER ID   IMAGE          COMMAND   CREATED       STATUS       NAMES
```

Building our own first Image: Dockerfile

- A Dockerfile is a set of commands to assemble an Image
- Start **FROM** a base image
- **RUN** commands, **COPY** files, **EXPOSE** ports, set **ENVIRONMENT** vars, …
- Dockerfile reference

```
# Start from the ubuntu:20.04 base image
FROM ubuntu:20.04
# Update the list of packages and install fortune and cowsay
RUN apt update -qq && apt install -y -q fortune cowsay
# Copy file.txt from the Dockerfile dir. to /home/file.txt in the Container
COPY ./file.txt /home/file.txt
# Default entrypoint for executing containers
CMD bash
```

Building our own first Image

```
$> cd ubuntu-fortune-cowsay

$> dir /b
Dockerfile
file.txt

$> docker build -t "ubuntu-fortune-cowsay" .

[+] Building 25.8s (8/8) FINISHED
 => [internal] load build definition from Dockerfile                0.0s
 => [internal] load metadata for docker.io/library/ubuntu:20.04     0.0s
 => CACHED [1/3] FROM docker.io/library/ubuntu:20.04                0.0s
 => [2/3] RUN apt update -qq && apt install -y -q fortune cowsay   24.9s
 => [3/3] COPY ./file.txt /home/file.txt                            0.1s
 => => writing image ha256:6e05a97a366b87c98a2[...]26678046c        0.0s
 => => naming to docker.io/library/ubuntu-fortune-cowsay            0.0s
```

Building our own first Image

```
$> docker image list --all
REPOSITORY            TAG              IMAGE ID       CREATED          SIZE
ubuntu-fortune-cowsay latest           6e05a97a366b   23 seconds ago   159MB
