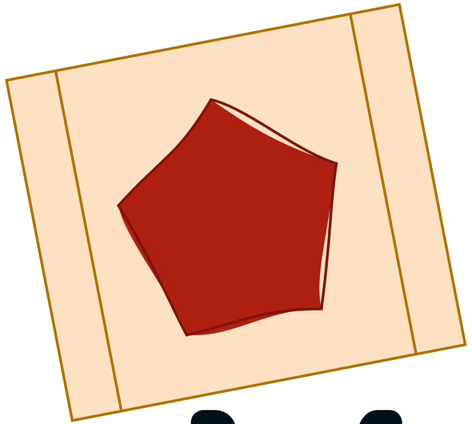
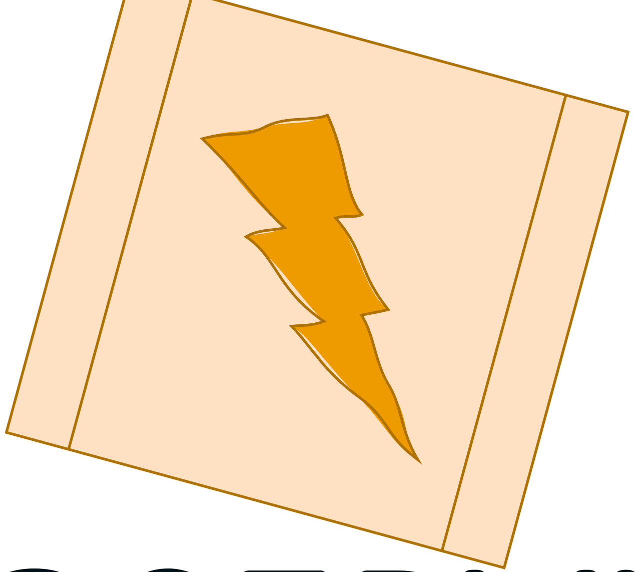
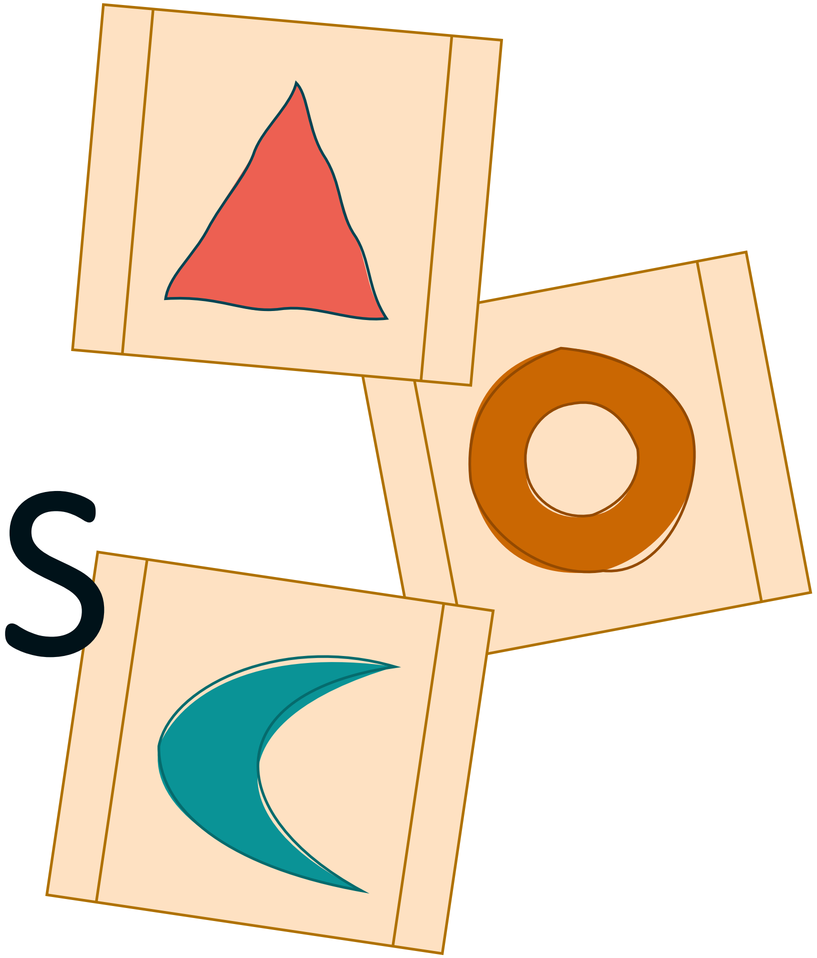






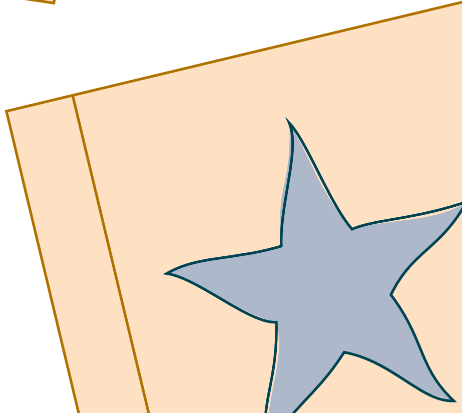
Microservices

Luigi Libero Lucio Starace, Ph.D.

https://luistar.github.io

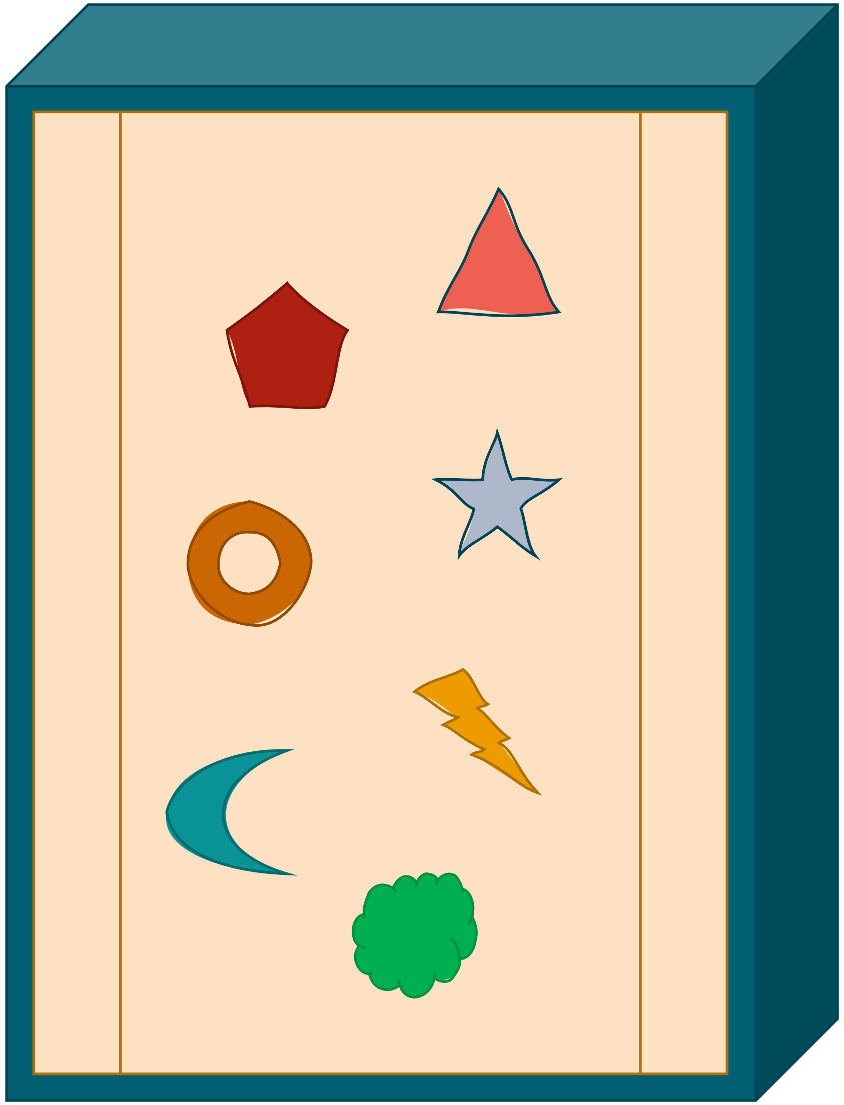
luigiliberolucio.starace@unina.it

May 17, 2023 – Tecnologie Web – Software Project Management & Evolution



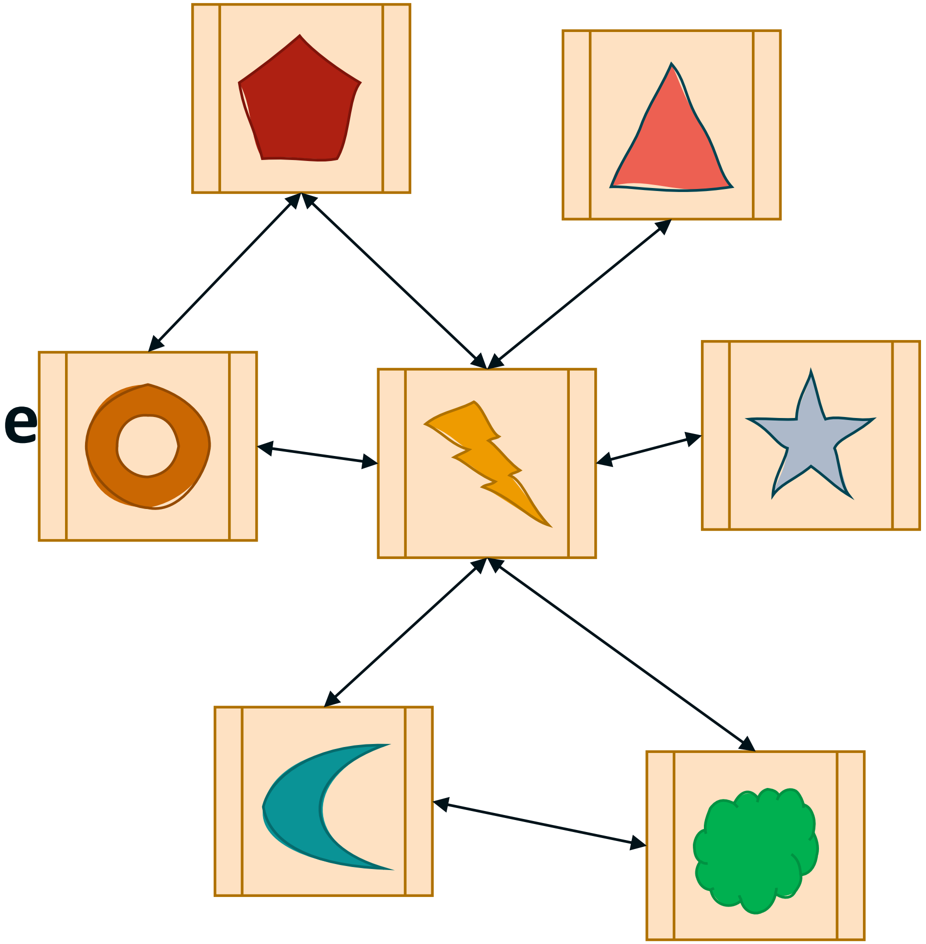
Monolithic Architecture

- Different components/modules tightly packaged in a **single unit**
- Single codebase
- Single, self-contained, deployment unit
- Business concerns are coupled

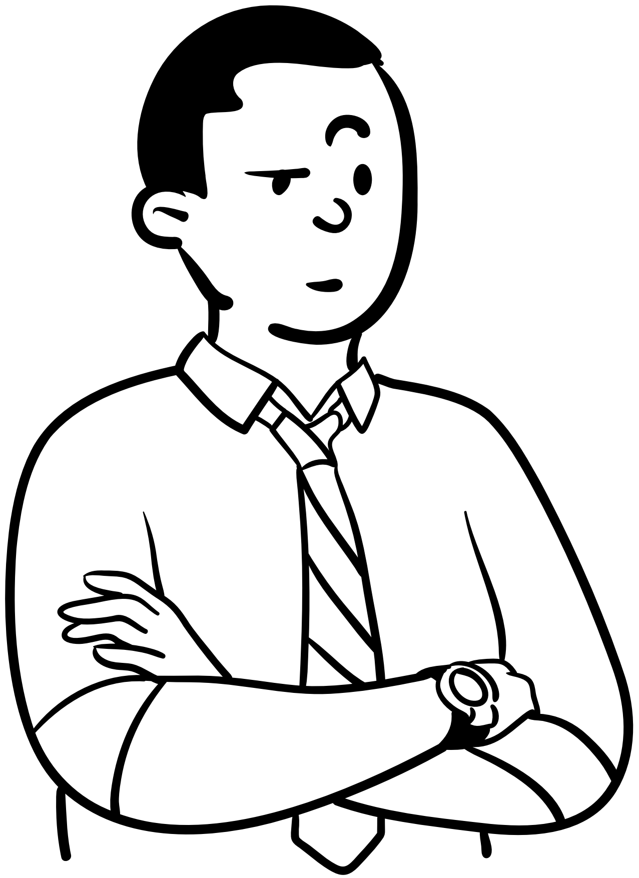


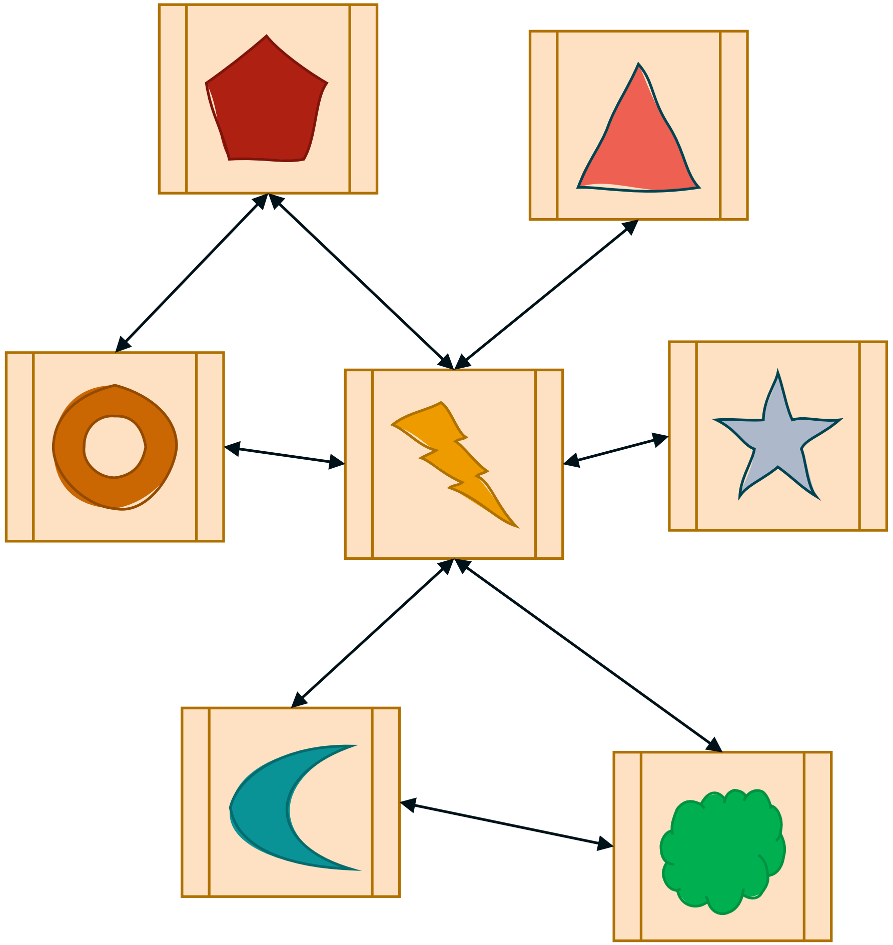
Microservice Architecture

- Software structured as a set of services
- Each service has its own codebase
- Possibly **polyglot**
- Each service is **independently deployable**
- Services organized around business capabilies
- Services are **loosely coupled**

