P.PORTO

# Methods and Techniques for Software Development

Ricardo Santos | 2019/2020
rjs@estg.ipp.pt

P.PORTO

# *From Monoliths to Microservices* Lesson 4

Methods and Techniques for
Software Development

2019/2020

P.PORTO

# *Evoluir Aplicações Monolíticas para Microserviços* Aula 4

Methods and Techniques for
Software Development

2019/2020

P.PORTO

ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO

*It is very difficult to manage complex enterprise systems*

- Understanding **how** any one **change** will **affect the entire system**
- New developers spend months learning the system's codebase before they can even begin to work on it
- **Even the most knowledgeable of development teams is fearful of making changes** or adding new code that would disrupt operation in some unforeseen way

*Microservices can turn this scenario*, *enabling a new, more agile world in which developers and operations teams work hand in hand to* ***deliver small, loosely coupled bundles of software quickly and safely***

P.PORTO

ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO

*The key challenge is to design and develop the integration between the existing system and the new microservices*

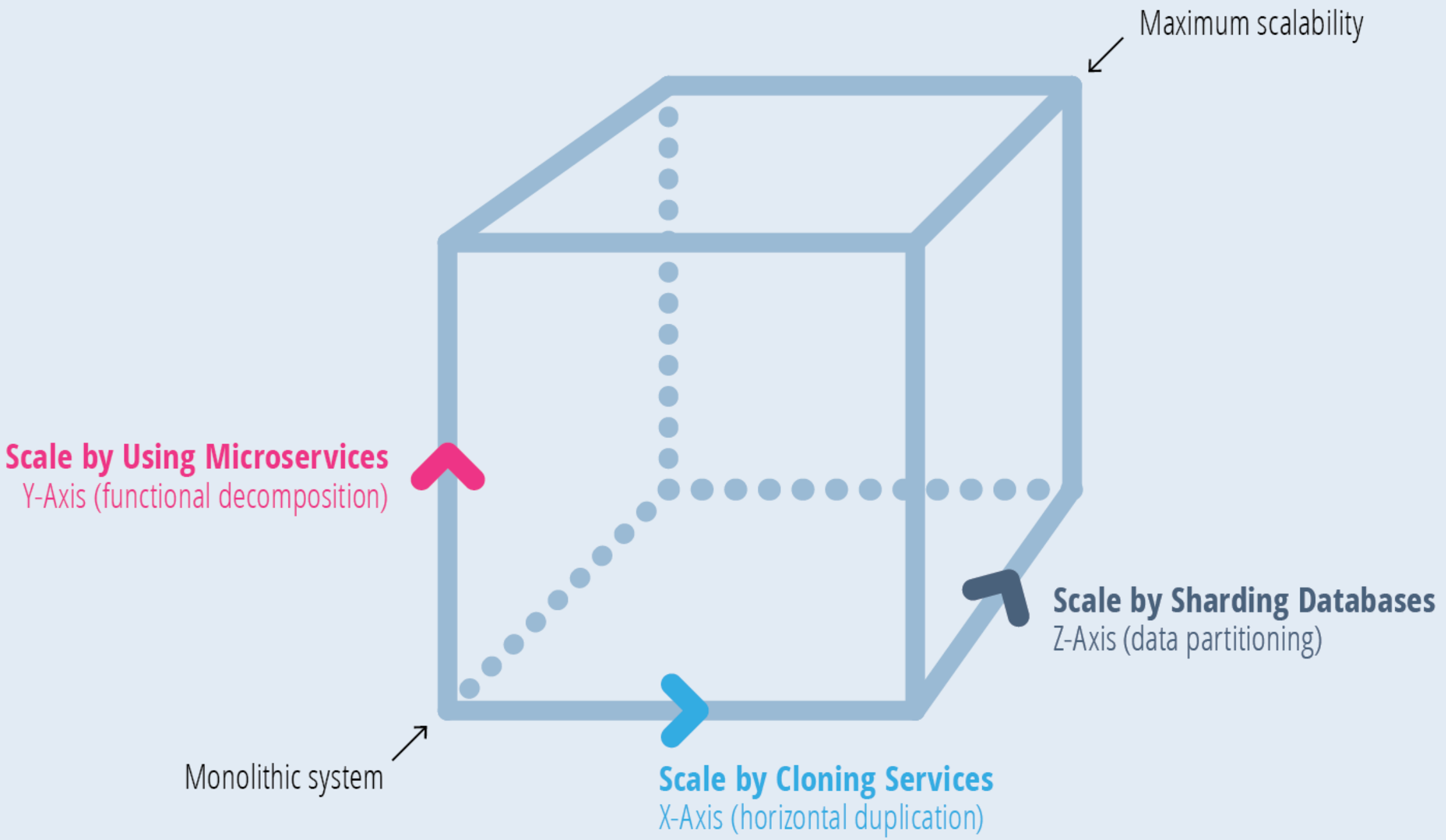
If you can't feed a team with two pizzas, it's too large.

Startup Quote!

**JEFF BEZOS**

*FOUNDER, AMAZON*

# The Scale Cube and Microservices: 3 Dimensions to Scaling

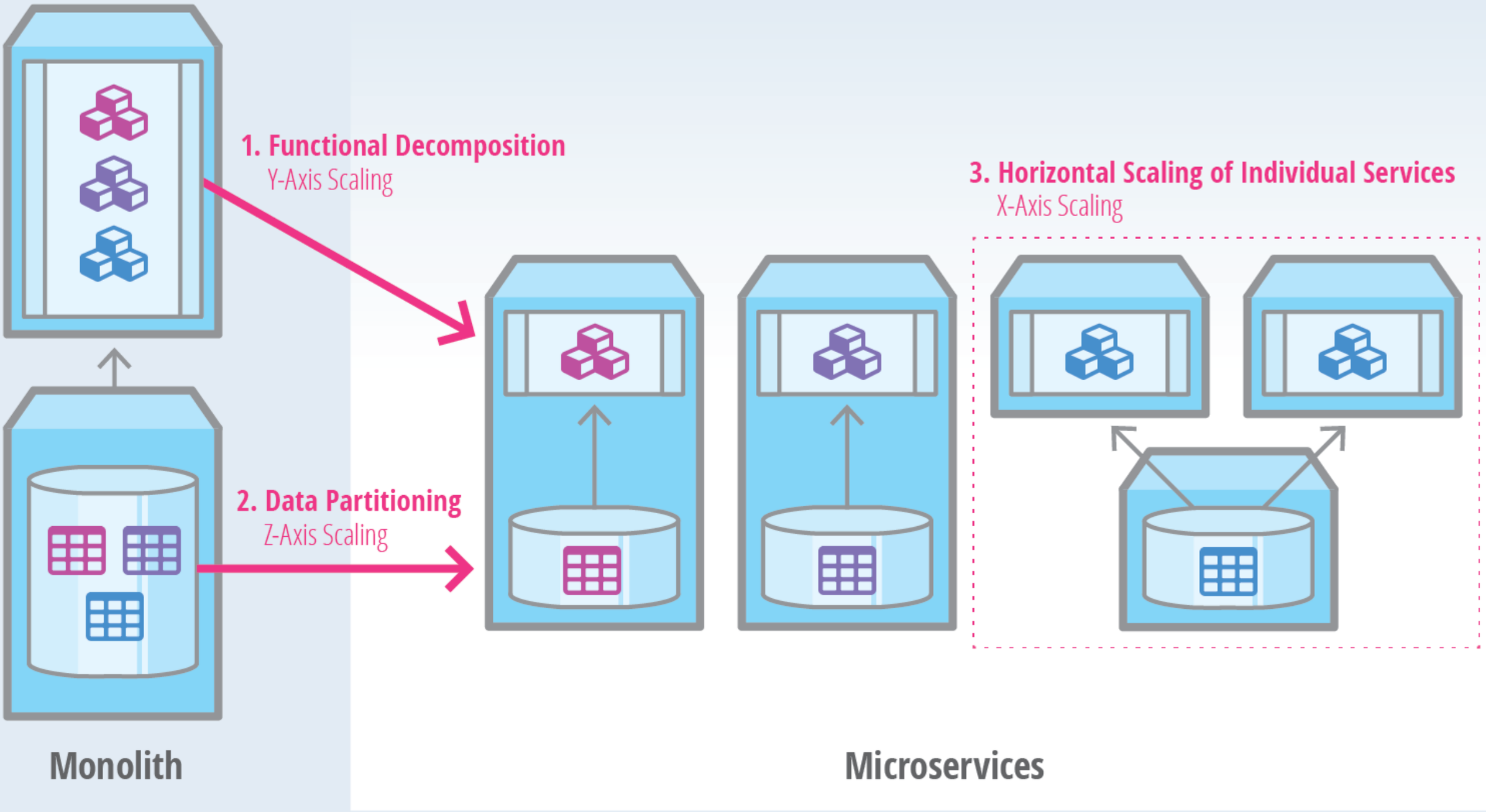



Source: The New Stack. Based on the "The Art of Scalability," by Martin Abbott & Michael Fisher.

THENEWSTACK

# Scaling With Microservices

Microservice architectures have 3 dimensions of scalability




Source: The New Stack. Based on a diagram appearing in "Microservices," by James Lewis & Martin Fowler.

THENEWSTACK

# Monolith First

*you shouldn't start a new* ***project with microservices***, *even if you're sure your application will be big enough to make it worthwhile*

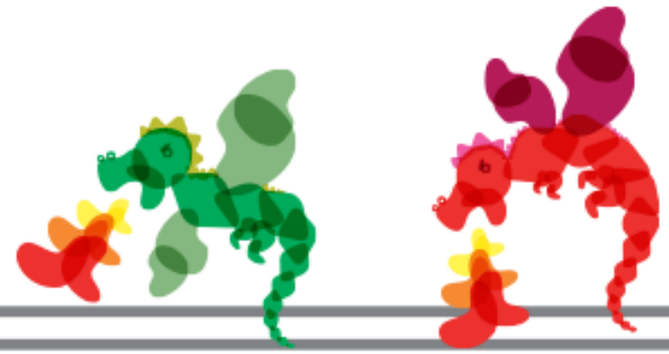
Almost all the successful microservice stories **have started with a monolith** that got too big and was broken up

*Almost all the cases where I've heard of a system that was built as a microservice system from scratch, it has ended up in serious trouble*

P.PORTO

ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO

*Going directly to a microservices architecture is risky*



*A monolith allows you to explore both the complexity of a system and its component boundaries*

*As complexity rises start breaking out some microservices*

*Continue breaking out services as your knowledge of boundaries and service management increases*

# Refactoring to Microservices

- The process of transforming a monolithic application into microservices is a **form of application modernization**
- That is something that developers have been doing for decades
- There are some ideas that we can reuse when refactoring an application into microservices

# Big Bang Strategy

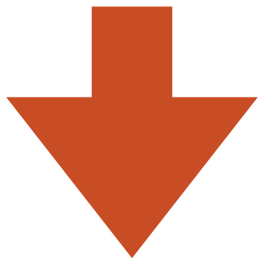
- you focus all of your development efforts on building a new microservices–based application from scratch
- it is extremely risky and will likely end in failure

*the only thing a Big Bang rewrite guarantees is a Big Bang!"*

P.PORTO

ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO

*you should incrementally refactor your monolithic application*



# *Strangler Application Pattern*

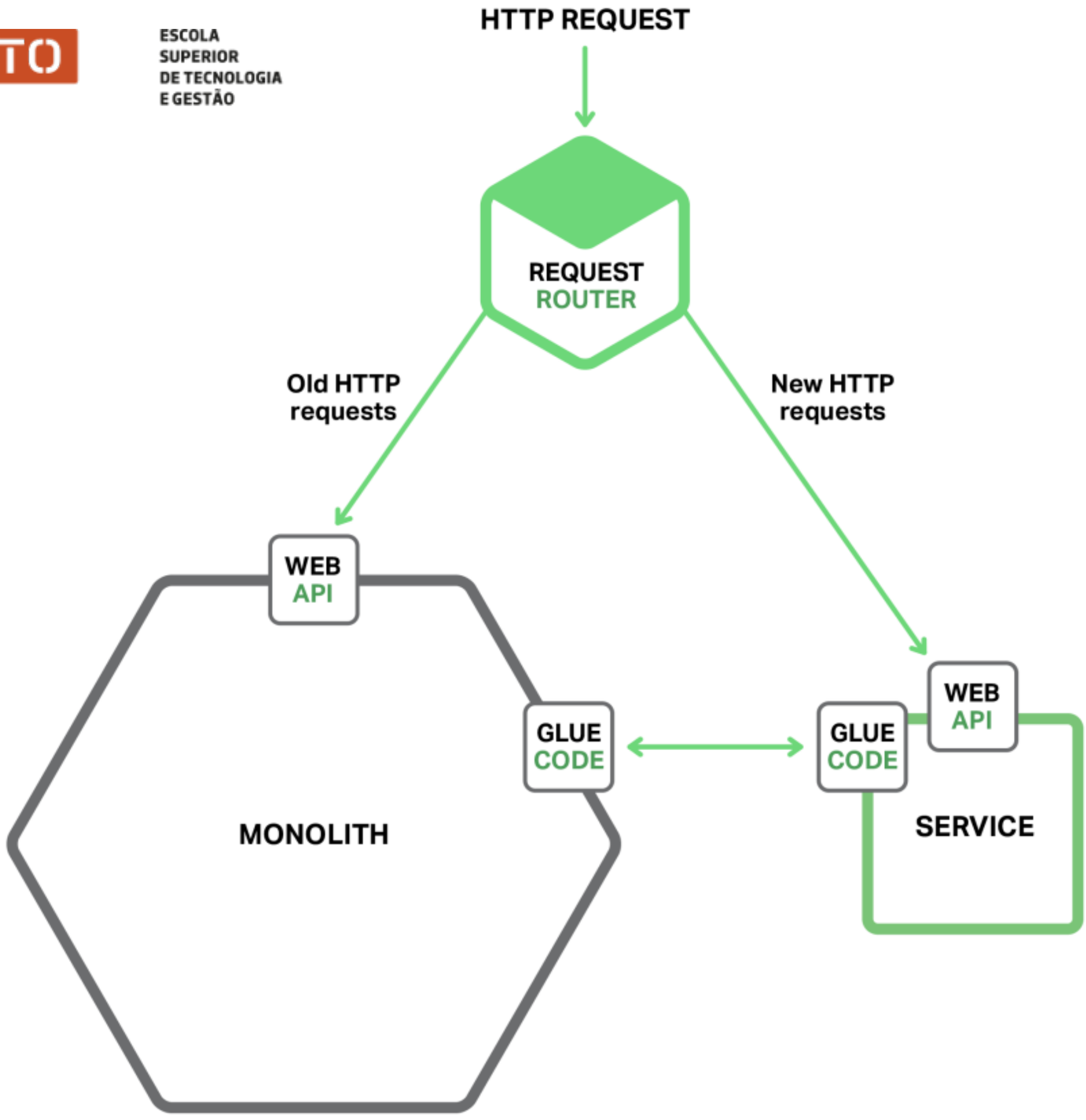
Martin Fowler, 2004

# Strategy 1 - Stop Digging

- you should **stop making the monolith bigger**
- when you are implementing new functionality you **should not add more code to the monolith**
- put that **new code in a standalone microservice**

P.PORTO

ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO



- Monolith and new microservice only occasionally live next to each other without a need of accessing their data

- To ensure that they can talk one to each other, glue code is needed and it can be provided in three ways:

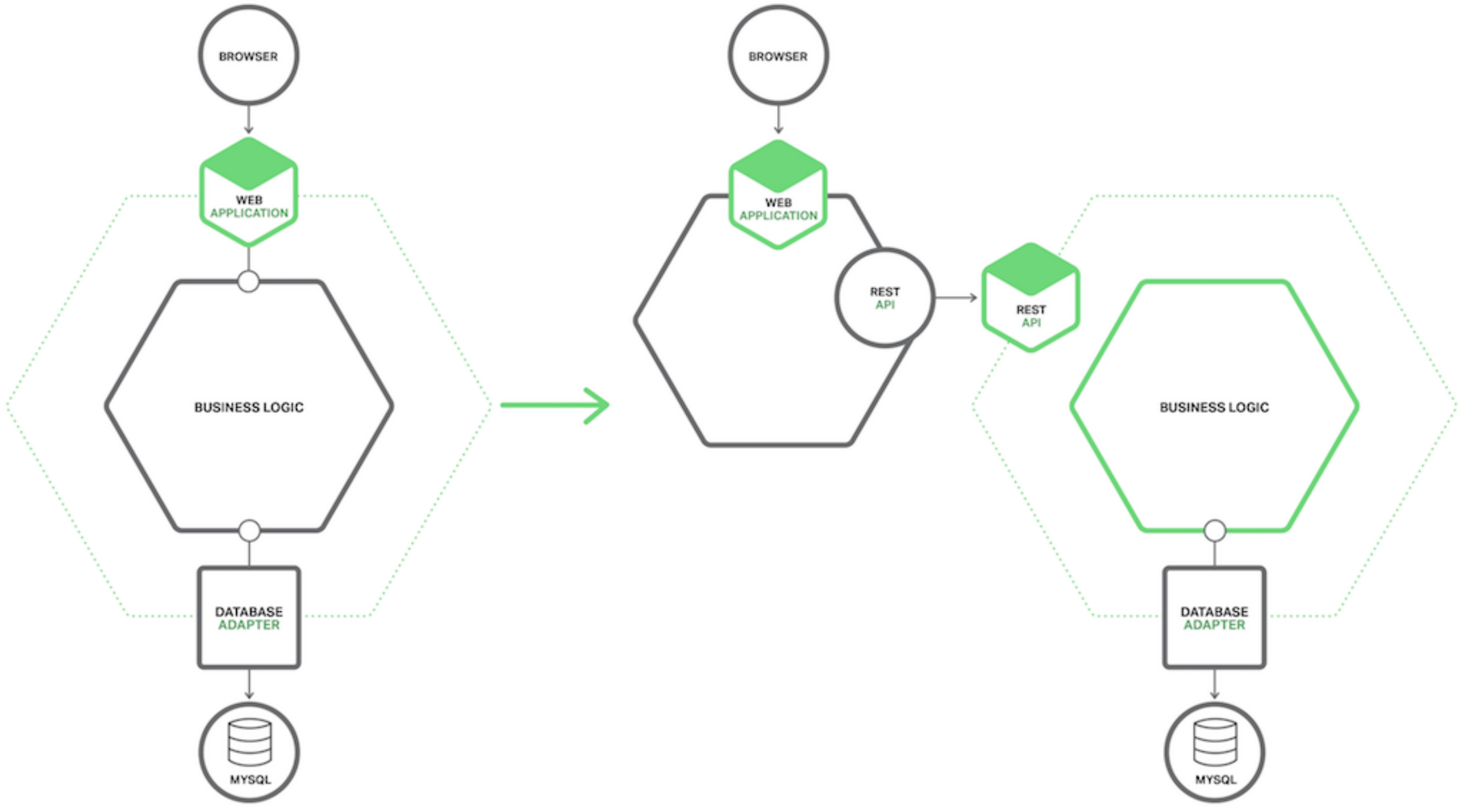
  - Allowing direct database access

  - Maintaining its own copy of the data

  - Creating a remote API

# **Strategy 2 -** Split Frontend and Backend

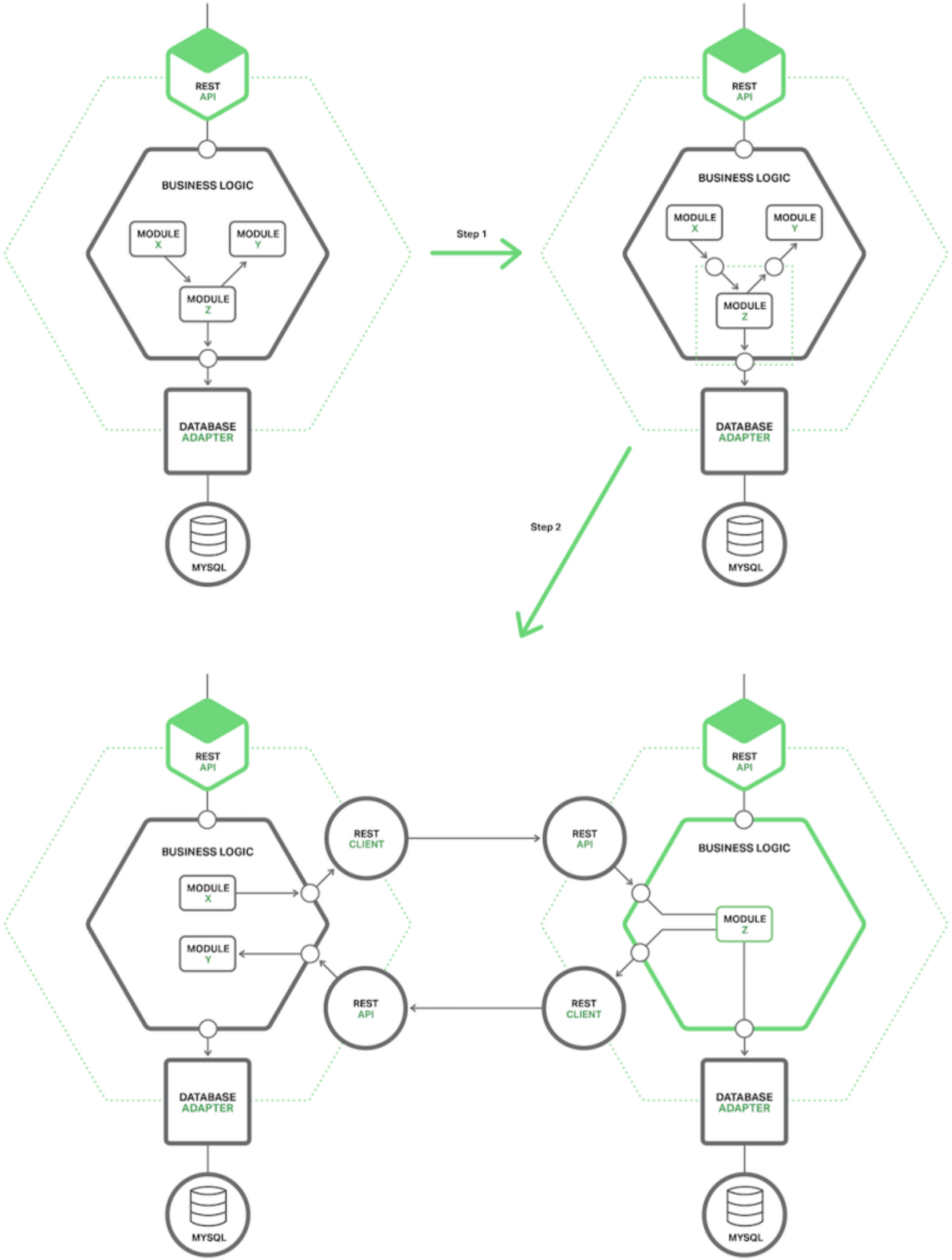
- split the presentation layer from the business logic and data access layers
- splitting a monolith in this way has two main benefits:
  - It enables you to develop, deploy, and scale the two applications independently of one another
  - another benefit of this approach is that it exposes a remote API that can be called by the microservices that you develop


BROWSER
WEB
APPLICATION
BUSINESS LOGIC
DATABASE
ADAPTER
MYSQL
BROWSER
WEB
APPLICATION
REST
API
REST
API
BUSINESS LOGIC
DATABASE
ADAPTER
MYSQL

# **Strategy 3 -** Extract Services

- Turn existing modules within the monolith into standalone microservices
- Each time you extract a module and turn it into a service, the monolith shrinks
- Once you have converted enough modules, the monolith will cease to be a problem
- Either it disappears entirely or it becomes small enough that it is just another service.

P.PORTO

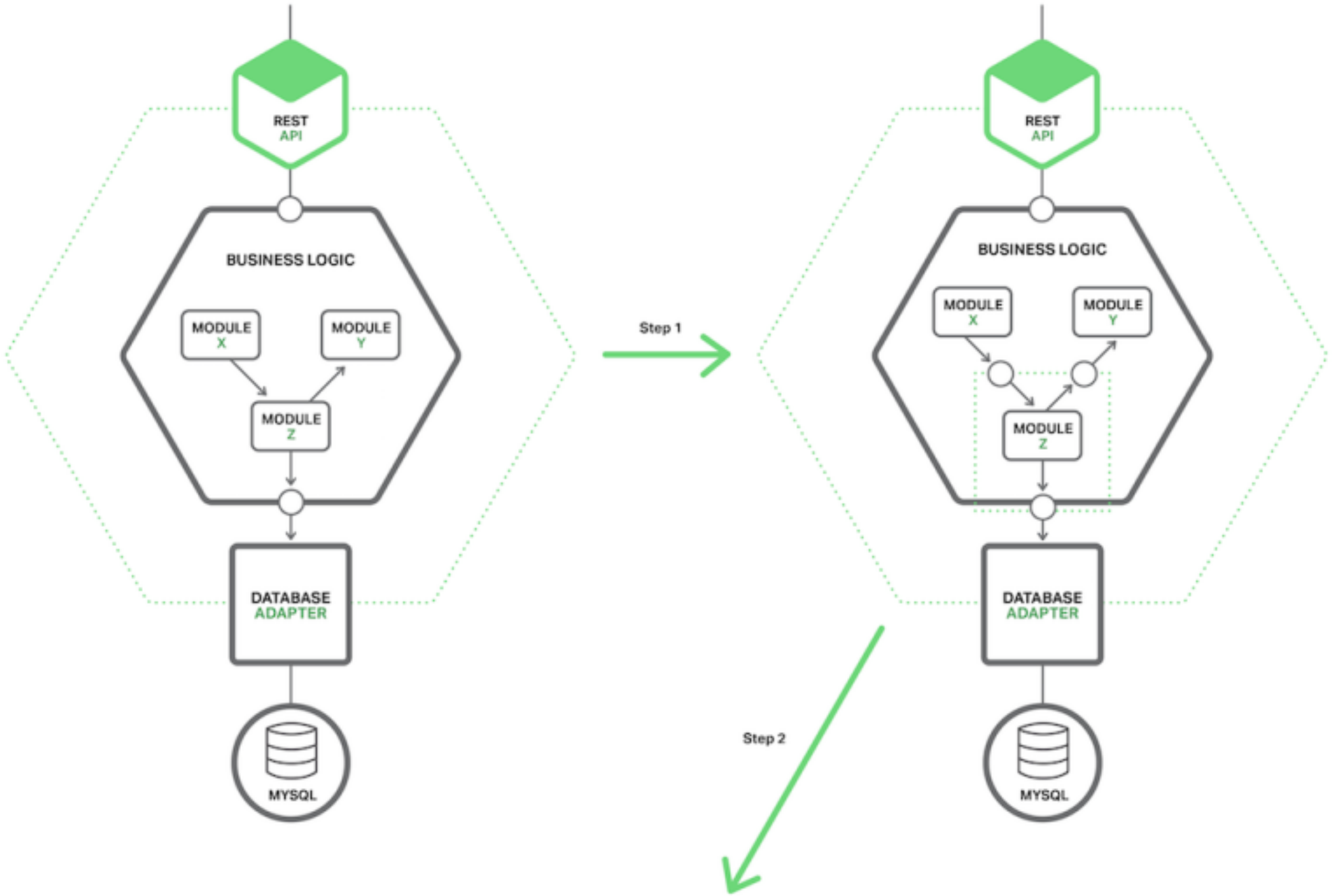

REST
API
BUSINESS LOGIC
MODULE
X
MODULE
Y
MODULE
Z
DATABASE
ADAPTER
MYSQL
Step 1
Step 2
REST
CLIENT

P.PORTO

ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO

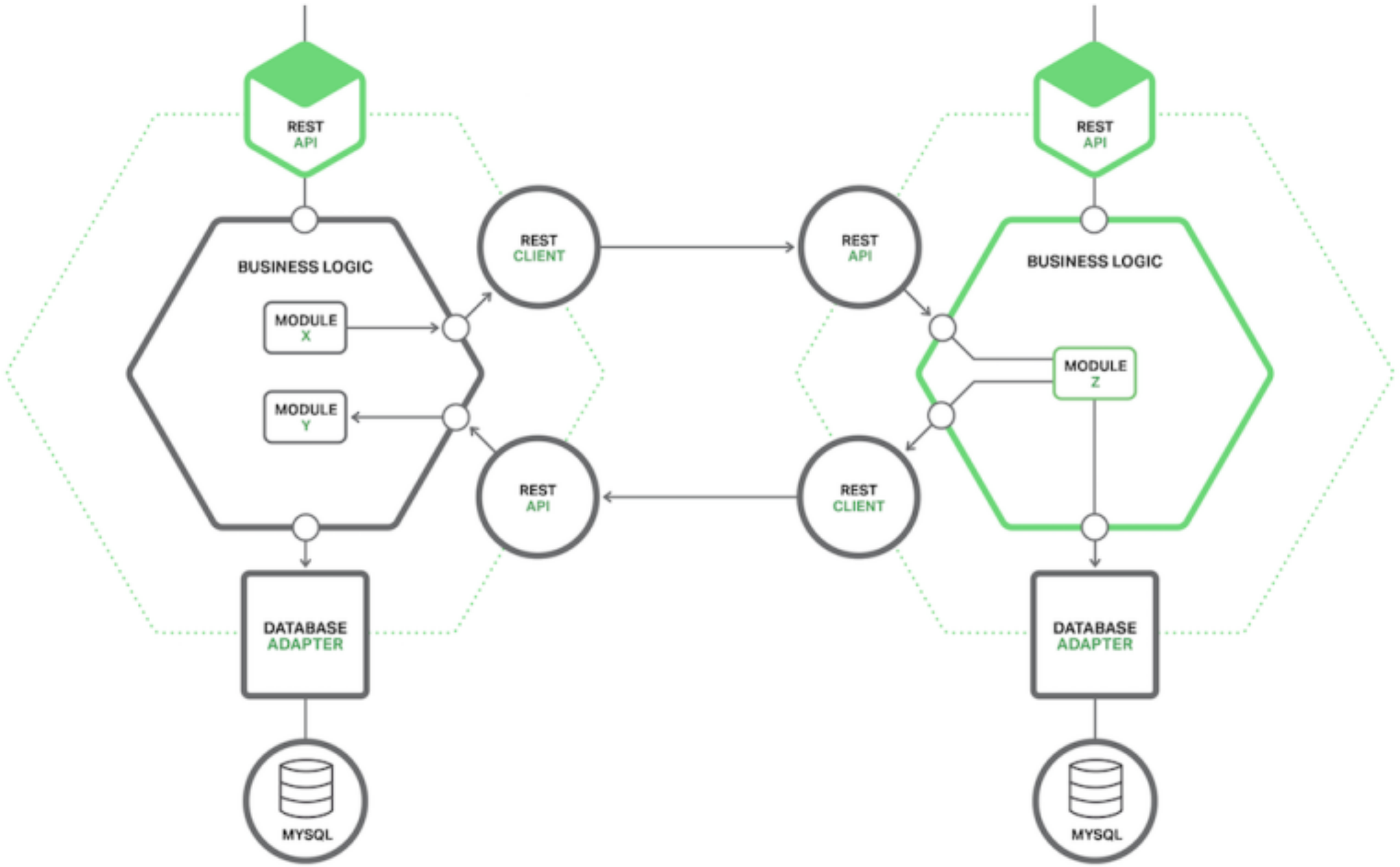
# *The final phase of breaking up a monolith to microservices consists of two steps*