ubuntu                20.04            a0ce5a295b63   3 weeks ago      72.8MB
$> docker run -it --name my-ubuntu-container ubuntu-fortune-cowsay
root@a87b47206c9a:/# /usr/games/fortune | usr/games/cowsay
 ________________
< You look tired. >
 ----------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
root@a87b47206c9a:/# cat /home/file.txt
Hello UniNA!
```

Using Docker as a Developer

Let’s build a container-native, microservice-based application

Container-native software

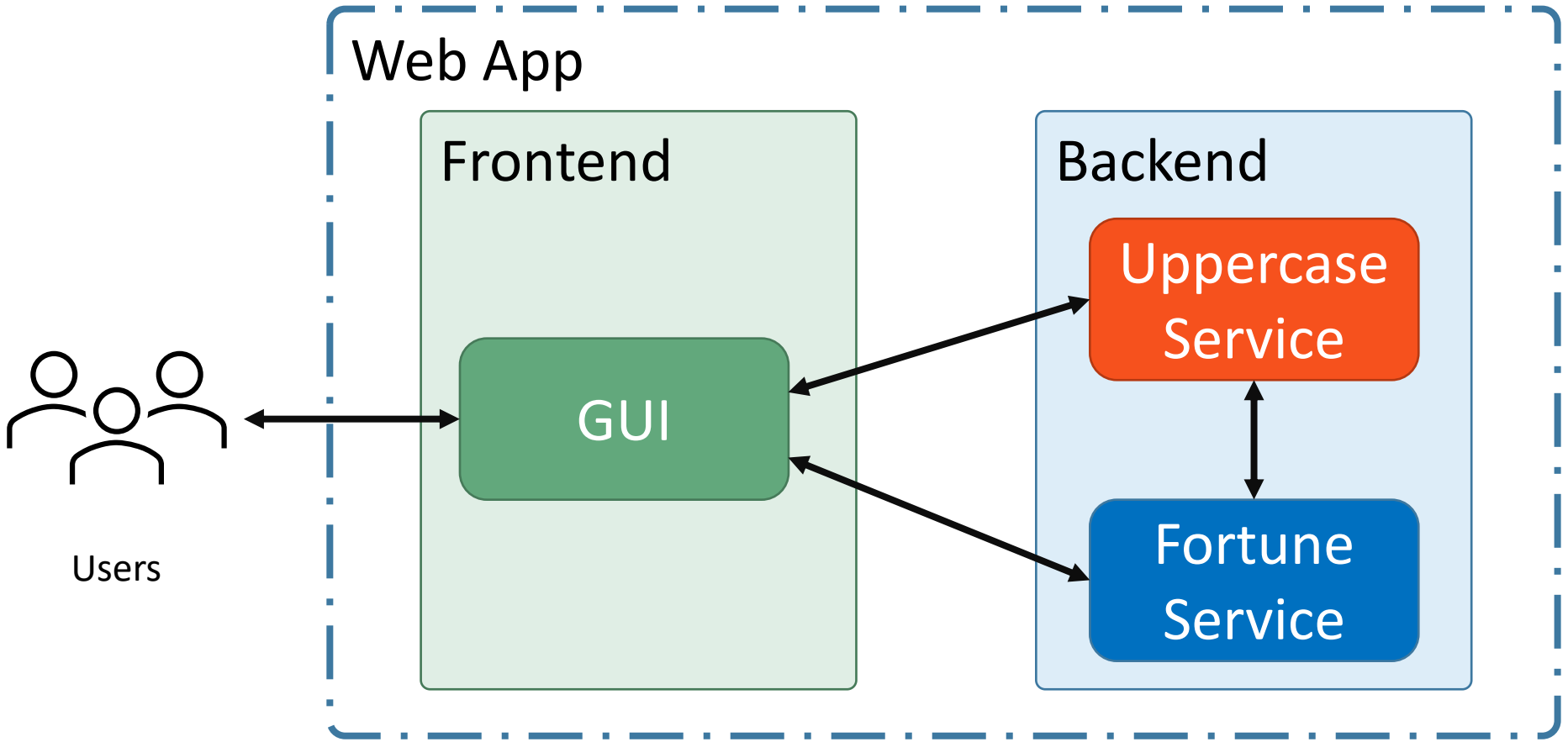
- Containers are first-class units of infrastructure (not servers or VMs)
- Purposefully designed for containers from the beginning

A Container-native Web App

We want to develop a web app that offers three main features:

- Users can get a fortune message
- Users can generate uppercase versions of any string they like
- Users can get an uppercase version of their fortune message

A Container-native Web App: Architecture



A Container-native Web App: Technologies

- The components of our app need to communicate
  - We'll have them communicate using a standard web procol (HTTP)
- Uppercase service
  - Python and Flask
- Fortune service
  - Nodejs and Express
- GUI
  - Single web page with a bit of Javascript

Fortune service

- Offers one feature: returns a carefully-chosen fortune message for the current user
- Users access this feature by sending a GET HTTP request to the /fortune endpoint (i.e.: http:/{fortune-service-location}/fortune)
- The service returns a JSON containing the fortune message:

```
{
  fortune: "Terrible fortune! Be careful! 😰"
}
```

Fortune service: Implementation (snippet)

```
const express = require('express')
const app = express()
const port = 3000

app.get('/fortune', (req, res) => {
  const fortunes = [
    "Great fortune! 😋",
    "Meh, average fortune 😐",
    "Terrible fortune! Be careful! 😭"
  ]
  var fortune = fortunes[Math.floor(Math.random()*fortunes.length)];
  res.send({
    'fortune': fortune
  })
})
```

Uppercase service

Offers two features:

- Given a string, it makes it uppercase
- It returns an uppercase fortune message for the user

Uppercase service: Making a string uppercase

- Users access this feature by sending a POST HTTP request to the /uppercase endpoint (i.e.: http:/{uppercase-service-location}/uppercase)
- The request must contain in its body a JSON like this:

```
{
  message: "hello world!"
}
```

- The server returns a JSON like this:

```
{
  original: "hello world!",
  uppercase: "HELLO WORLD!"
}
```

Fortune service: Uppercase Implementation (snippet)

```
@app.route('/uppercase', methods=['POST'])
def uppercase():
    content_type = request.headers.get('Content-Type')
    print(content_type)
    if(content_type == 'application/json'):
        data = request.json    # get json data in request body
        message = data["message"]  # get message field
        print(data)
        return jsonify({
            'original' : message,'uppercase': message.upper()
        })
    else:
        return 'Content-Type not supported!'
```

Uppercase service: Uppercase fortune

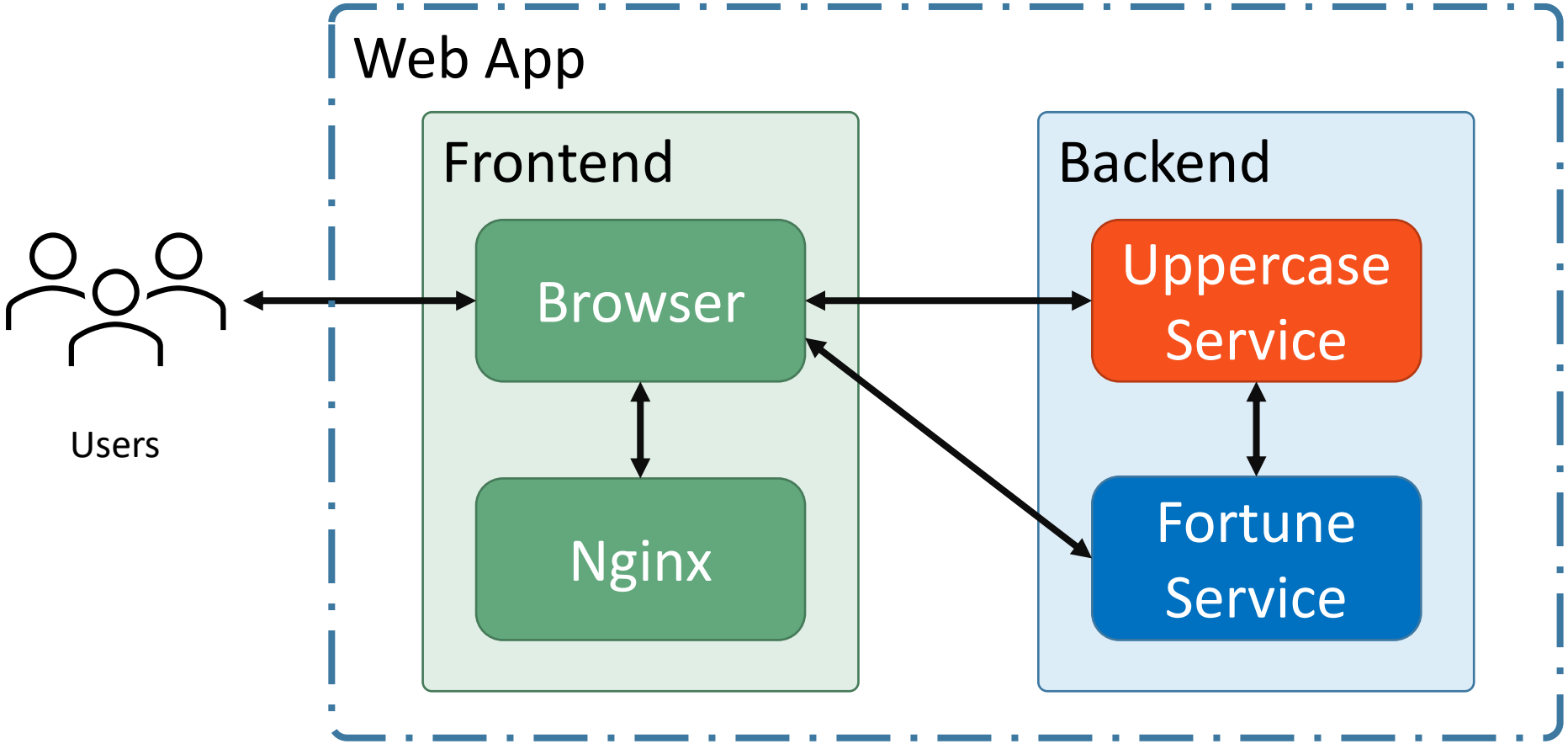
- Users access this feature by sending a GET HTTP request to the /uppercase-fortune endpoint (i.e.: http:/{uppercase-service-location}/uppercase)
- The service gets a fortune using the Fortune service, and makes it uppercase
- The service returns a JSON containing the uppercase fortune

