Online Multimedia

Winter Semester 2019/20

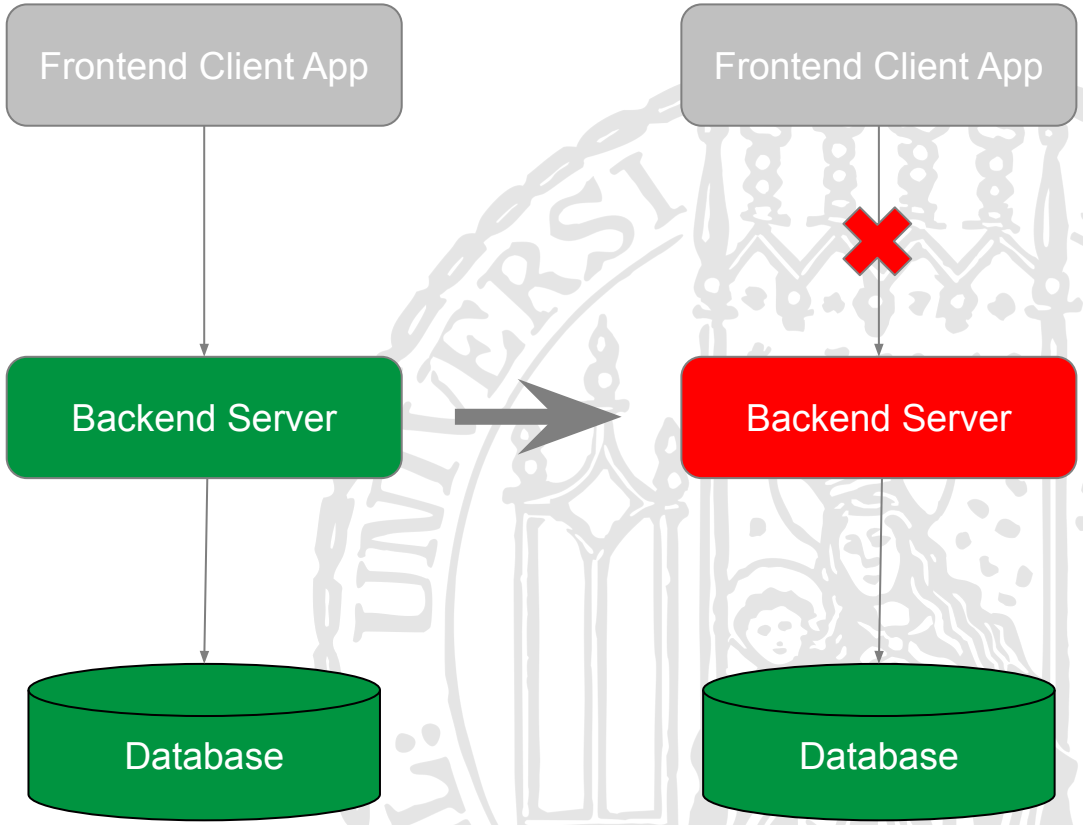
Tutorial 11 – Web Infrastructures

Today's Agenda

- Web Application Architectures
- Containerization and Orchestration
- Web Infrastructure: Load Balancing
- Web Infrastructure: Deployment Strategy
- Web Infrastructure: CDN

Architecture: Monolithic

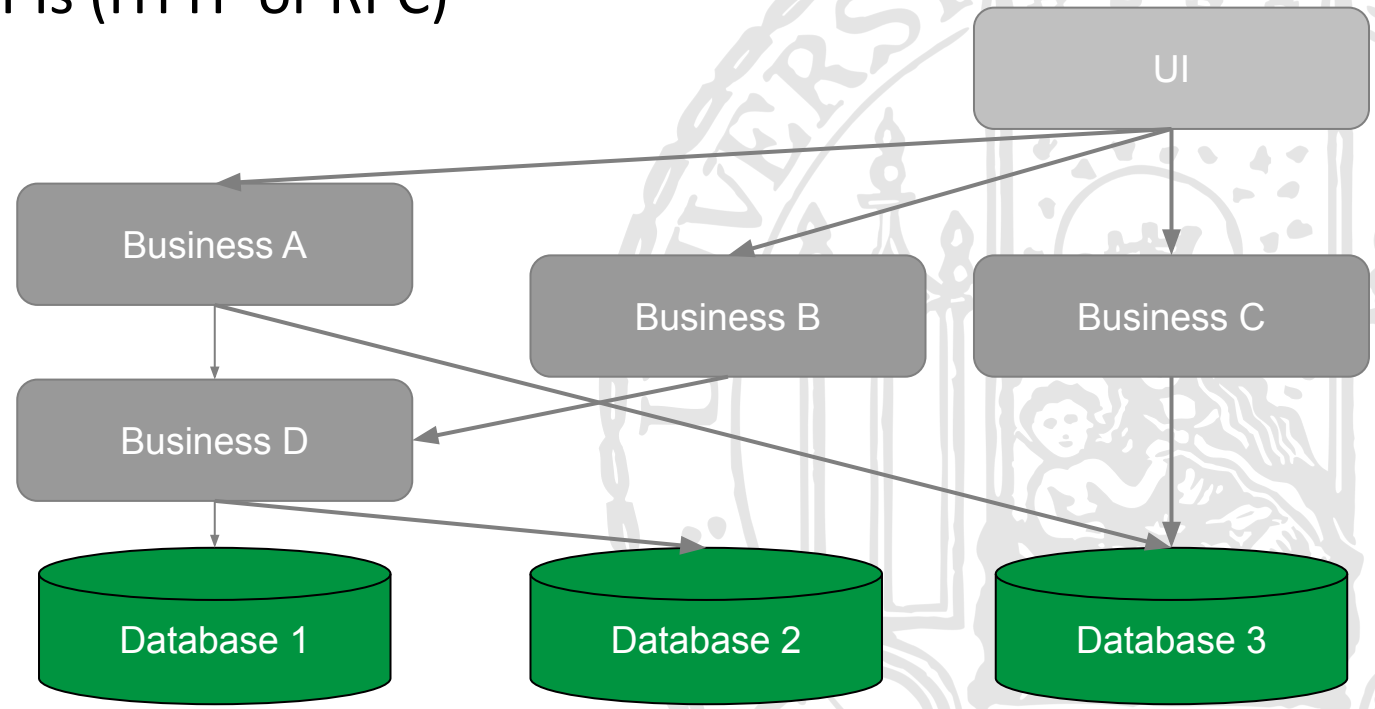
- Big giant all in one application
- Communicate in a single flow
- Pros
  - Easy to manage
  - Single flow for debugging
- Cons
  - Poor scalability
  - One failure crash the whole app