- Step 1
  - All dependencies between the monolith and the new service are untangled by defining the API between them
  - Once the API is defined, communication with the monolith can be restricted to API only, which prepares the service to be completely detached


REST
API
BUSINESS LOGIC
MODULE
X
MODULE
Y
MODULE
Z
DATABASE
ADAPTER
MYSQL
Step 1
REST
API
BUSINESS LOGIC
MODULE
X
MODULE
Y
MODULE
Z
DATABASE
ADAPTER
MYSQL
Step 2

## Step 2

- the service is moved to standalone microservice running in a separate process
- After this step, a new microservice can be developed, scaled and deployed independently to monolith


REST
API
BUSINESS LOGIC
MODULE
X
MODULE
Y
REST
CLIENT
REST
API
REST
API
BUSINESS LOGIC
MODULE
Z
REST
API
REST
CLIENT
DATABASE
ADAPTER
DATABASE
ADAPTER
MYSQL
MYSQL

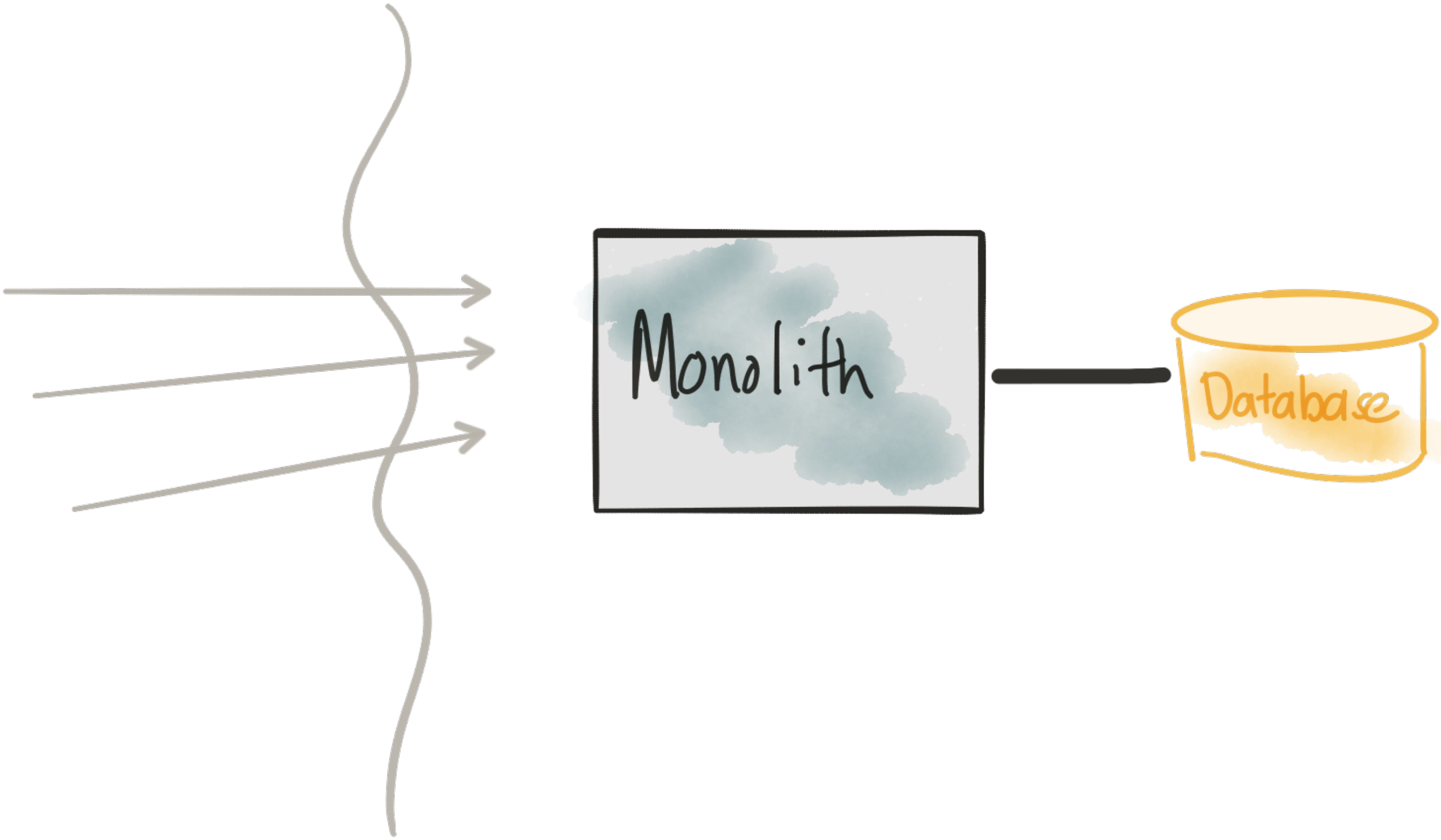
P.PORTO

ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO

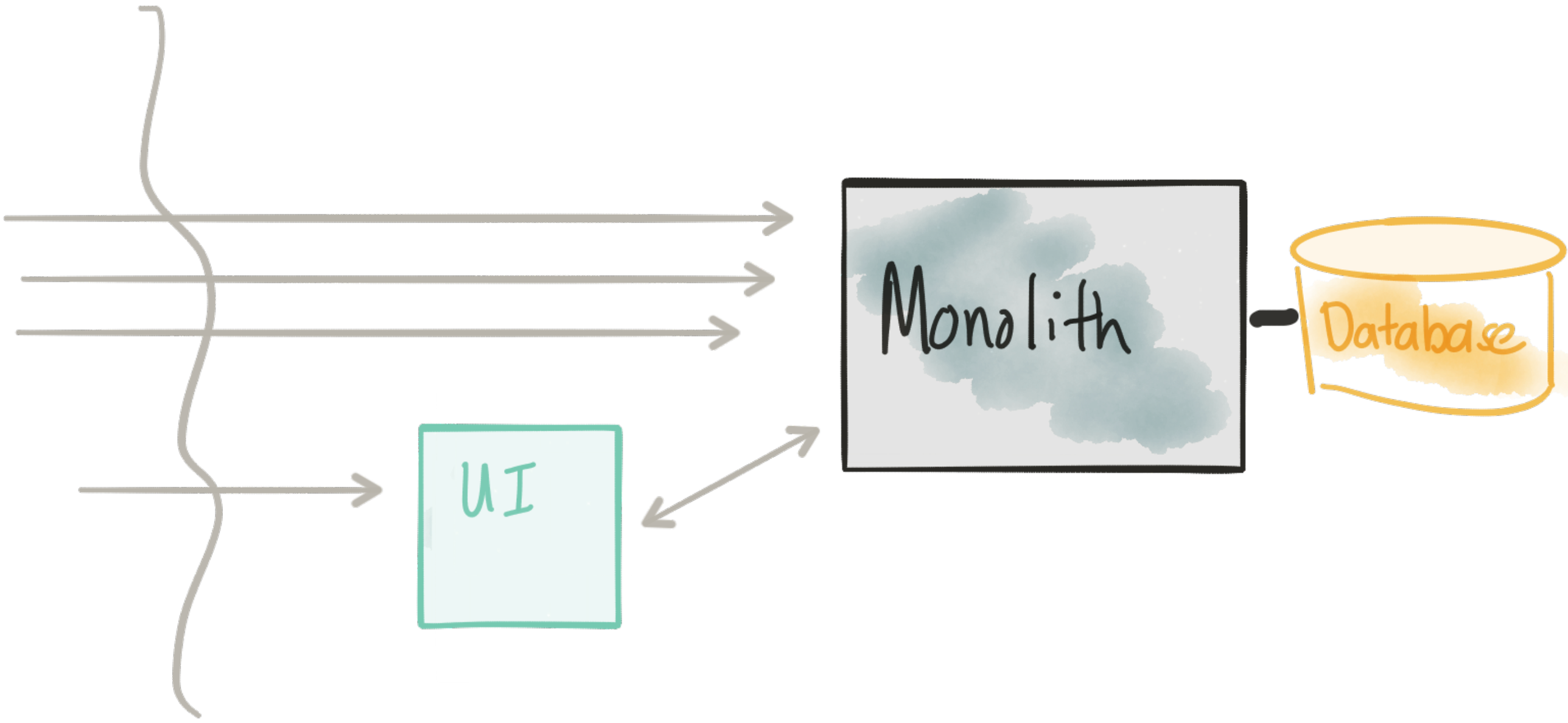
# Monolith to Microservice Evolution

## *Concrete example*

# Meet the monolith



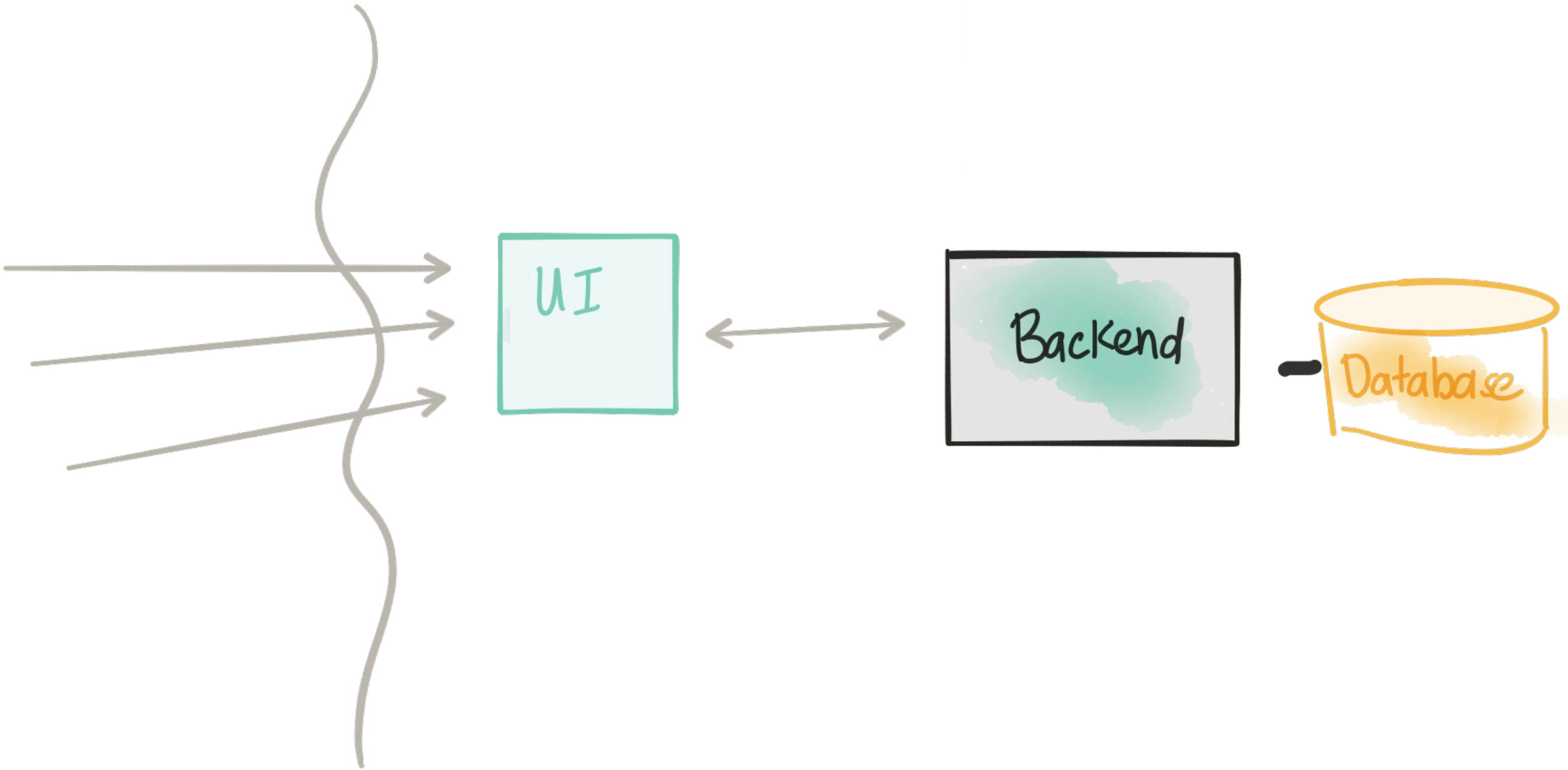
# Extract the UI



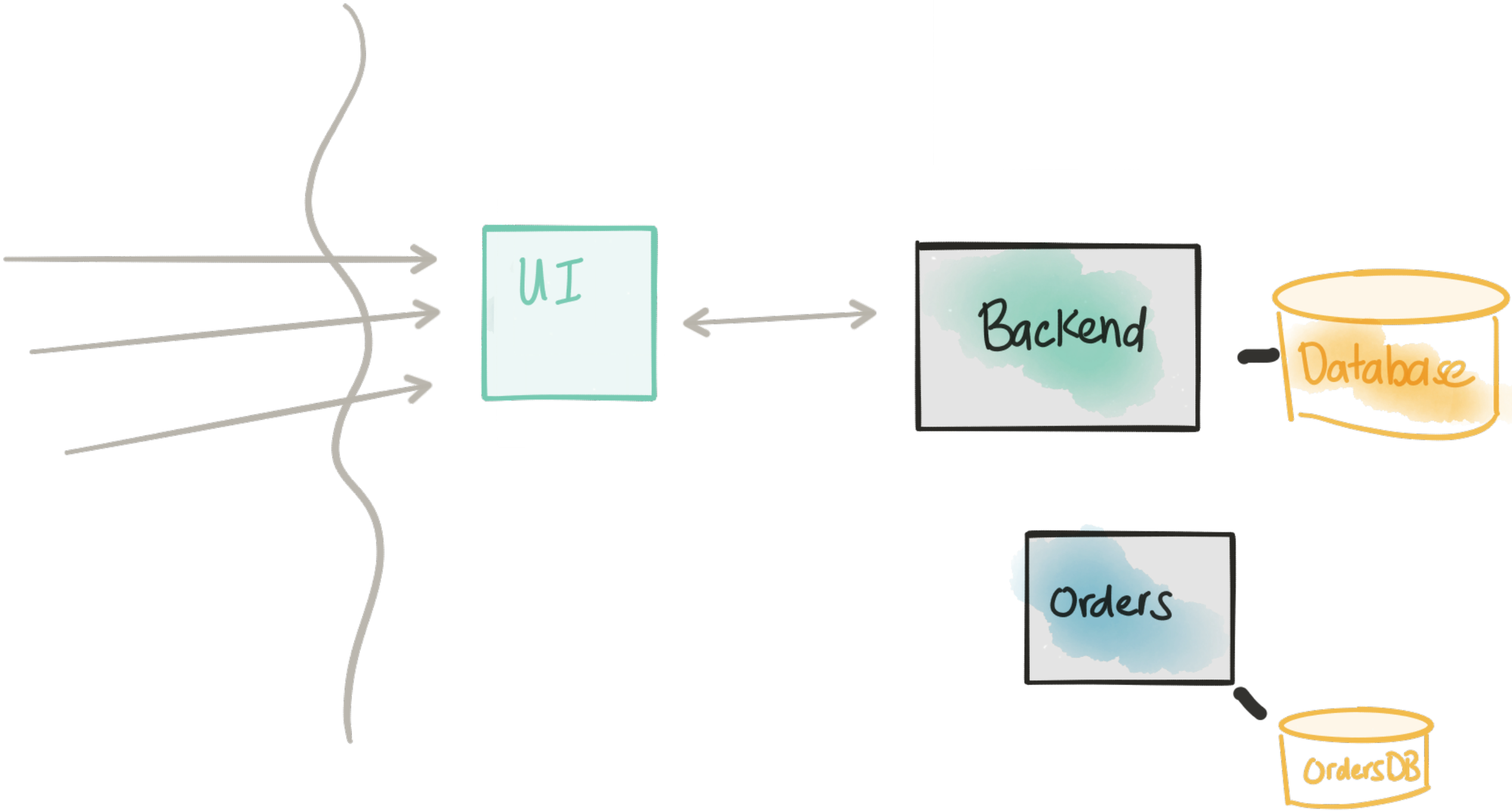
P.PORTO

ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO

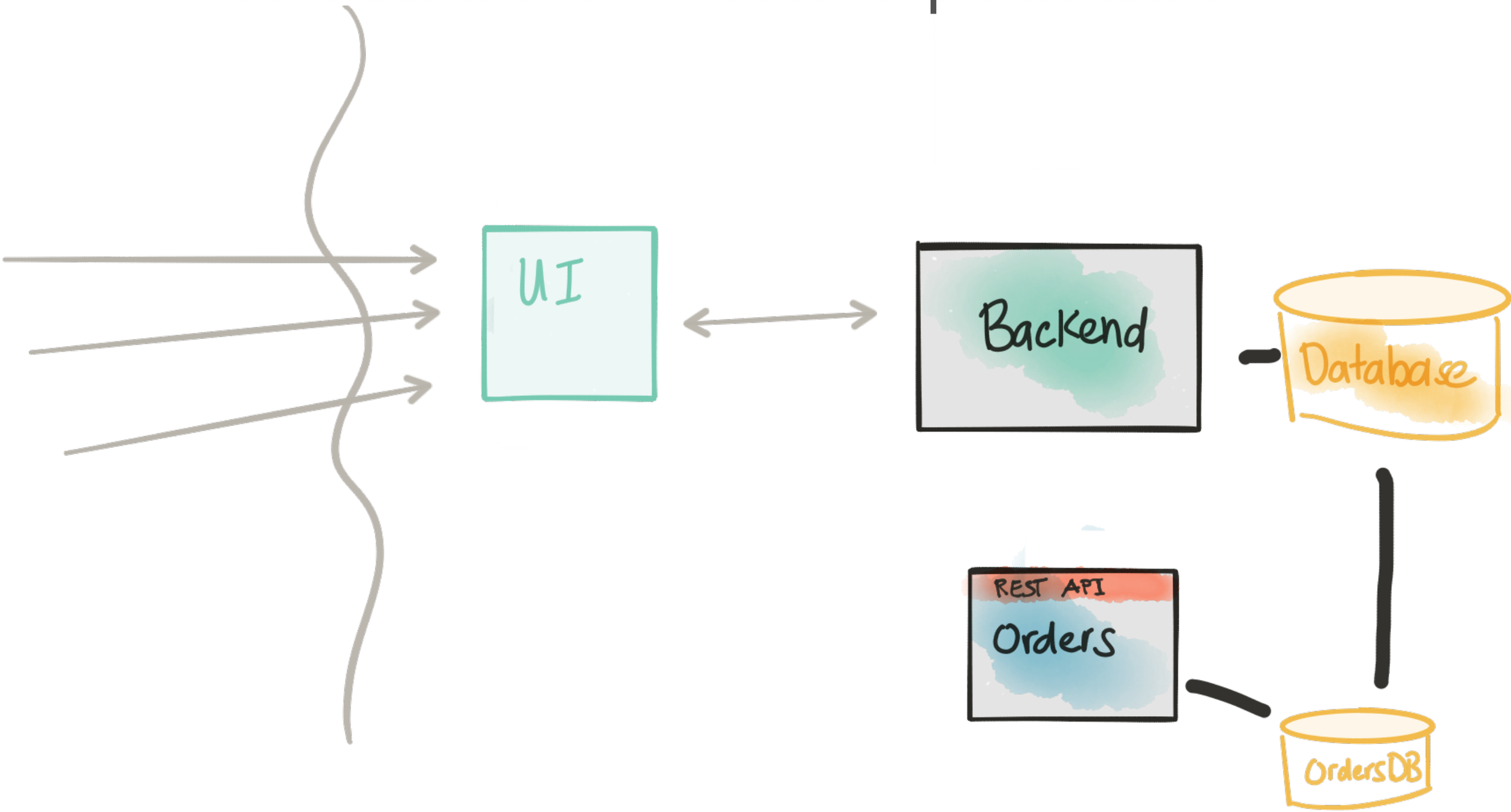
# Drop the UI from the monolith



# Introduce a new service

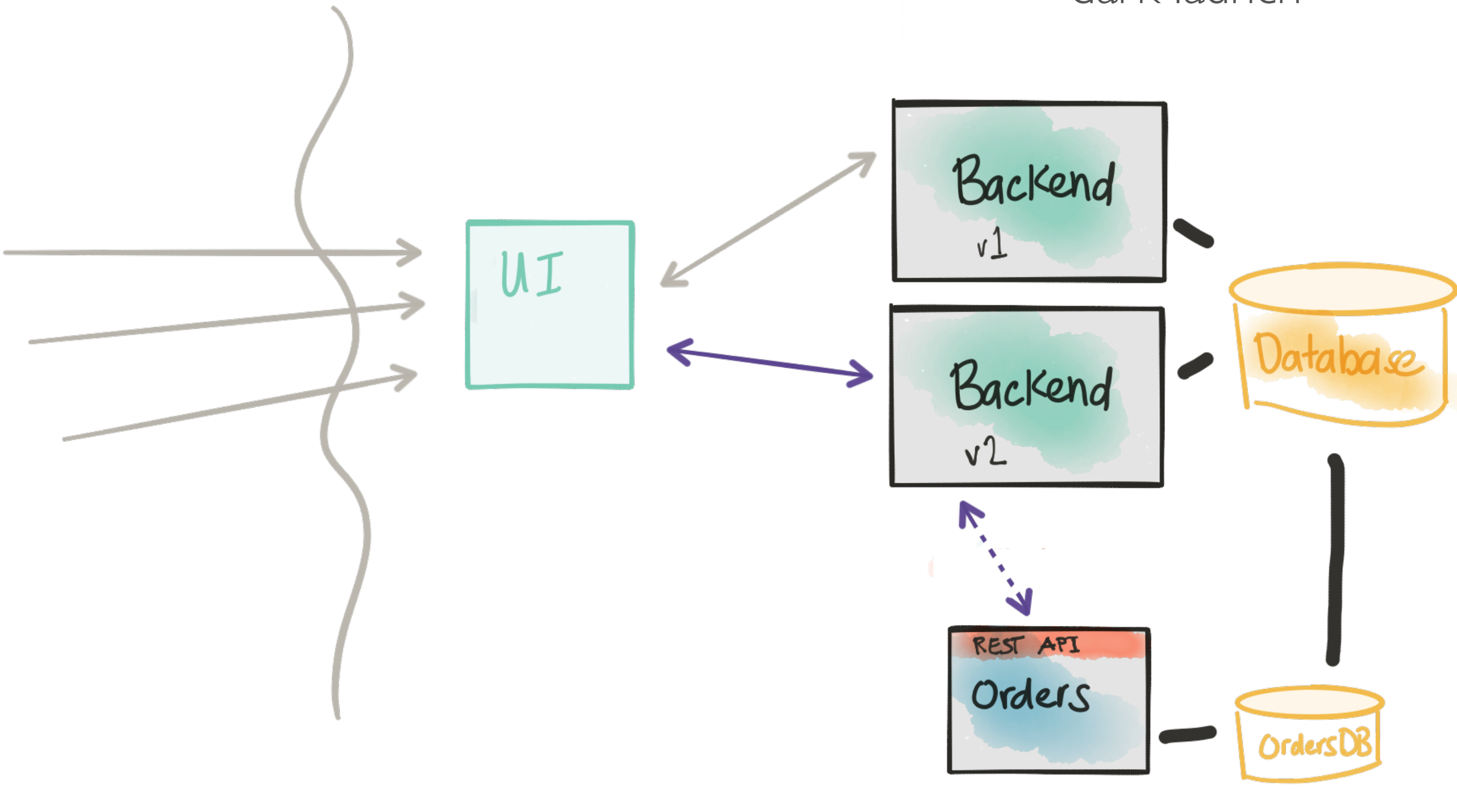


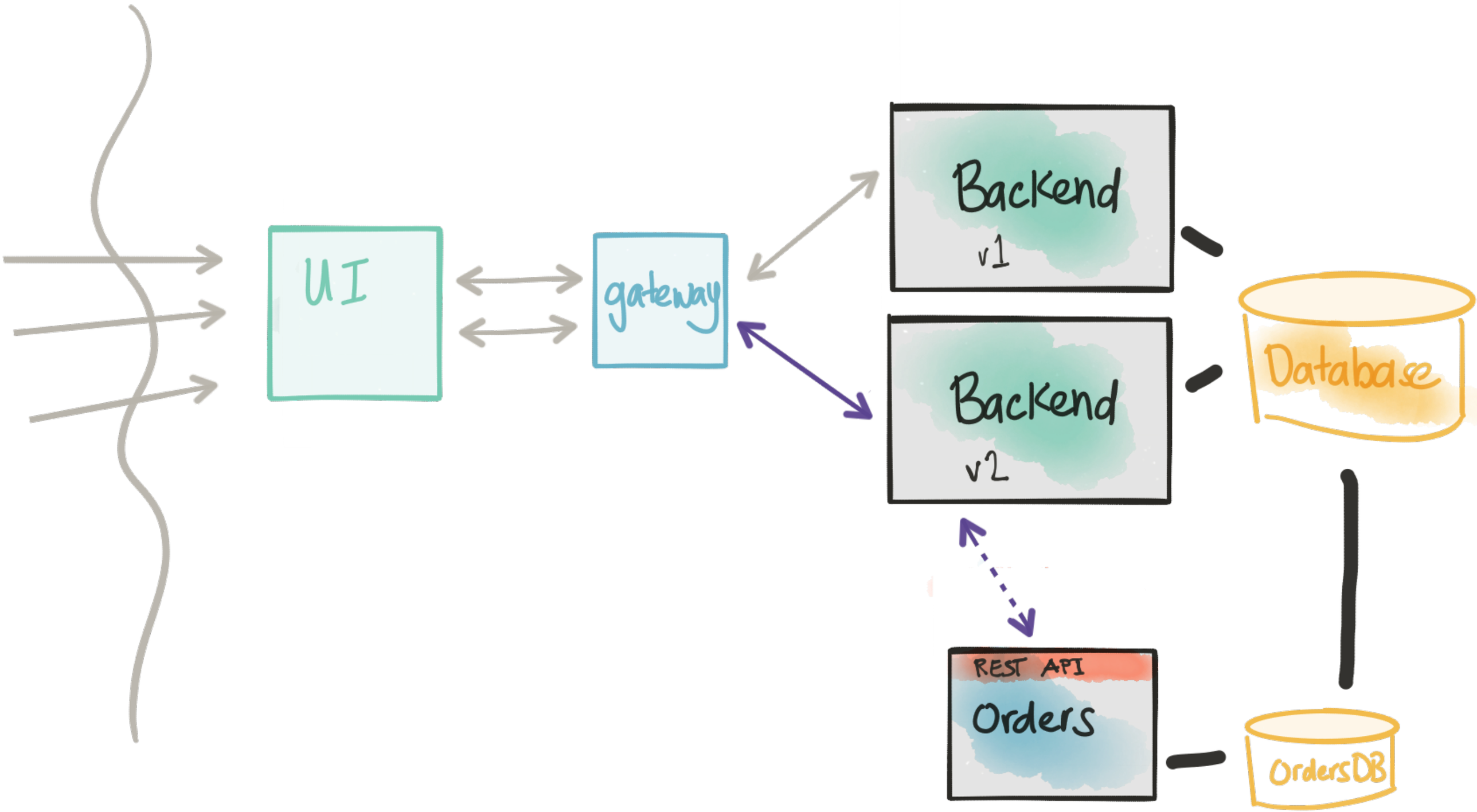
# Connect the API with an implementation