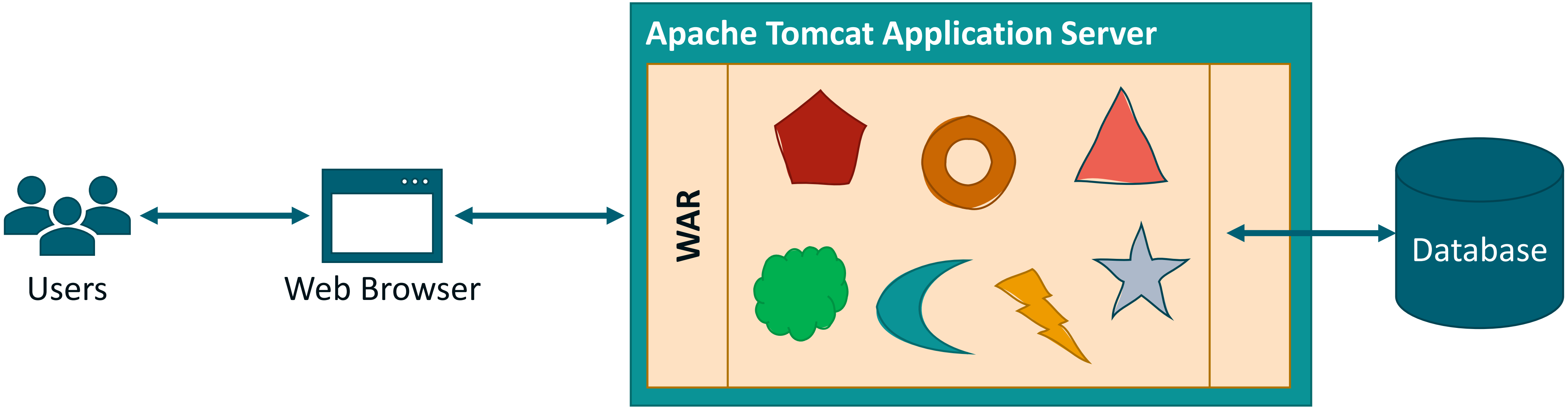


Microservices: Why on hell would I do that?!





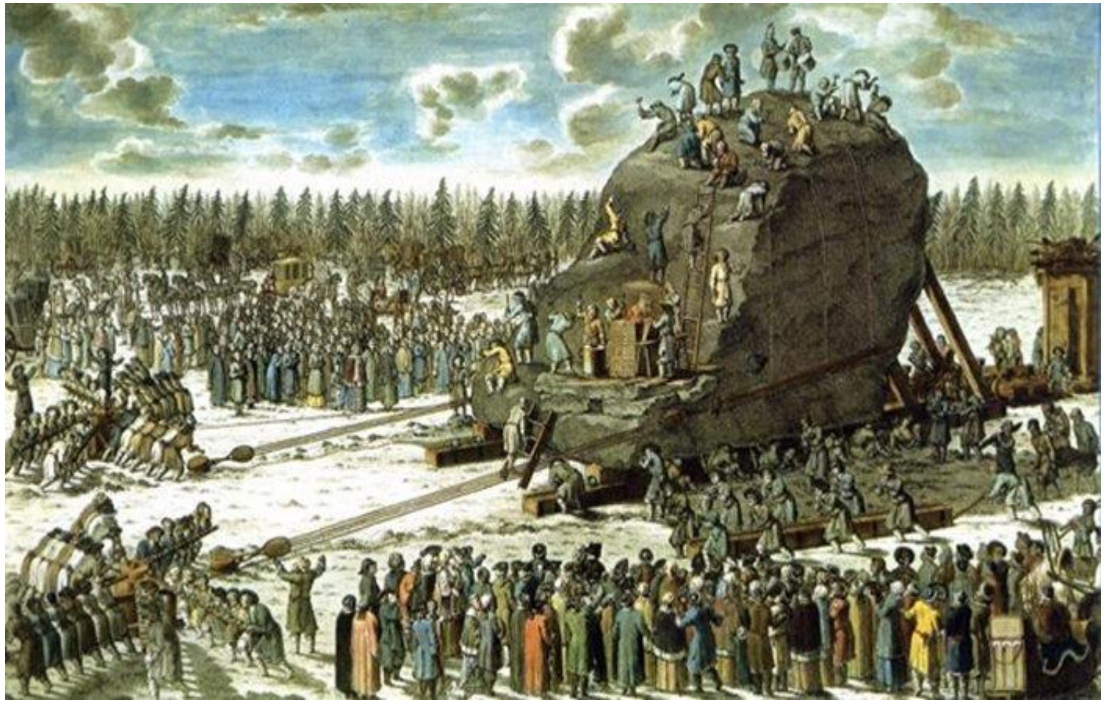
Monolithic Architecture



At the beginning (or for software with low complexity):

- **Easy Deployments:** need to build just one artifact
- **Easy Development:** single codebase, easy data management

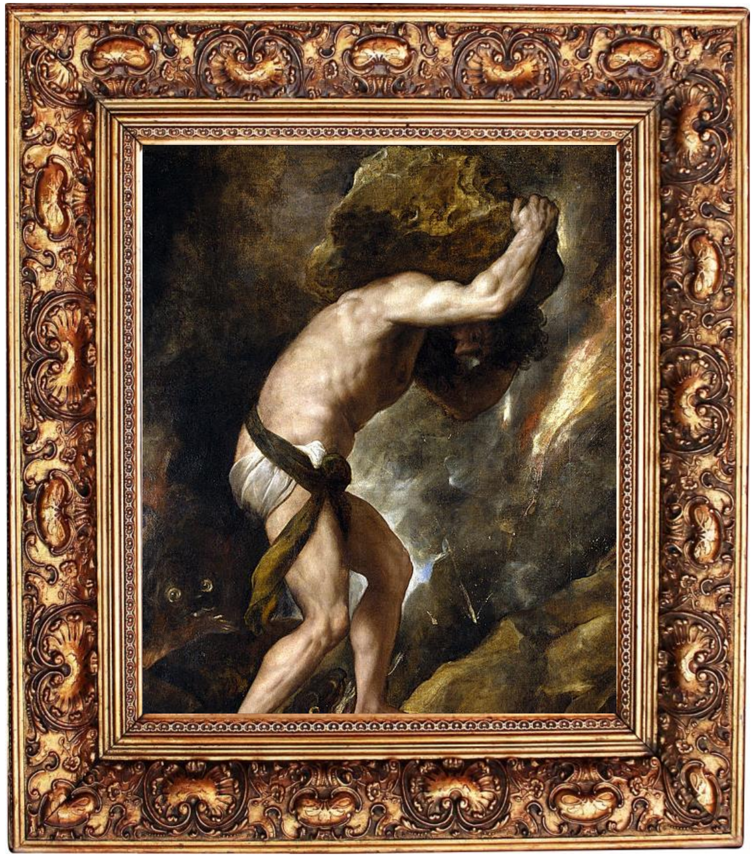
There’s Monolith and Monolith



Challenges with Monolithic Software (1/3)

When a Monolith increases in complexity:

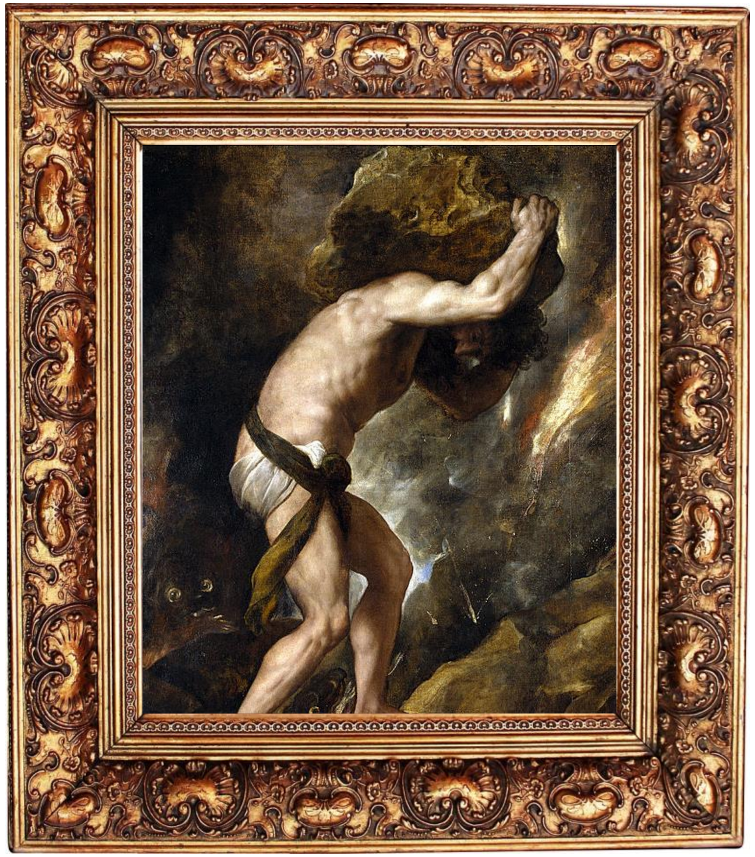
- Inner architecture becomes harder to maintain and evolve
- New releases take longer and longer (months)
- Long time to add new features



Sisyphus, Titian
Oil painting, 1548 ca.
Museo del Prado, Madrid

Challenges with Monolithic Software (2/3)

- Long and increasingly complex build/test/release lifecycle
  - Who broke the build?
- Deployments become increasingly difficult
  - Who's the owner of the failing module?
- Lack of innovation



Sisyphus, Titian
Oil painting, 1548 ca.
Museo del Prado, Madrid

Microservices: Motivations

- Today’s world is **volatile** and **changes rapidly**
- Businesses must be **agile** and **innovate faster**
- Software must be delivered **rapidly**, **frequently**, and **reliably**

DevOps Research and Assessment (DORA) metrics (link):	
Deployment Frequency	How often releases to production are made
Lead Time for Changes	How much time it takes a commit to get into production
Changes Failure Rate	Percentage of deployments failing in production
Time to Restore Service	How long it takes to recover from a failure in production

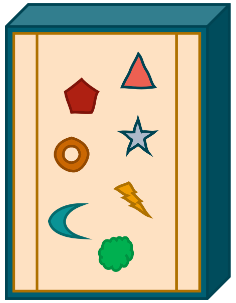
Microservices: Handling Complexity

Splitting the Monolith in a number of Microservices can help make larger systems manageable.

- Each Microservice is significantly smaller than the entire system
  - Easier to understand, to maintain, to evolve, to test…
- Each Microservice can be built and deployed independently
- New features should likely impact a single Microservice
- More frequent deployments (redeploy a single Microservice to add a new feature)

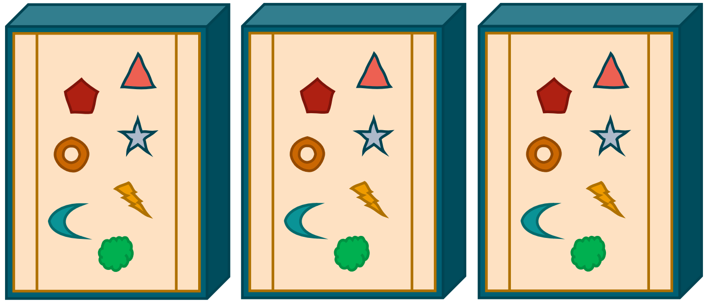
Monolith vs Microservices: Scaling

- Monoliths put all functionality in a single process and can scale by **replicating** the monolith on multiple servers



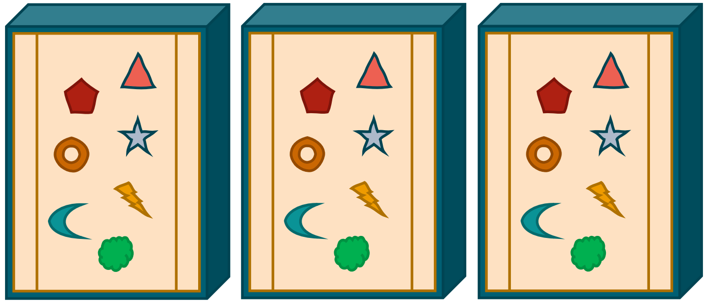
Monolith vs Microservices: Scaling

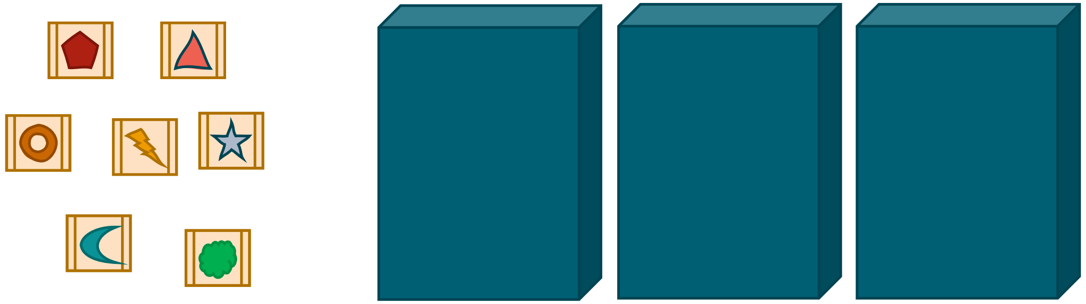
- Monoliths put all functionality in a single process and can scale by **replicating** the monolith on multiple servers



Monolith vs Microservices: Scaling

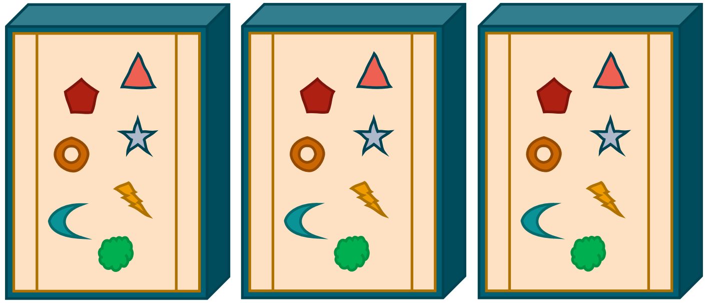
- Monoliths put all functionality in a single process and can scale by **replicating** the monolith on multiple servers
- Microservices put each functionality in a distinct process, and scale by **distributing** services on different servers, replicating as needed