Example: Poor scalability and fault tolerance

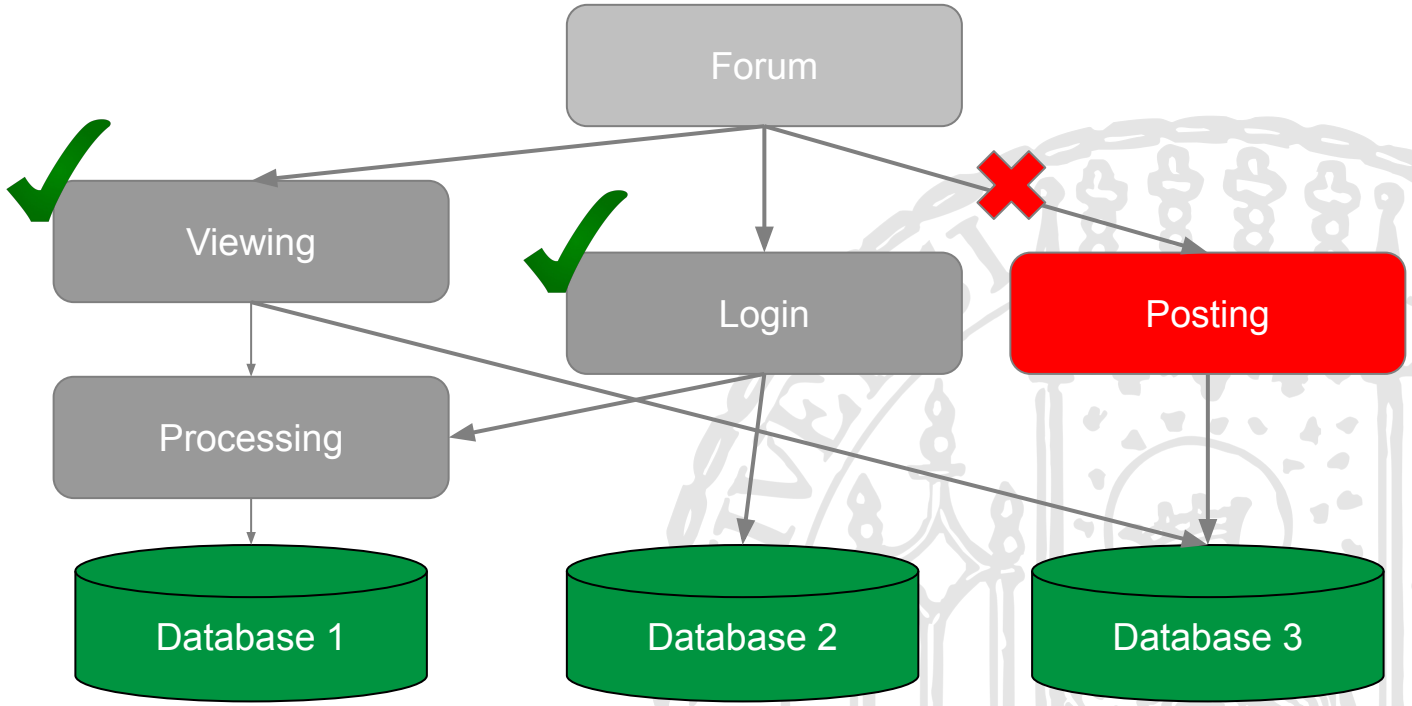
Architecture: Microservice

- Separate business logic functions
- Instead of one big app, several smaller applications
- Communicate via well defined APIs (HTTP or RPC)



Microservices: Pros and Cons

- Pros
  - Fungibility
  - Scalable
  - Fault Isolation
- Cons
  - Complex (mesh) organizations
  - Complex call (bug) tracing



Example: In fault isolation, when posting is failed but login and viewing APIs still survived and operates

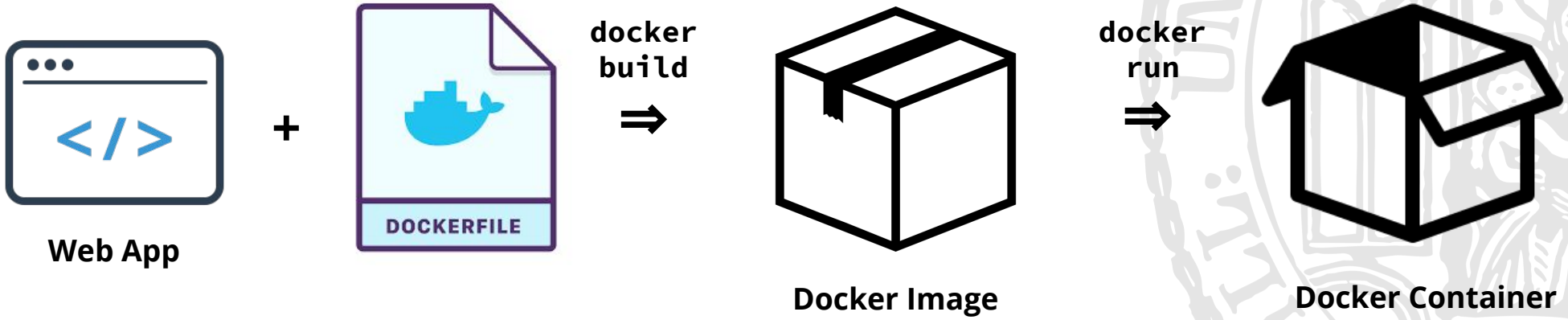
Web Application Deployment Problems

- Deployment is a hard in general
- Because
  - Build dependencies: system environment is different
  - Breaking changes: backend API changes can break frontend
  - Upgrade consistency: cached frontend assets is not updated
  - Traffic control: unbalanced web requests
  - Error rollback: fault tolerance if newer version fails
  - ...

Containerization

Containerization

- Describes the build process for an application
- The built image can be run automatically everywhere with a standard container runtime
- Contains all the command necessary to build the image and run an application
- A containerization solution: **Docker**

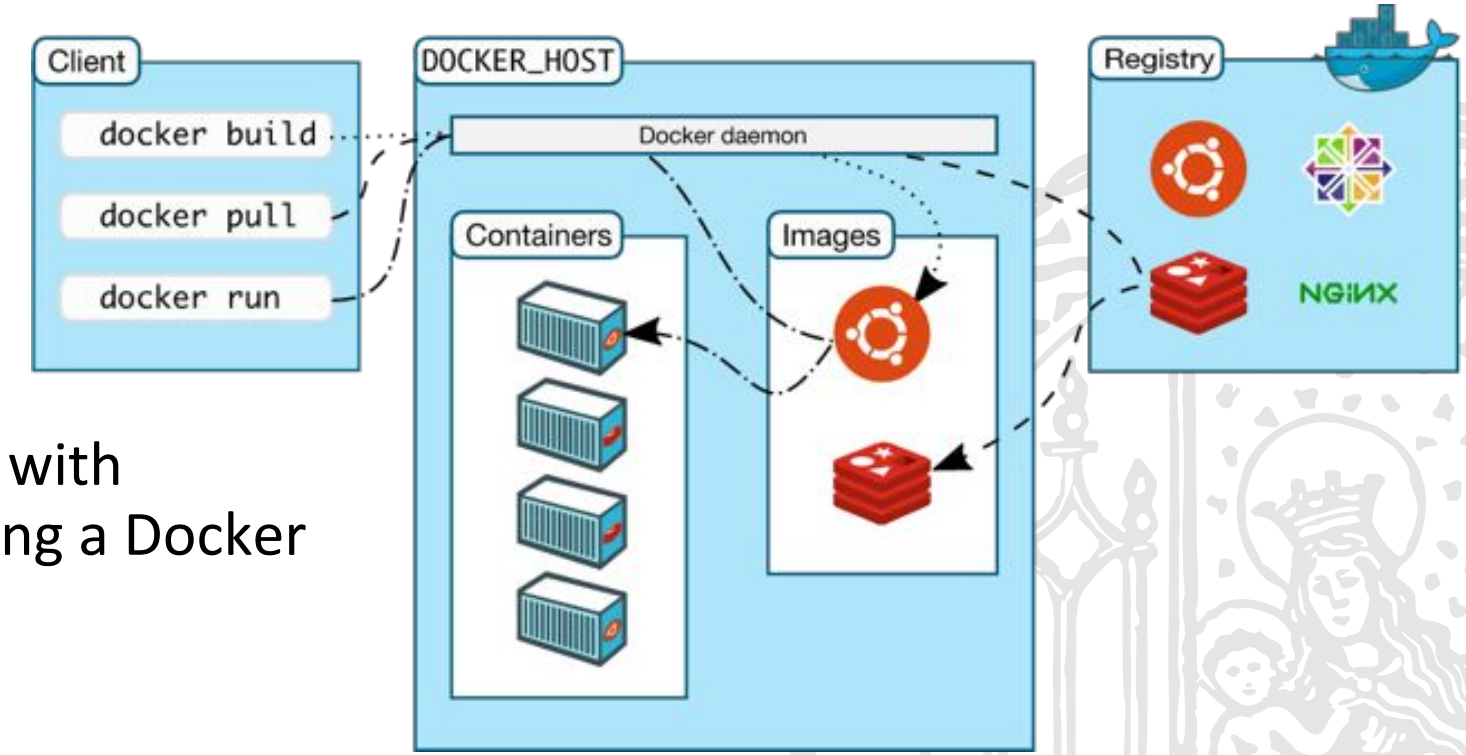


Docker

- Run applications securely isolated in a container, packaged with all its dependencies and libraries.
- Installation: https://docs.docker.com/install/
  - macOS & Windows users: register your self to Docker Hub for downloading
  - Linux user can install docker by `apt install docker.io`
  - We will use Docker Hub for lately