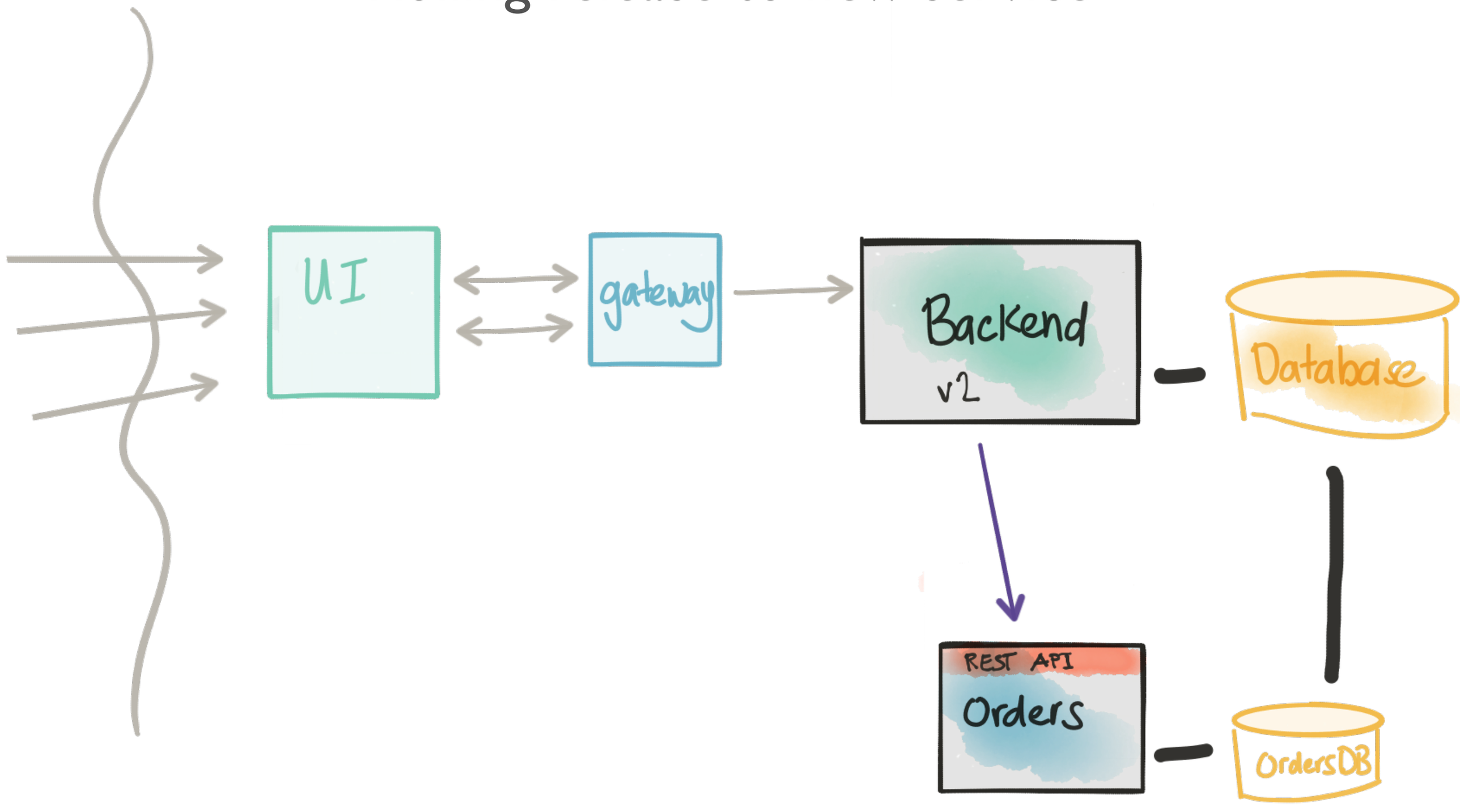
# Start sending shadow traffic to the new service

## dark launch

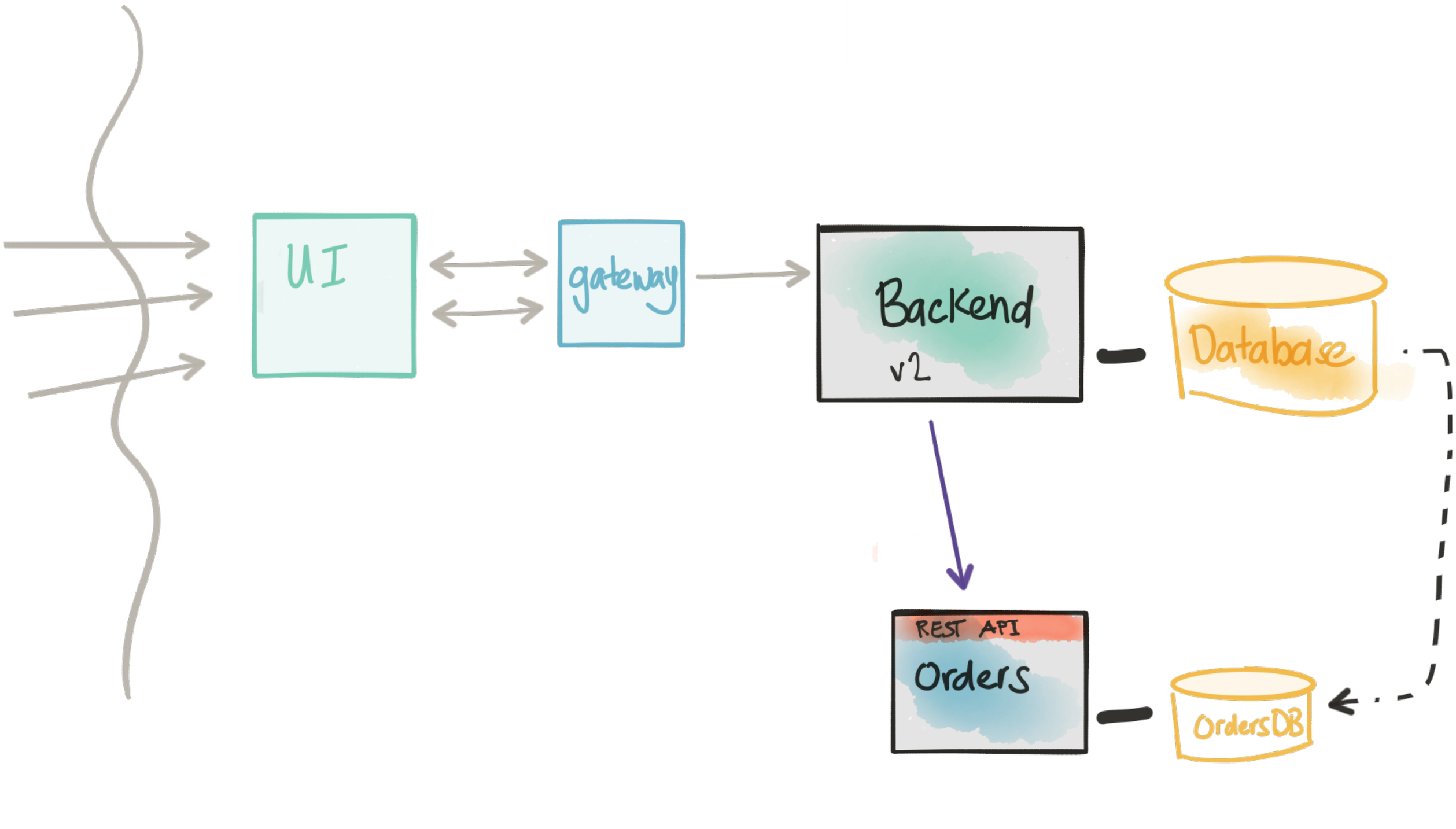



UI
gateway
Backend
v1
Backend
v2
Database
REST API
Orders
OrdersDB

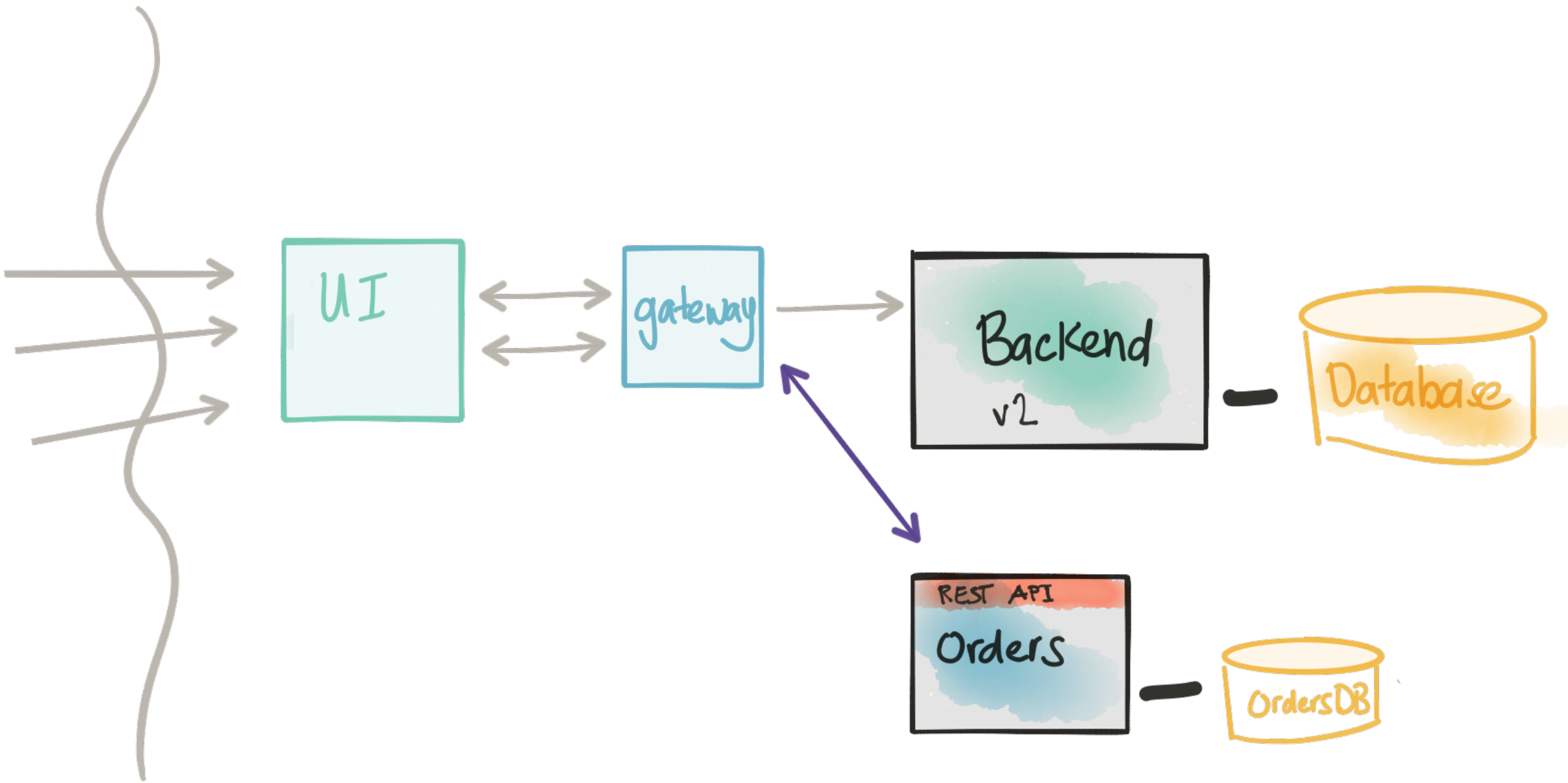
# Rolling release to new service



# Offline data ETL/migration



# Disconnect/decouple the datastore



kubernetes

# *Hands on lab*

## Kubernetes

- Pods
- Services

kubernetes

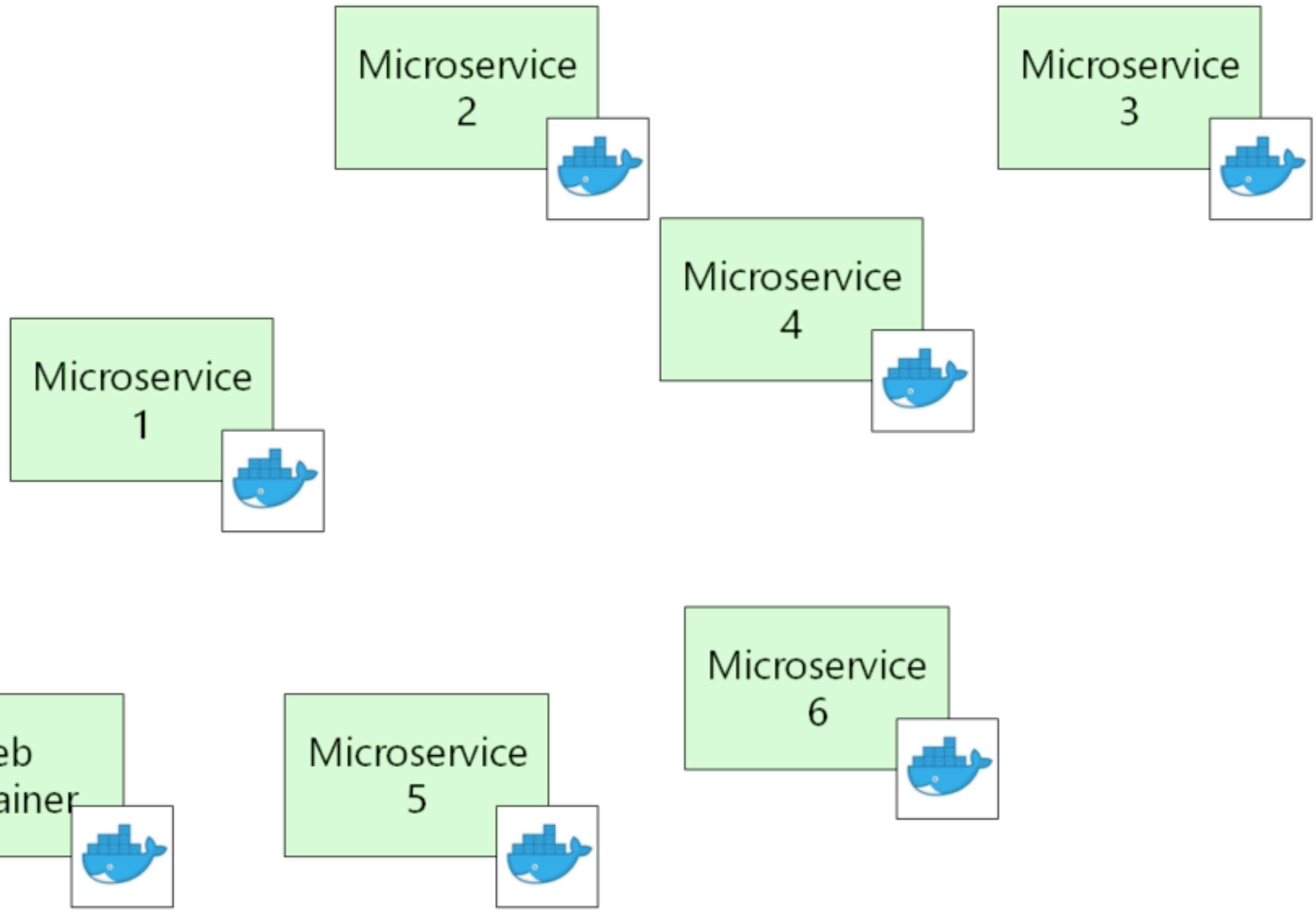
# *PODs*

# What is a Pod?

https://kubernetes.io/docs/concepts/workloads/pods/pod-overview/

*Pods are the most basic concept on kubernetes*

# What are we planning to do?



- Our job is to deploy a micro service architecture to the cloud
- Can we manage it ourselves?

***We could log on to an Amazon AWS node and deploy each Microservice***

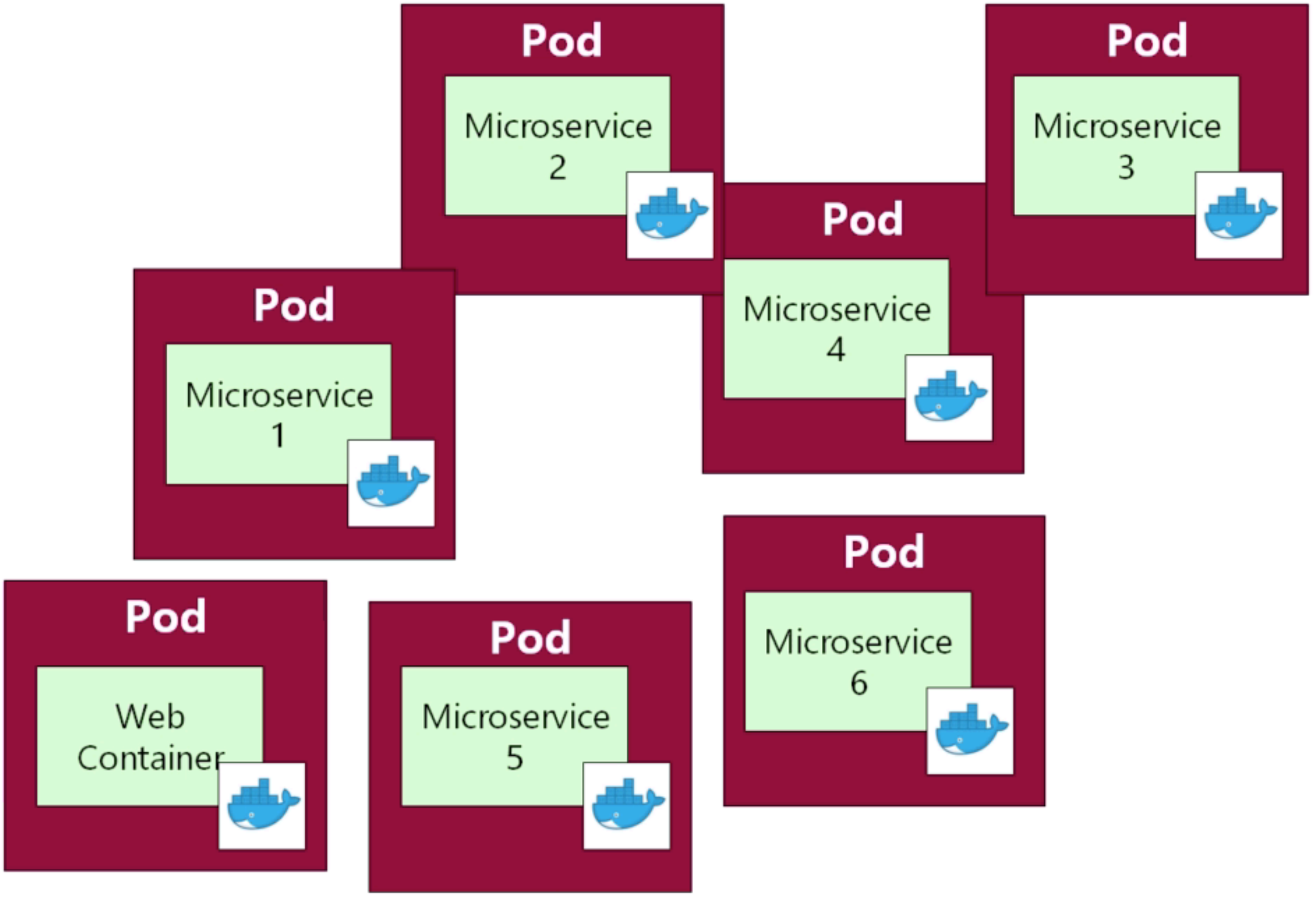
P.PORTO

ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO

# *but who manages it?*

## ***It will be to much work...***

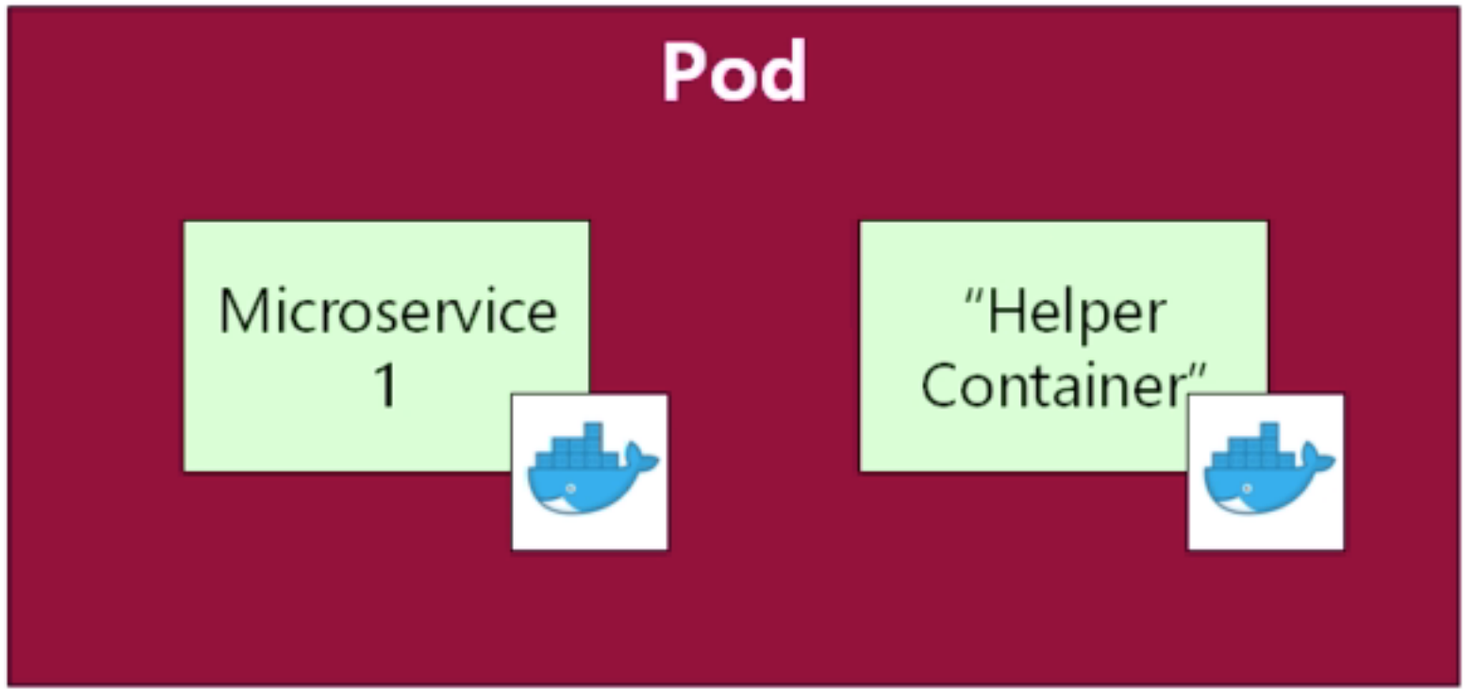
- We will use kubernetes to orchestrate this system
- Responsible for starting and stoping these containers


Pod
Microservice 2
Pod
Microservice 3
Pod
Microservice 4
Pod
Microservice 1
Pod
Web Container
Pod
Microservice 5
Pod
Microservice 6

P.PORTO

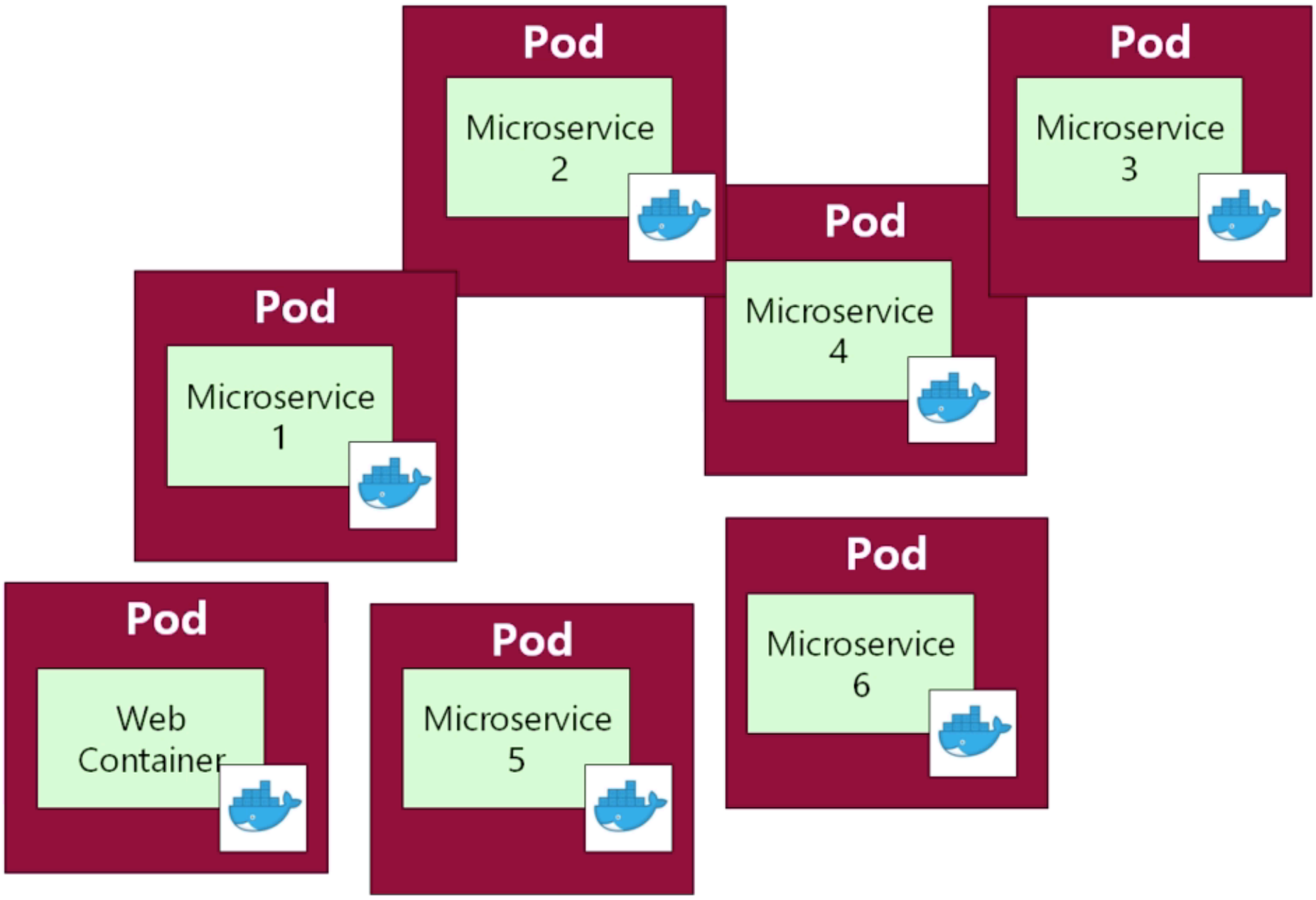
ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO

# *each pod can be seen as a wrapper to the container*




- We can have more that one container per pod
  - but it's not recommended
  - It's rare to see an architecture like that

- Kubernetes will manage these pods



- Pods are the most basic unit of deployment on kubernetes

*Now let's write our first pod!*

P.PORTO

ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO

**release0** 9.76 MB

Last updated **a year ago** by richardchesterwood

| DIGEST | ARCHITECTURE | OS | SIZE |
|---|---|---|---|
| 9b98fec20772 | amd64 | linux | 9.76 MB |

**release0-5** 9.76 MB

Last updated **a year ago** by richardchesterwood

| DIGEST | ARCHITECTURE | OS | SIZE |
|---|---|---|---|
| 636f57ab20c3 | amd64 | linux | 9.76 MB |

**release2** 25.07 MB

Last updated **a year ago** by richardchesterwood

| DIGEST | ARCHITECTURE | OS | SIZE |
|---|---|---|---|
| ed7d720878ac | amd64 | linux | 25.07 MB |

**release1** 25.07 MB

Last updated **a year ago** by richardchesterwood

| DIGEST | ARCHITECTURE | OS | SIZE |
|---|---|---|---|
| 46bef951fc3c | amd64 | linux | 25.07 MB |

# How do we make a pod to deploy this image?

***We want kubernetes to manage it!***

- **`Release0`** image is the first attempt to show a map

# Go to the documentation of a Pod

## API Object

Pod is a top-level resource in the Kubernetes REST API. The Pod API object definition describes the object in detail.

## Pod v1 core

kubectl example | curl example

Pod Config to print "Hello World".