```
{
  fortune: "TERRIBLE FORTUNE! BE CAREFUL! 😰"
}
```

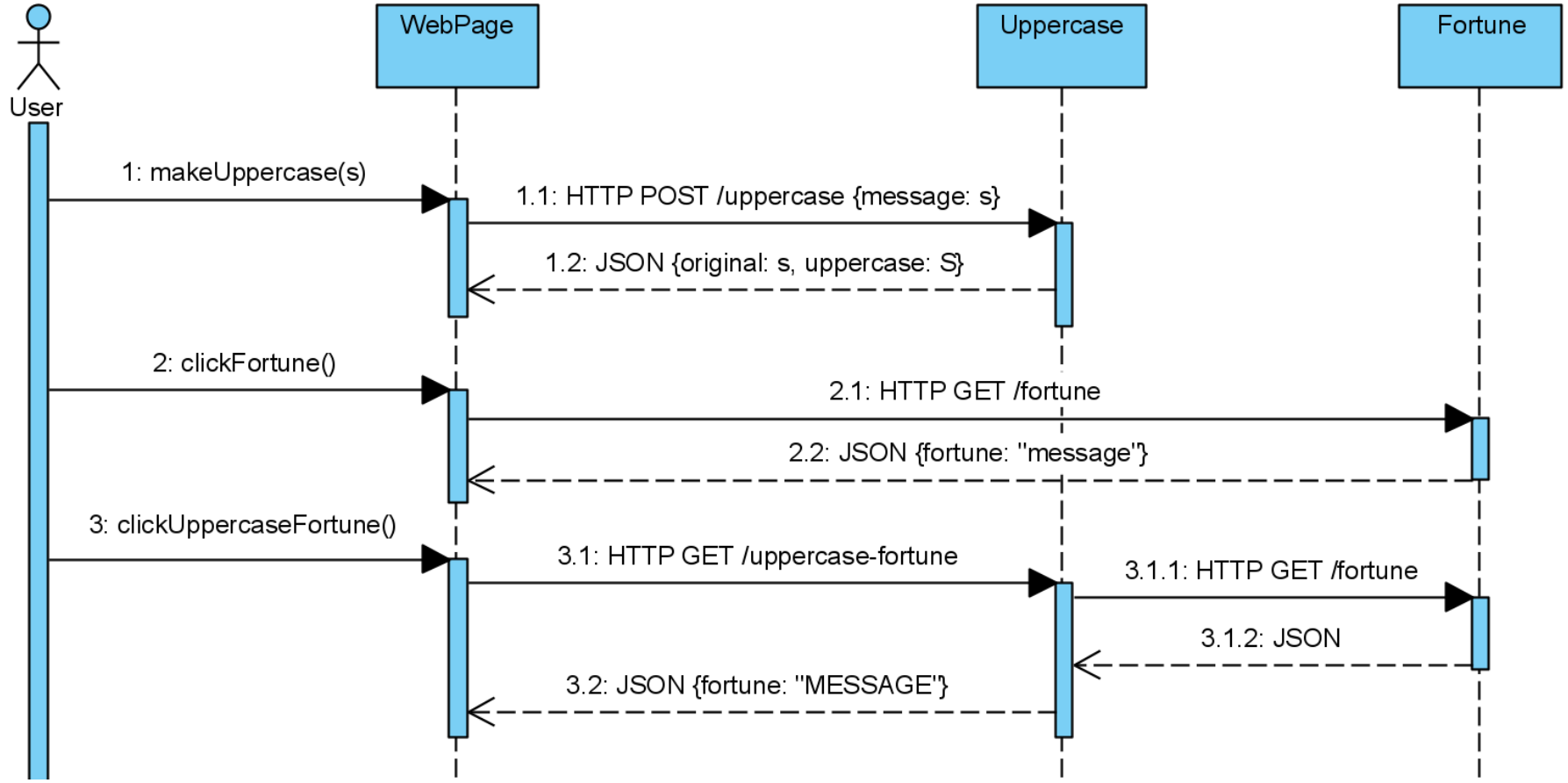
Fortune service: Uppercase Fortune Implementation (snippet)

```
@app.route('/uppercase-fortune')
def uppercase_fortune():
    url = f"{fortune_baseurl}/fortune"
    req = requests.get(url)
    data = req.json()
    return jsonify({
        'fortune' : data['fortune'].upper()
    })
```

A Container-native Web App



Sequence diagram



Preparing Docker images for our services

```
# Fortune service

# Start from base Nodejs 14 image
FROM node:14
# Create app directory and set it as base work directory
WORKDIR /home/app
# Copy package.json file (contains list of dependencies) to work directory
COPY package*.json .
# Copy app.js file to work directory
COPY app.js .
# Install dependencies
RUN npm install
# Expose port 3000 of the container to the Docker host
EXPOSE 3000
# Start the app
CMD node app.js
```

Preparing Docker images for our services

```
# Uppercase service

# Start from base Python 3.8 image
FROM python:3.8
# Copy requirements.txt file with dependencies
COPY ./requirements.txt /home/requirements.txt
# Install dependencies with pip
RUN pip3 install -r /home/requirements.txt
# Copy app files
COPY app.py /home/app.py
# Expose port 5000 of the container to the docker host
EXPOSE 5000
# Start the Flask app
CMD flask --app /home/app.py run --host 0.0.0.0
```

Preparing Docker images for our services

```
# Web server hosting our GUI (web page)

# Start from nginx image
FROM nginx:latest
# COPY our web page to the default document root of the web server
COPY ./website /usr/share/nginx/html
```

Container Orchestration with docker-compose

```yaml
version: "3.9"
services:

  frontend:
    build: ./frontend/
    container_name: "uppercase-fortune-frontend"
    depends_on:
      - fortune-service
      - uppercase-service
    ports:
      - "8080:80"

  fortune-service:
    build: ./fortune/
    container_name: "fortune-service-container"
    ports:
      - "3000:3000"
```

```yaml
  uppercase-service:
    build: ./uppercase/
    container_name: "uppercase-service-container"
    environment:
      FORTUNE_URL: http://fortune-service:3000
    depends_on:
      - fortune-service
    ports:
      - "5000:5000"
```

Running our docker-compose project

```
$> docker-compose -p "uppercase-fortune-app" up -d
[+] Building 3.6s (24/24) FINISHED
...
[+] Running 4/4
- Network uppercase-fortune-app_default  Created   0.7s
- Container fortune-service-container    Started   1.7s
- Container uppercase-service-container  Started   2.9s
- Container uppercase-fortune-frontend   Started   4.1s
docker run –it --name my-first-container ubuntu:20.04
```