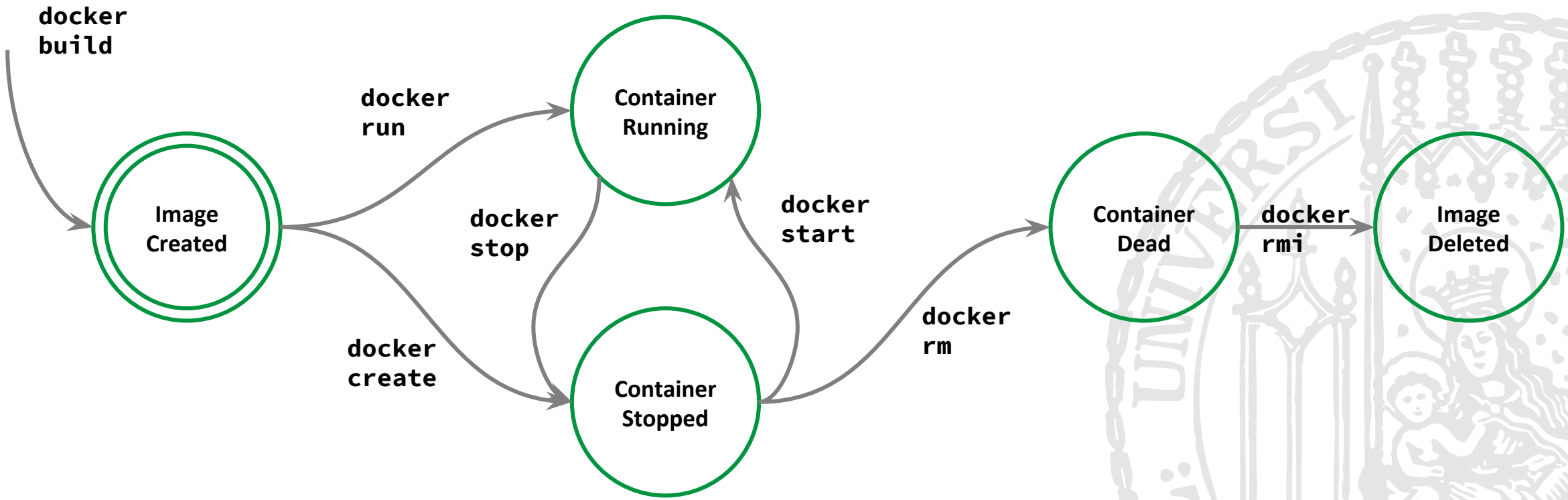
Docker Core Concepts

- Registry
  - A repository to store docker images
- Image
  - A read-only template with instructions for creating a Docker container
- Container
  - A runnable instance of an image



https://docs.docker.com/engine/docker-overview/

Lifecycle of A Docker Container



Container Management Commands

- Use `docker [command] --help` to check manual

Command	Description
`docker build` `-t image .`	build an image
`docker create` `image …` **`docker run`** `image …`	create the container create+start the container
`docker start` `container …` **`docker stop`** `container …`	start the container graceful stop
`docker rm` `container …`	destroy the container
`docker rmi` `image …`	delete specified iamge

Build Docker Image with Dockerfile

01-docker/Dockerfile

```
# build starting from a node.js base image
FROM node:alpine
# define your application working directory
WORKDIR /app
# in the next command, ADD adds ./src from host to WORKDIR in container
ADD src .
# install dependencies for your application
RUN npm install
# define commands to run your container (app)
CMD ["npm", "start"]
```

Build Image

Explanation:

command
naming the image
build from the current folder

- To build a container: `docker build -t imagename:version .`

```
$ docker build -t helloworld:v0.0.1 .
Sending build context to Docker daemon  22.02kB
Step 1/5 : FROM node:alpine
 ---> 364fb8e7f28a
...
Step 5/5 : CMD ["npm", "start"]
 ---> Using cache
 ---> ef3bbc292036
Successfully built ef3bbc292036
```

Push to Registry

Explanation:

command
local image name
registry image name
docker hub username

- To build a container: `docker build -t imagename:version .`
- Then push to Docker Hub registry*:

```
$ docker tag helloworld:v0.0.1 mimuc/helloworld:v0.0.1
$ docker push mimuc/helloworld:v0.0.1
The push refers to repository [docker.io/mimuc/helloworld]
9d50f58b665f: Pushed
1f0ff57a0279: Pushed
...
v0.0.1: digest:
sha256:deeaeb483e51341ae54da6c45f9b6989ac6ff481cc11c5e55769ad284f857e6f
size: 1783
```

* you need register to docker hub for using docker registry.
In production, you can setup your own image registry.

Run Container

Explanation:

command
port mapping from host to container
running as a daemon process
image name

- To run a container:

```
$ docker run -p 1234:3000 -d mimuc/helloworld:v0.0.1
802c076b2895e3974c91138c8a41a5462eeeab12df090f4f49714566867f5db2
$ docker ps
CONTAINER ID        IMAGE                          COMMAND
CREATED             STATUS              PORTS                    NAMES
802c076b2895        mimuc/helloworld:v0.0.1
"docker-entrypoint.s…"   About a minute ago   Up About a minute
0.0.0.0:1234->3000/tcp   zealous_diffie
$ curl 0.0.0.0:1234
{"hello":"world","name":"Z2vTvgmxb0"}
```

Make a request (or do it in a browser)