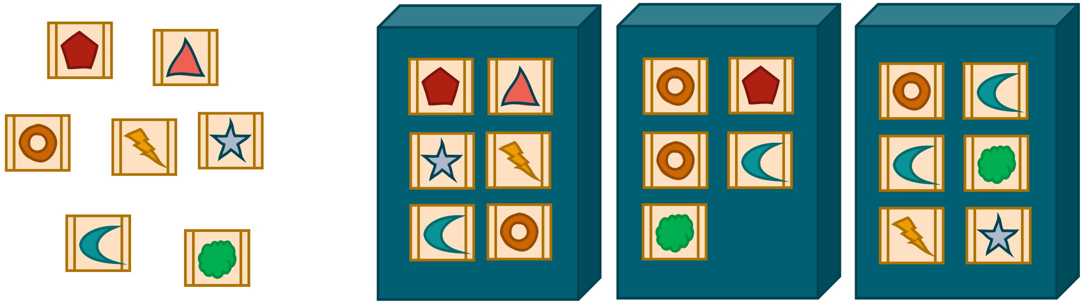




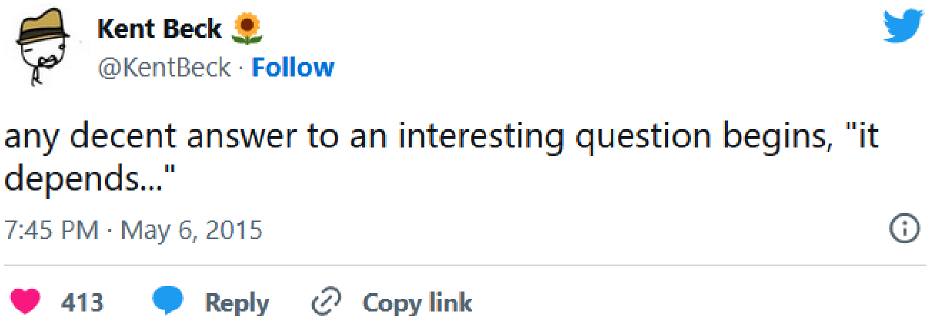
Monolith vs Microservices: Scaling

- Monoliths put all functionality in a single process and can scale by **replicating** the monolith on multiple servers
- Microservices put each functionality in a distinct process, and scale by **distributing** services on different servers, replicating as needed





Should I use a Microservice Architecture?



Kent Beck 🌻

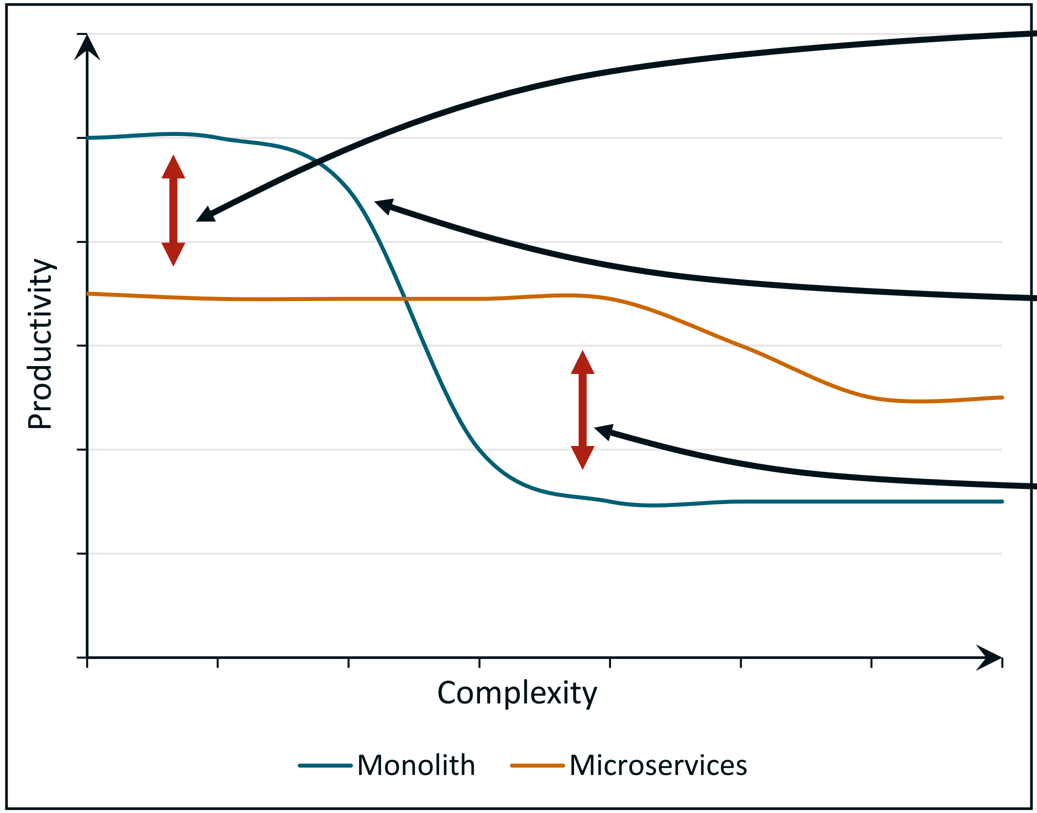
@KentBeck · Follow

any decent answer to an interesting question begins, "it depends..."

7:45 PM · May 6, 2015

- Don’t even consider it unless the system is **complex**
- The complexity that drives us to microservices can come from:
  - Sheer size, dealing with large development teams
  - Multi-tenancy
  - Supporting different user interaction models
  - Need for scaling

Monoliths vs Microservices: Productivity



- For less-complex systems, the overhead of microservices reduces productivity
- Complexity kicks in, Monolith productivity drops
- Decreased coupling of microservices reduces the attenuation of productivity

Source: https://www.martinfowler.com/bliki/MicroservicePremium.html

Adopting Microservices

Key points to address

- How do I **organize** the development teams?
- How do I **design** the microservices?
- How do I manage inter-service **communication**?
- How do I manage **data**?

Organization

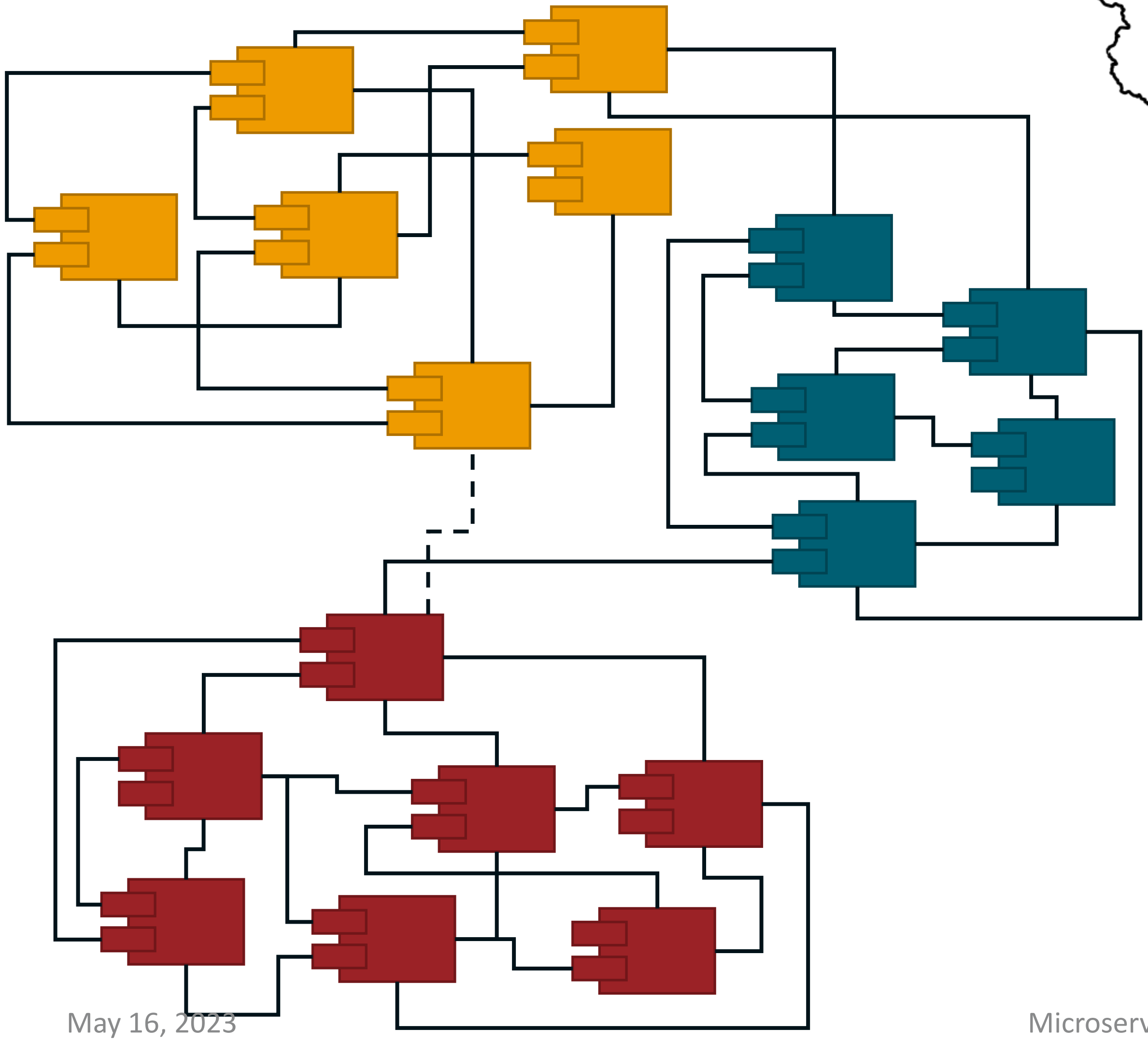
Development Team Organization

- What does Organization have to do with Architecture?
- Conway’s Law:

> *Any organization that designs a system will produce a design whose structure is a copy of the organization's communication structure.*
>
> -- Melvin Conway [1]

[1] Conway, M. E. (1968). How do committees invent. *Datamation*, *14*(4), 28-31.

Conway’s Law

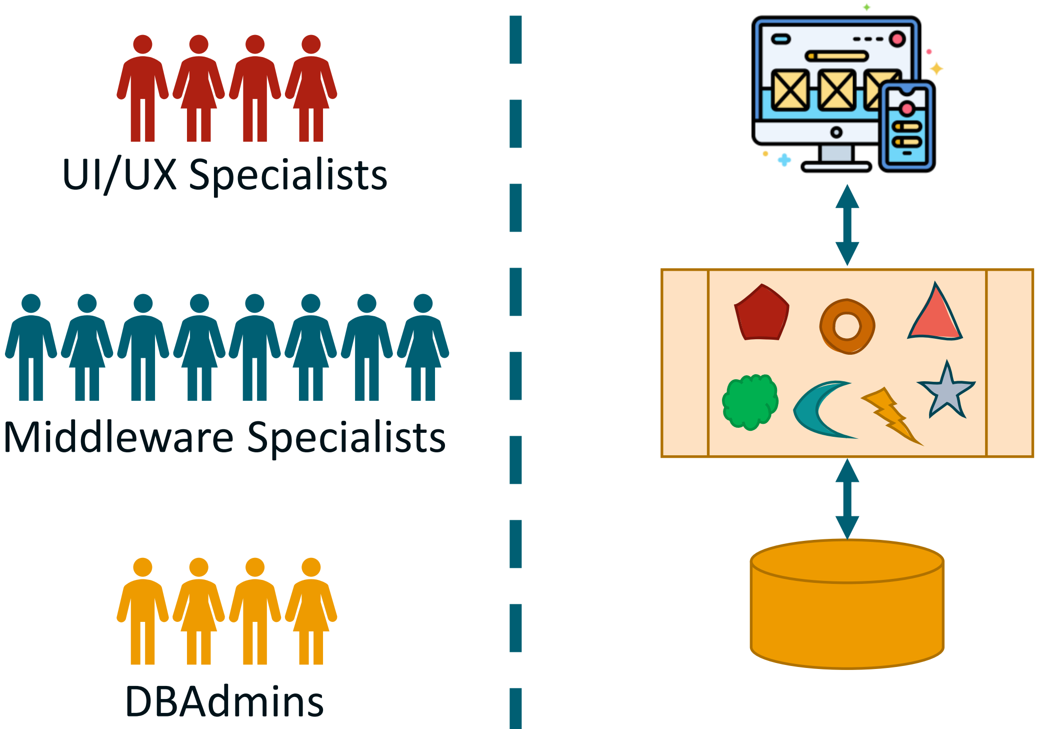


Conway’s Law

- Software coupling is **enabled** and **encouraged** by **human communications**
- If you can talk to the author of some code, it’s easier to truly understand that code
- Which means it is easier to for your code to interact with (and thus be coupled to) that code

How do we organize a large team?

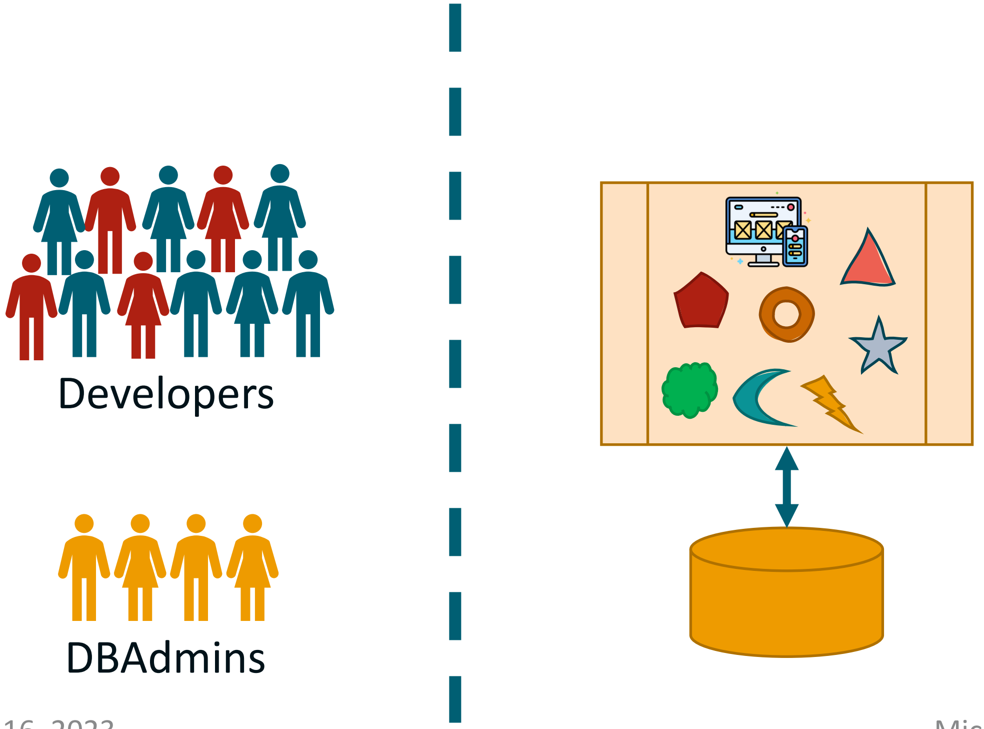
- Often management focuses on the technology layer
- Layered functional teams lead to layered architectures



- What if we need to add a new feature?
- Logic everywhere!