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-example
spec:
  containers:
  - name: ubuntu
    image: ubuntu:trusty
    command: ["echo"]
    args: ["Hello World"]
```

***on the API page***

# YAML file for a basic pod

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-example
spec:
  containers:
  - name: ubuntu
    image: ubuntu:trusty
    command: ["echo"]
    args: ["Hello World"]
```

**give the pod a name - think in a good name**

**name of the container in docker**

**image that is going to be build from**

**optional**

- First you need to see if minikube is up and running - **`minikube status`**

```
~                                                                    ✓ 86 10:21:22
$ minikube status
host: Running
kubelet: Running
apiserver: Running
kubectl: Correctly Configured: pointing to minikube-vm at 192.168.99.100
```

- Create a YAML text file for the pod - **`pod.yaml`**

pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: webapp
  labels:
    app: webapp
    release: "0"
spec:
  containers:
  - name: webapp
    image: richardchesterwood/k8s-fleetman-webapp-angular:release0
```

- `ls` on the folder of the file

```
~/mtsds
$ ls
pod.yaml
```

- `kubectl get all`

```
~/mtsds
$ kubectl get all
NAME                         READY   STATUS    RESTARTS   AGE
service/kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP
```

# kubectl apply -f pod.yaml

```
~/mtsds
$ kubectl apply -f pod.yaml
pod/webapp created
```

# kubectl get all

```
~/mtsds
$ kubectl get all
NAME                     READY   STATUS    RESTARTS   AGE
pod/webapp               1/1     Running   0          58s

NAME                     TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
service/kubernetes       ClusterIP   10.96.0.1       <none>        443/TCP
```

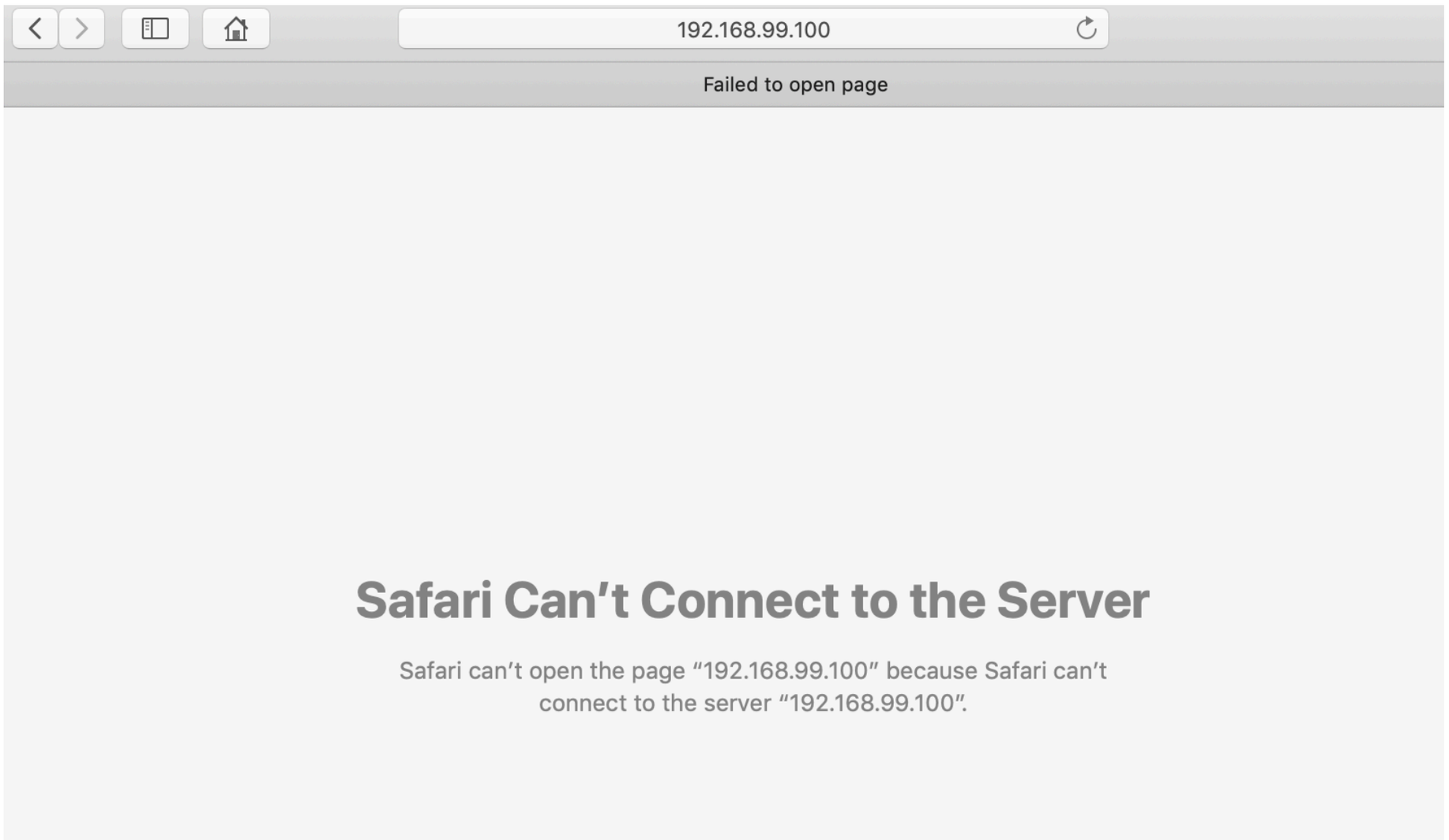
P.PORTO

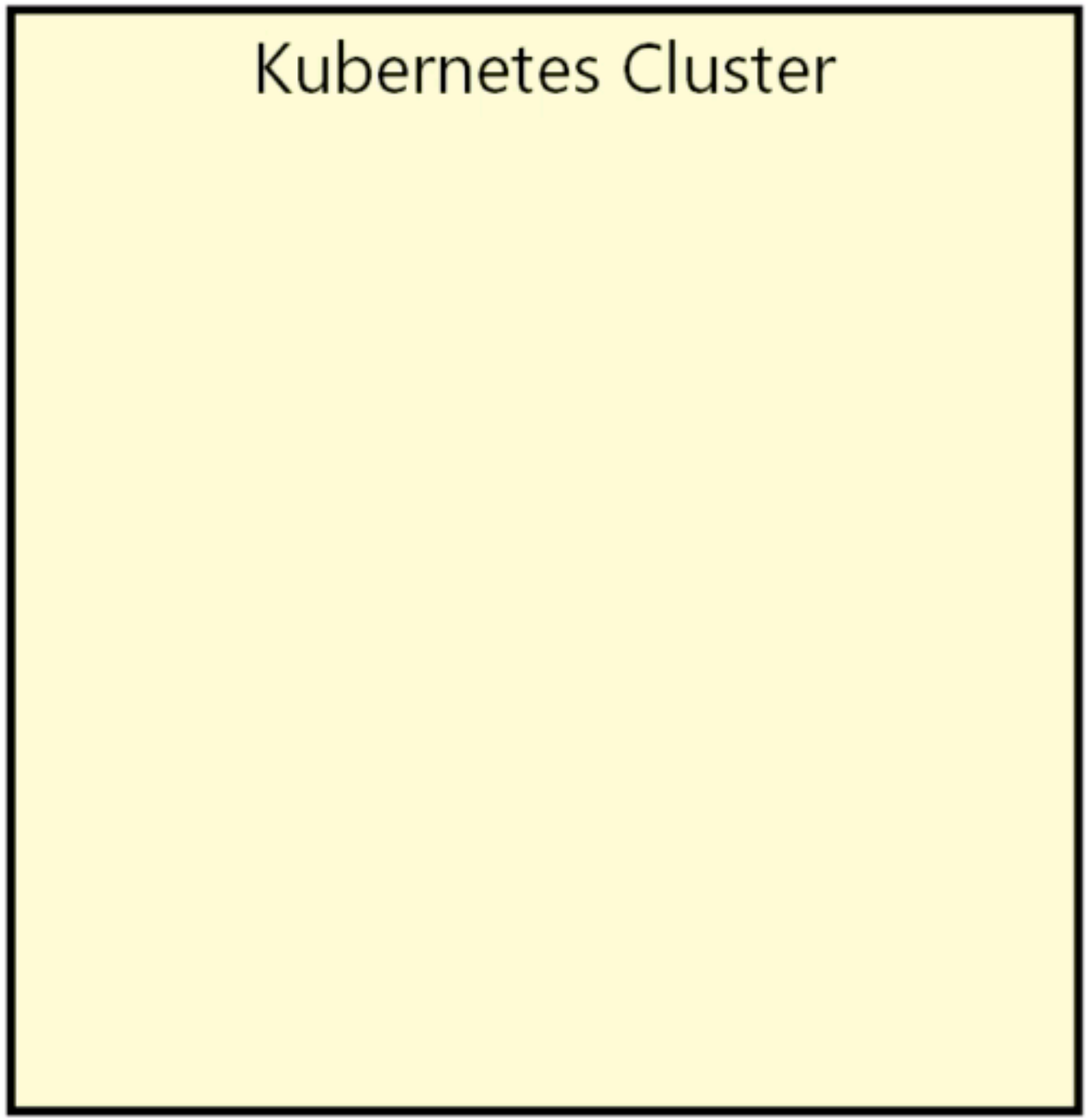
ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO

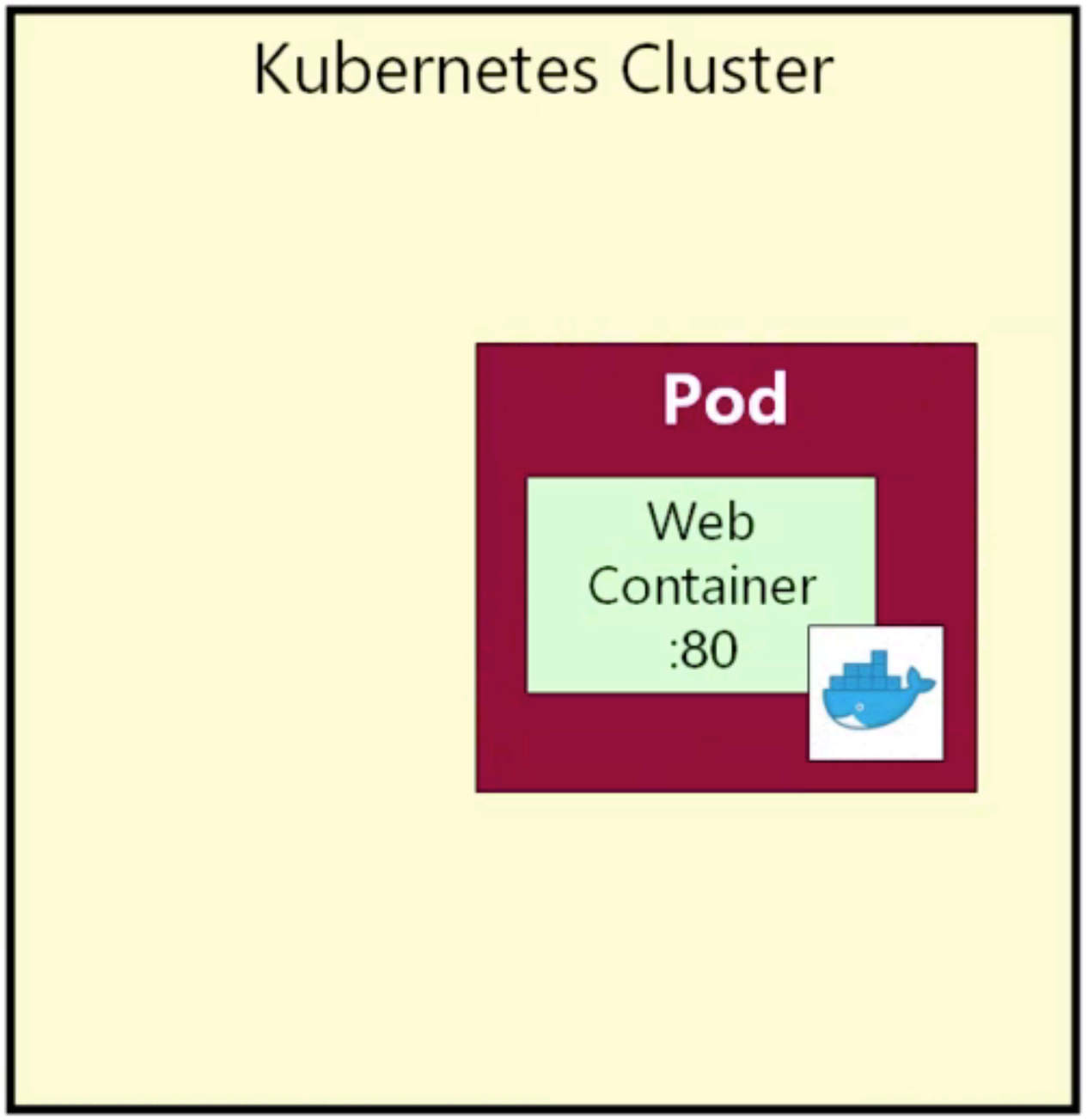
# *Now we have our first Kubernetes pod running on a cluster!*

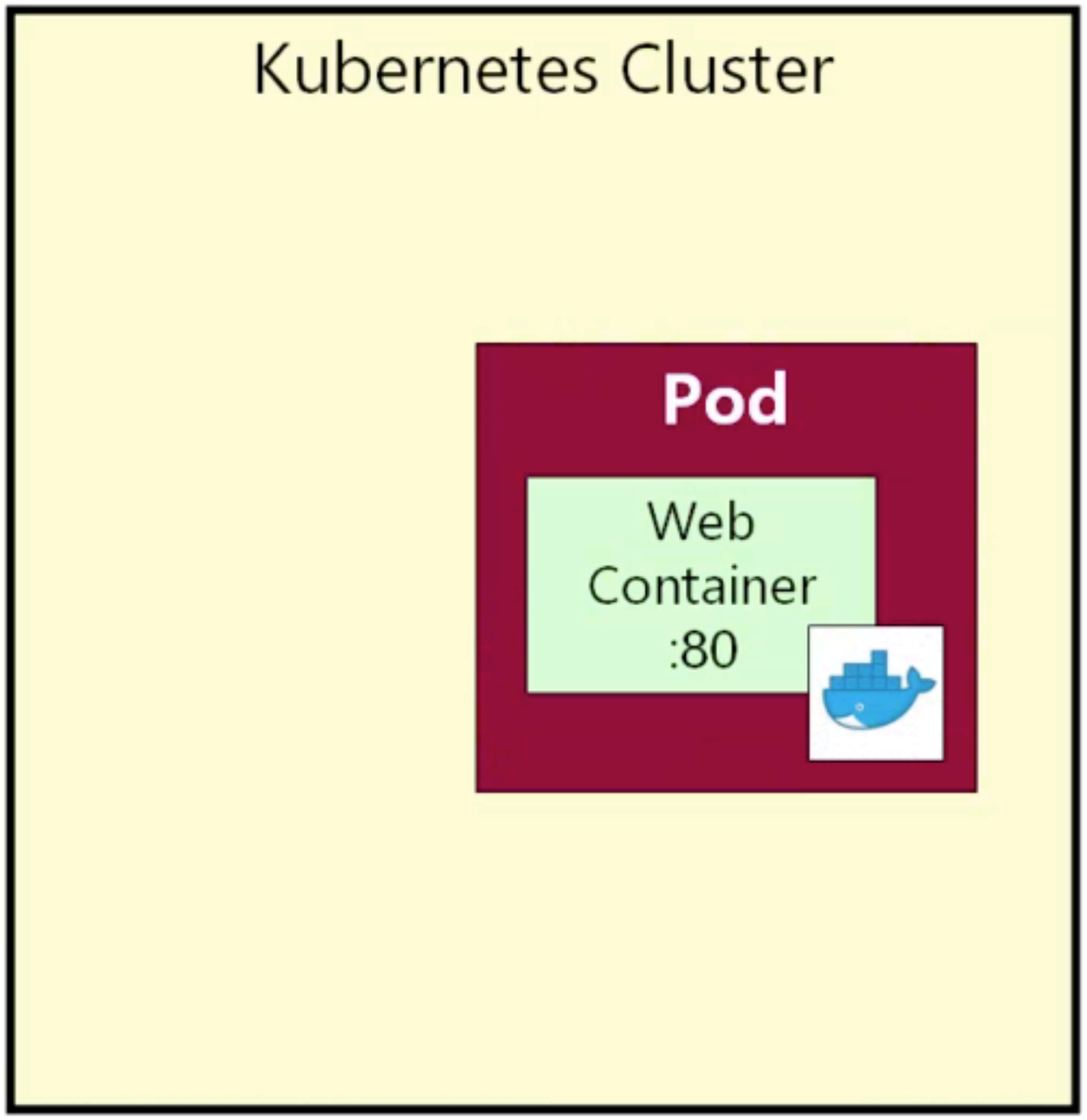
# Let's try it - `minikube ip`

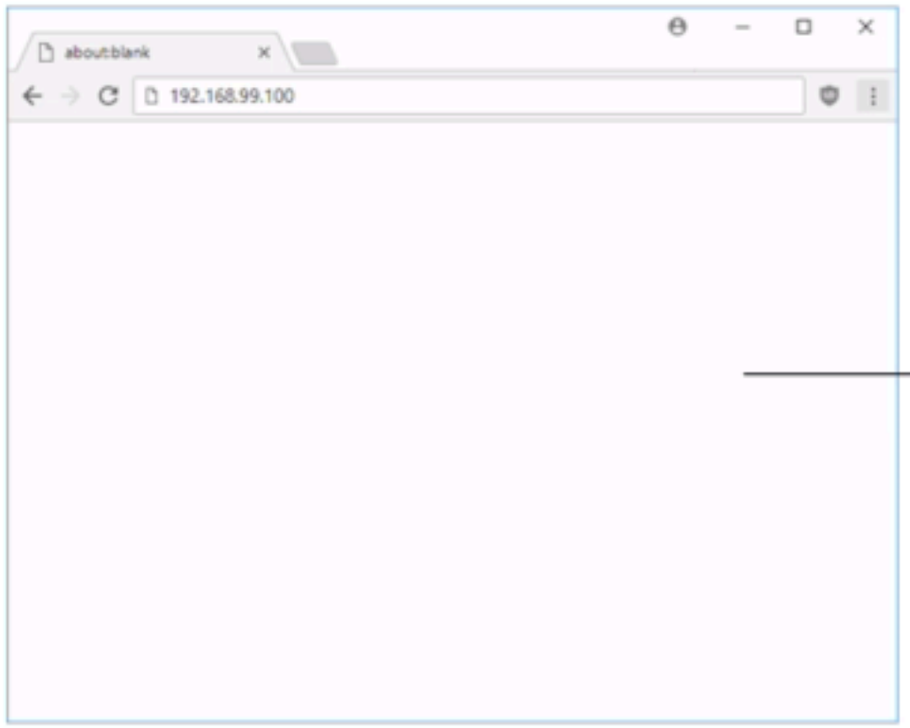
```
~/mtsds
$ minikube ip
192.168.99.100
```

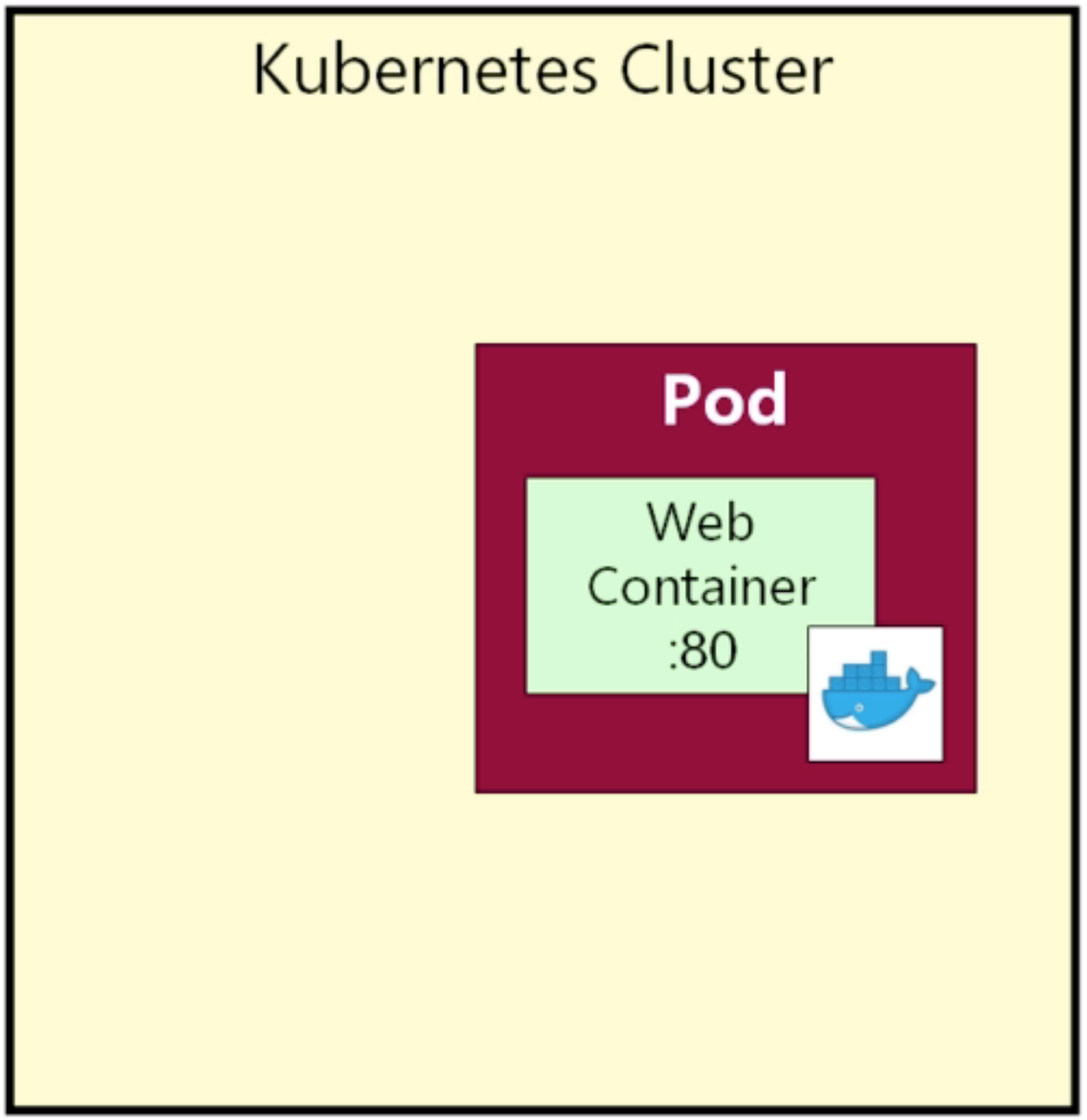



Kubernetes Cluster


Kubernetes Cluster
Pod
Web
Container
:80

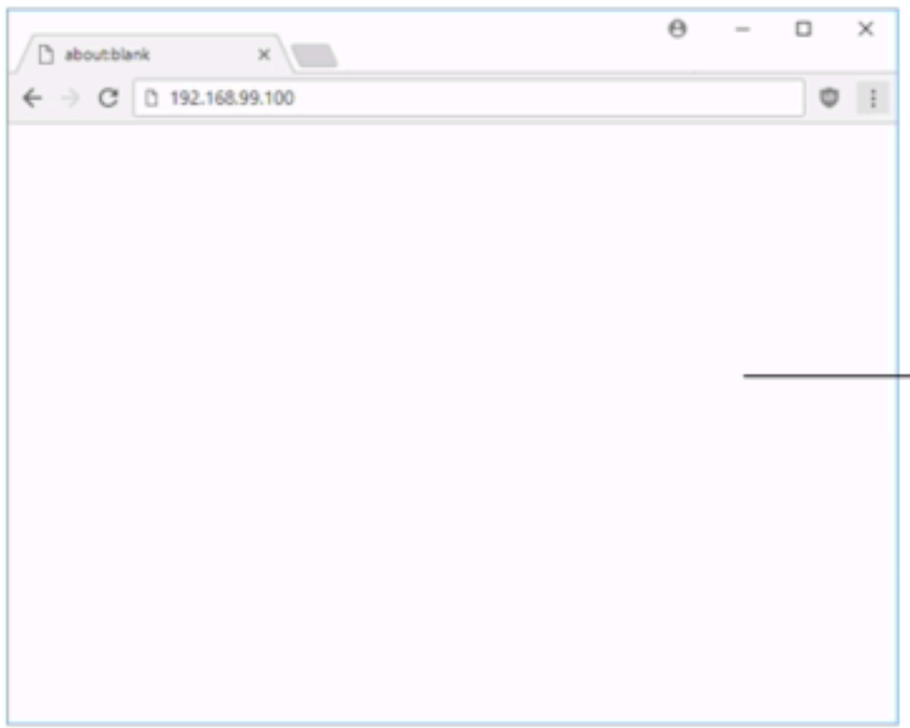

Kubernetes Cluster
Pod
Web
Container
:80

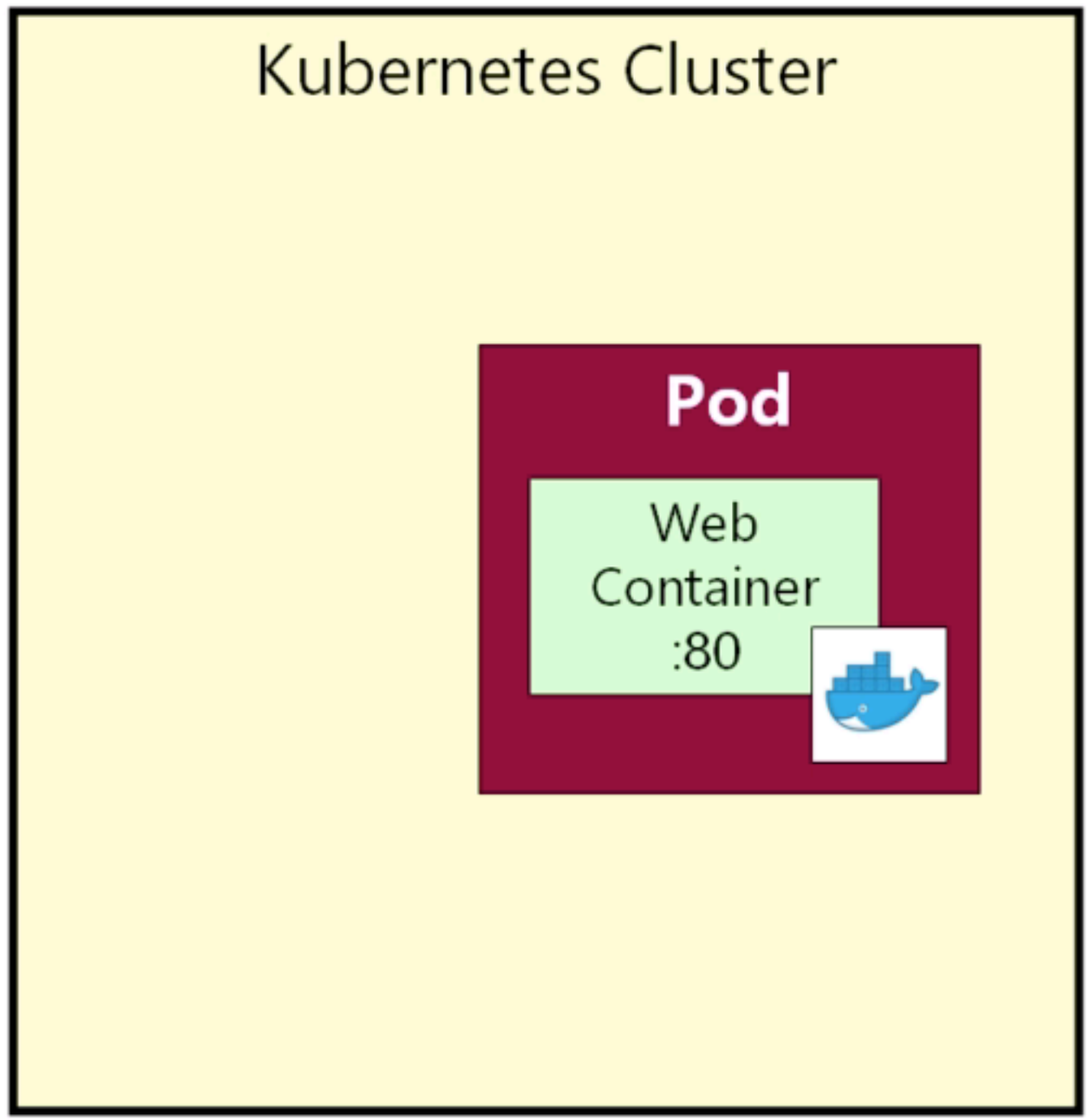

about:blank
192.168.99.100


Kubernetes Cluster
Pod
Web
Container
:80

ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

# Pods are not visible outside the cluster!



# *Pods are not visible outside the cluster!*

- Part of the design of pods is that they are completely isolated and only accessible from inside the cluster itself
- We can't visit this pod from a web browser

# *Let's try some useful commands*

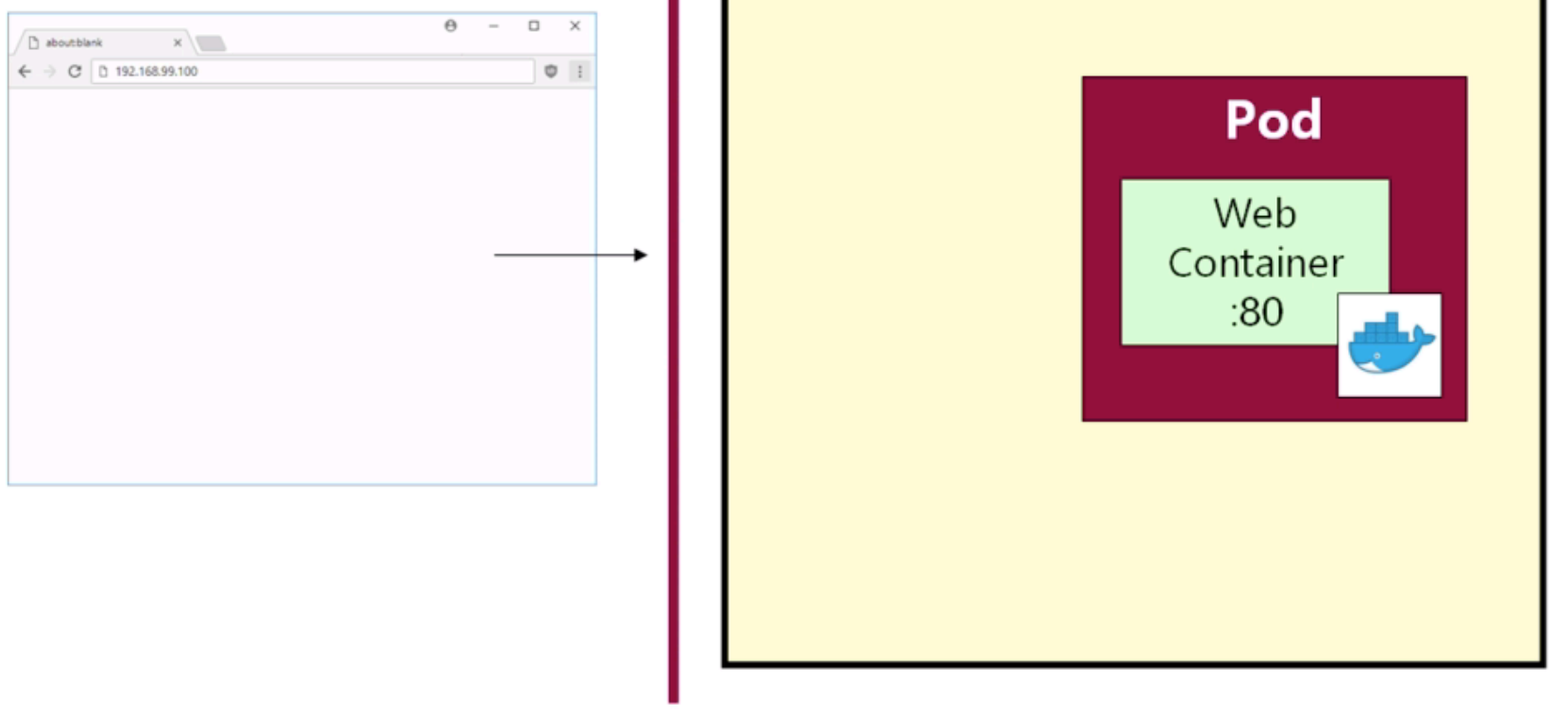
# kubectl describe pod webapp