Server responsed

Breakout #1

Install Docker, and get the the previous example running locally

Timeframe: 10 Minutes

Orchestration


KUBERNETES
LIGHTYEAR
KUBERNETES EVERYWHERE
imgflip.com

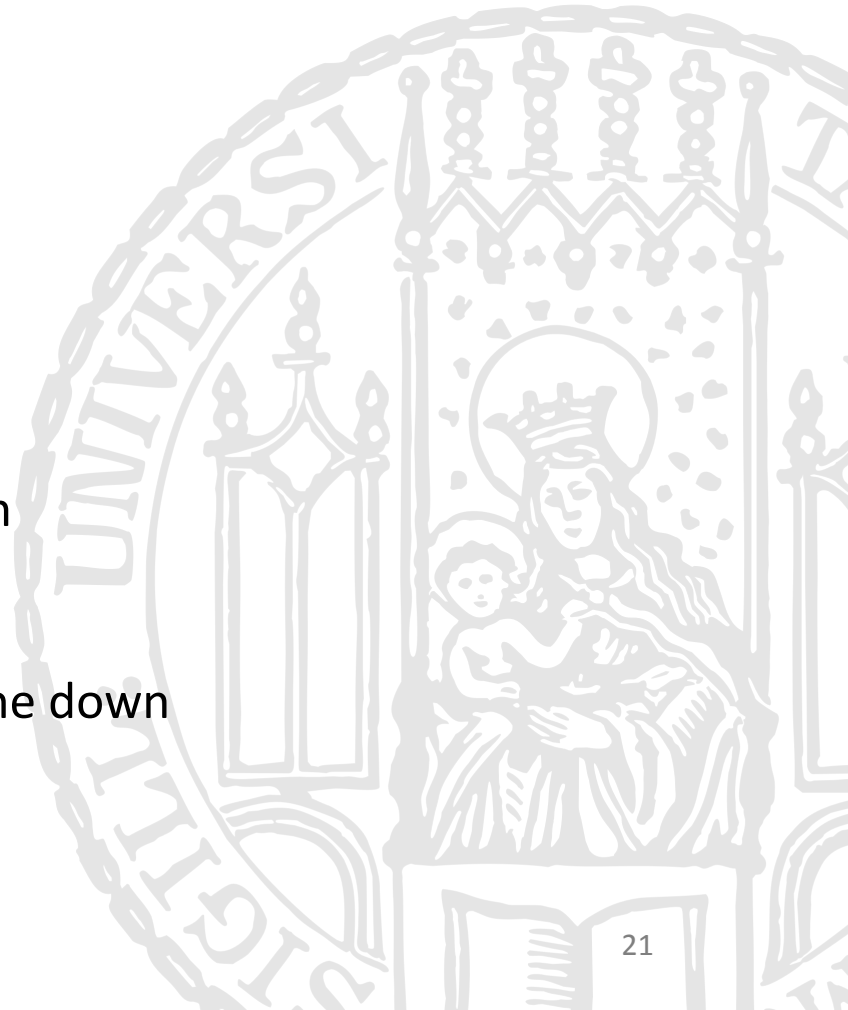
Kubernetes (k8s)

kubernetes

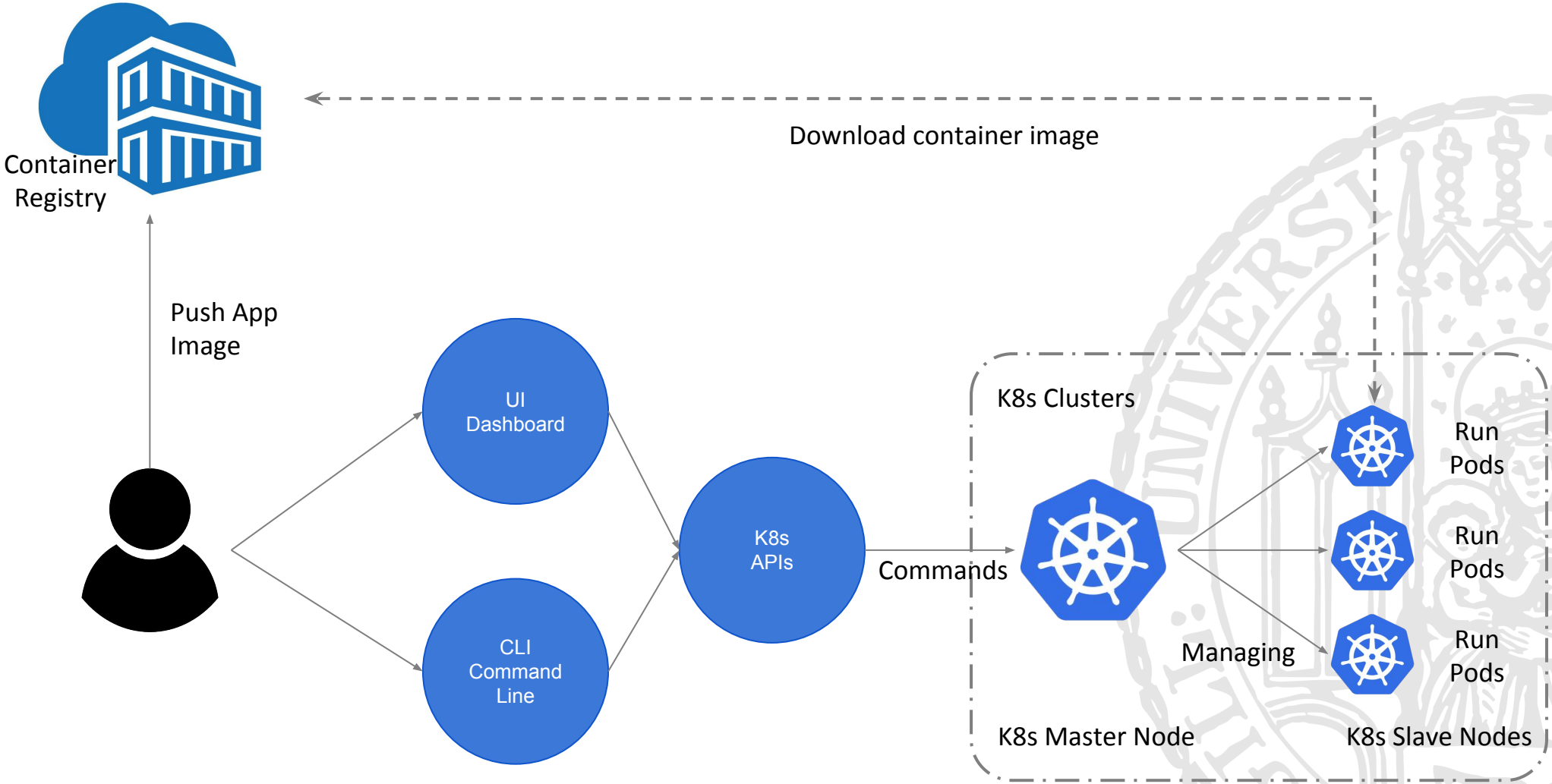
- An open source container orchestration engine for automating deployment, scaling, and management of containerized applications
- In production, k8s is deployed on in a cluster, which requires at least a Master node and a Slave node.
- To use k8s locally (single node):
  - Quickly enable K8s in Docker's Preference panel (macOS & Windows)
  - Linux user can install minikube.
- Core Command: `kubectl [command] …`
- Use YAML configuration file to define application orchestration

Kubernetes (k8s) Core Concepts

- We cover the following concepts today:
  - **Node**
    - Machine runs your application pods
  - **Pod**
    - Runs 1 more containers
  - **Service**
    - A logical set of Pods and a policy by which to access them
  - **Deployment**
    - Defines desired application state, e.g. what happens if one down
    - Configured in a YAML file



K8s Orchestration Overview



K8s Deployment

- Config file defines desired deployment state
- Two major parts
  - metadata about the kind
  - specifications about the kind
    - replicas: number of pod replications
    - containers: containers in the a pod

02-k8s-deployment/kubernetes/deployments.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
 name: mimuc-app
spec:
 replicas: 3
 selector:
   matchLabels:
     app: mimuc-app
 template:
   metadata:
     labels:
       app: mimuc-app
   spec:
     containers:
       - name: mimuc-app
         image: mimuc/helloworld:v0.0.1
         ports:
           - containerPort: 3000
```

```
$ kubectl apply -f kubernetes/deployment.yaml
deployment.apps/mimuc-app configured
$ kubectl get pods