How do we organize a large team?

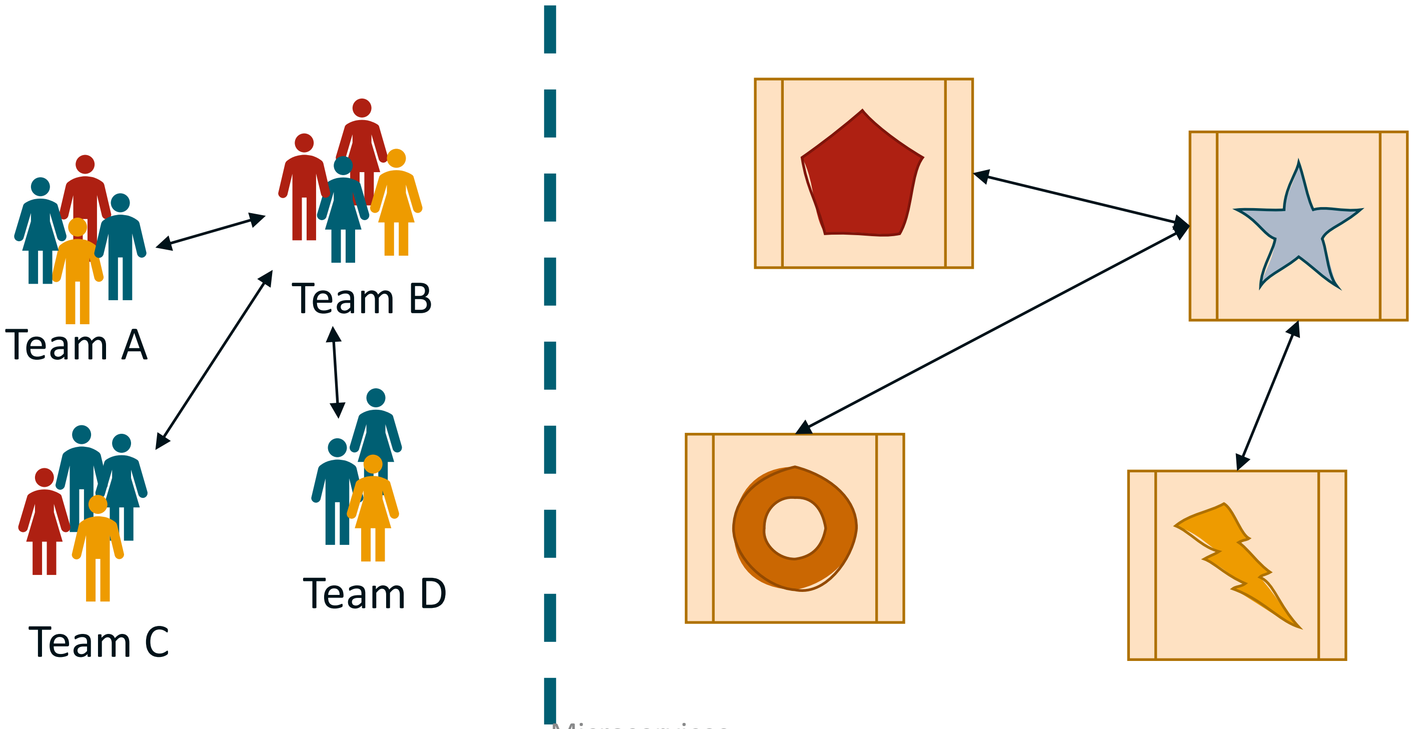
- Often management focuses on the technology layer
- Layered functional teams lead to layered architectures



- What if we need to add a new feature?
- Logic everywhere!

How do we organize a large team?

- Microservices favor small, independent teams
- Teams are **cross-functional**: include full range of skills required



Dealing with Conway’s Law

- **Ignore it:** Don’t take it into account, because you’ve never heard of it, or you don’t think it applies (spoiler: it does)
- **Accept it:** Recognize its impact, make sure that you architecture doesn’t clash with the team’s communicational patterns.
- **Inverse Conway Maneuver** [1]**:** Change the communication patterns of the development team to encourage the desired architecture.

[1] https://martinfowler.com/bliki/ConwaysLaw.html

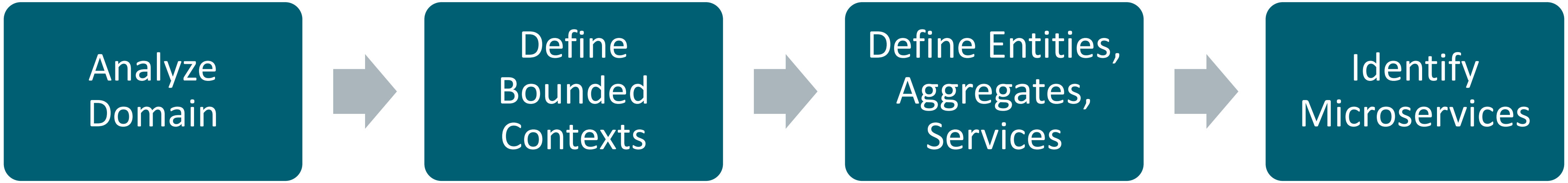
Conway’s Law: Take Home Message

- The decomposition of a system and the decomposition of the development organization cannot be done independently.
- Not only at the beginning of a project, but throughout the entire lifecycle of the project.
- Evolving the architecture and re-organizing the human organization must go hand-in-hand.

Design

Microservices: Design

- Defining boundaries for each service is one of the biggest challenges
- General rule: a service should just do «**one thing**»
- No mechanical process can guarantee the ***right*** result
- **Domain-Driven Development** (DDD) comes in handy
- Nice example here: [Using domain analysis to model microservices]()

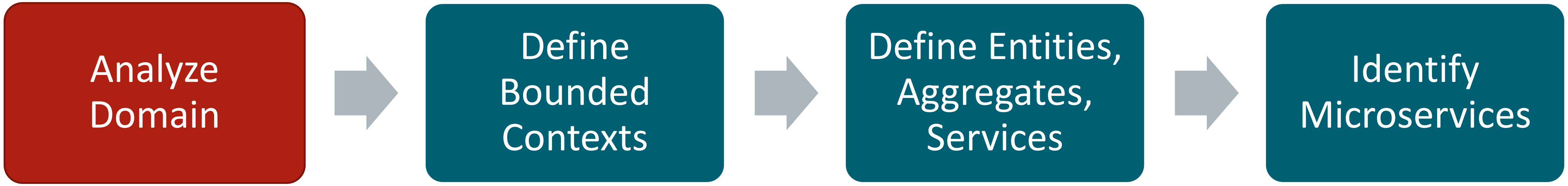

Analyze Domain

Define Bounded Contexts

Define Entities, Aggregates, Services

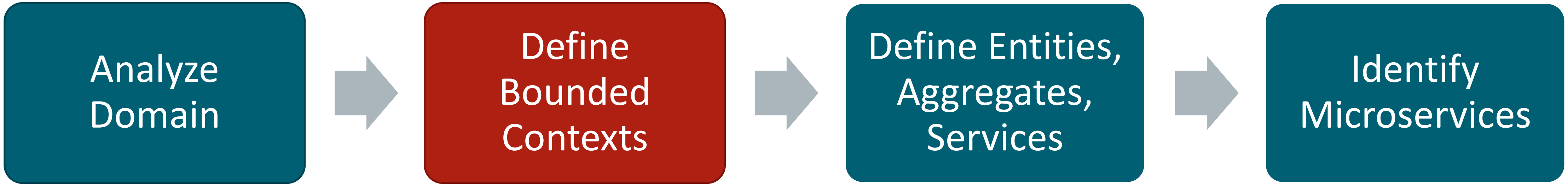
Identify Microservices

Microservices: Design



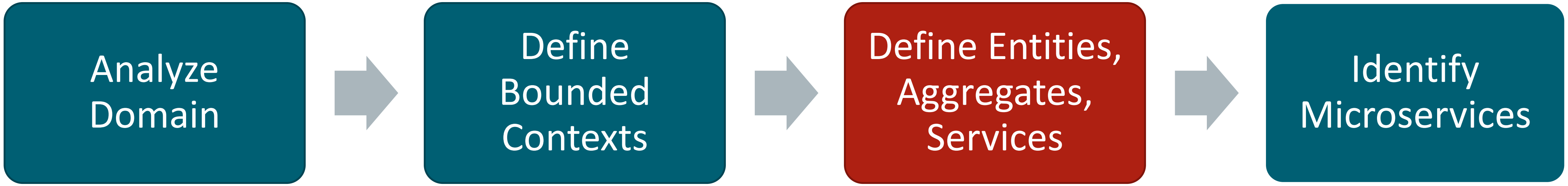
- Understand functional requirements
- Output is an informal definition of the system domain
- Informal definition can be refined into a set of domain models

Microservices: Design



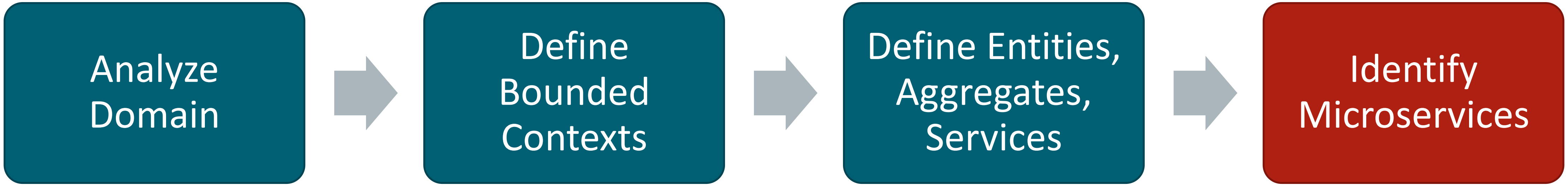
- Each Bounded Context contains a domain model representing a particular subdomain
- Total domain unification might not be feasible nor convenient

Microservices: Design



- For each Bounded Context, apply DDD patterns to define:
  - **Entities**: Objects with an «identity» that persists over time.
  - **Aggregates**: Consistency boundaries around one or more entities. Model transactional invariants.
  - **Domain Services**: Objects that implement logic without holding any state

Microservices: Design



Analyze Domain → Define Bounded Contexts → Define Entities, Aggregates, Services → Identify Microservices

- Start with a Bounded Context. Functionality in a microservice should not span over multiple bounded contexts.
- Aggregates and Domain Services are good candidates for becoming microservices

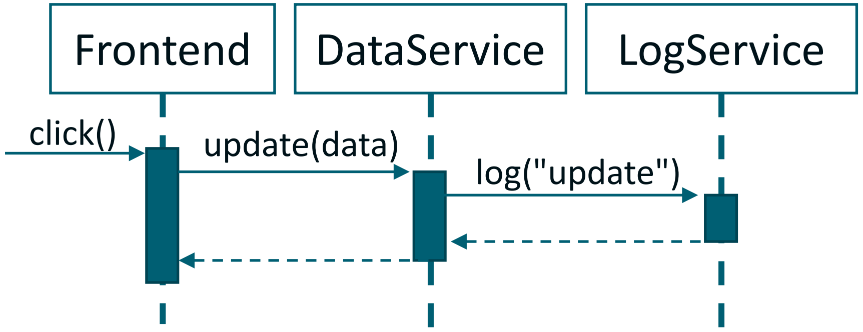
Communication

Microservices: Communication

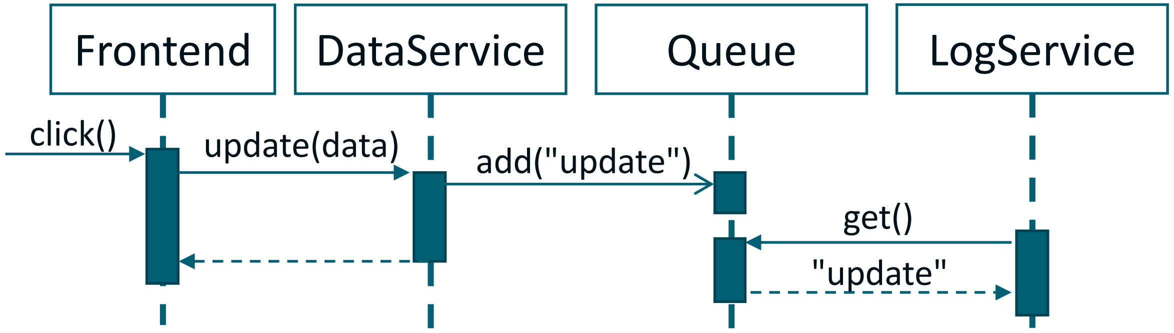
- In Monoliths, communication between different components is achieved with method invocations
- In a Microservice architecture, communication should be **technology agnostic**
- Often, teams leverage protocols and technology on which the World Wide Web is built

Microservices: Communication

- Can be done **synchronously**
  - **REST** (http resource API)
  - **Apache Thrift** (link). Supports multiple protocols (binary, JSON-based, …)
- Can be done **asynchronously**
  - **Message queues**



Synchronous Communication



Asynchronous Communication

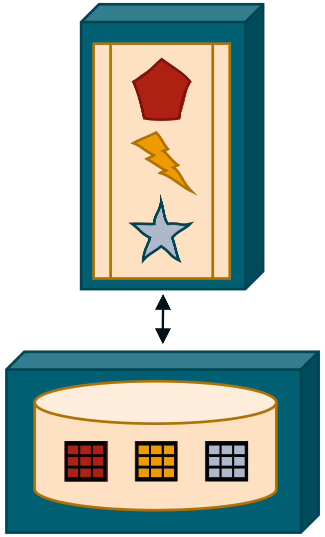
Microservices: Sync vs Async Communication

- Synchronous calls can hinder performance
- In presence of many synchronous calls, the **multiplicative effect of downtime** might manifest.
  - Downtime of the systems becomes the product of the downtimes of individual services
- Either you make your communications asynchronous, or you manage the downtimes

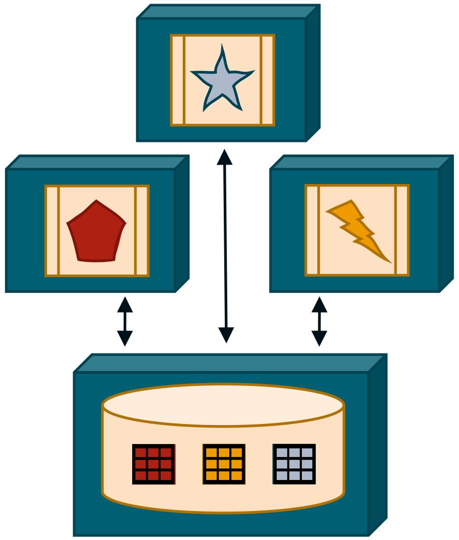
Data management

Microservices: Data management

- Monoliths are typically associated with a **single, centralized database**
  - Often driven by DB licensing models and costs
- Microservices can work with a centralized database…
  - Need to coordinate schema changes across teams