```
~
$ kubectl describe pod webapp
Name:         webapp
Namespace:    default
Priority:     0
Node:         minikube/10.0.2.15
Start Time:   Tue, 22 Oct 2019 11:35:21 +0100
Labels:       app=webapp
              release=0
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"v1","kind":"Pod","metadata":{"annotations":{},"labels":{"
"webapp","namespace":"default"...
Status:       Running
IP:           172.17.0.7
IPs:
  IP:  172.17.0.7
Containers:
```

```
Events:
  Type     Reason           Age        From               Message
  ----     ------           ----       ----               -------
  Normal   Scheduled        <unknown>  default-scheduler  Successfully assigned default/webapp
  Normal   Pulled           5d23h      kubelet, minikube  Container image "richardchesterwood/k
ease0" already present on machine
  Normal   Created          5d23h      kubelet, minikube  Created container webapp
  Normal   Started          5d23h      kubelet, minikube  Started container webapp
  Warning  FailedMount      2m12s      kubelet, minikube  MountVolume.SetUp failed for volume "
to sync secret cache: timed out waiting for the condition
  Normal   SandboxChanged   2m10s      kubelet, minikube  Pod sandbox changed, it will be kille
  Normal   Pulled           2m9s       kubelet, minikube  Container image "richardchesterwood/k
ease0" already present on machine
  Normal   Created          2m8s       kubelet, minikube  Created container webapp
  Normal   Started          2m7s       kubelet, minikube  Started container webapp
```

- We can't visit that pod with a browser
- But there is one thing we can do
- We can connect to the pod
- `kubectl exec webapp ls`

```
~
$ kubectl exec webapp ls
bin
dev
etc
home
lib
media
mnt
proc
root
run
sbin
srv
sys
tmp
usr
var
```




- We could get a shell to the container
- Do a curl to localhost port 80

- kubectl -it exec webapp sh

```
~
$ kubectl -it exec webapp sh
/ #
```

- wget http://localhost:80

```
~
$ kubectl -it exec webapp sh
/ # wget http://localhost:80
Connecting to localhost:80 (127.0.0.1:80)
index.html           100% |********************************************************
/ #
```

# cat index.html

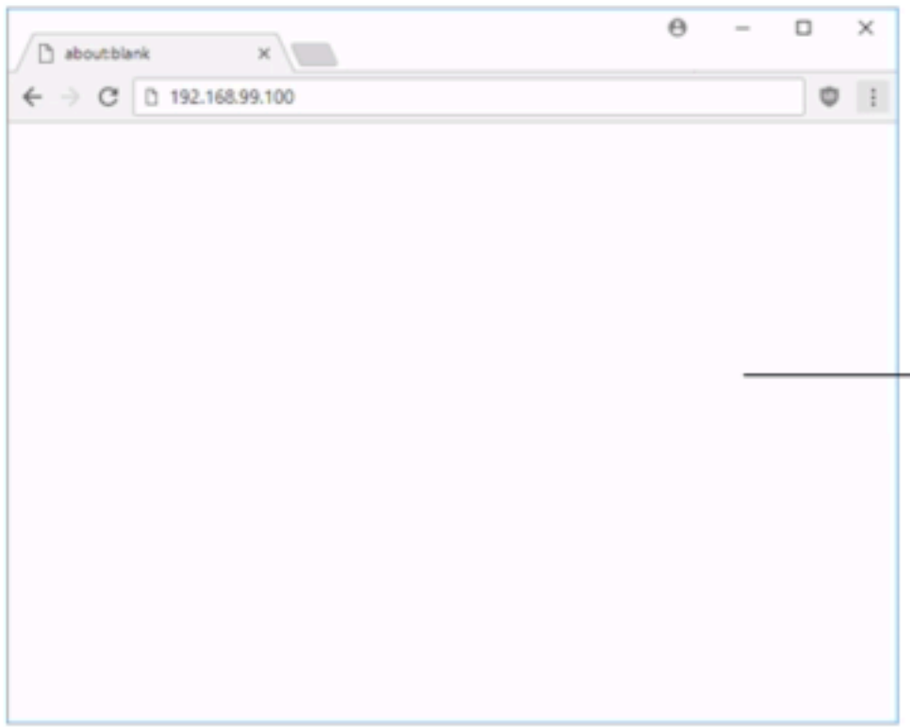
```
~
$ kubectl -it exec webapp sh
/ # wget http://localhost:80
Connecting to localhost:80 (127.0.0.1:80)
index.html           100% |*****************************************
/ # cat index.html
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Fleet Management</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.png">
</head>
<body>
  <app-root></app-root>
<script type="text/javascript" src="runtime.js"></script><script type=
type="text/javascript" src="styles.js"></script><script type="text/jav
avascript" src="main.js"></script></body>
</html>
/ #
```

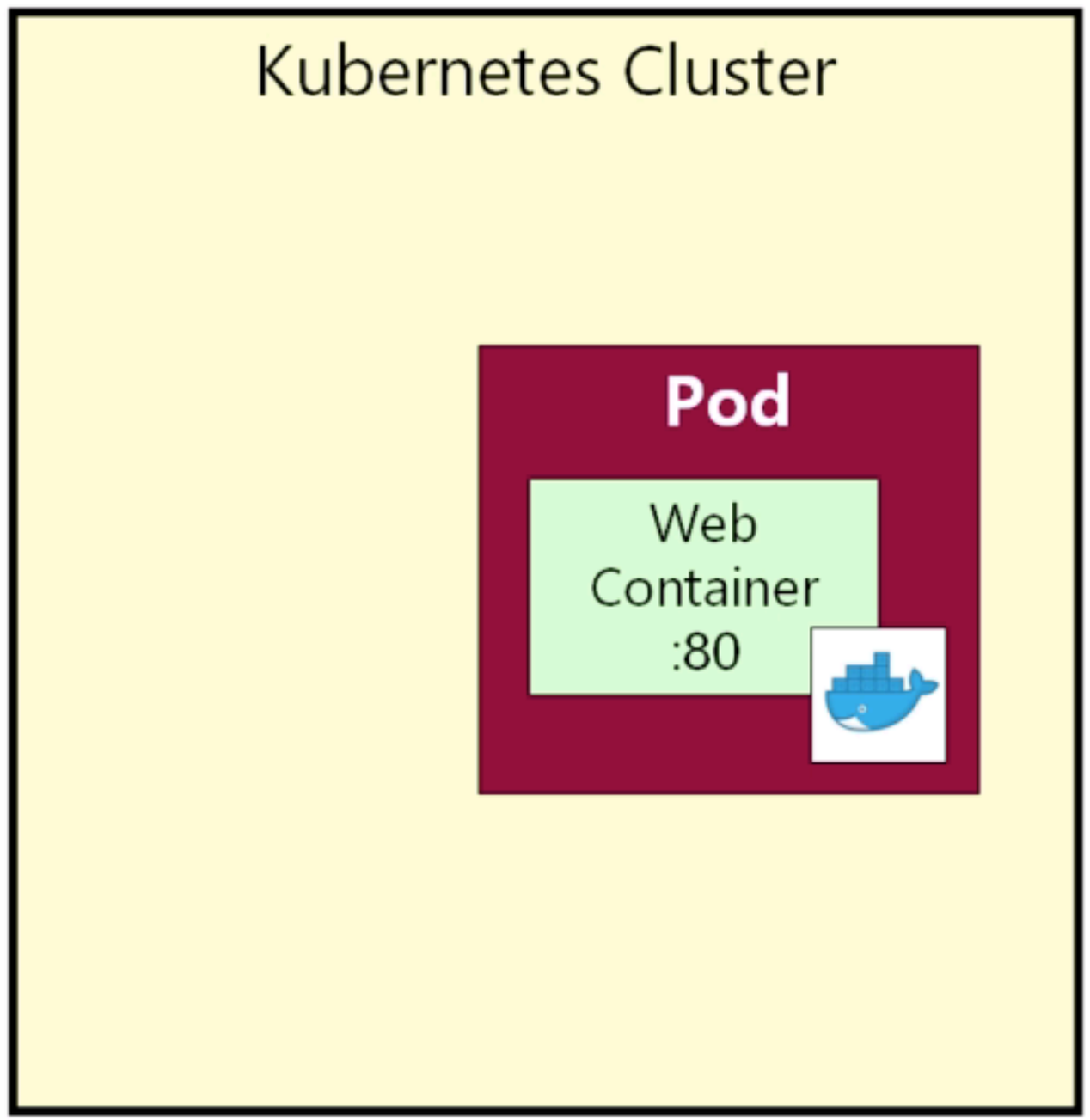
# exit

kubernetes

# *Services*

- We started our first pod
- It has a web container inside it
- We also know that the web container is running on port 80
- We tried to access the web container through a web browser - it failed


about:blank
192.168.99.100


Kubernetes Cluster
Pod
Web
Container
:80

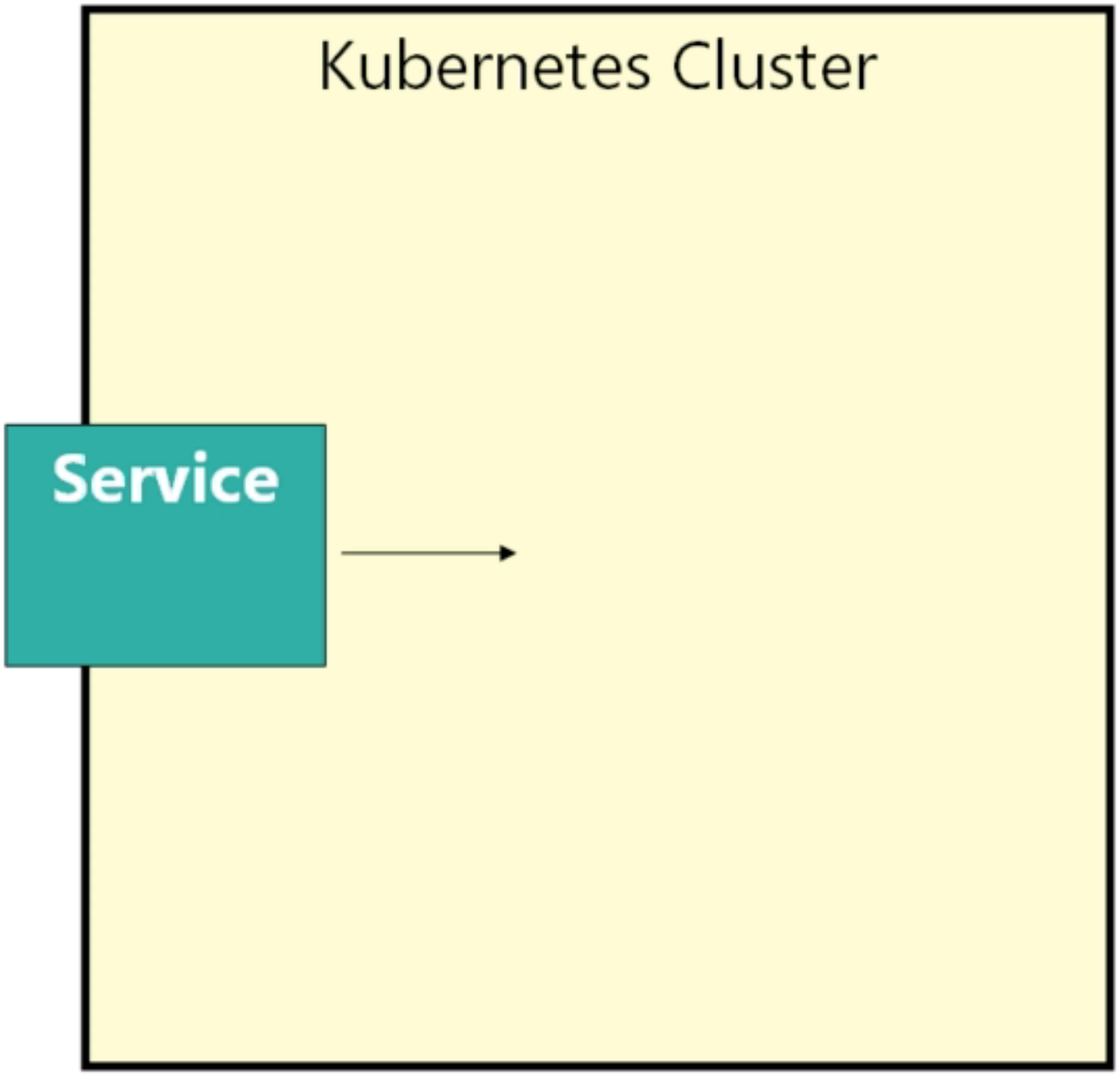
- Pods are designed to be disposable
- Pods have short live times
- Pods regularly die
- Pods are regularly recreated

*We treat pods like cattle*

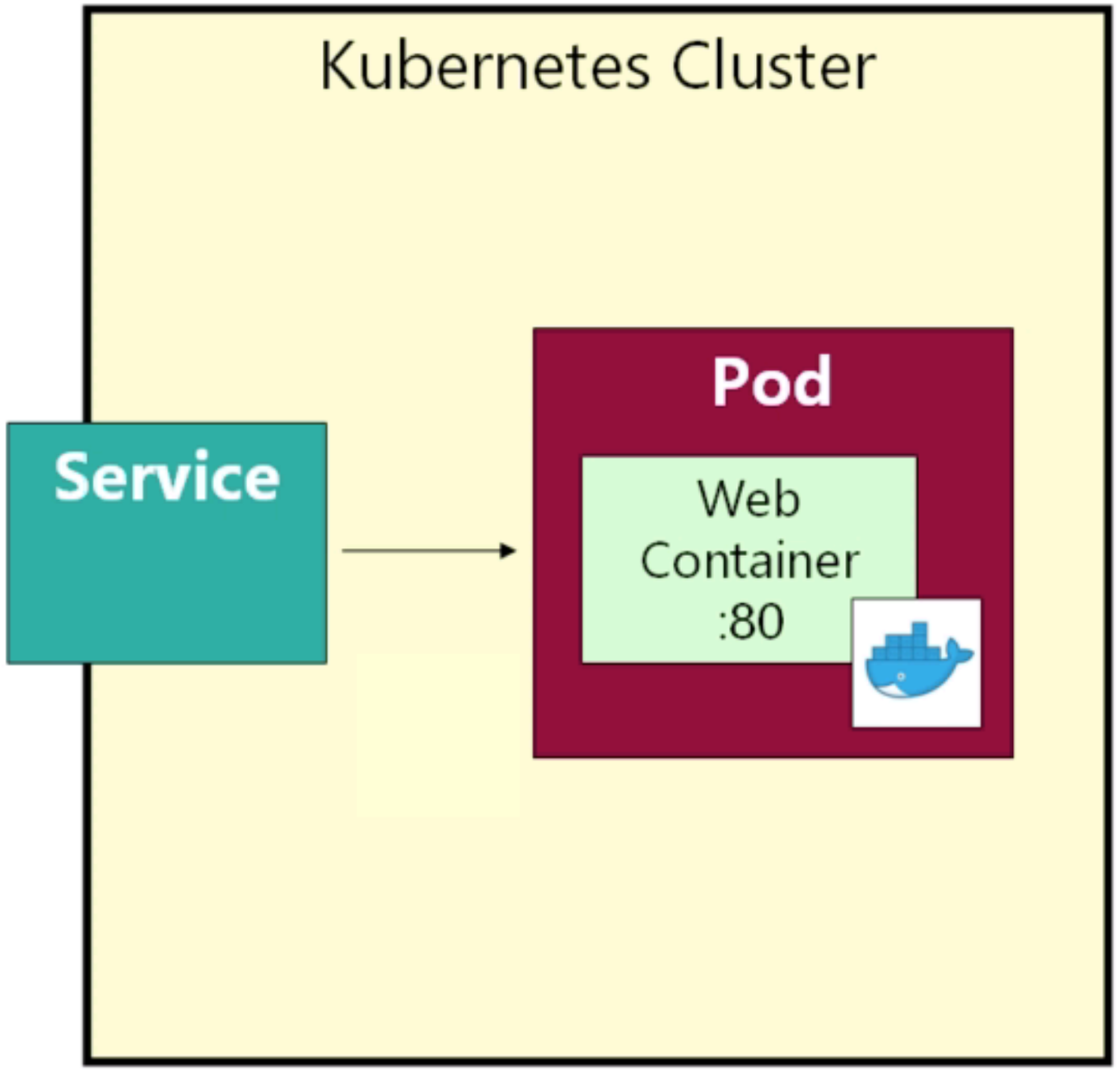
P.PORTO

ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO

# *Kubernete has a new concept to solve this that is a - Service*

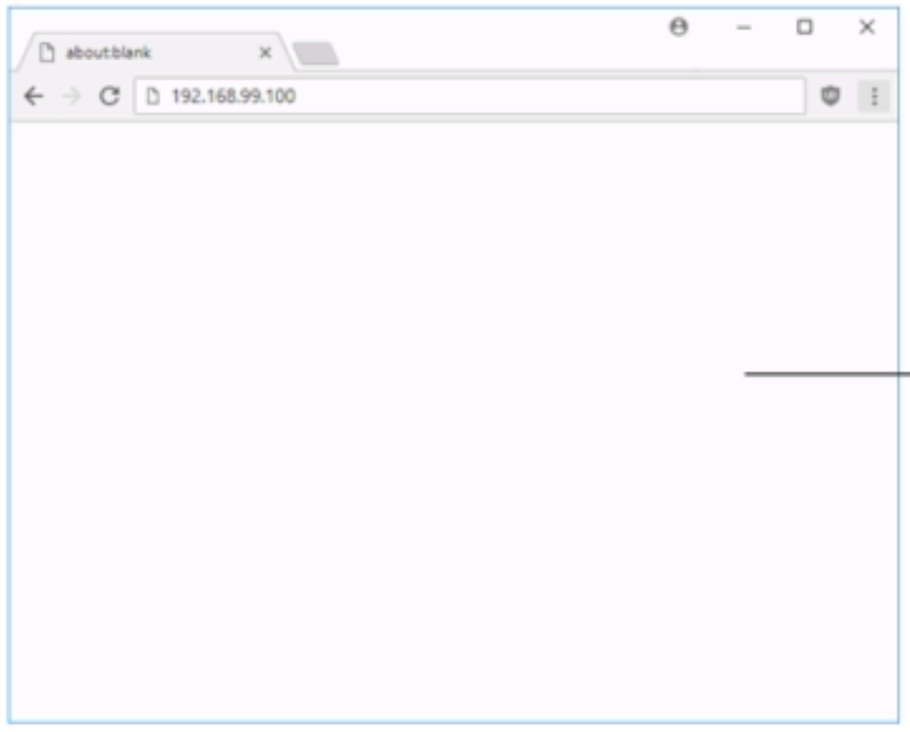

Kubernetes Cluster
Service

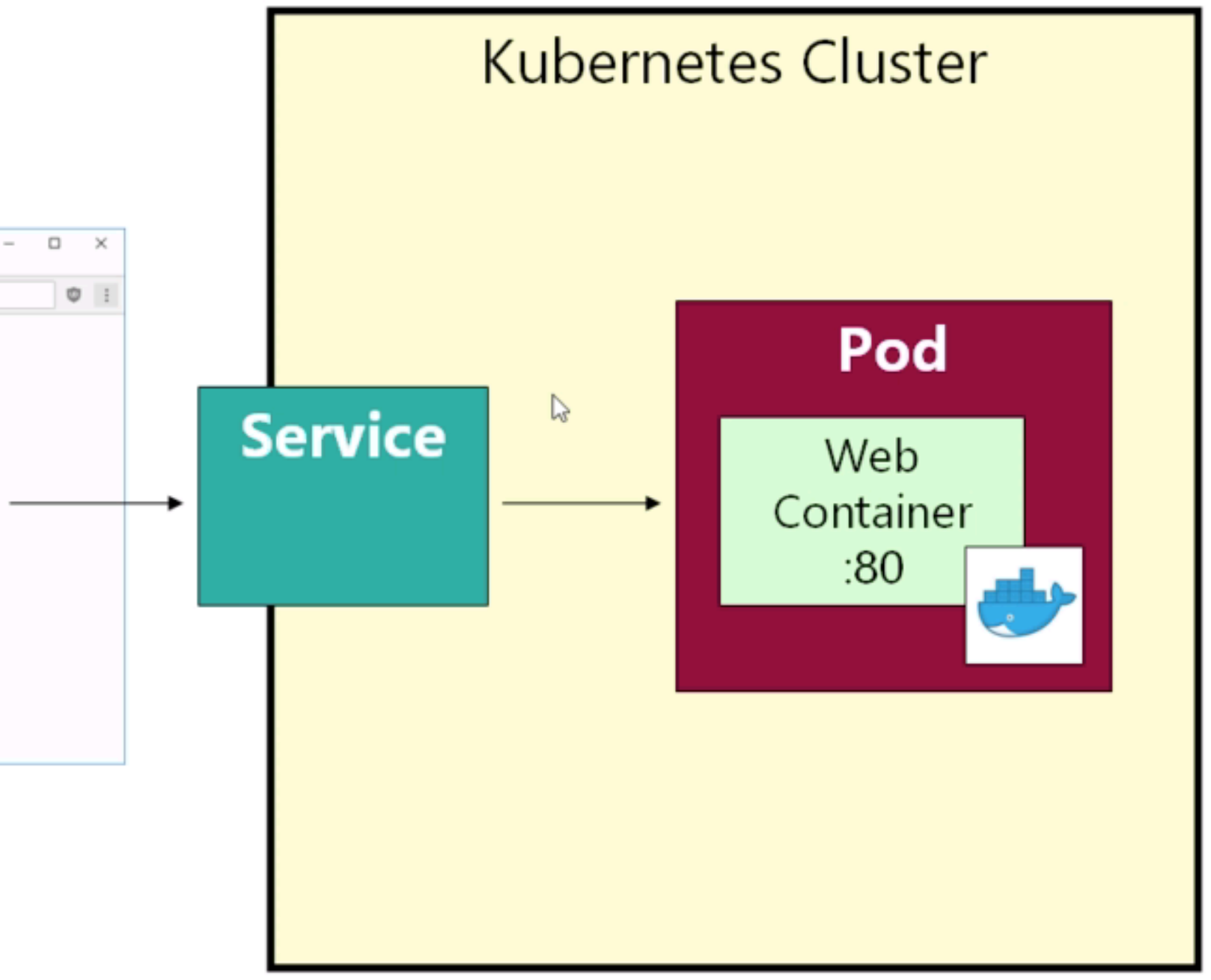
- A service is a long running object in Kubernetes
- A service will have an IP address
- And a service will have a stable fixed port




- We can attach services to pods

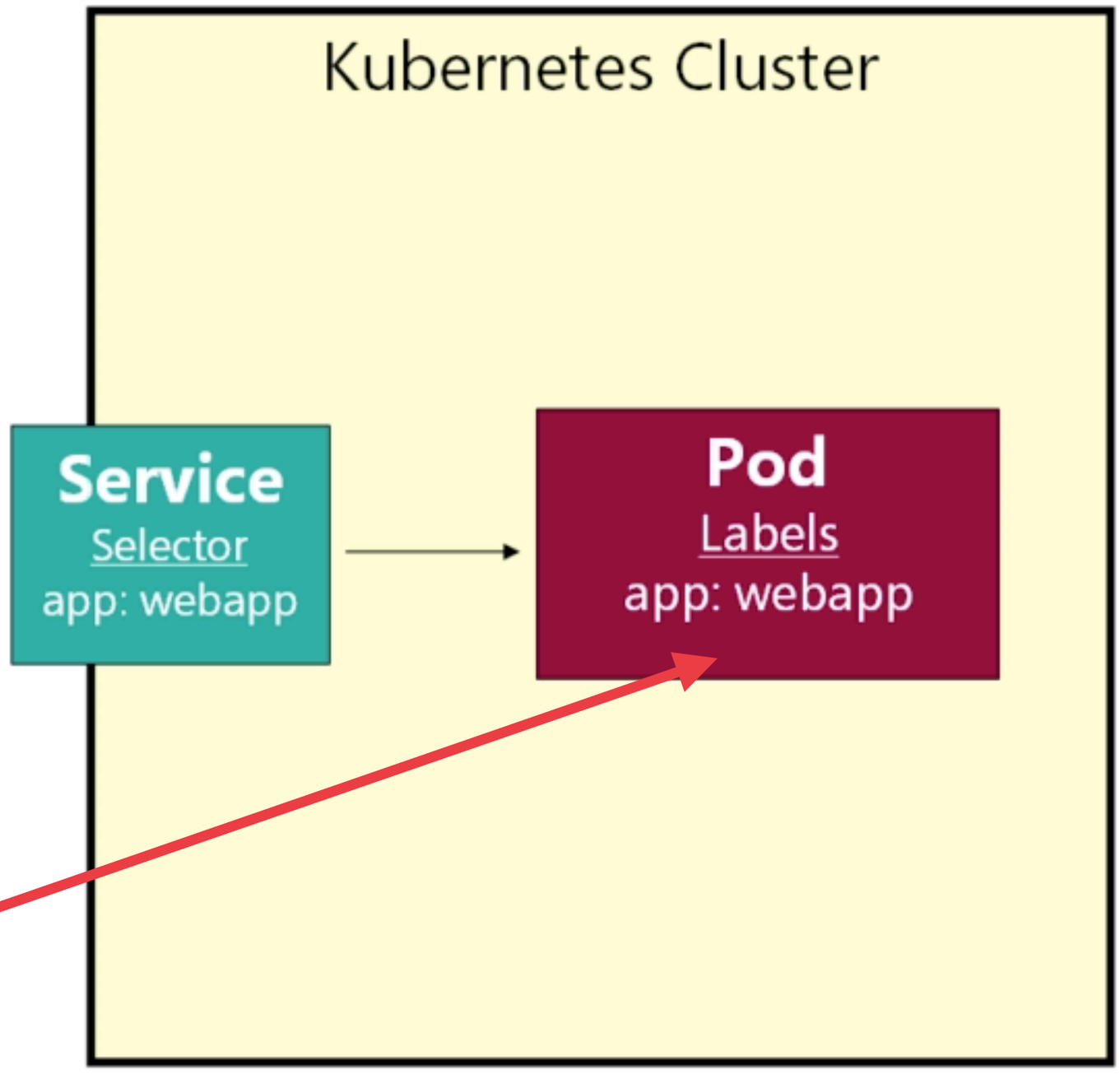
- With a service we can connect to the kubernetes cluster
- The service will find a suitable port to service that request

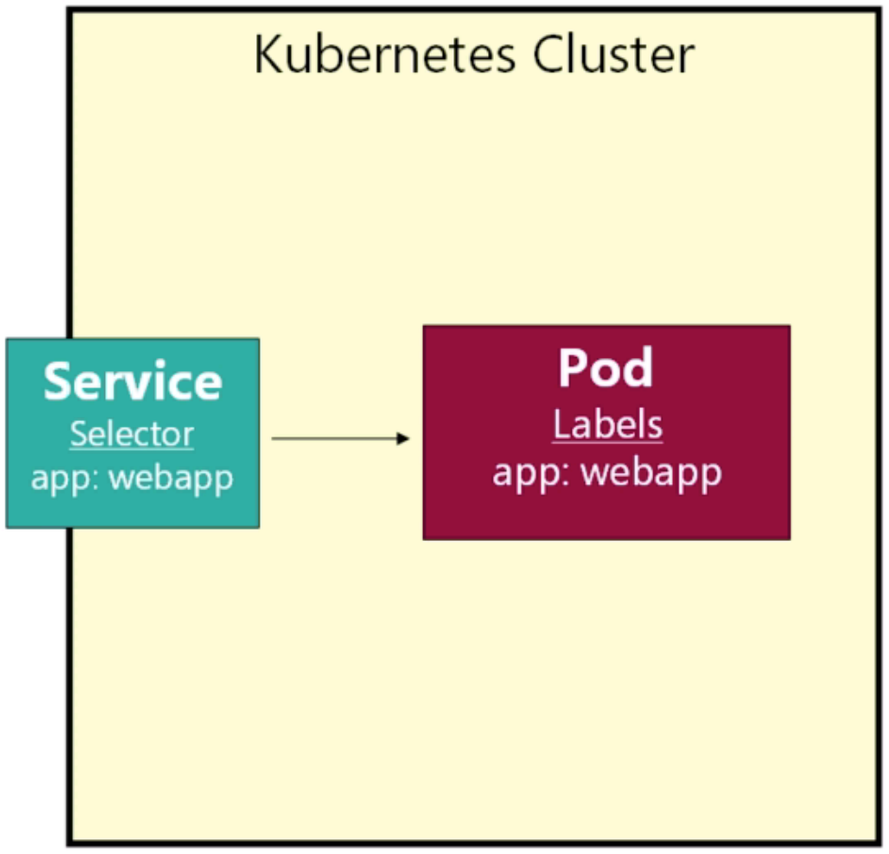

about:blank
192.168.99.100


Kubernetes Cluster
Service
Pod
Web
Container
:80

- The next important concept - Pod label

- We can setup a series of key-value pairs



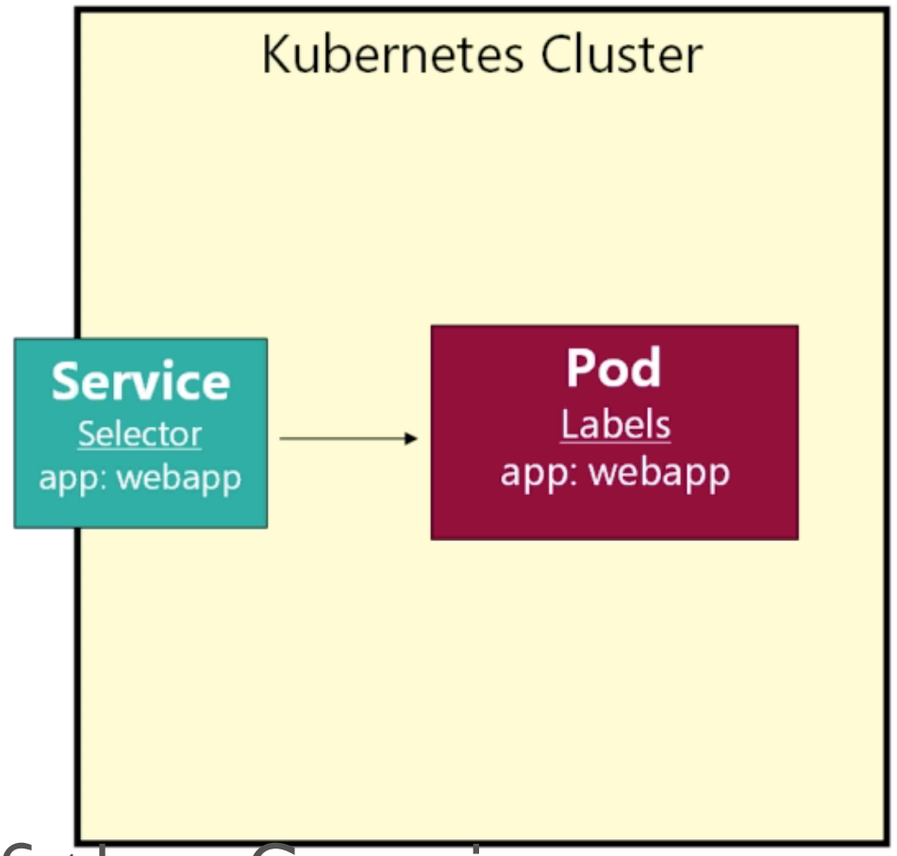



- In this example we have:
  - Key - app
  - Value - web app

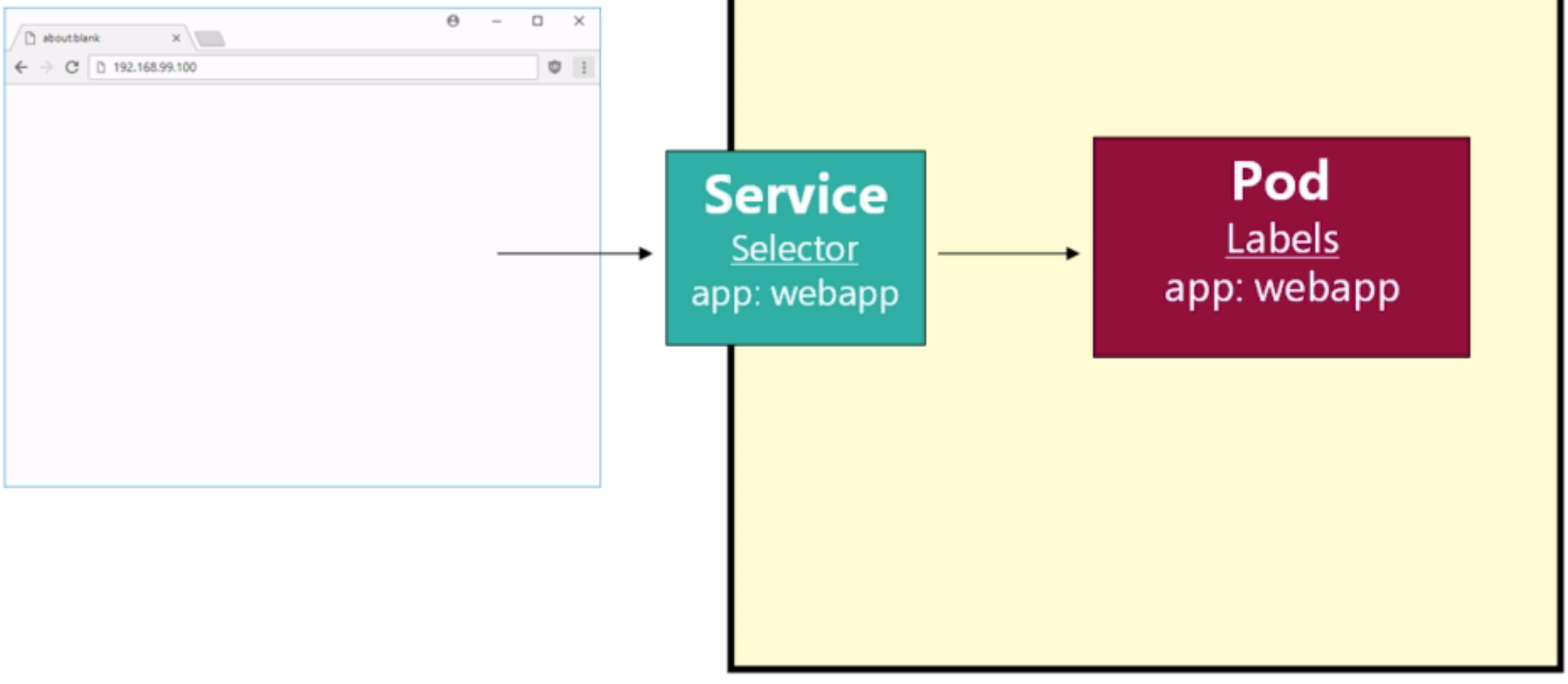
P.PORTO

ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO

# *This is the way we connect Services to Pods*




- In the definition of the Service we provide a selector
  - Key - app
  - Value - web app

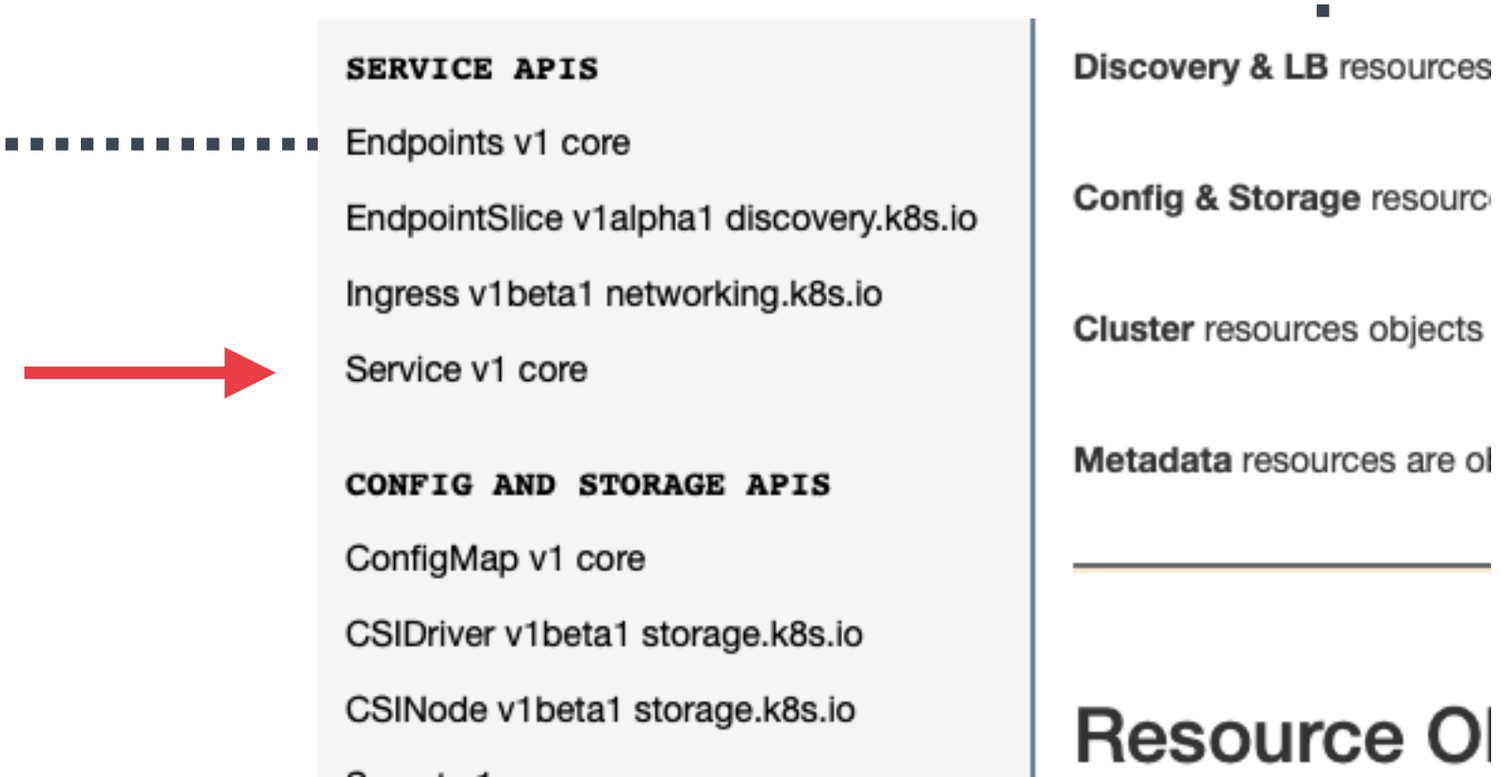

about:blank
192.168.99.100
Kubernetes Cluster
Service
Selector
app: webapp
Pod
Labels
app: webapp

# kubectl get all