NAME                          READY   STATUS    RESTARTS   AGE
mimuc-app-5fb7fc9df5-5xlgm    1/1     Running   0          12s
mimuc-app-5fb7fc9df5-77l47    1/1     Running   0          8s
mimuc-app-5fb7fc9df5-z88z7    1/1     Running   0          10s
```

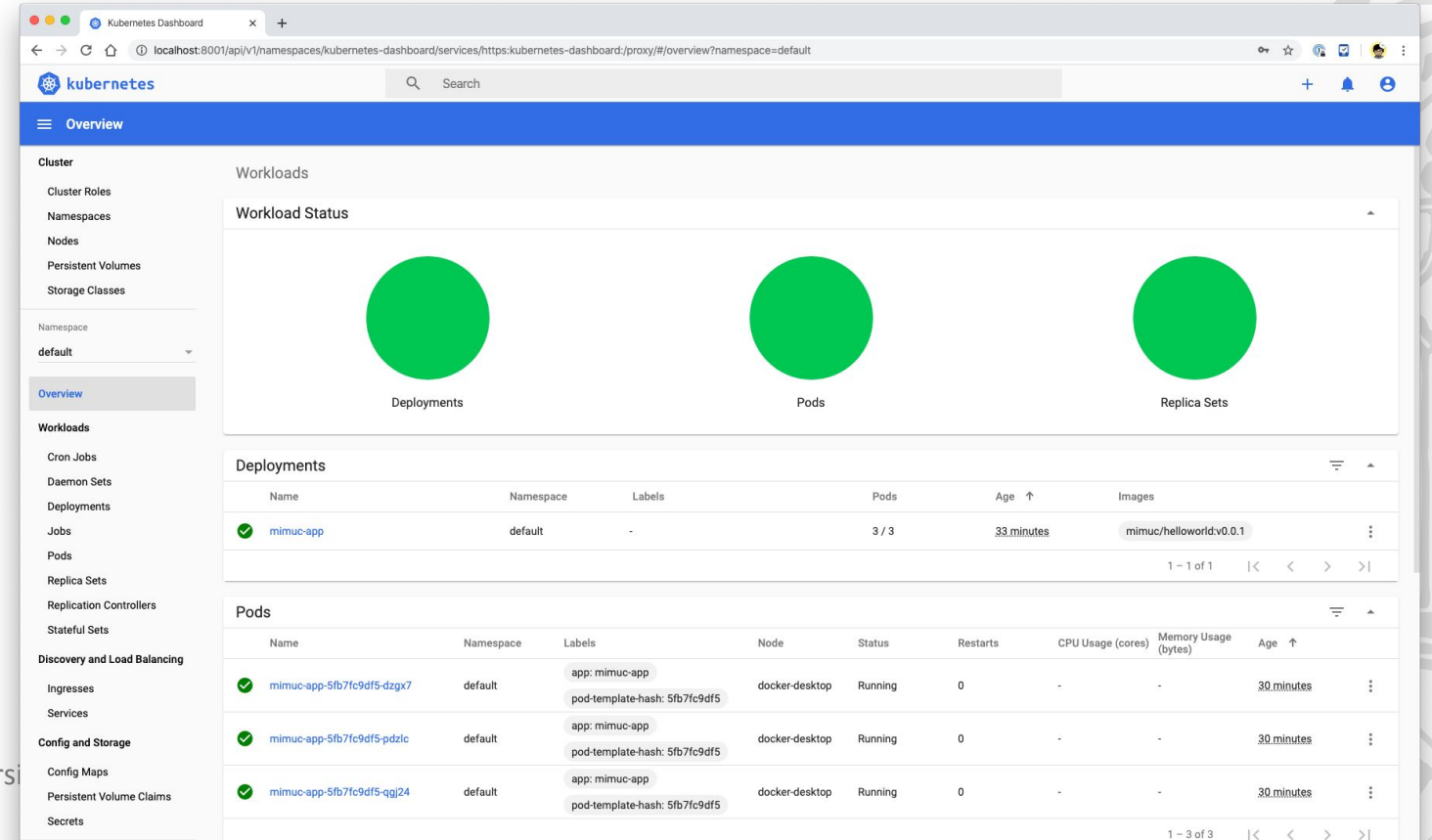
K8s Management Commands

- Use `kubectl [command] --help` to check manual

Command	Description
`kubectl get …`	Display one or many resources, e.g. pods
`kubectl apply …`	Apply a configuration to a resource
`kubectl describe …`	Show details of a specific resource or group of resources
`kubectl delete …`	Delete resources

K8s Dashboard

- A web-based K8s user interface, provides information on the state of k8s resources in the cluster on any errors that may have occurred.



K8s Dashboard: Create Dashboard

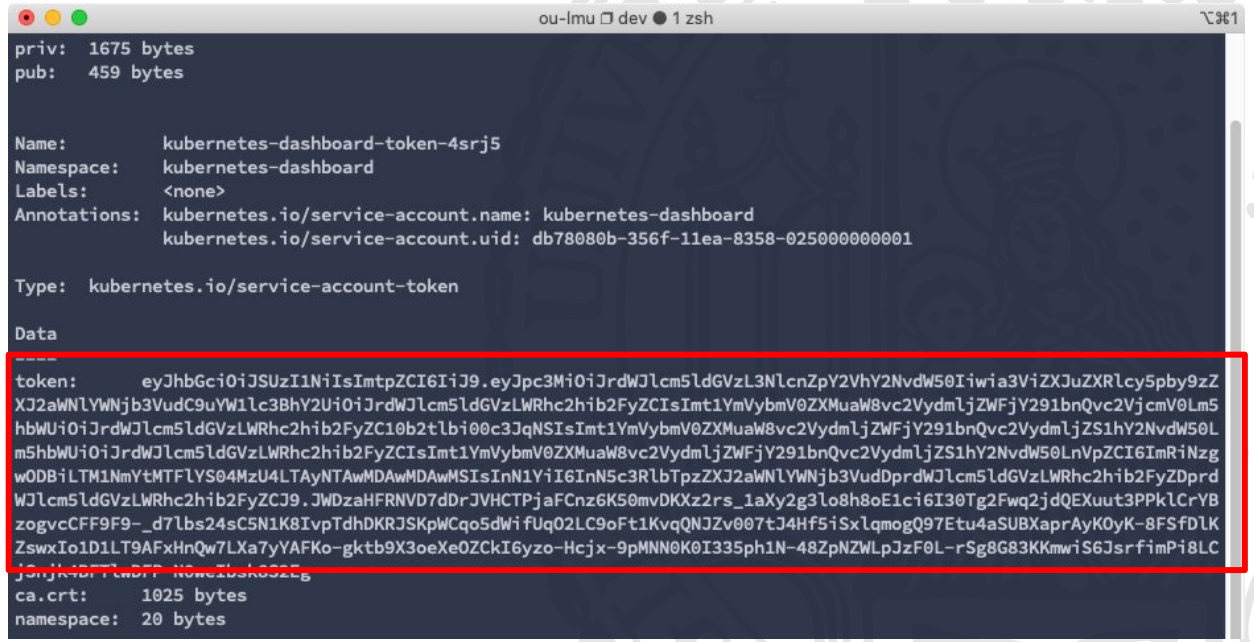
```
$ kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/
v2.0.0-rc1/aio/deploy/recommended.yaml

namespace/kubernetes-dashboard created
serviceaccount/kubernetes-dashboard created
service/kubernetes-dashboard created
secret/kubernetes-dashboard-certs created
...
```

K8s Dashboard: Access Credentials

- Kubernetes Dashboard requires access credentials for security reason:

```
$ kubectl -n kubernetes-dashboard describe secret
$(kubectl -n kubernetes-dashboard get secret | grep
admin-user | awk '{print $1}')
```

Dashboard Access Token



K8s Dashboard: Open

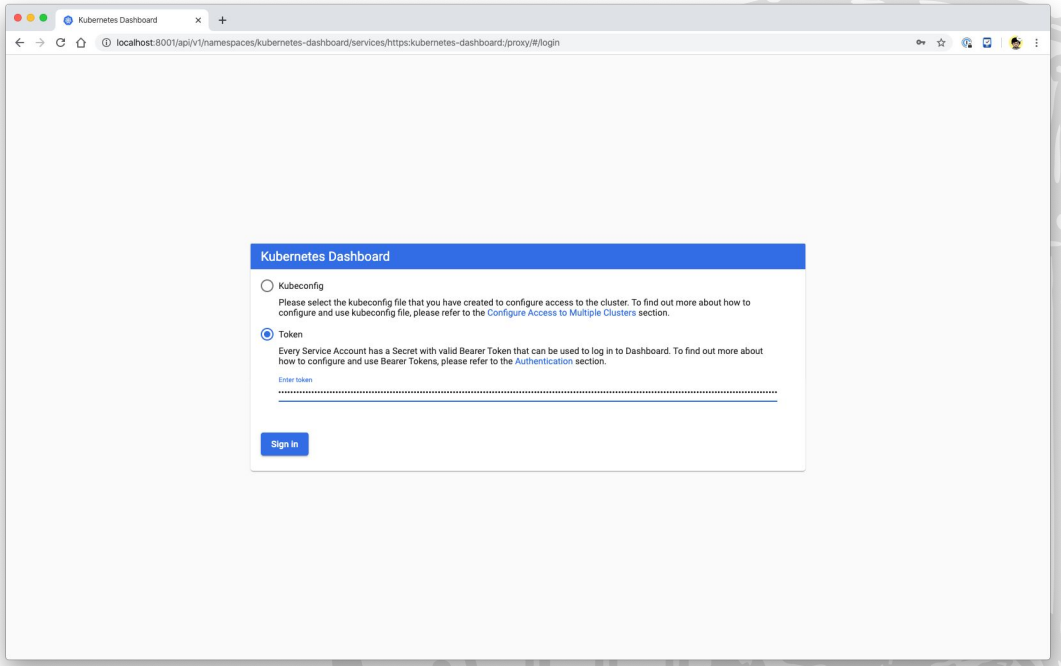
- Start proxy

  ```
  $ kubectl proxy --port=8001
  ```

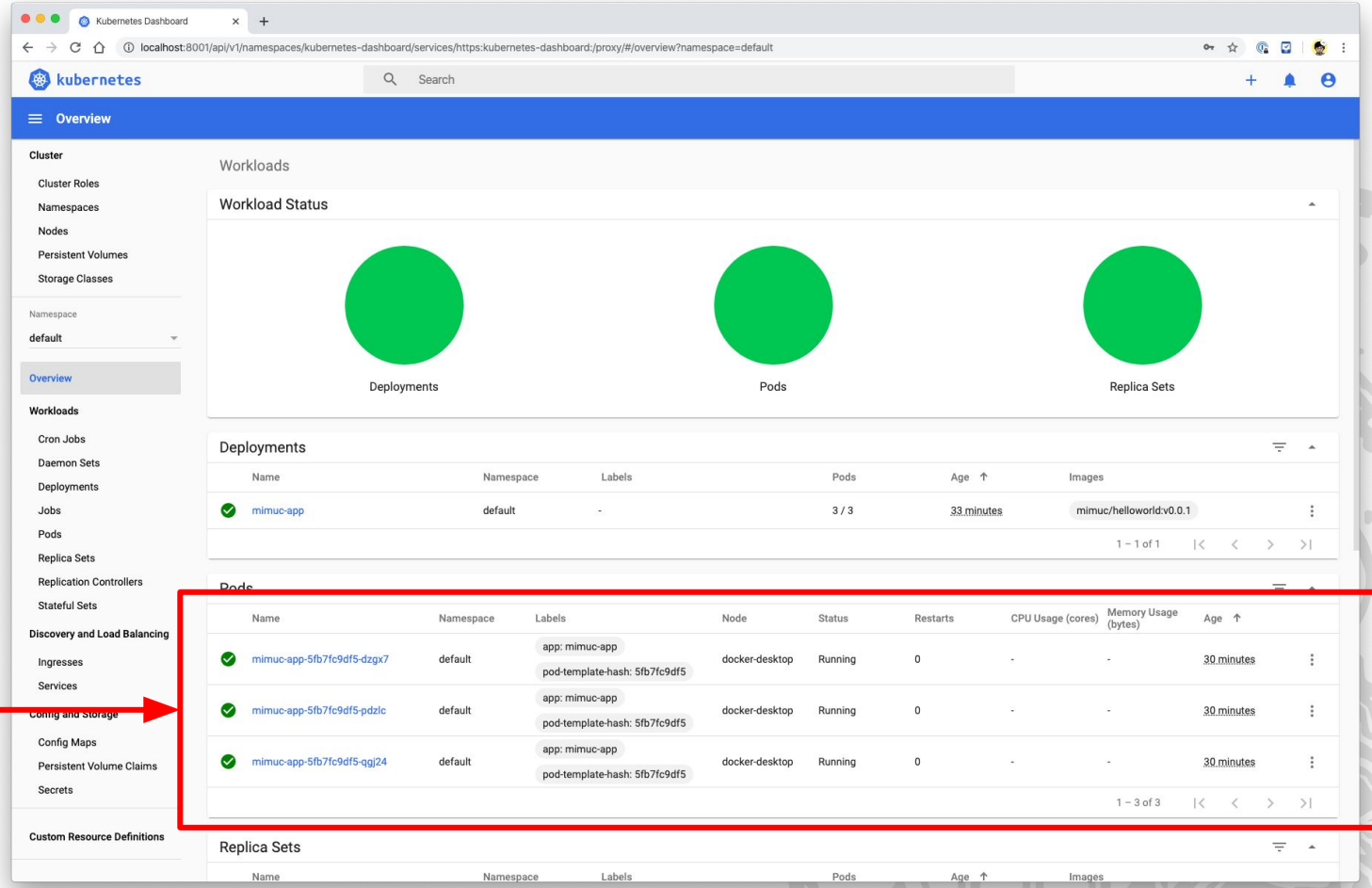
- Open the link in a browser:

  ```
  http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/
  ```

- Input the token you get from last step



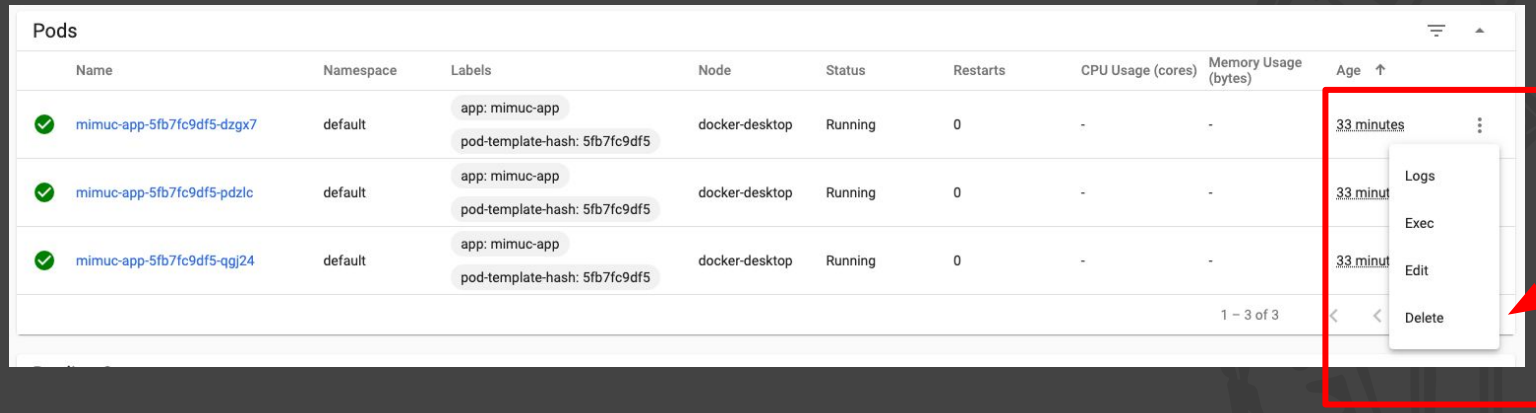
Deployed Application Pods (Replicas)



Breakout #2

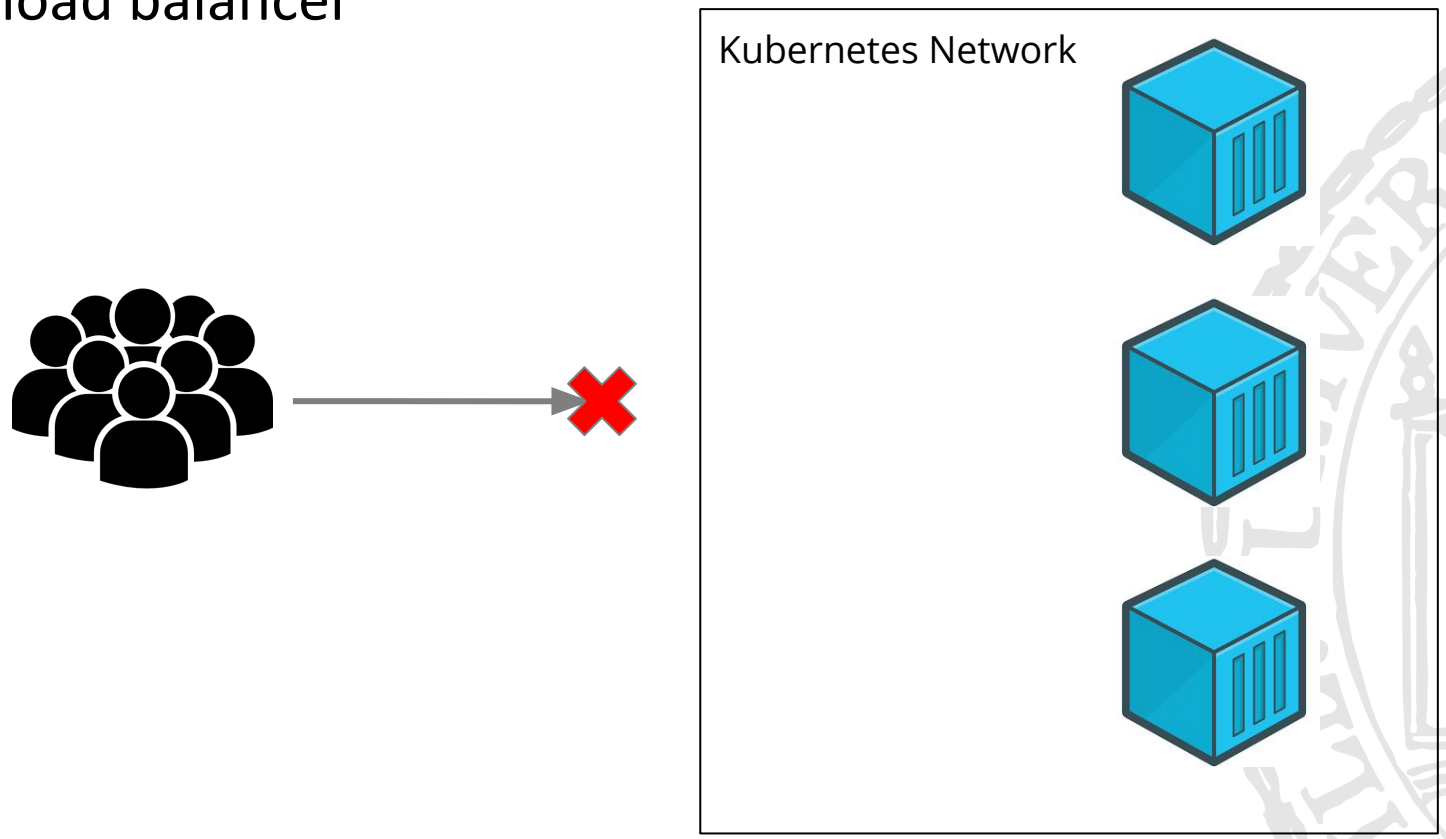
- Get Kubernetes and dashboard running locally
- Try to kill one container manually, observe what did k8s reacts to it.

Timeframe: 10 Minutes



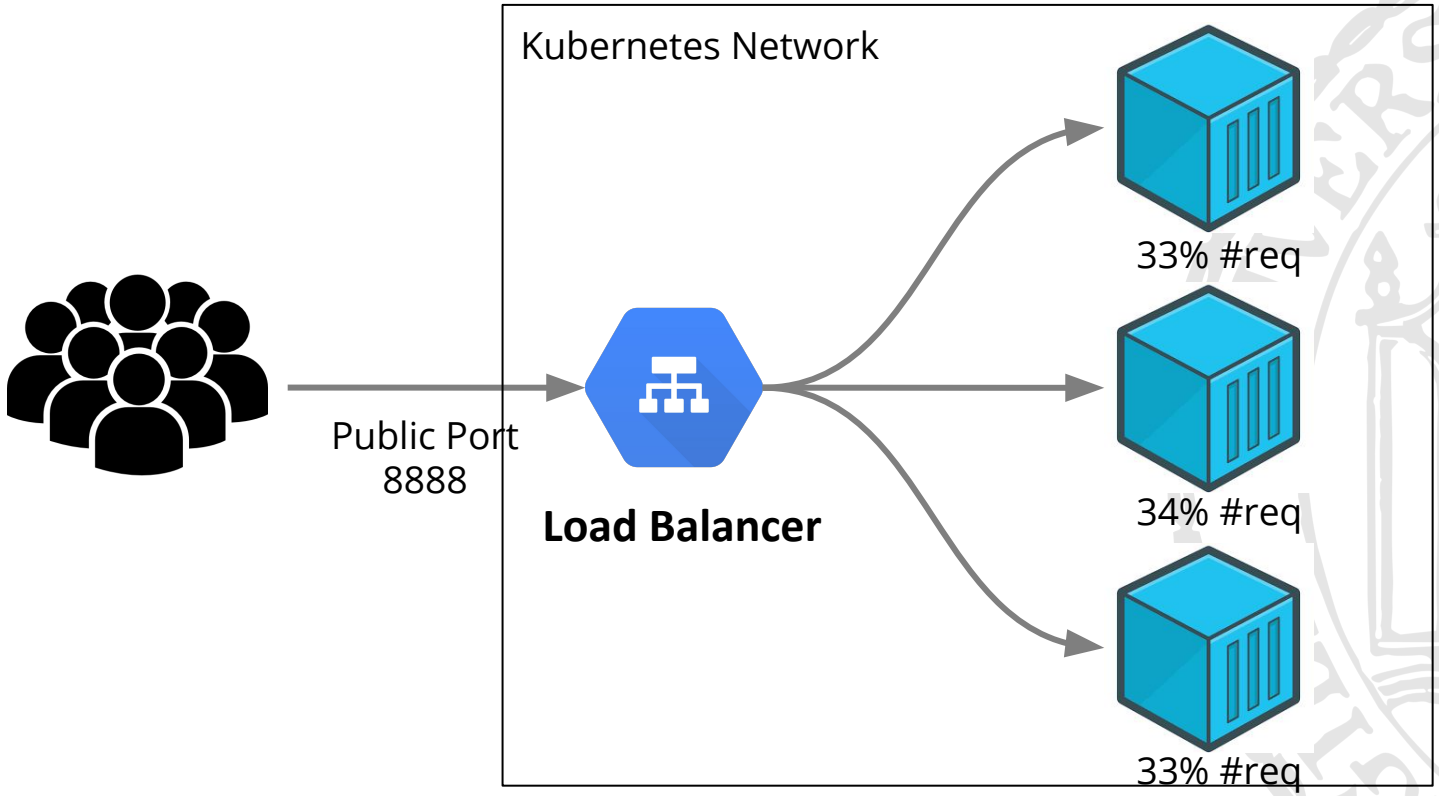
Expose Pods to Public

- The replicated applications is not accessible outside kubernetes yet, we need deploy a load balancer



Infrastructure: Load Balancer

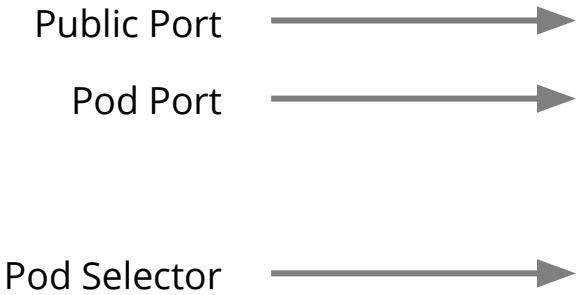
- Improves the distribution of workloads across multiple computing resources, such maximize throughput, minimize response time, and avoid overload of any single resource, e.g. monolithic server



K8s Service: Load Balancer

- Define a Service in the deployment.yaml configuration file
- Apply the new configuration