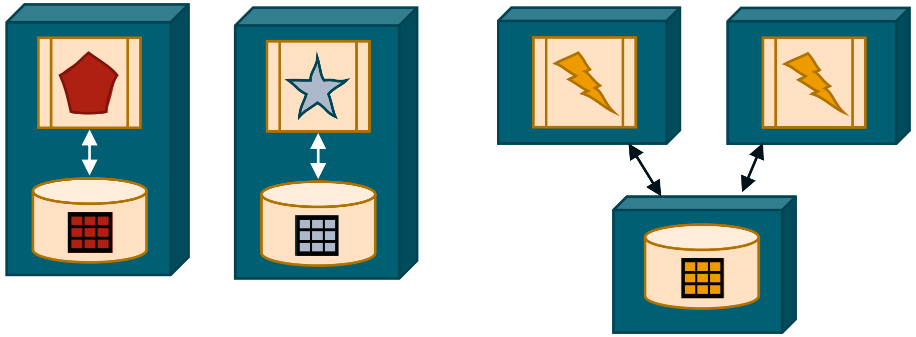
Monolith



Microservices, with shared database

Microservices: Data management

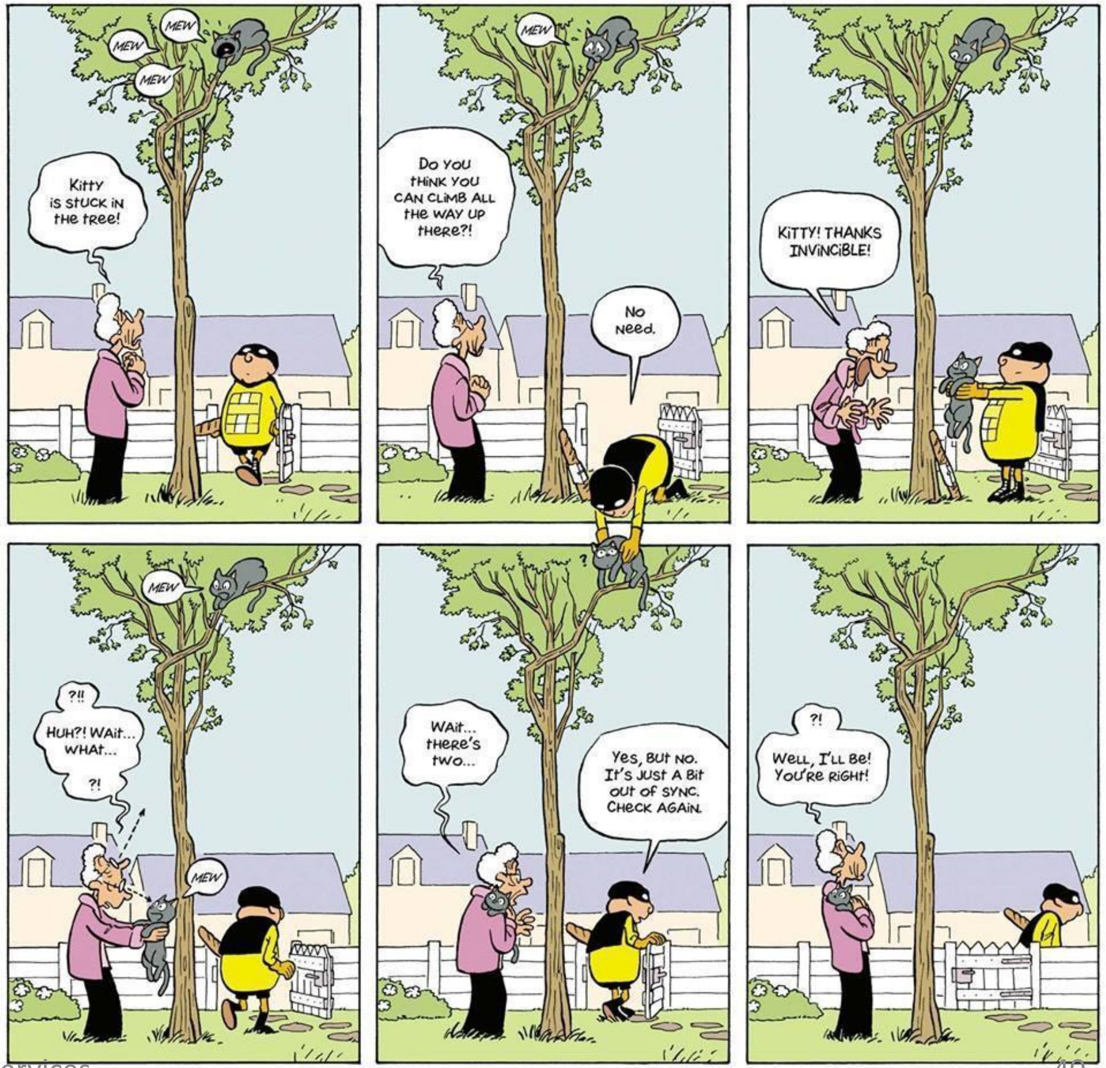
- **Single database per service**
  - Each microservice (and Team) is actually independent
  - Enables polyglot persistence



Microservices, with per-service databases

Microservices: Data management

- When dealing with distributed systems, ensuring **strong consistency** can become harder and harder.
- Might want to settle with **eventual consistency.**
  - At some point in the future, all reads on an entity will return the updated value


MEW
MEW
MEW
Kitty is stuck in the tree!
MEW
Do you think you can climb all the way up there?!
No need.
Kitty! Thanks Invincible!
MEW
?!!
Huh?! Wait... What... ?!
MEW
?
Wait... there's two...
Yes, but no. It's just a bit out of sync. Check again.
?!
Well, I'll be! You're right!

Imbattable, Pascal Jousselin

Microservices: Conclusions

Microservices: It’s about trade offs

Microservices provide benefits…

✓ Strong module boundaries

✓ Independent deployments

✓ Technology diversity

… but not for free

× Increased complexity

× Eventual consistency

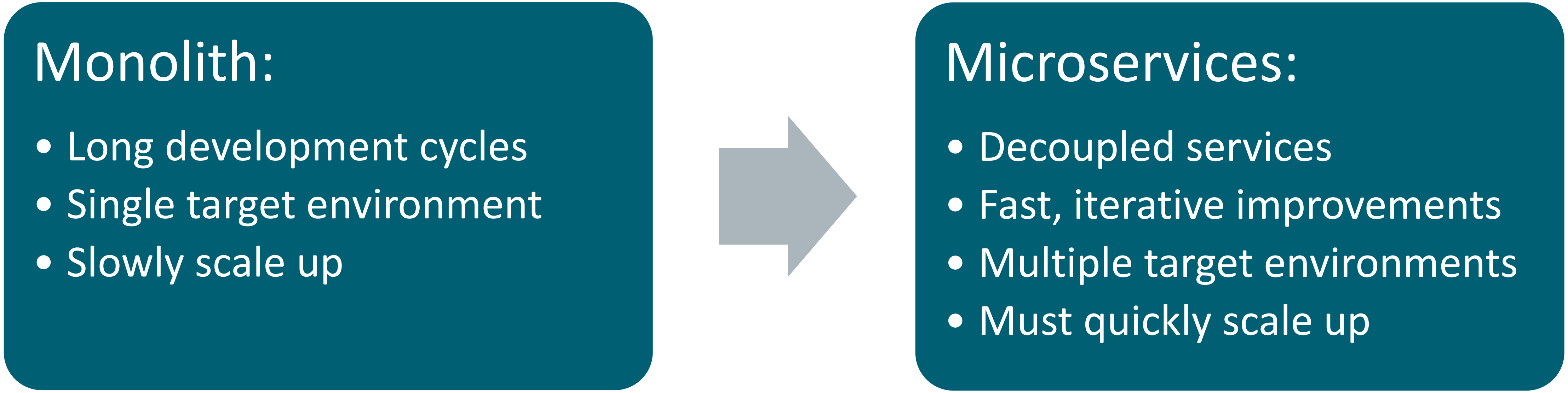
× Operational complexity

So, make sure you have a good reason to adopt microservices! [1]

[1] Mendonça, N. C., Box, C., Manolache, C., & Ryan, L. (2021). The monolith strikes back: Why Istio migrated from microservices to a monolithic architecture. *IEEE Software*, *38*(05), 17-22. https://ieeexplore.ieee.org/document/9520758

Containerization

Microservices trend



Monolith:

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

Microservices:

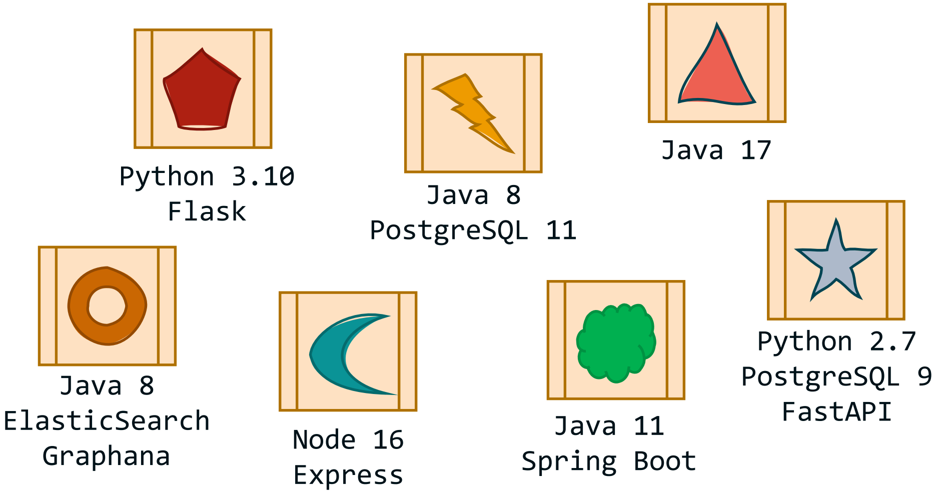
- Decoupled services
- Fast, iterative improvements
- Multiple target environments
- Must quickly scale up

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



Python 3.10
Flask

Java 8
PostgreSQL 11

Java 17

Java 8
ElasticSearch
Graphana

Node 16
Express

Java 11
Spring Boot

Python 2.7
PostgreSQL 9
FastAPI

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»

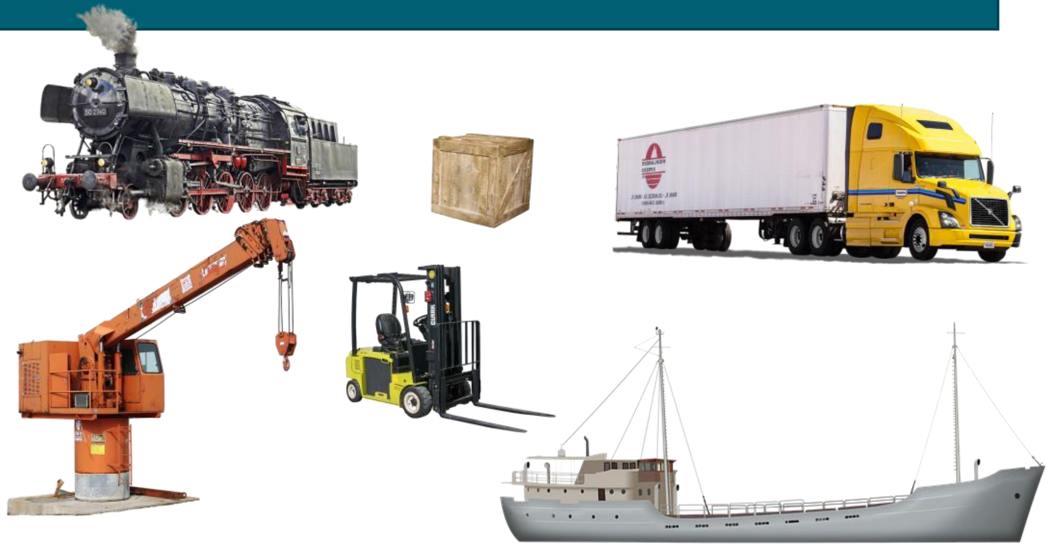
		Environments				
		Dev Laptop	Production Server	Distaster Recovery	Public Cloud	QA Server
Stacks	Web App	?	?	?	?	?
	API Endpoint	?	?	?	?	?
	Analytics	?	?	?	?	?
	App DB	?	?	?	?	?
	Queue	?	?	?	?	?
	Preprocessing	?	?	?	?	?

Cargo Transportation before 1960s

Multiplicity of goods

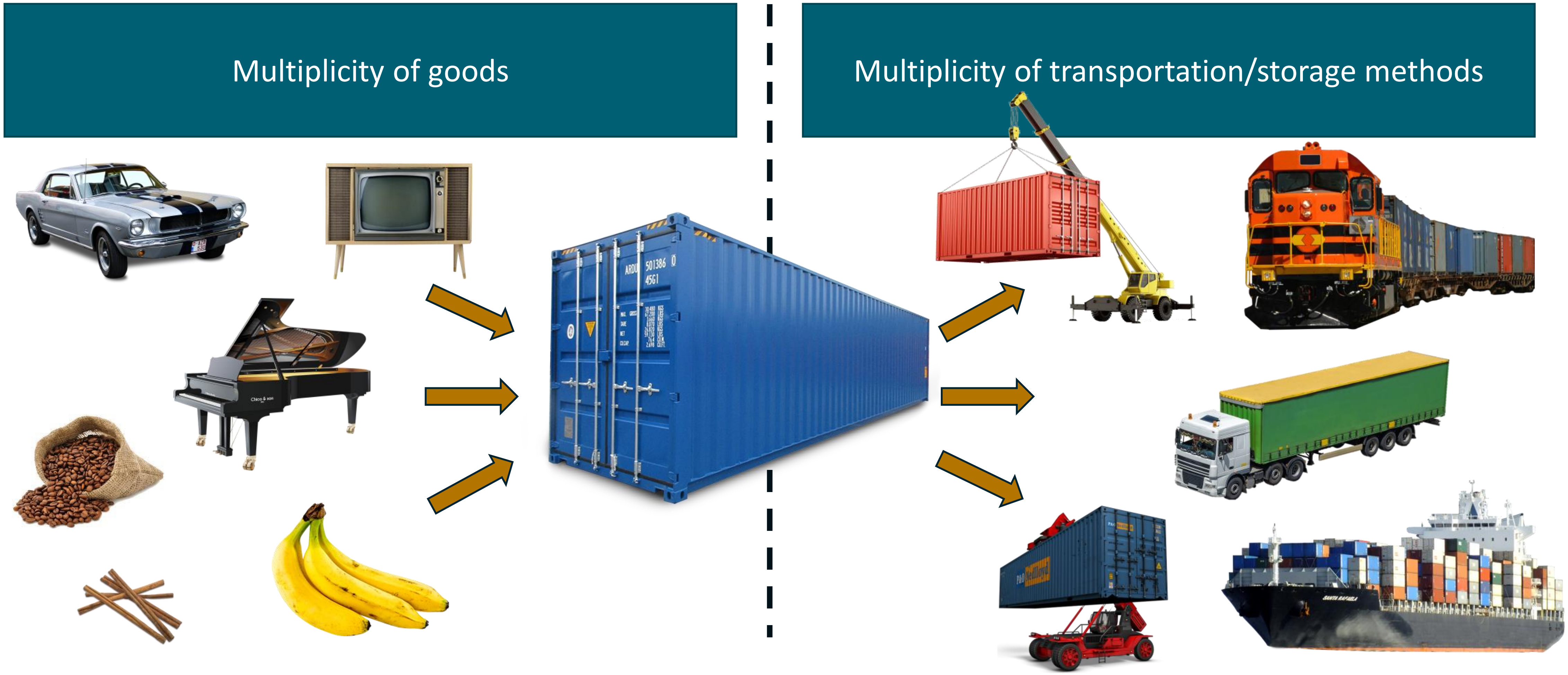
Do goods interact with each others?