```
~/mtsds
$ kubectl get all
NAME                     READY   STATUS    RESTARTS   AGE
pod/webapp               1/1     Running   0          58s

NAME                 TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)
service/kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP
```

- Create a service for that pod (**`webapp`**)
- Create a file **`webapp-service.yaml`**

API Overview - find the description for services



```
kind: Service
apiVersion: v1
metadata:
  # Unique key of the Service instance
  name: service-example
spec:
 ports:
    # Accept traffic sent to port 80
    - name: http
      port: 80
      targetPort: 80
 selector:
    # Loadbalance traffic across Pods matching
    # this label selector
    app: nginx
 # Create an HA proxy in the cloud provider
 # with an External IP address - *Only supported
 # by some cloud providers*
 type: LoadBalancer
```

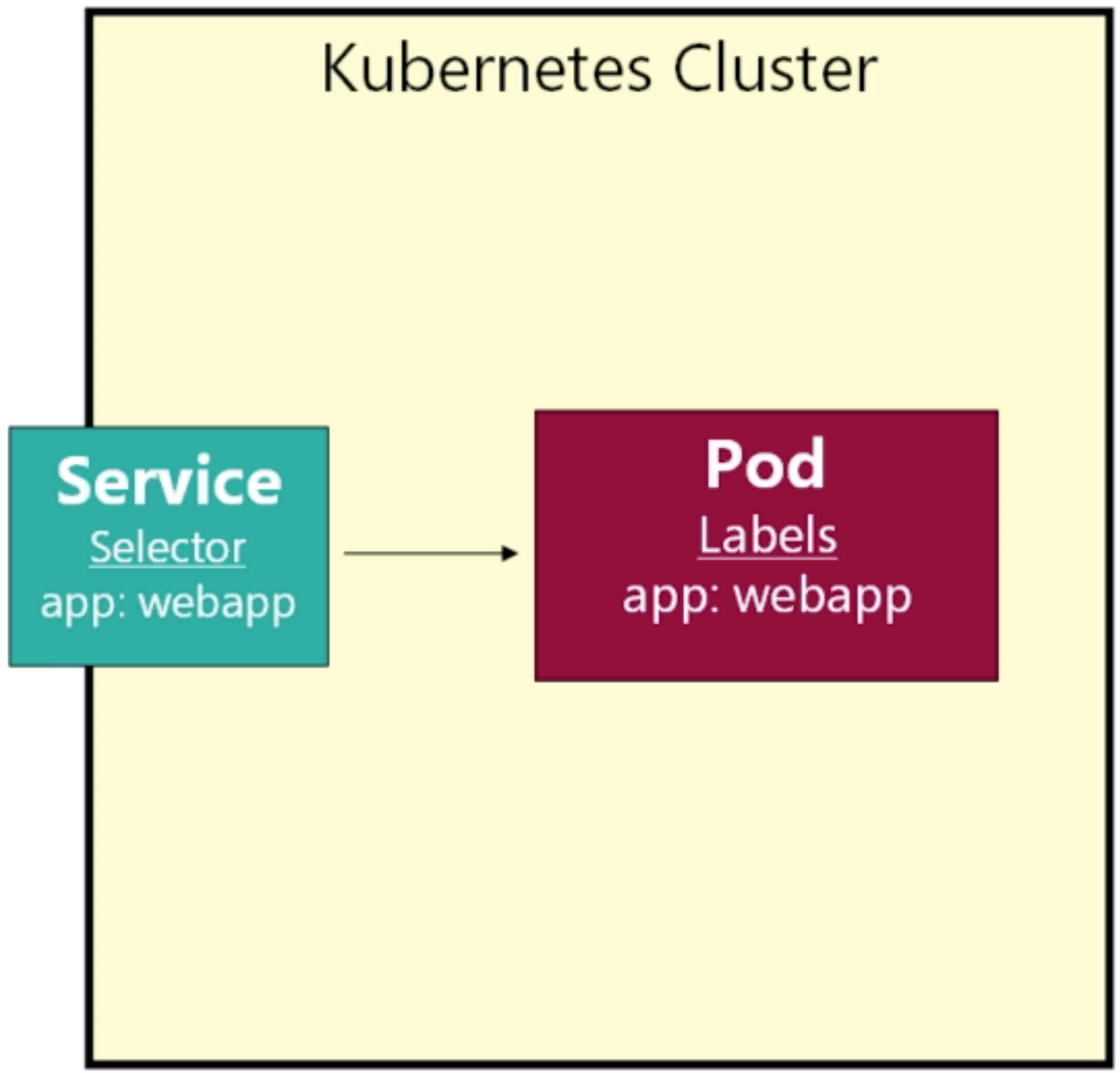
**give the service a name - think in a good name**

**later**

**selector: key - app; value - nginx**

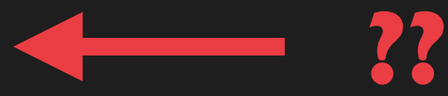
**later**

- We are going to give to the pod a label app:webapp
- The service will select that key-value pair - app:webapp



webapp-service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: fleetman-webapp

spec:
  selector:
    app: webapp

  ports:
  - name: http
    port: 80

  type:
```



??

- **Type** options for **Services**:
  - **`LoadBalancer`** - we will learn that later and is not available on a local cluster
  - **`ClusterIP`** - This service will only be accessible from inside the cluster
    - We will only be using this if it is an internal service (private service)
  - **`NodePort`** - expose a port through the node
    - We can choose what port we want to expose to the outside world (>30000)

webapp-service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: fleetman-webapp

spec:
  selector:
    app: webapp

  ports:
  - name: http
    port: 80
    nodePort: 30080

  type: NodePort

```

- With this definition we are saying that we are going to have a service called fleetman-webapp
- It is going to be a network endpoint for any pod that finds in the cluster with the key value pair - app:webapp
- It is going to be talking to the port inside the pod (80) and it is going to expose to the outside world through the port 30080

- # kubectl apply -f webapp-service.yaml

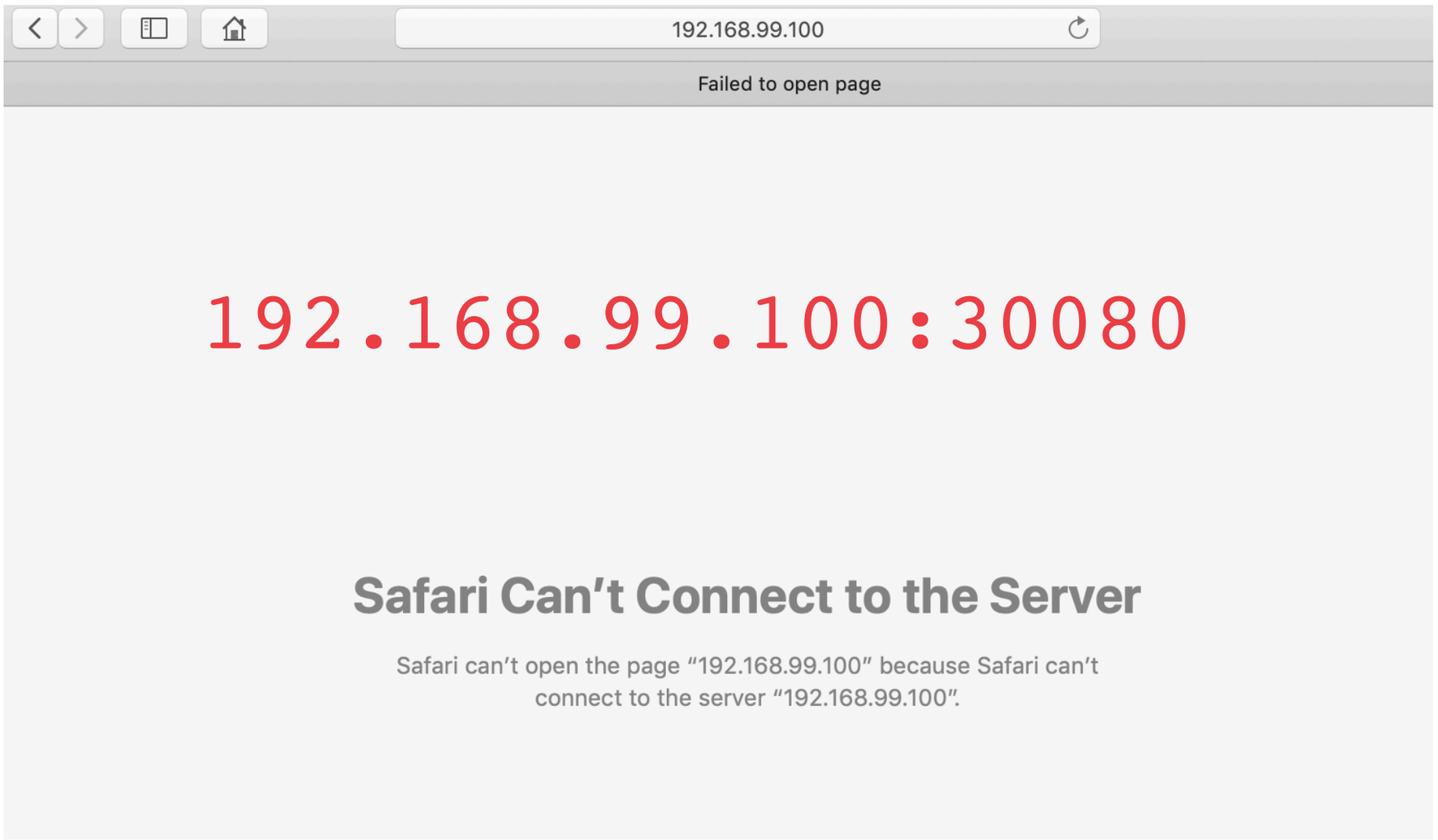
```
~
$ kubectl apply -f webapp-service.yaml
```

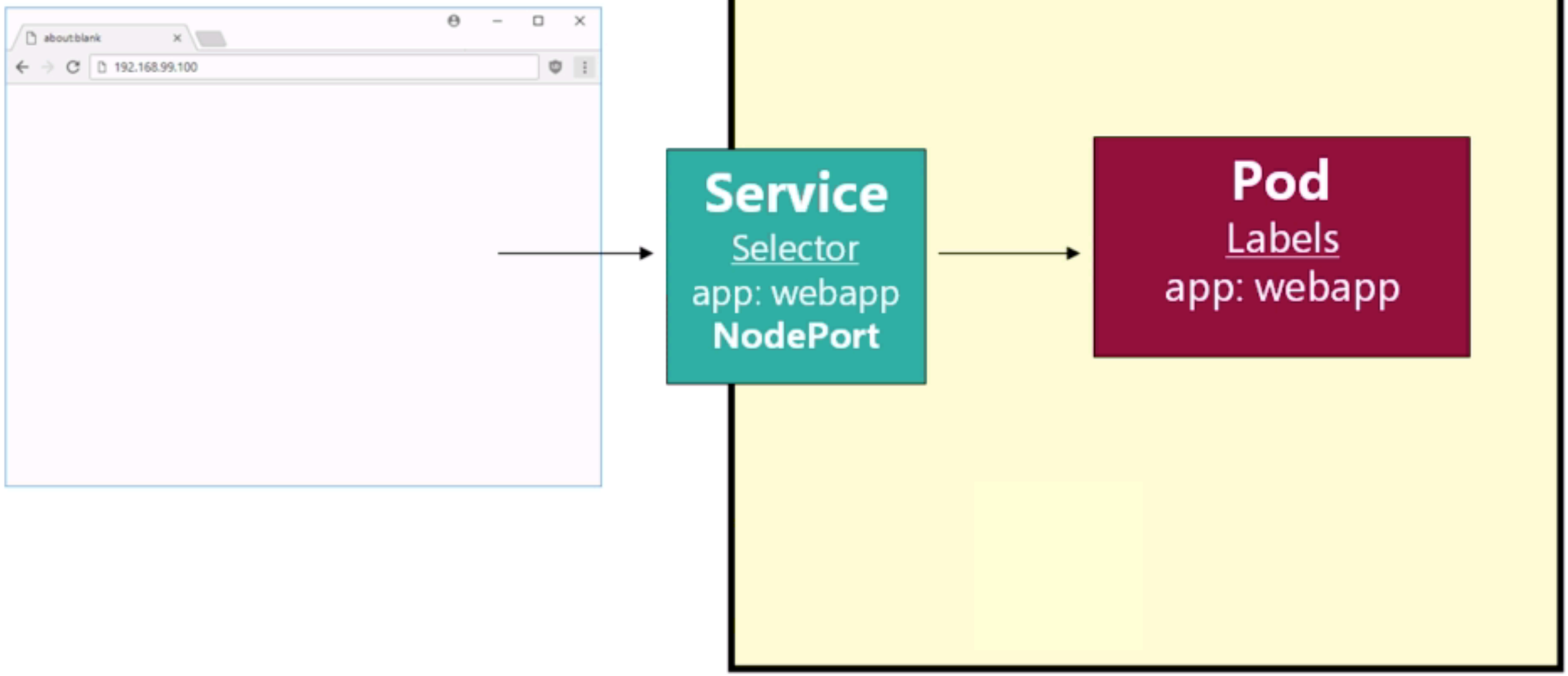
- # kubectl get all

```
~
$ kubectl get all
NAME                          READY   STATUS    RESTARTS   AGE
pod/webapp                    1/1     Running   1          6d5h

NAME                          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
service/fleetman-webapp       NodePort    10.106.54.9     <none>        80:30080/TCP
service/kubernetes            ClusterIP   10.96.0.1       <none>        443/TCP
```

# Let's try it - `minikube ip`

```
~/mtsds
$ minikube ip
192.168.99.100
```



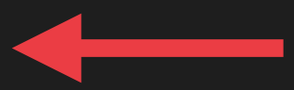

about:blank
192.168.99.100
Kubernetes Cluster
Service
Selector
app: webapp
NodePort
Pod
Labels
app: webapp

- The service looks for any pod that finds in the cluster with the key value pair - app:webapp

*We didn't define any label on our Pod*

pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: webapp
  labels:
    app: webapp

spec:
  containers:
  - name: webapp
    image: richardchesterwood/k8s-fleetman-webapp-angular:release0