Multiplicity of transportation/storage methods



Is it possible to smoothly change transport mode?

Solution: Containers



Multiplicity of goods

Multiplicity of transportation/storage methods

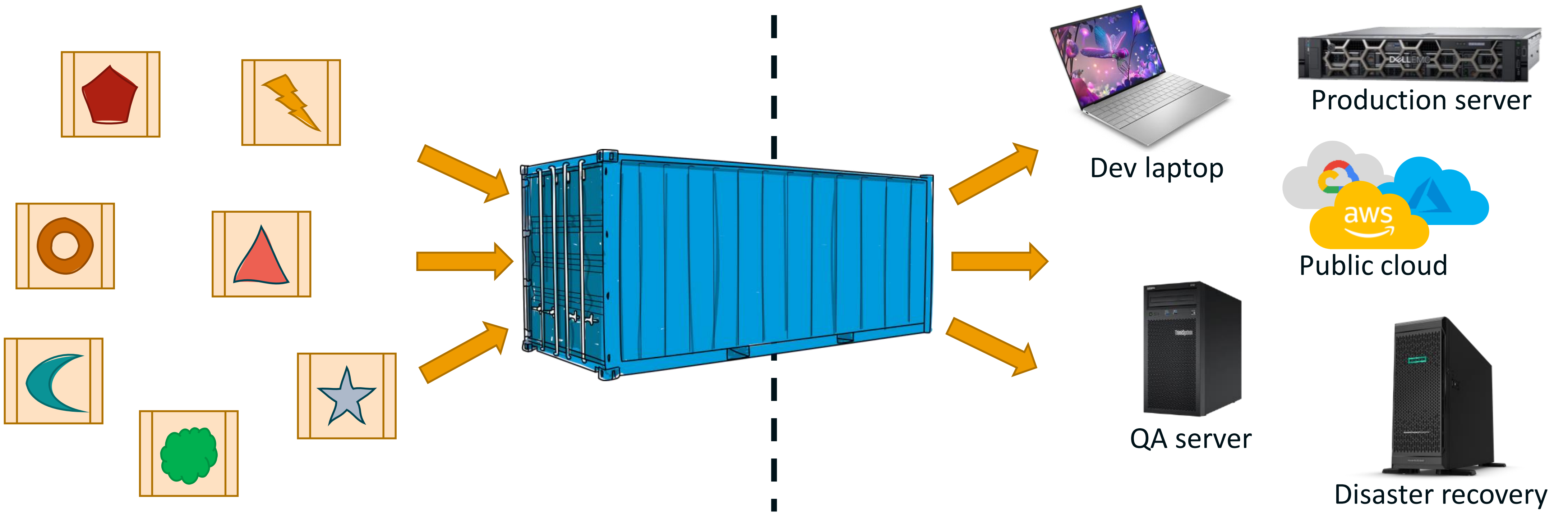
Containers

- **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

Multiplicity of stacks

Multiplicity of Hardware Environments



Dev laptop

Production server

Public cloud

QA server

Disaster recovery

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***

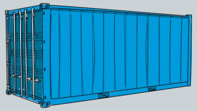
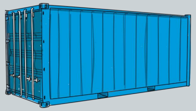
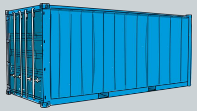
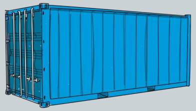
***anywhere with the same architecture and a modern Linux kernel**

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**

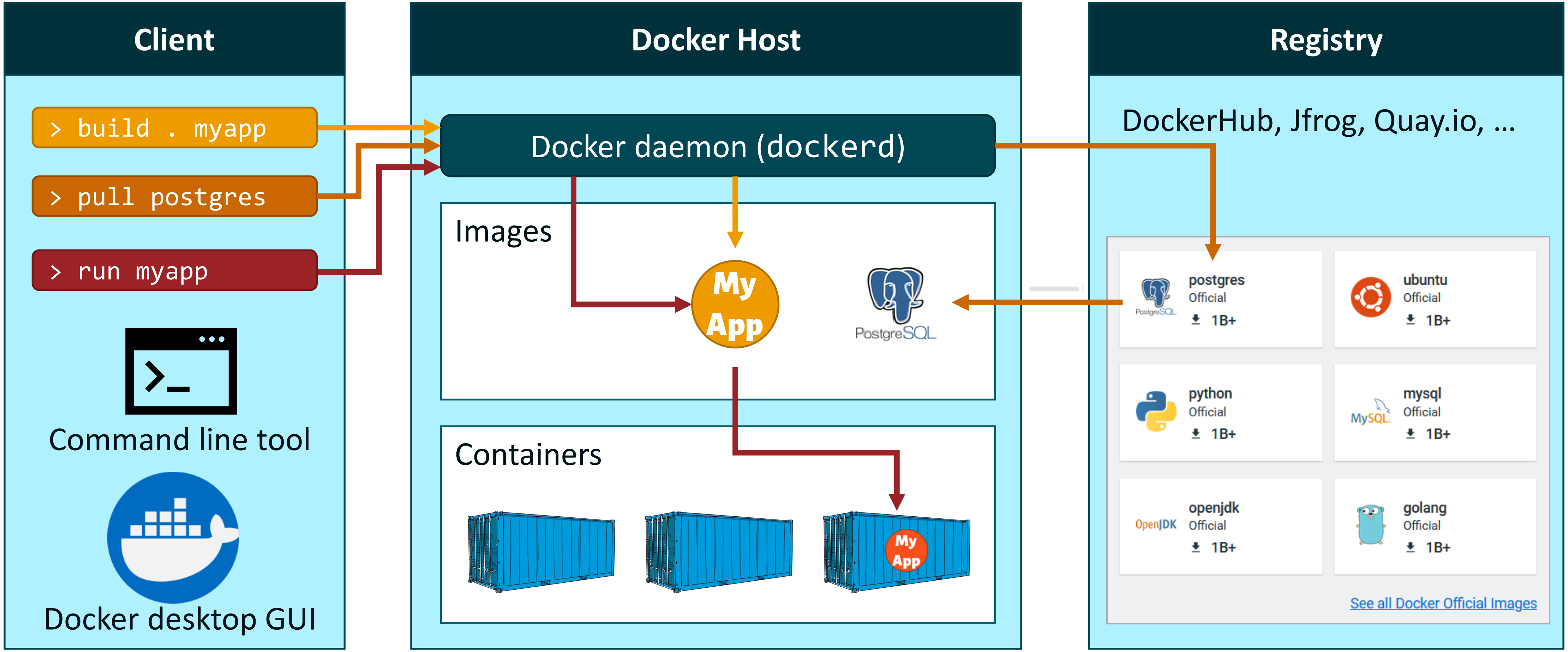
The «Matrix from Hell», solved

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

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



Client

```
> build . myapp
> pull postgres
> run myapp
```

Command line tool

Docker desktop GUI

Docker Host

Docker daemon (dockerd)

Images

My App

PostgreSQL

Containers

My App

Registry

DockerHub, Jfrog, Quay.io, …

postgres
Official
1B+

ubuntu
Official
1B+

python
Official
1B+

mysql
Official
1B+

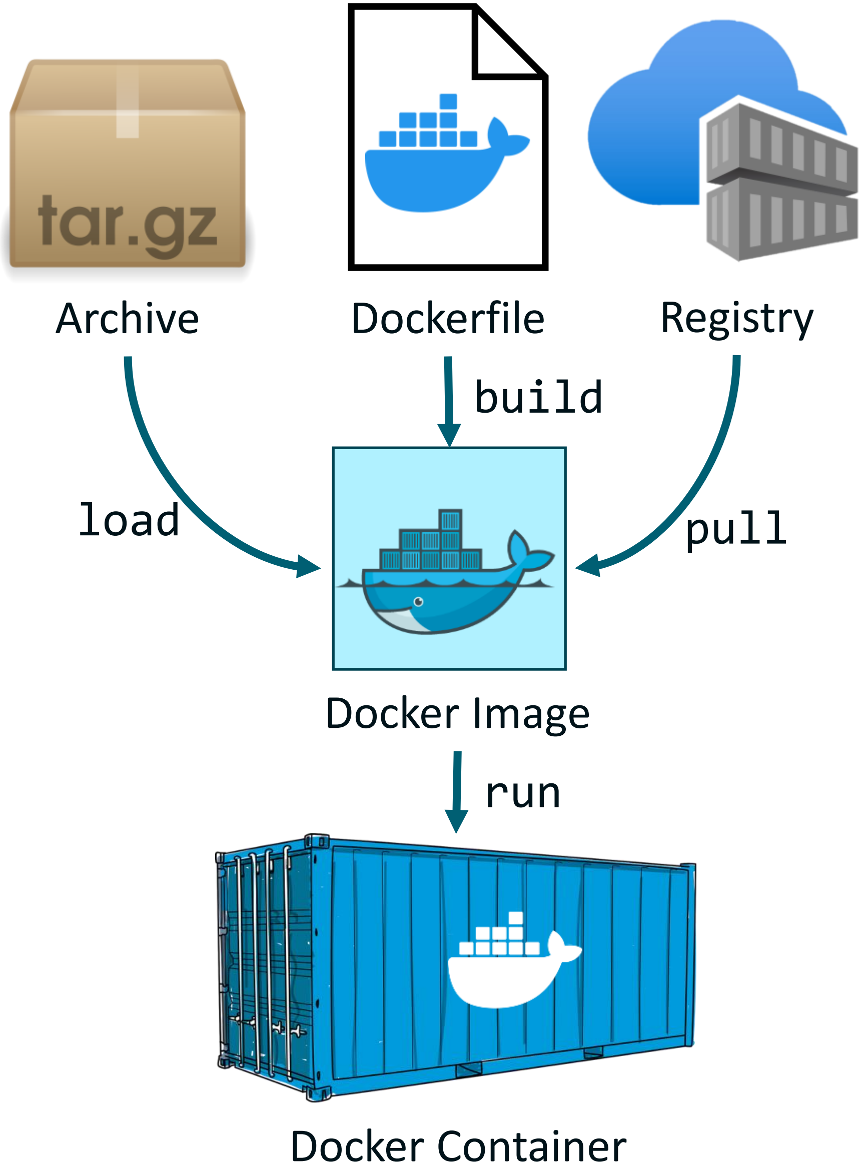
openjdk
Official
1B+

golang
Official
1B+

See all Docker Official Images

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

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:
```

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!
```

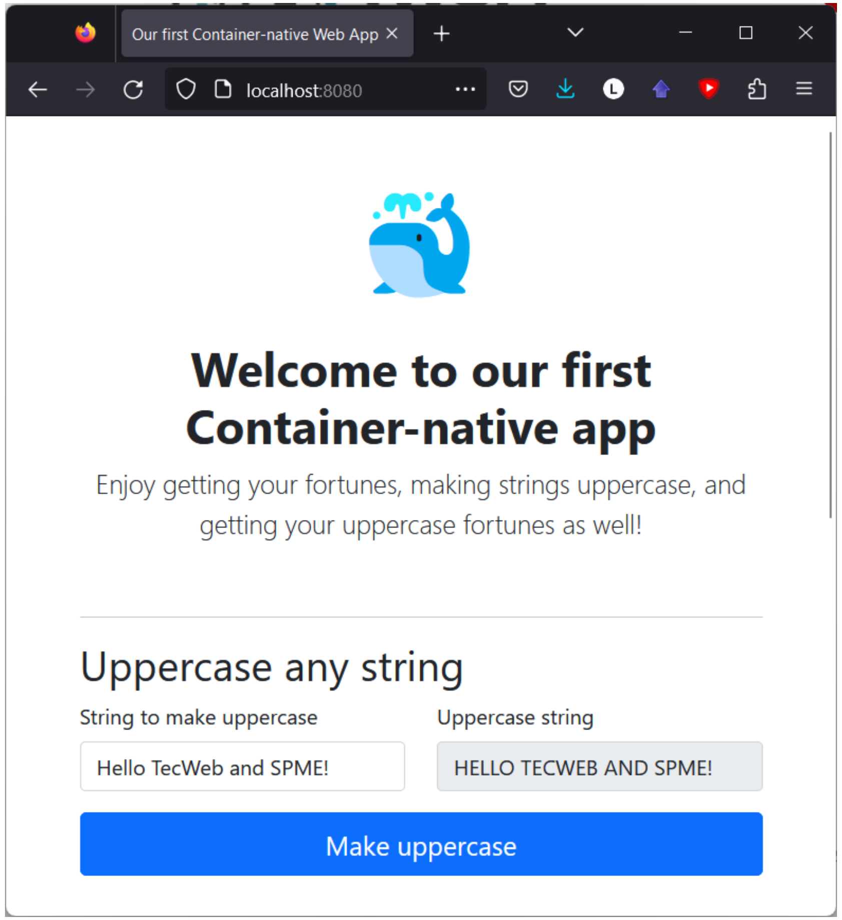
A Container-native Web App

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

A Container-native Web App

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

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



Welcome to our first Container-native app

Enjoy getting your fortunes, making strings uppercase, and getting your uppercase fortunes as well!

Uppercase any string

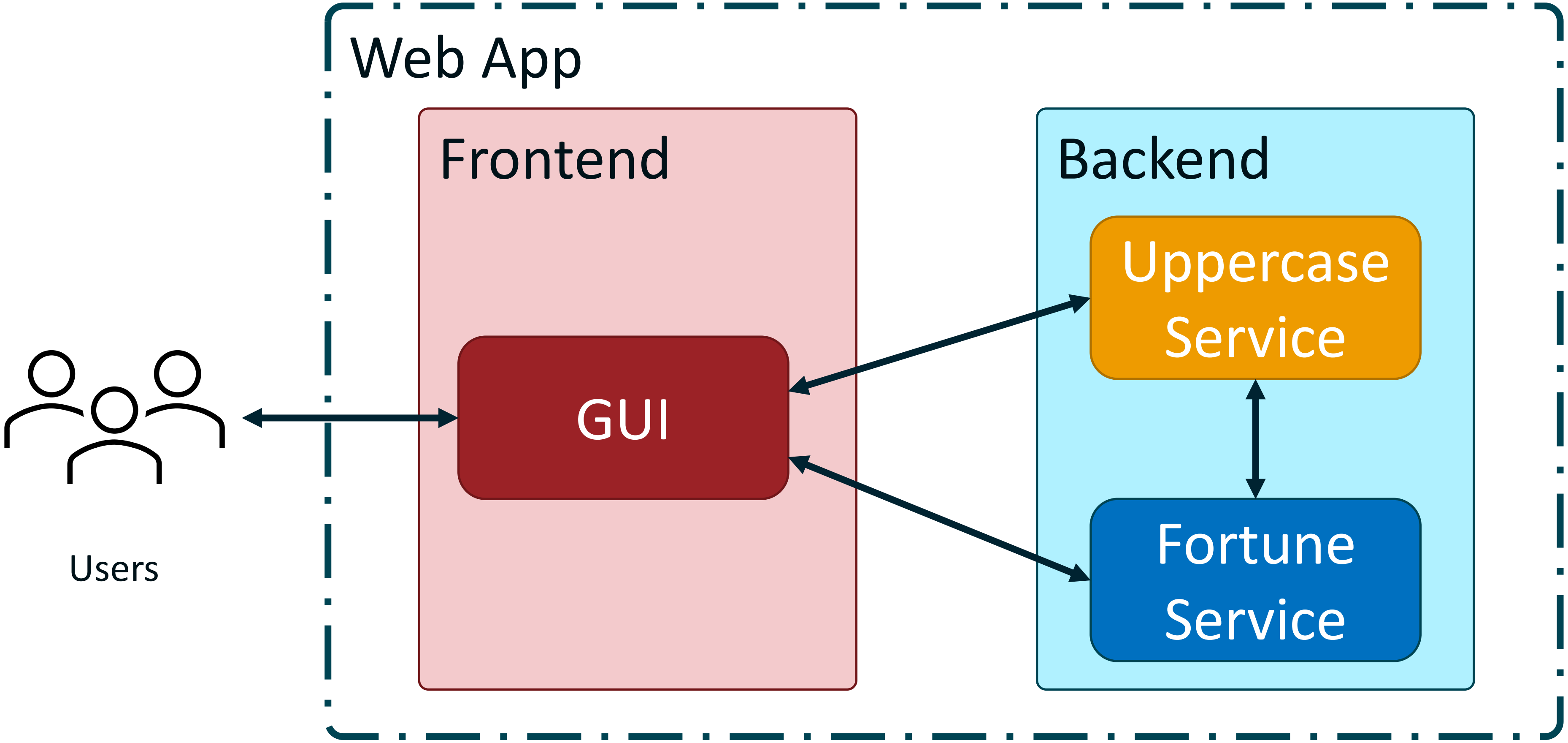
String to make uppercase: Hello TecWeb and SPME!

Uppercase string: HELLO TECWEB AND SPME!

CAUTION

- **This demo is just demostration of using containers to run a web app**
- **This demo is way too simple to be an effective use of microservices**
- **It’s intended to show the basics of container communication across different microservices**
- **Under no circumstances will the Presenter of these slides be held responsible or liable in any way for any claims, damages, losses, expenses, costs or liabilities whatsoever (including, without limitation, any direct or indirect damages for loss of profits, business interruption or loss of information) resulting or arising directly or indirectly from following this example :P**

A Container-native Web App: Architecture



A Container-native Web App: Technologies

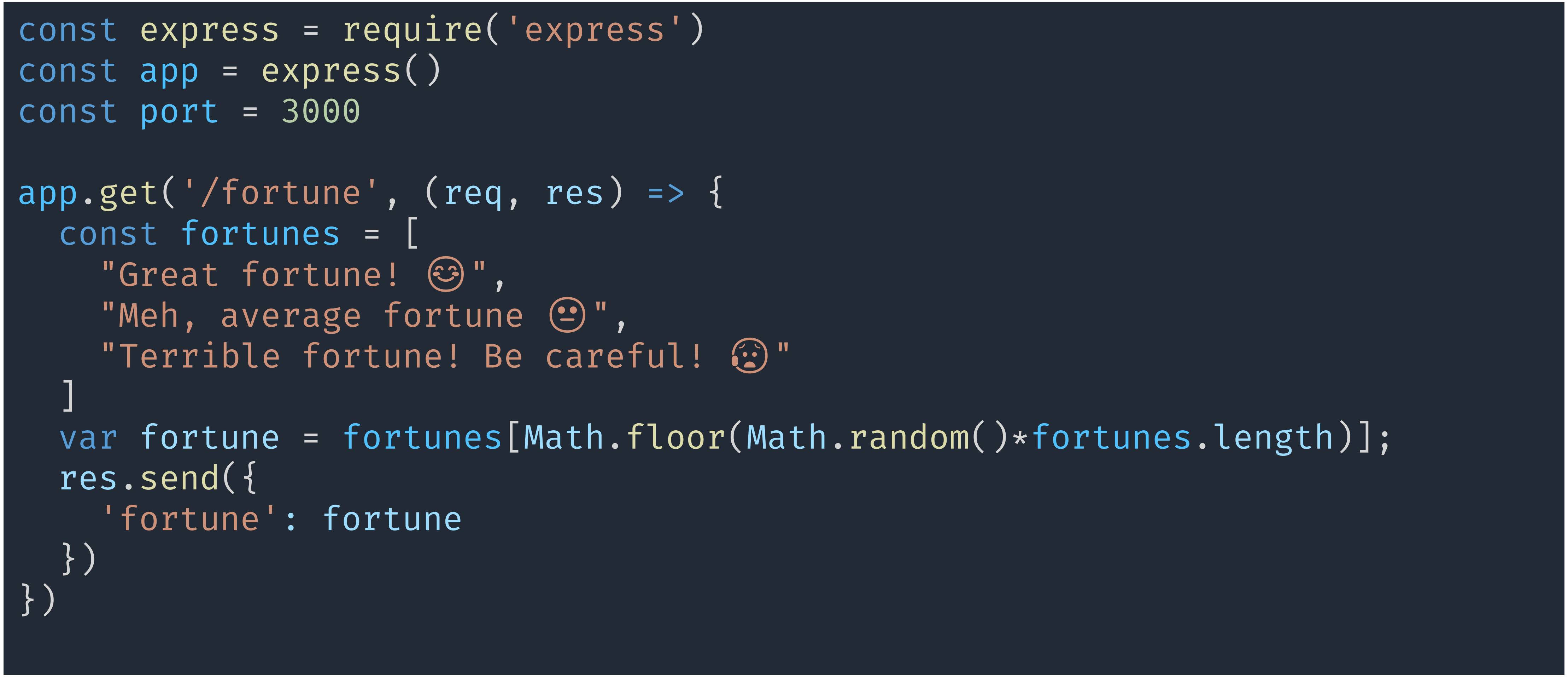
- The components of our app need to communicate
  - We’ll have them communicate using a standard web procol (HTTP)
- Uppercase service
  - Python and Flask
- Fortune service
  - Nodejs and Express
- GUI
  - Single web page with a bit of Javascript

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

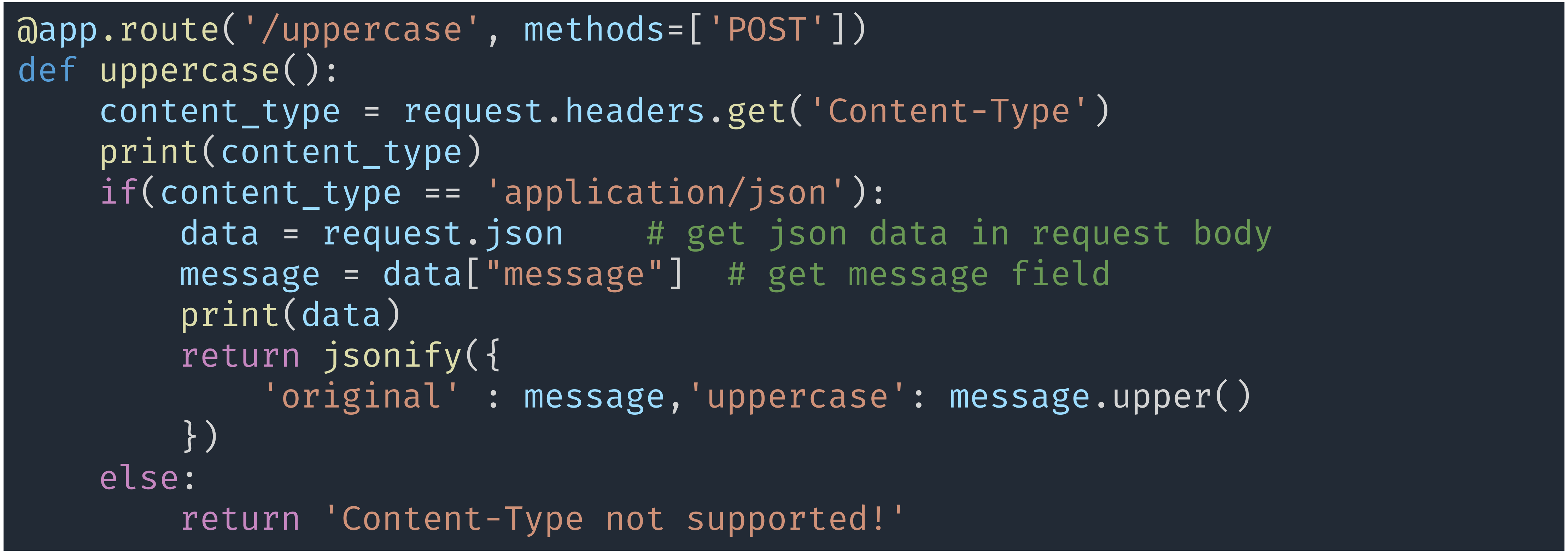
- Users access this feature by sending a POST HTTP request to the /uppercase endpoint (i.e.: http:/{uppercase-service-location}/uppercase)
- The request must contain in its body a JSON like this:

```
{
  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

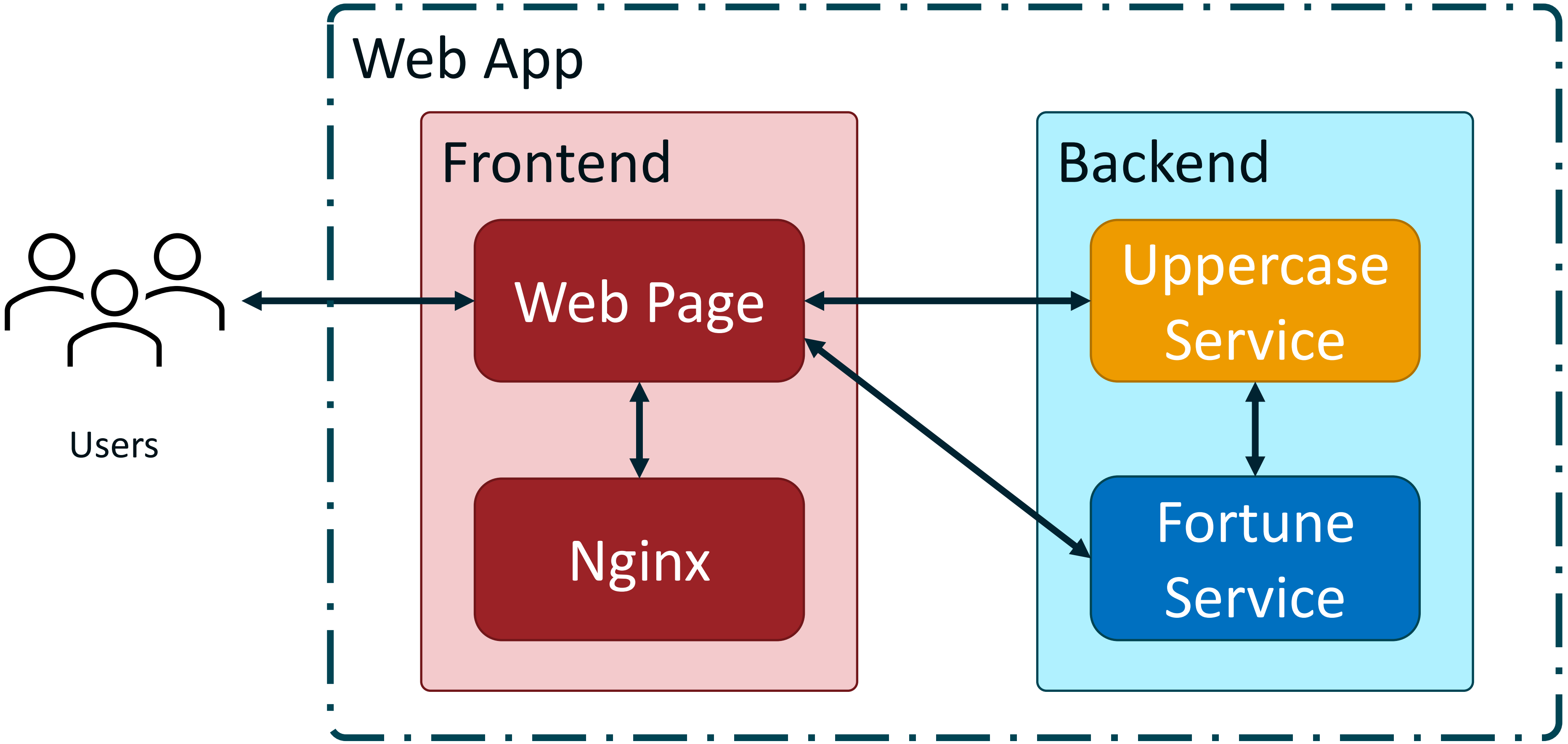
- Users access this feature by sending a GET HTTP request to the /uppercase-fortune endpoint (i.e.: http:/{uppercase-service-location}/uppercase)
- The service gets a fortune using the Fortune service, and makes it uppercase
- The service returns a JSON containing the uppercase fortune

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

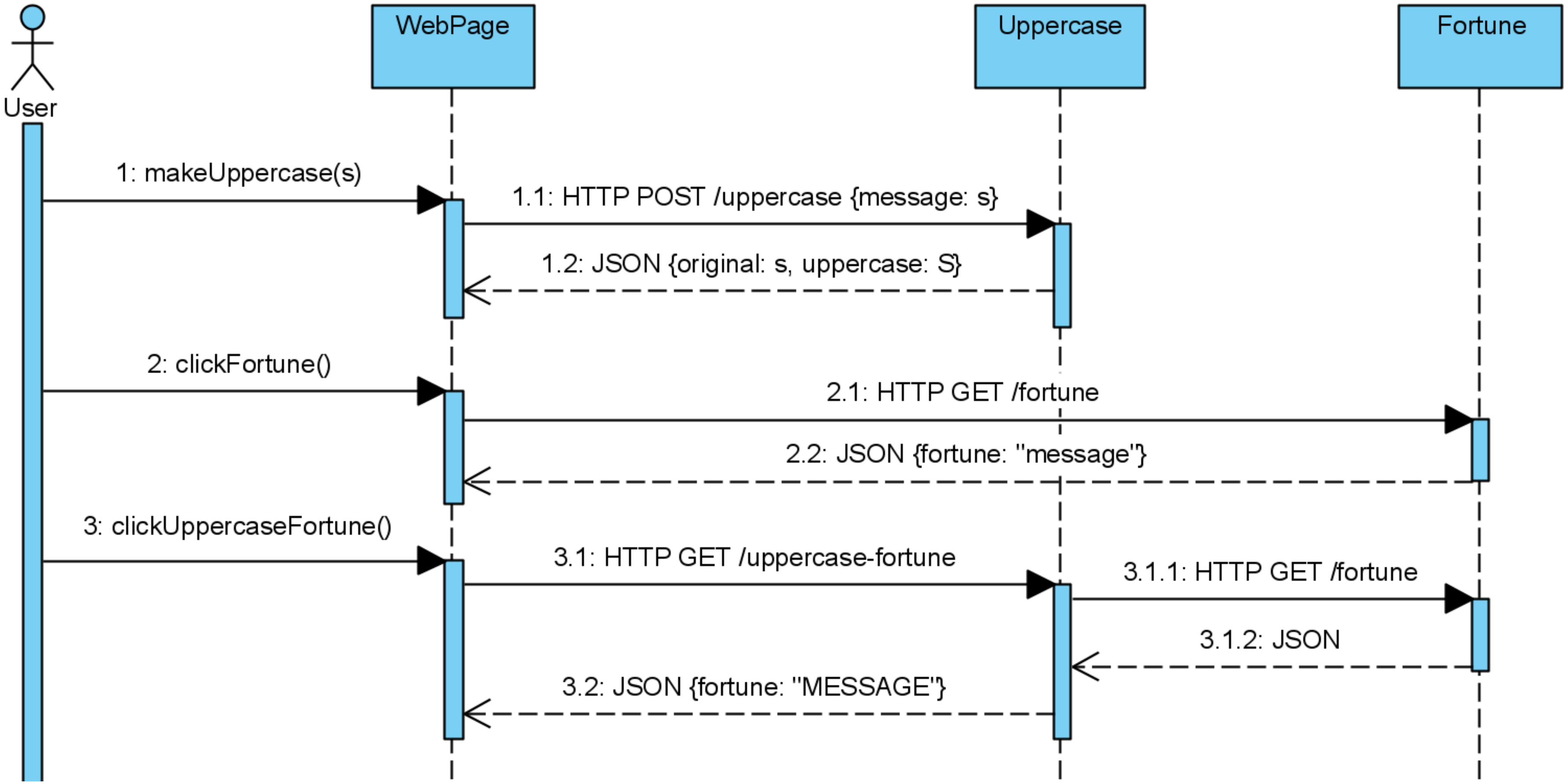
Uppercase service: Uppercase Fortune Implementation (snippet)