```



- `kubectl apply -f pod.yaml`

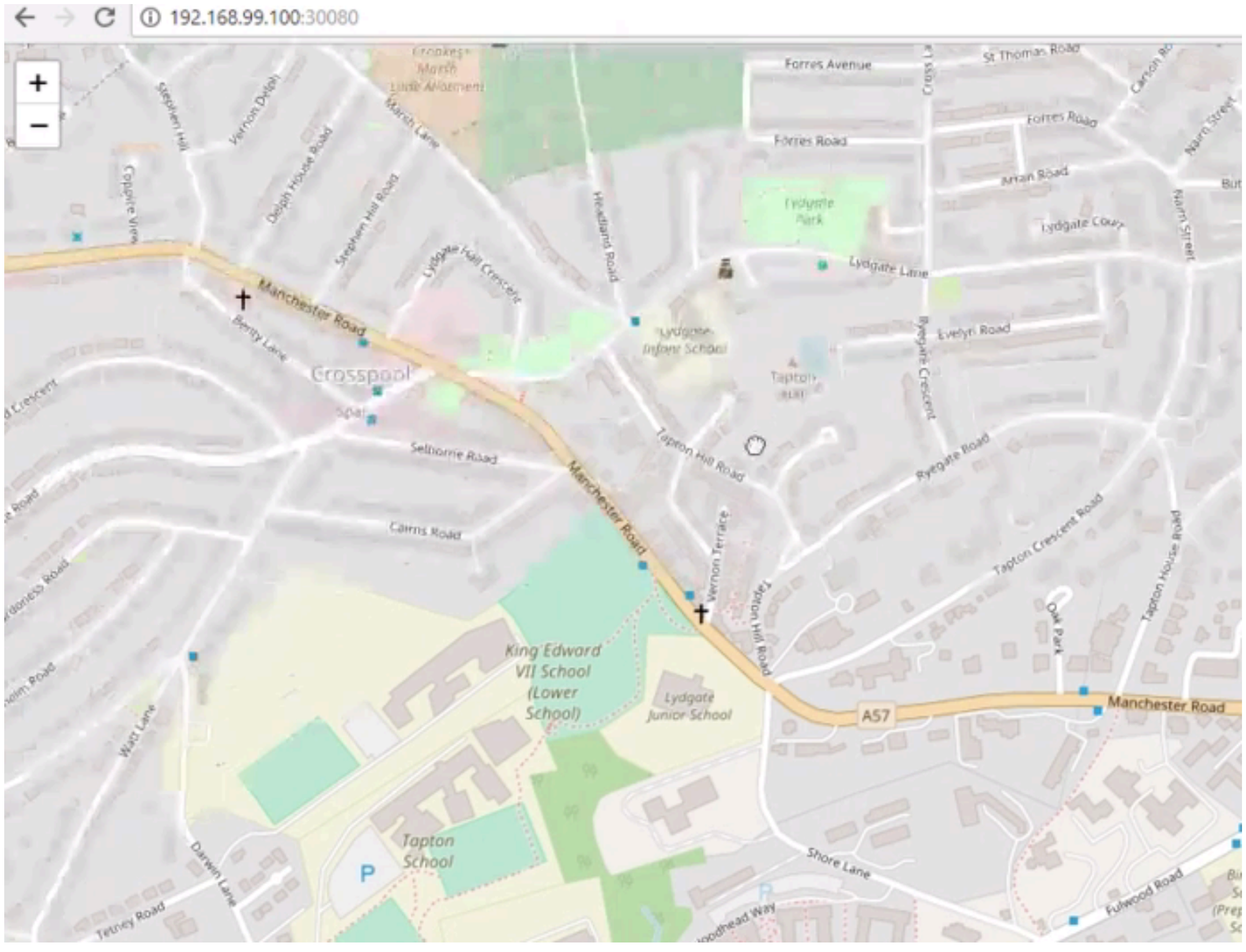
```
~
$ kubectl apply -f pod.yaml
```

- `kubectl apply -f webapp-service.yaml`

```
~
$ kubectl apply -f webapp-service.yaml
```

ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO

# *Let's try it now!*



- Now we have been told by the developer that a new version is available - `release0-5`

*Deploy the new version!*

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: webapp
  labels:
    app: webapp

spec:
  containers:
  - name: webapp
    image: richardchesterwood/k8s-fleetman-webapp-angular:release0

```

*Should we change this?*

release0 → release0-5

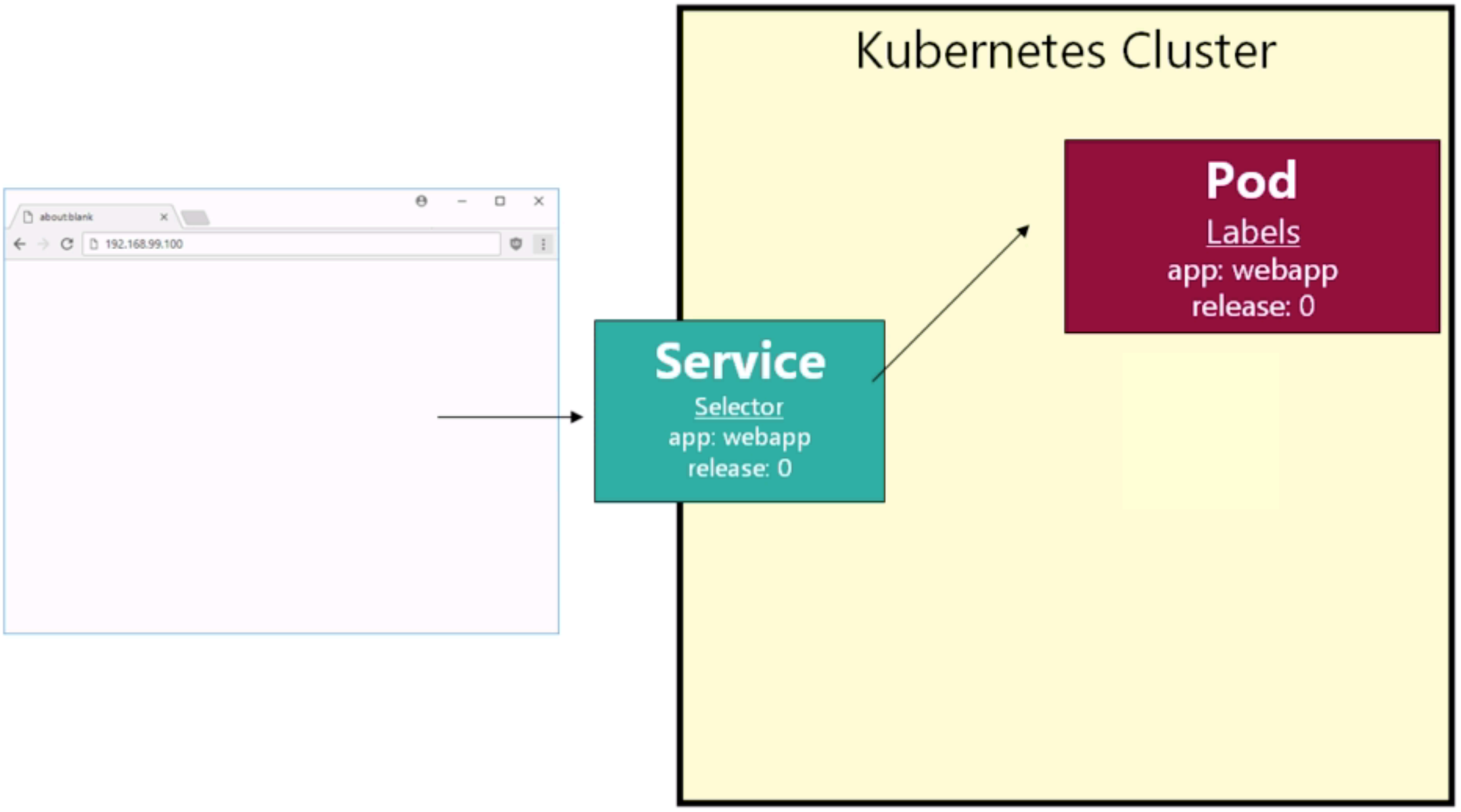
- That is not a good idea:
  - The pod is only a wrapper to a container, Kubernetes will have to stop this pod, start a new pod
  - Pull a new container image
  - Start that pod

*We are going to have some downtime! :(*

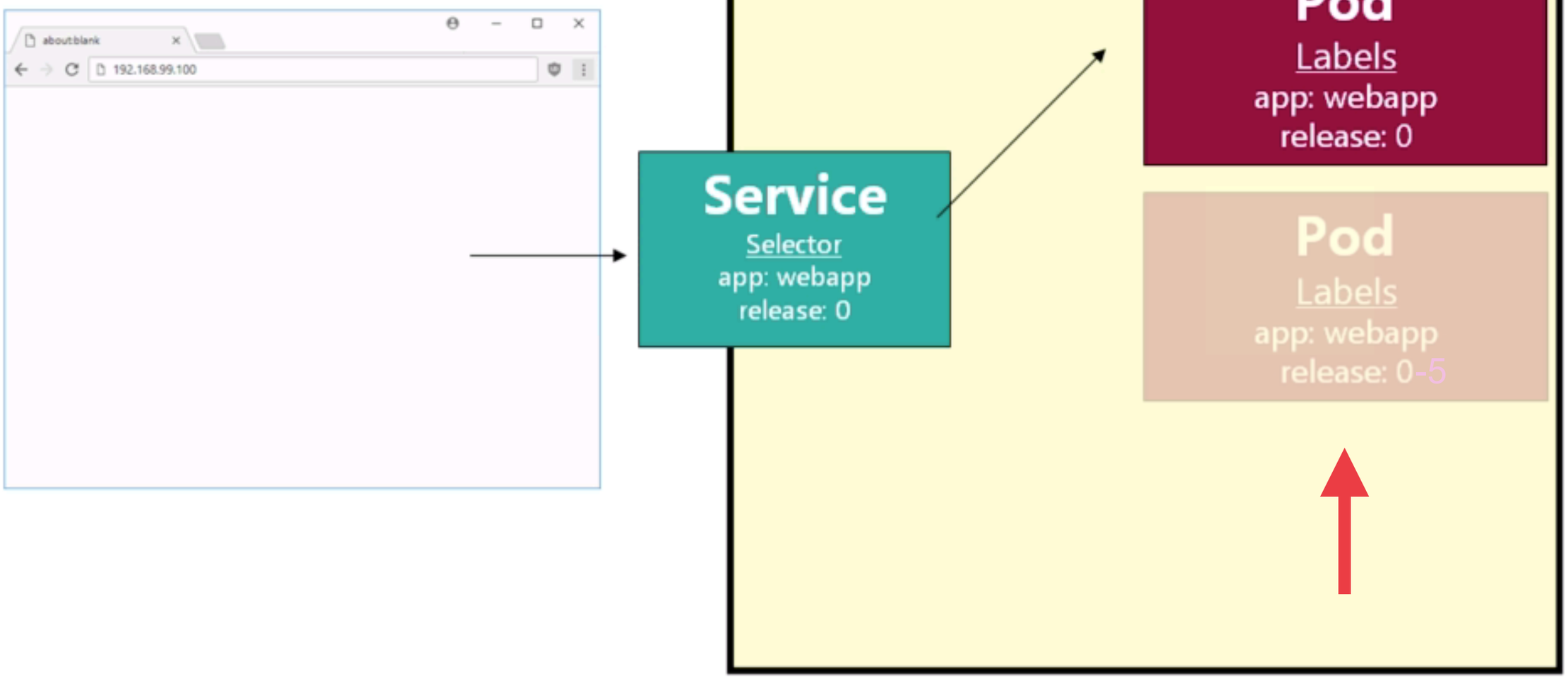
P.PORTO

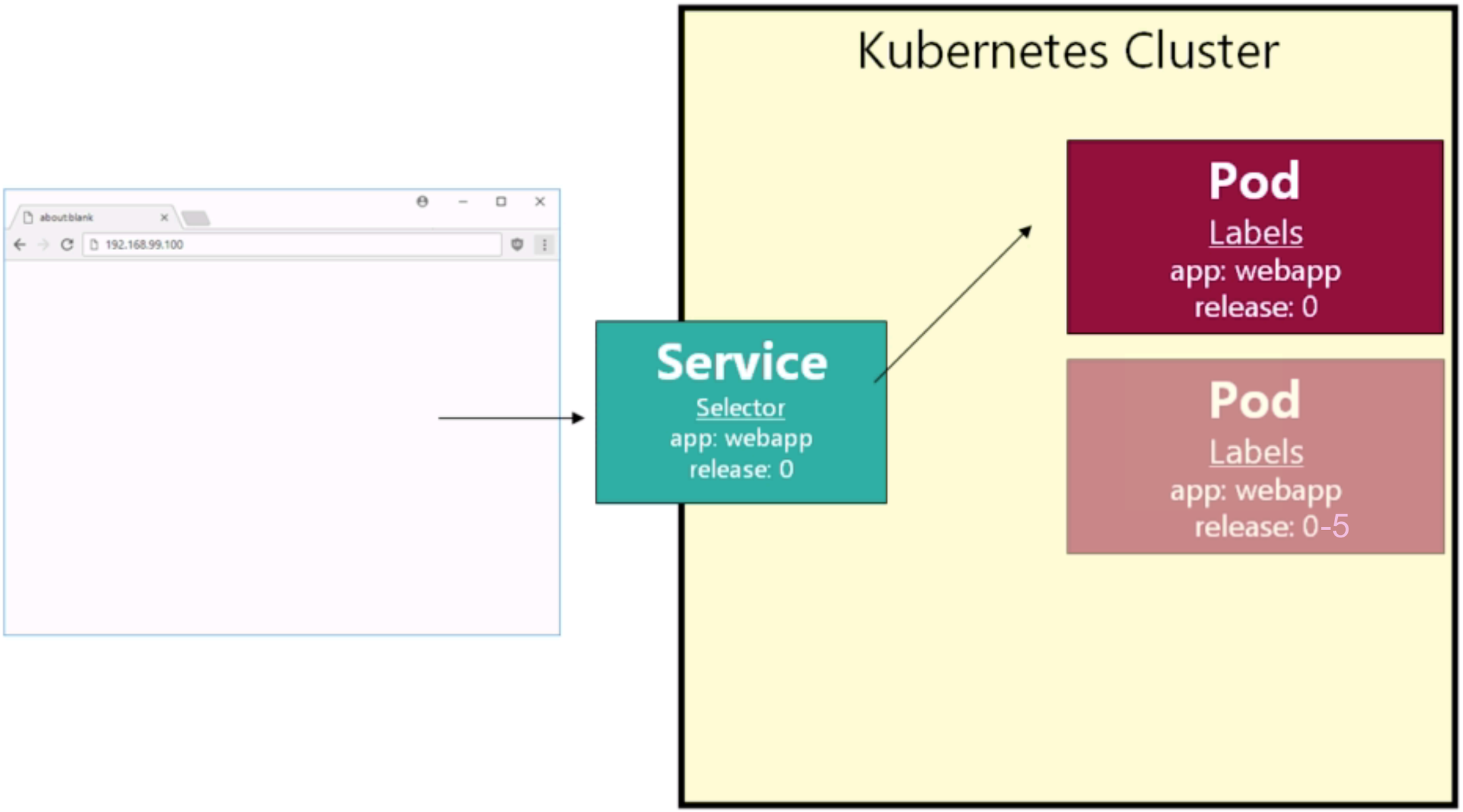
ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO

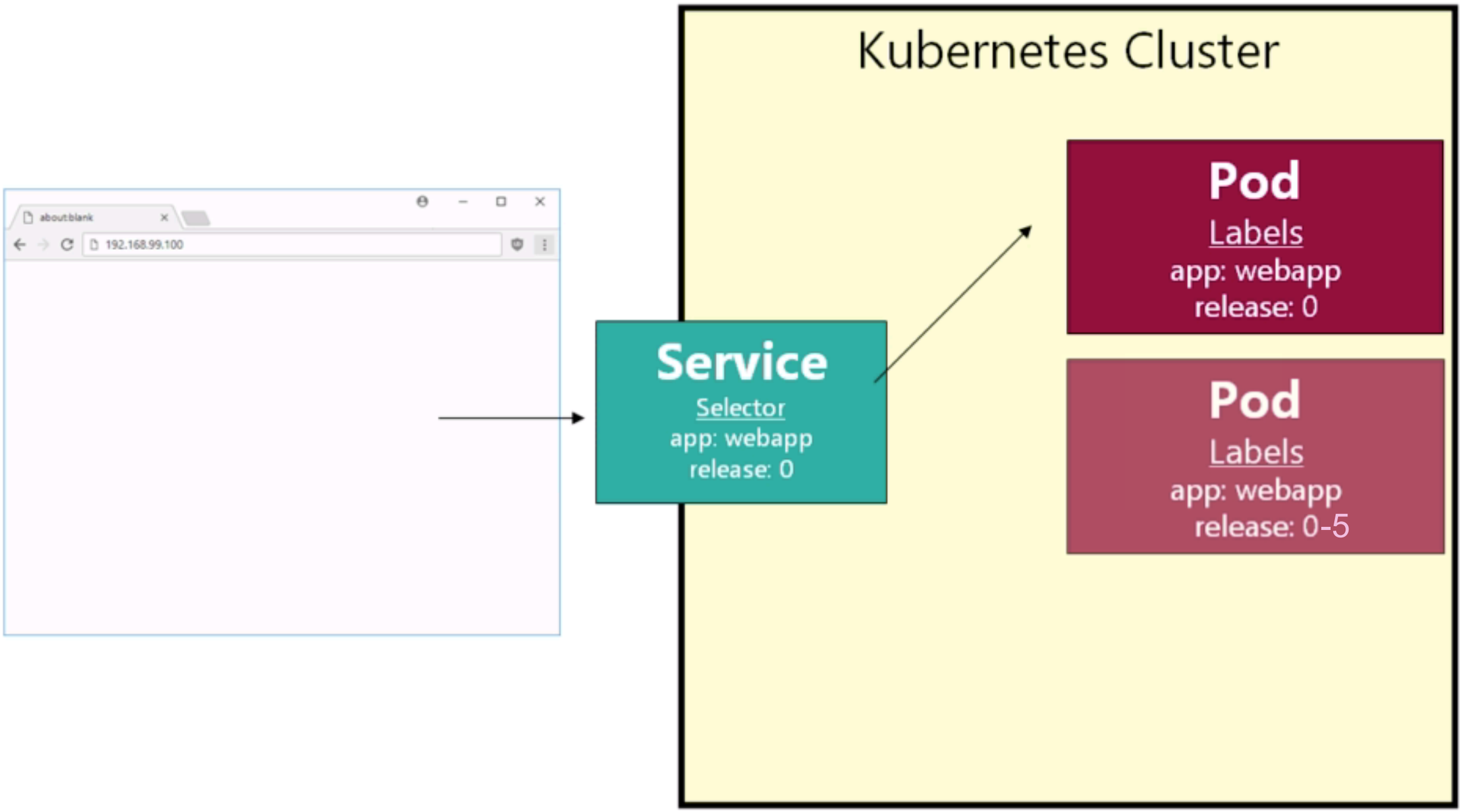
# *In the next lessons we are going to learn how to deploy with zero downtime*

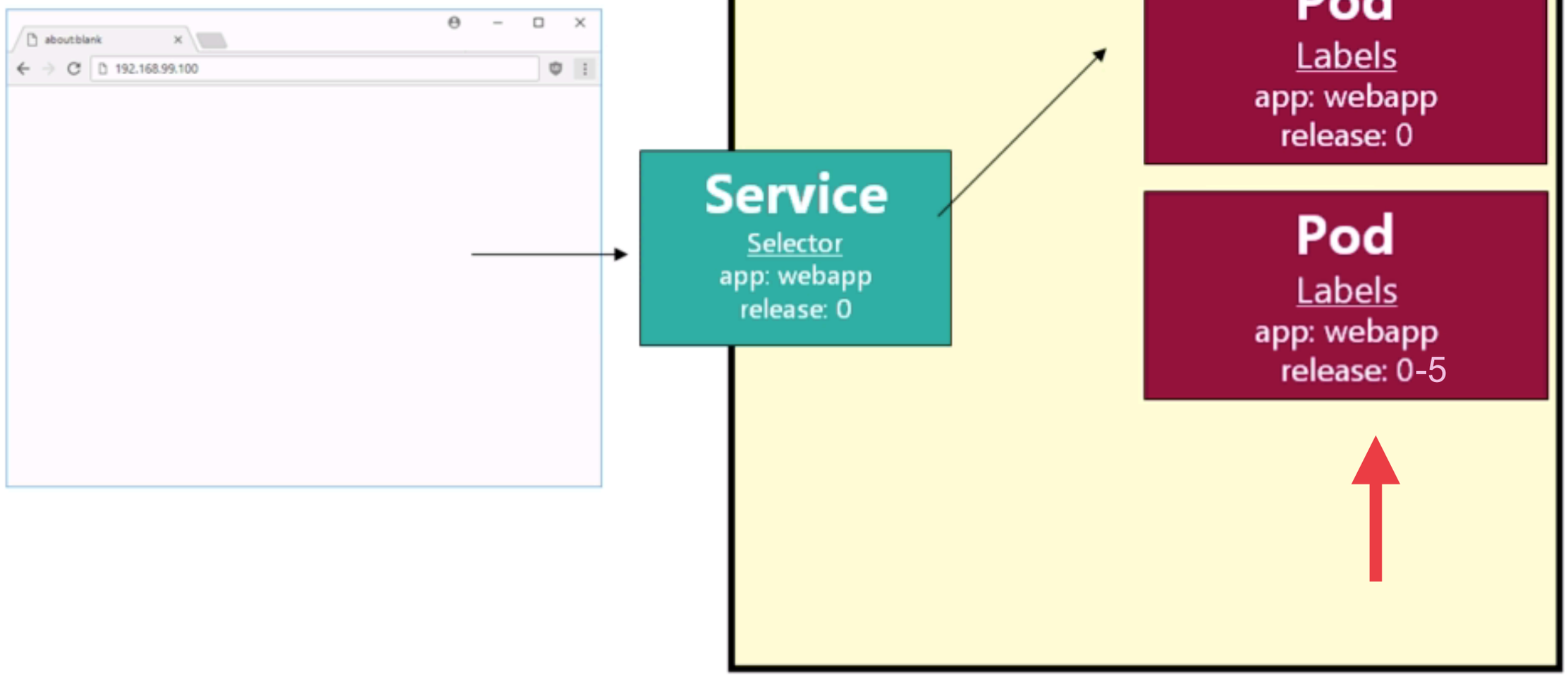


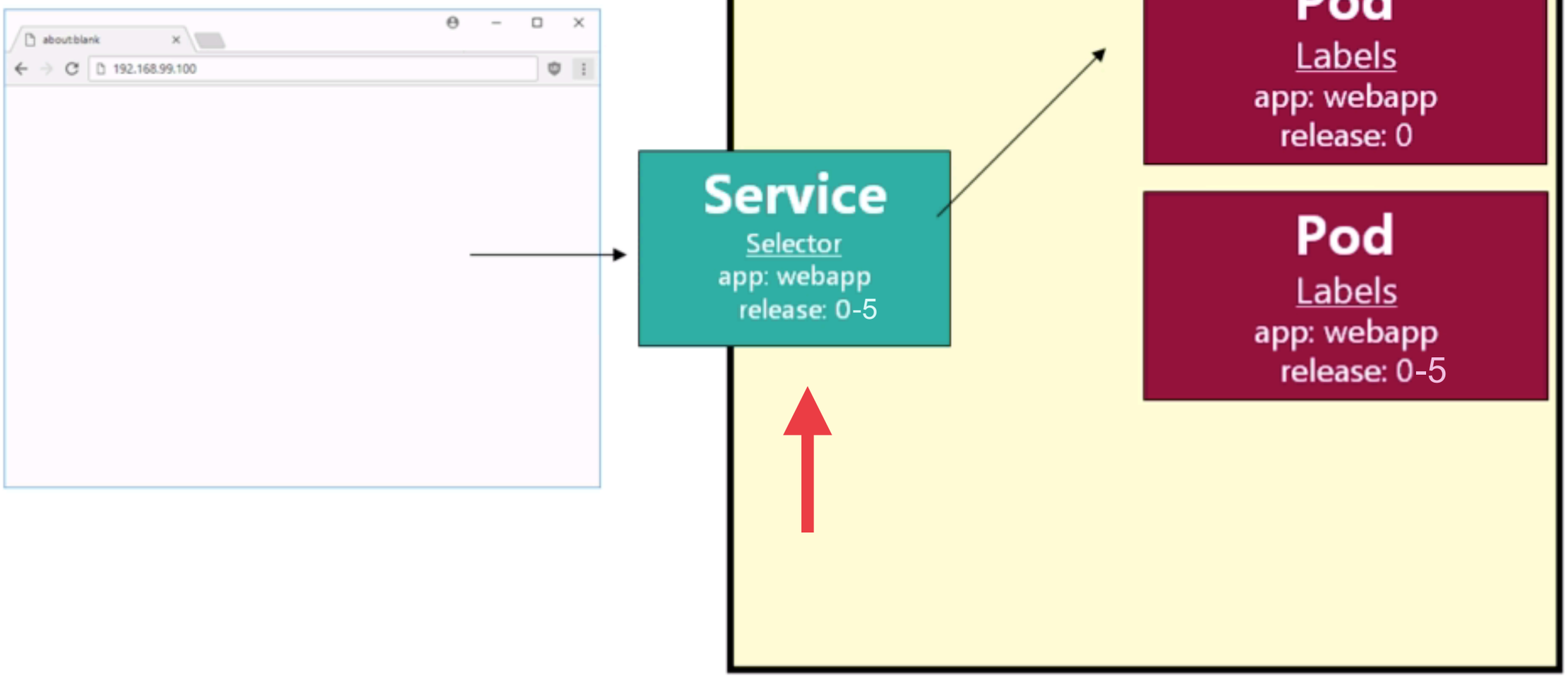

- Until we learn new concepts we can deploy with zero downtime using labels

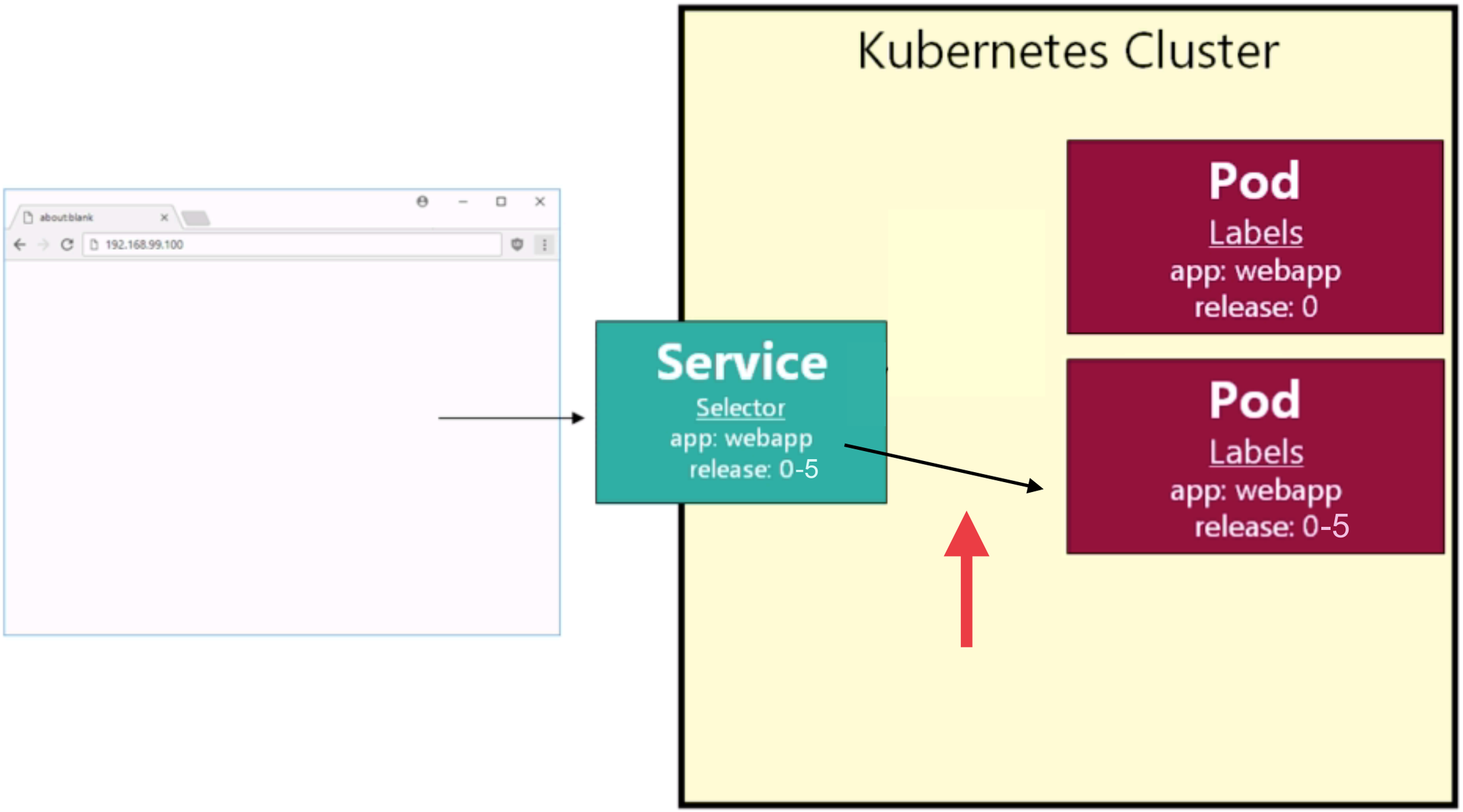

about:blank
192.168.99.100
Kubernetes Cluster
Service
Selector
app: webapp
release: 0
Pod
Labels
app: webapp
release: 0
Pod
Labels
app: webapp
release: 0-5


about:blank
192.168.99.100
Kubernetes Cluster
Service
Selector
app: webapp
release: 0
Pod
Labels
app: webapp
release: 0
Pod
Labels
app: webapp
release: 0-5


about:blank
192.168.99.100
Kubernetes Cluster
Service
Selector
app: webapp
release: 0
Pod
Labels
app: webapp
release: 0
Pod
Labels
app: webapp
release: 0-5


about:blank
192.168.99.100
Kubernetes Cluster
Service
Selector
app: webapp
release: 0
Pod
Labels
app: webapp
release: 0
Pod
Labels
app: webapp
release: 0-5


about:blank
192.168.99.100
Kubernetes Cluster
Service
Selector
app: webapp
release: 0-5
Pod
Labels
app: webapp
release: 0
Pod
Labels
app: webapp
release: 0-5




- This is a very fast operation the pod is already started

- We can switch between release0 and release0-5 at will

- We can start thinking that is possible support a production and development release on the same cluster (it is not the right way to do it or the more elegant one, but it is possible)

pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: webapp
  labels:
    app: webapp
    release: "0"

spec:
  containers:
  - name: webapp
    image: richardchesterwood/k8s-fleetman-webapp-angular:release0

---