```
@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()
    })
```

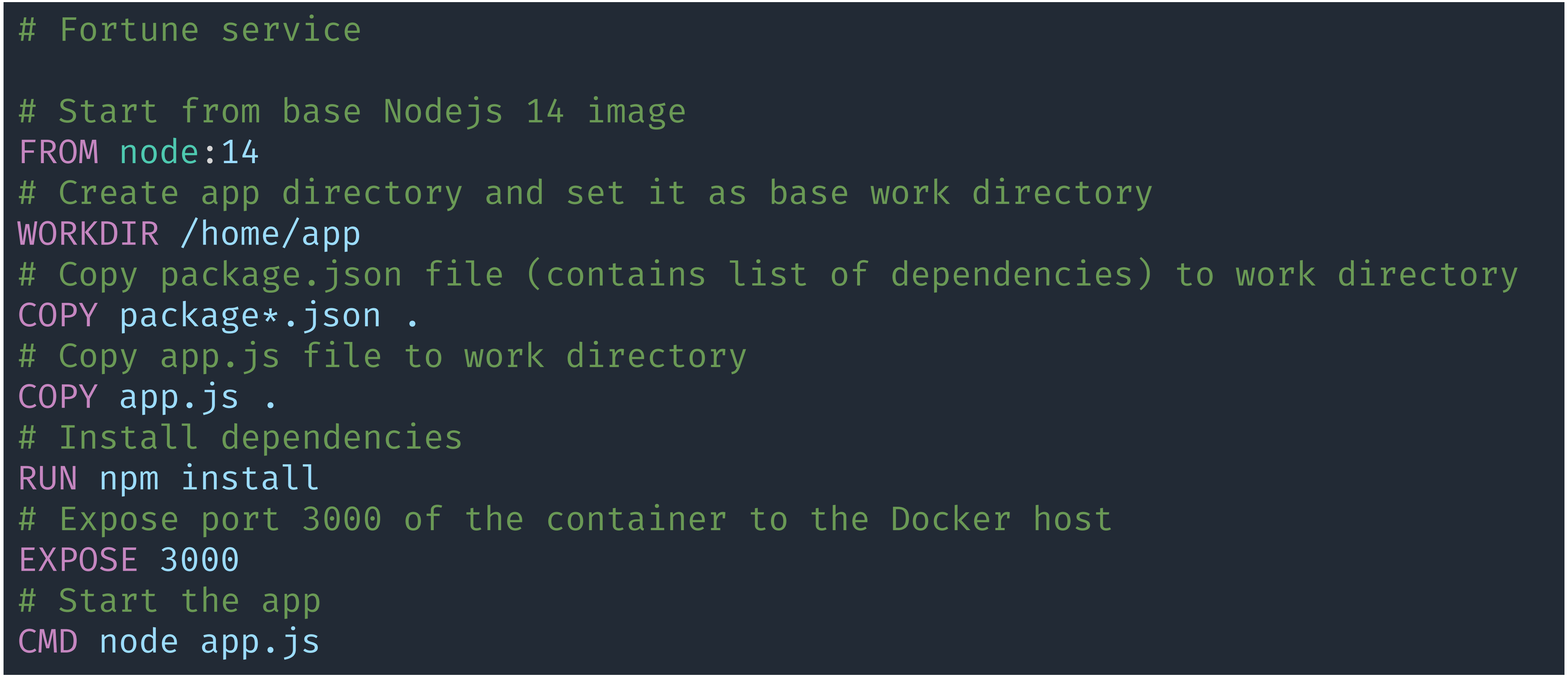
A Container-native Web App



Sequence diagram

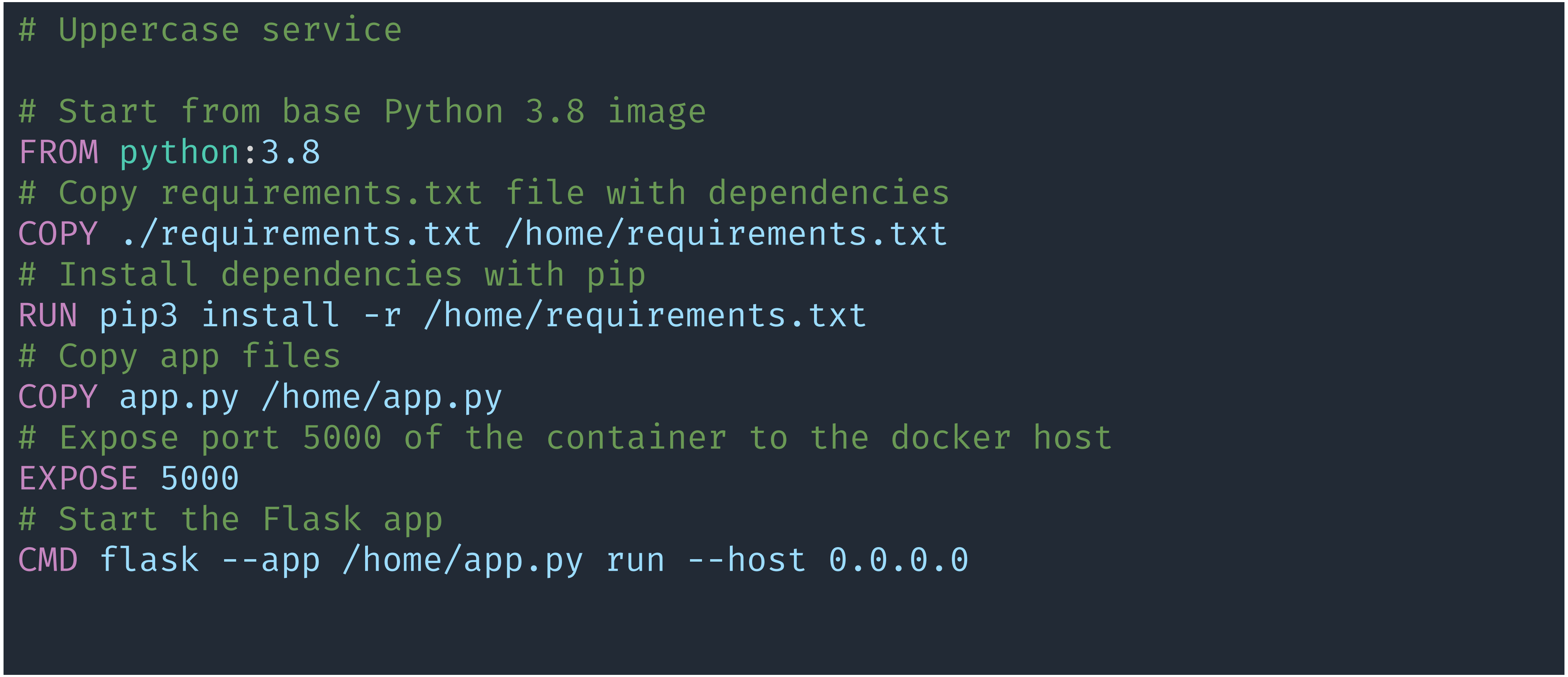

User
WebPage
Uppercase
Fortune
1: makeUppercase(s)
1.1: HTTP POST /uppercase {message: s}
1.2: JSON {original: s, uppercase: S}
2: clickFortune()
2.1: HTTP GET /fortune
2.2: JSON {fortune: "message"}
3: clickUppercaseFortune()
3.1: HTTP GET /uppercase-fortune
3.1.1: HTTP GET /fortune
3.1.2: JSON
3.2: JSON {fortune: "MESSAGE"}

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

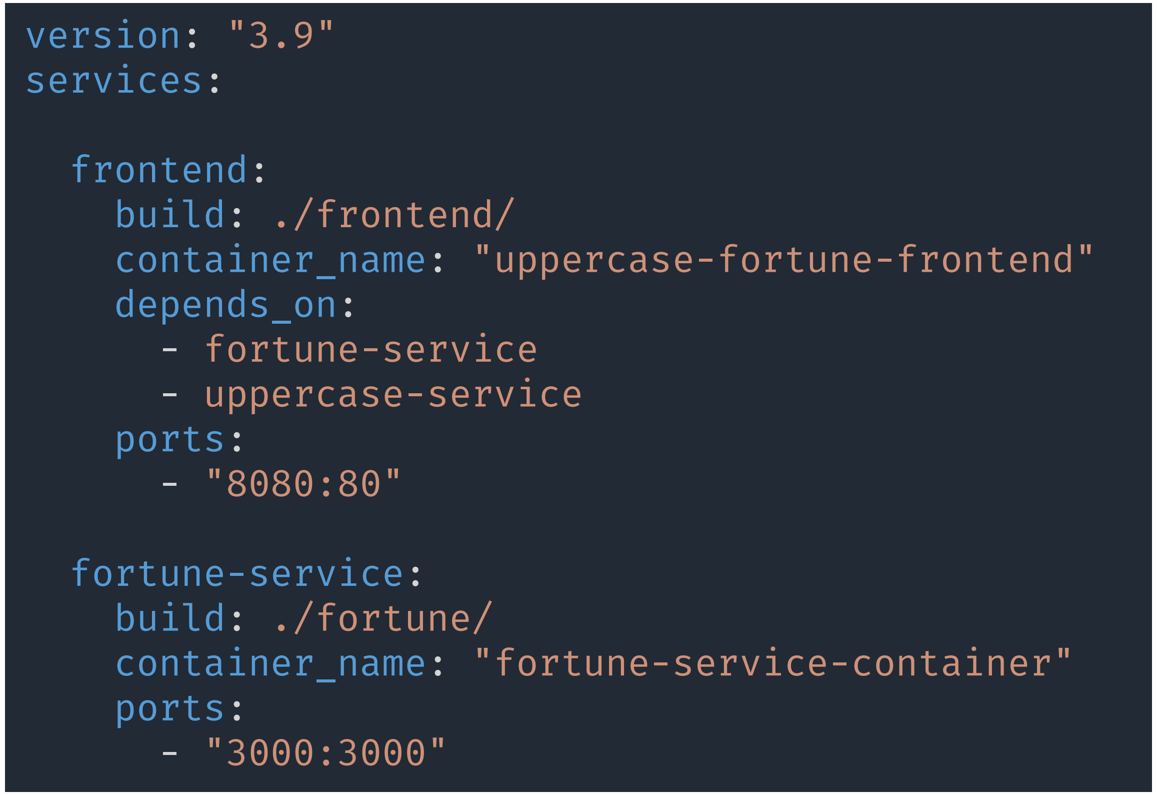


```
# Uppercase service

# 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
```

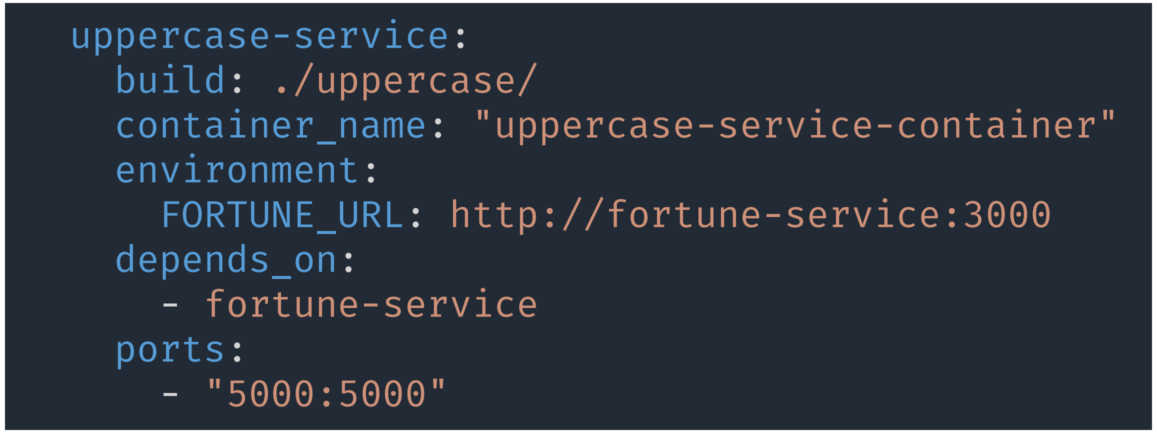
Preparing Docker images for our services

```
# 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
```