```

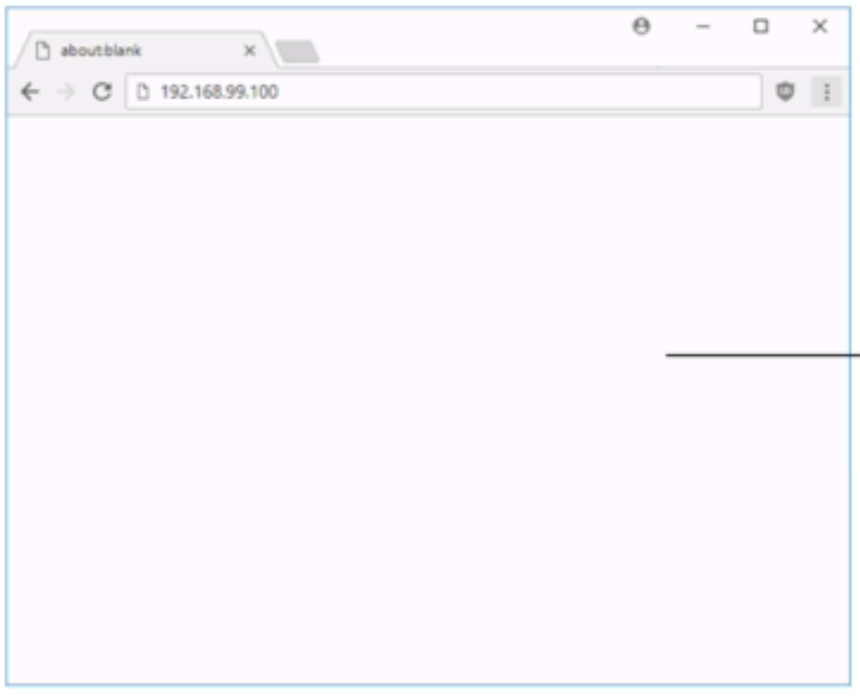
yaml document separator

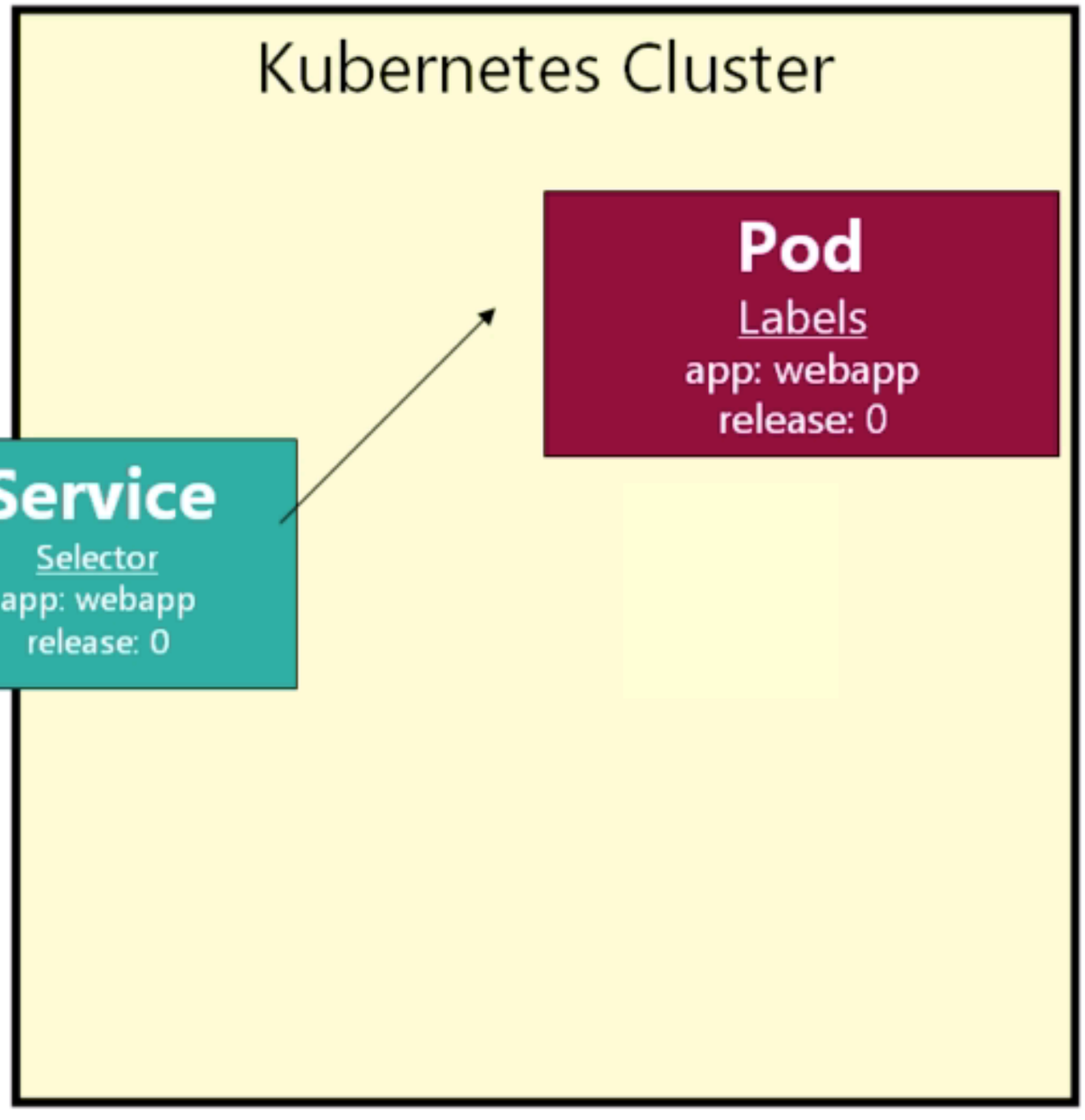
```
16  apiVersion: v1
17  kind: Pod
18  metadata:
19    name: webapp-release-0-5
20    labels:
21      app: webapp
22      release: "0-5"
23
24  spec:
25    containers:
26    - name: webapp
27      image: richardchesterwood/k8s-fleetman-webapp-angular:release0-5
28
```

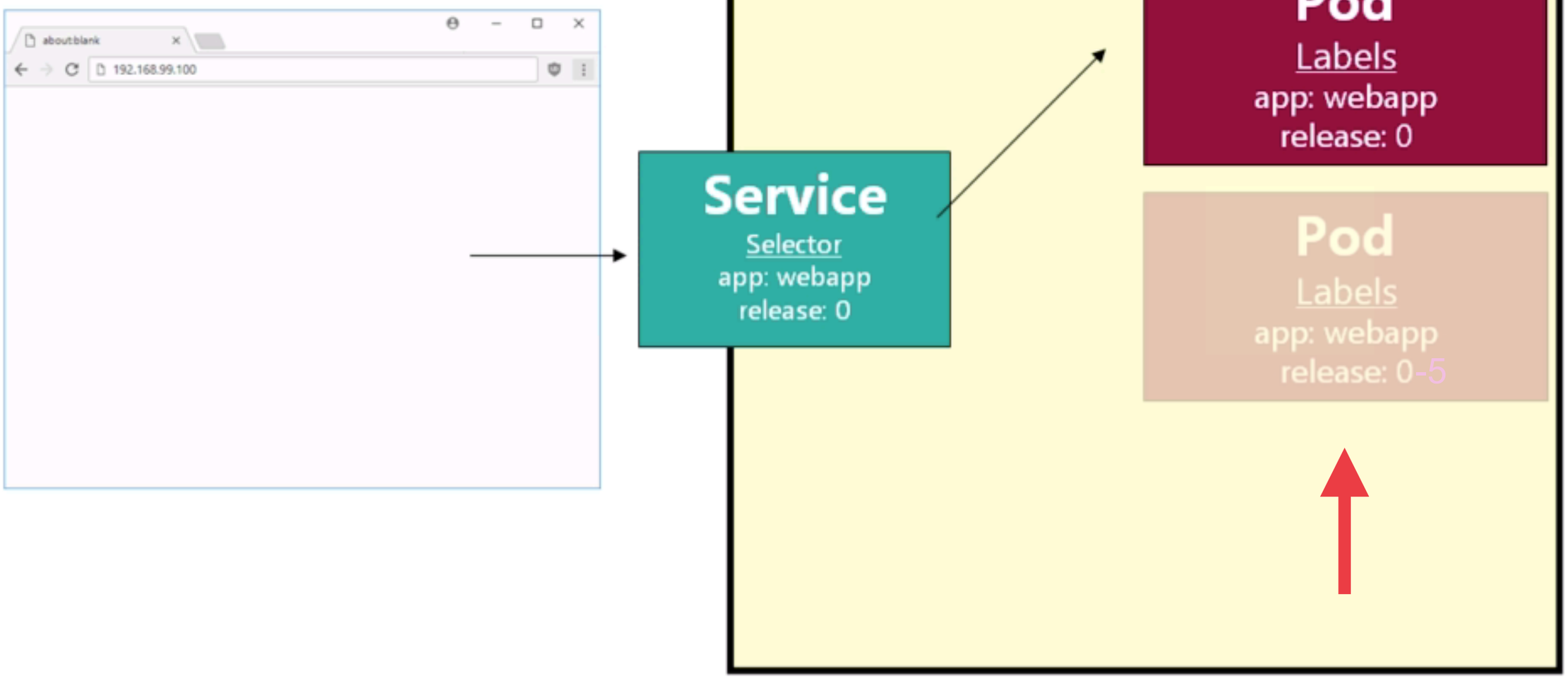
webapp-service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: fleetman-webapp

spec:
  selector:
    app: webapp
    release: "0"

  ports:
  - name: http
    port: 80
    nodePort: 30080

  type: NodePort

```

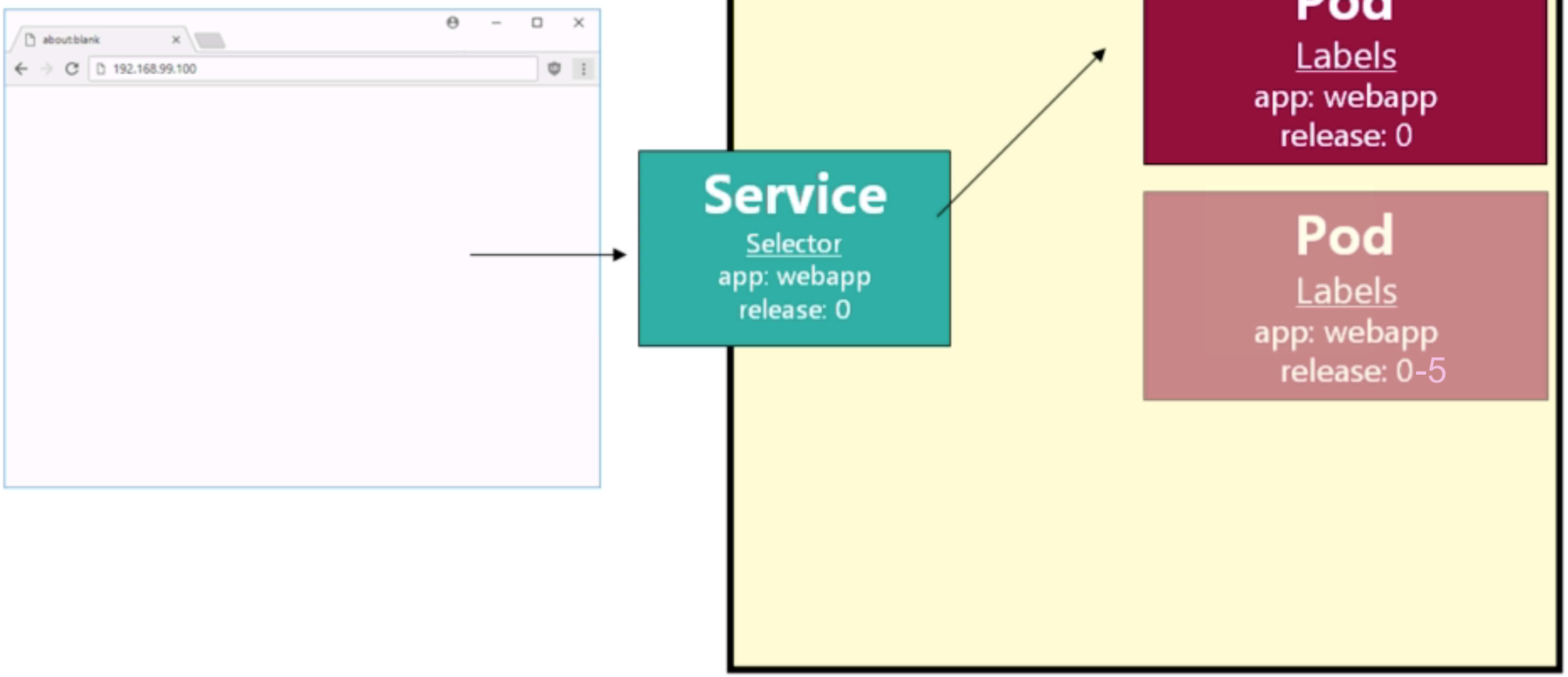
# *This is our scenario*



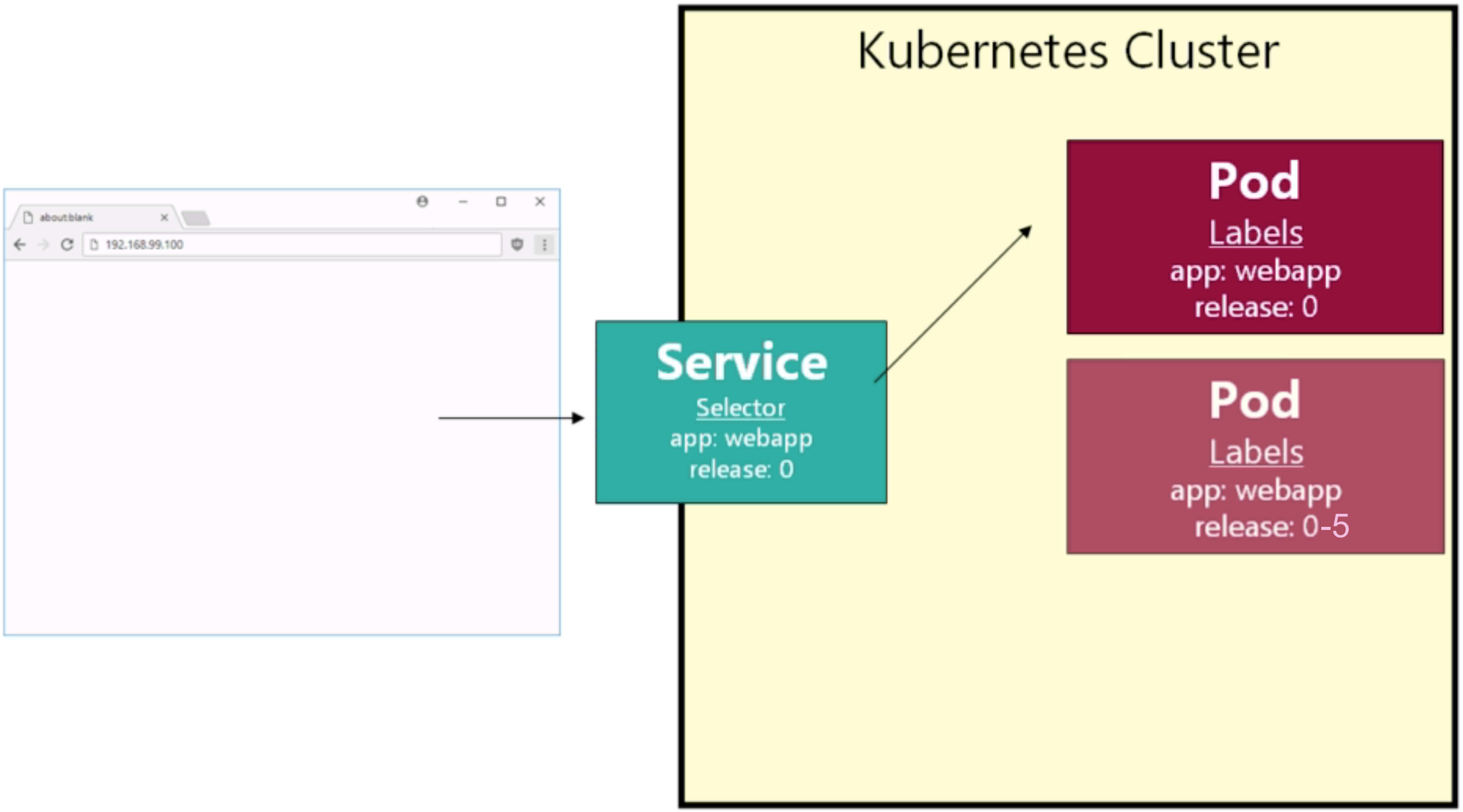





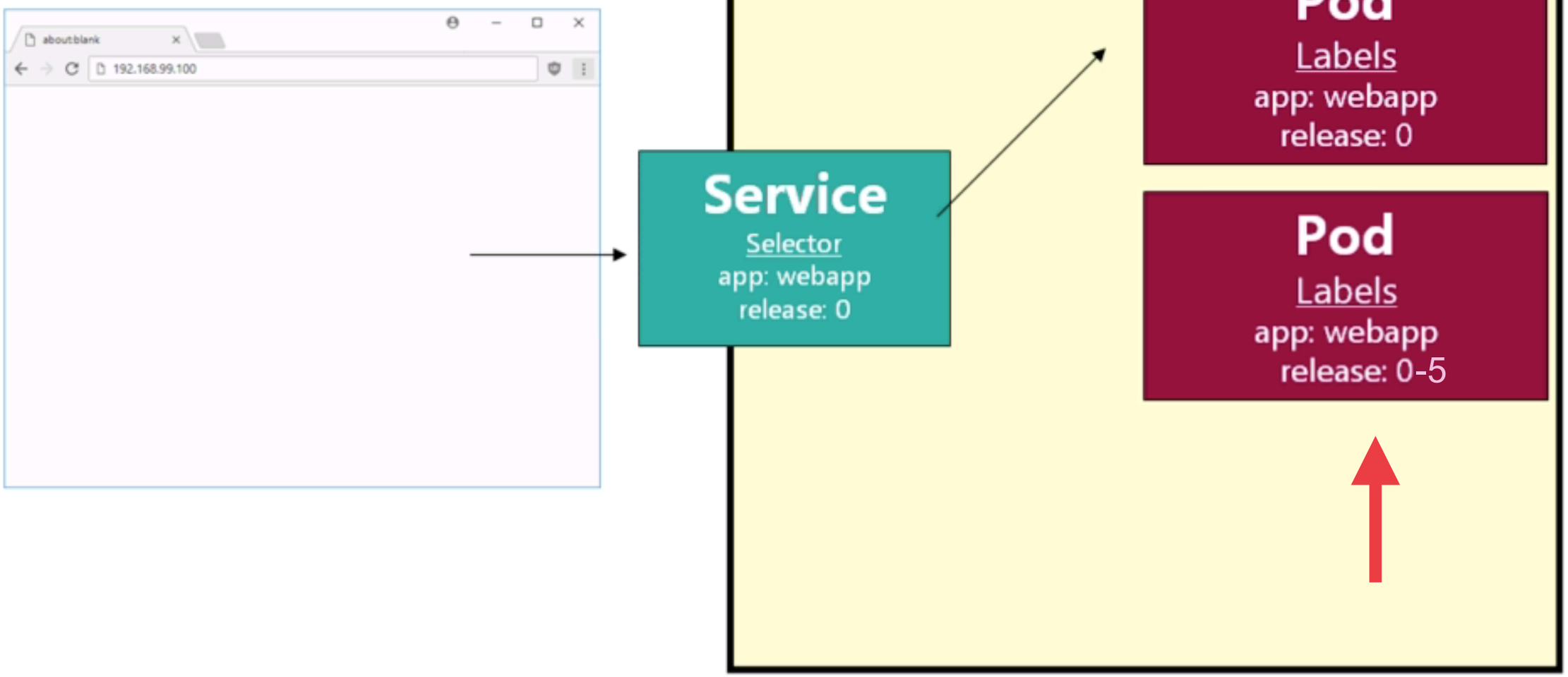

***Pod deploying...***




***Pod deploying...***




***Pod deploying...***




***Pod deployed!***

- `kubectl apply -f pod.yaml`

```
~
$ kubectl apply -f pod.yaml
```

- `kubectl apply -f webapp-service.yaml`

```
~
$ kubectl apply -f webapp-service.yaml
```

# kubectl get all

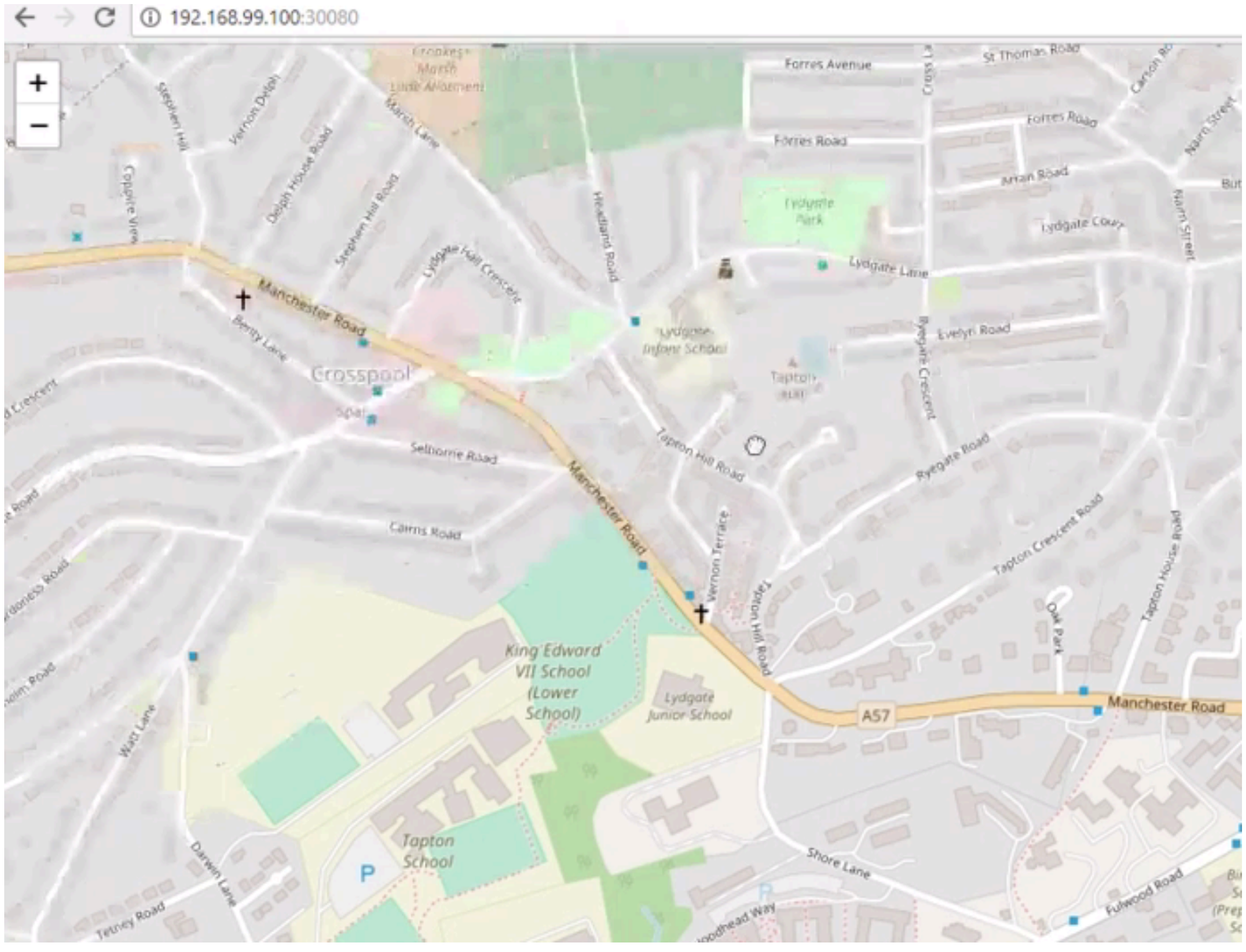
```
NAME                          READY     STATUS    RESTARTS   AGE
pod/webapp                    1/1       Running   1          20h
pod/webapp-release-0-5        1/1       Running   0          1m

NAME                           TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
service/fleetman-webapp        NodePort    10.107.11.35    <none>        80:30080/TCP
service/kubernetes             ClusterIP   10.96.0.1       <none>        443/TCP
```

- kubectl describe service fleetman-webapp

```
Name:                     fleetman-webapp
Namespace:                default
Labels:                   <none>
Annotations:              kubectl.kubernetes.io/last-applied-configuration=
otations":{},"name":"fleetman-webapp","namespace":"default"},"spec":{"ports
Selector:                 app=webapp,release=0
Type:                     NodePort
IP:                       10.107.11.35
Port:                     http  80/TCP
TargetPort:               80/TCP
NodePort:                 http  30080/TCP
Endpoints:                172.17.0.4:80
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```

ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO

# *Let's try it now!*



P.PORTO

ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO

# *Now let's change to releasase0-5*

webapp-service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: fleetman-webapp

spec:
  selector:
    app: webapp
    release: "0-5"

  ports:
  - name: http
    port: 80
    nodePort: 30080

  type: NodePort

```

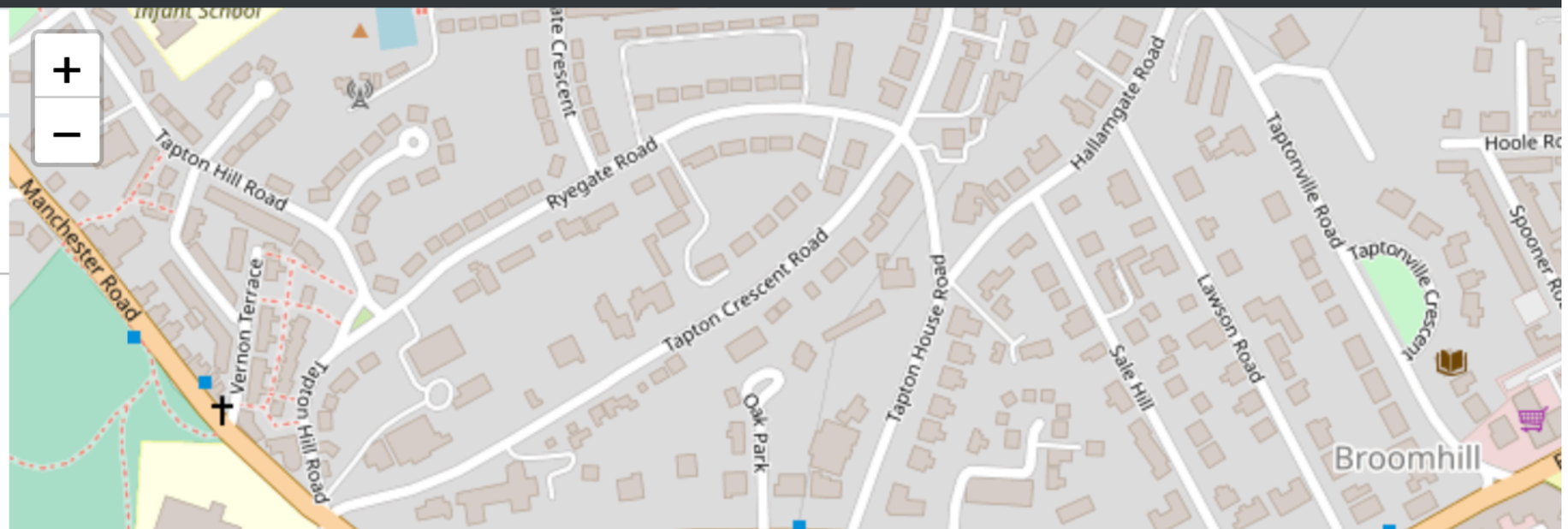
- `kubectl apply -f webapp-service.yaml`

```
~
$ kubectl apply -f webapp-service.yaml
```

Fleet Management System PROTOTYPE. Release 0.5

| Name | Last seen | Speed mph |
|---|---|---|

Live vehicle updates will appear here.
Once we've implemented it!



# Some useful commands

- `kubectl get all`
- `Kubectl get pods`
- `Kubectl get po`
- `Kubectl get po --show-labels`
- `Kubectl get po --show-labels -l release=0`

- https://www.martinfowler.com/bliki/StranglerApplication.html
- http://microservices.io/patterns/monolithic.html
- http://microservices.io/patterns/microservices.html
- Richardson, Ch. 2016. Refactoring a Monolith into Microservices
- Vivek Juneja, 2016. From Monoliths To Microservices: An Architectural Strategy

P.PORTO

Polytechnic of
Porto

ESTG - School of
Management and Technology