```
kubectl apply -f
kubernetes/deployment.yaml
```




```yaml
---
apiVersion: v1
kind: Service
metadata:
 name: mimuc-service
spec:
 type: LoadBalancer
 ports:
 - port: 8888
   targetPort: 3000
 selector:
   app: mimuc-app
```

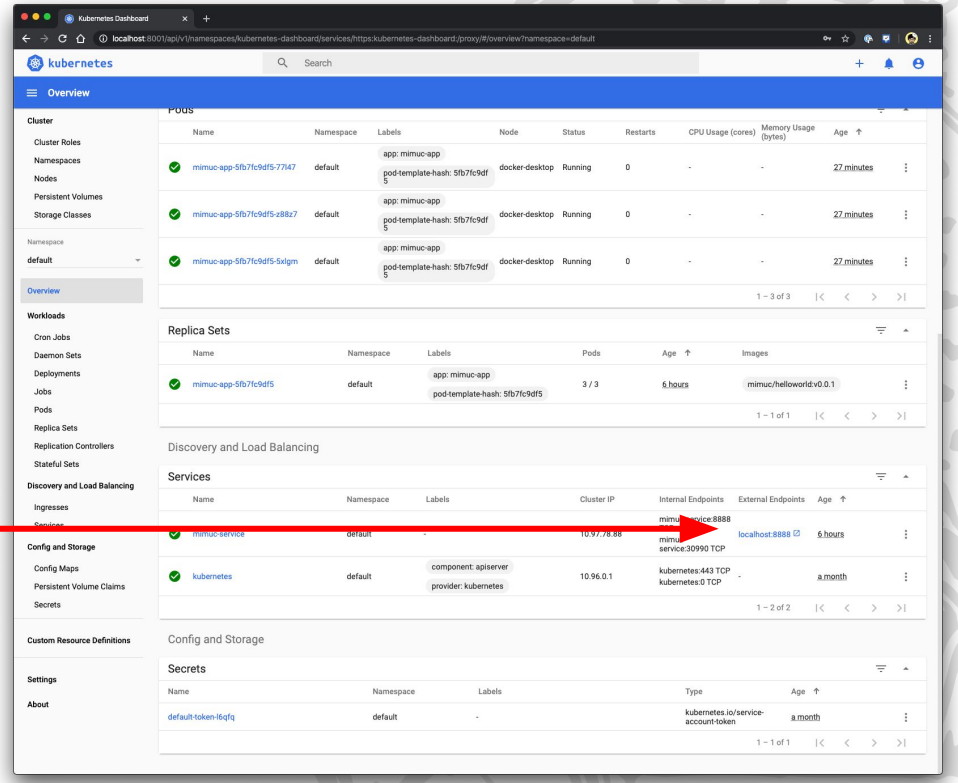
04-k8s-loadbalancing/kubernetes/deployments.yml

K8s Service: Load Balancer

- Perform 100 request and the server is roughly requested in three replicas:

```
$ sh request.sh > out.txt
$ cat out.txt | sort | uniq -c
  37 {"hello":"world","name":"D7P73SSxPS"}
  31 {"hello":"world","name":"MdQIiVyhKU"}
  32 {"hello":"world","name":"ODgqddk9Ok"}
```

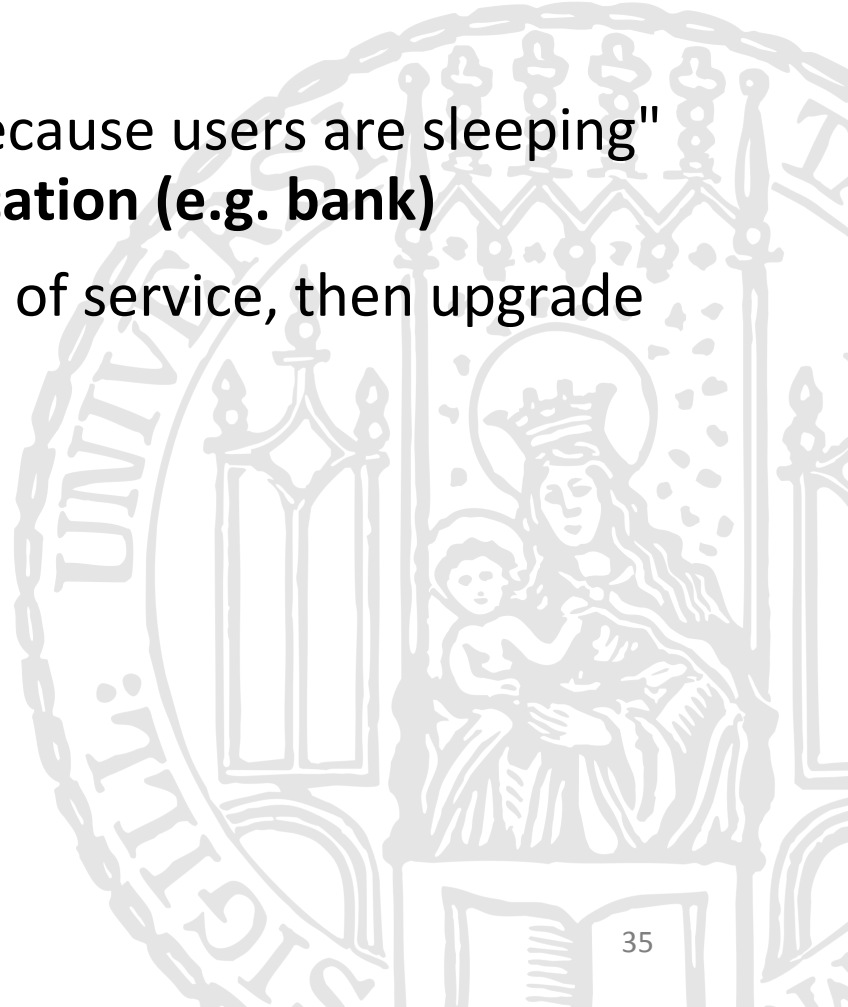
localhost:8888



Infrastructure: Deployment Strategy

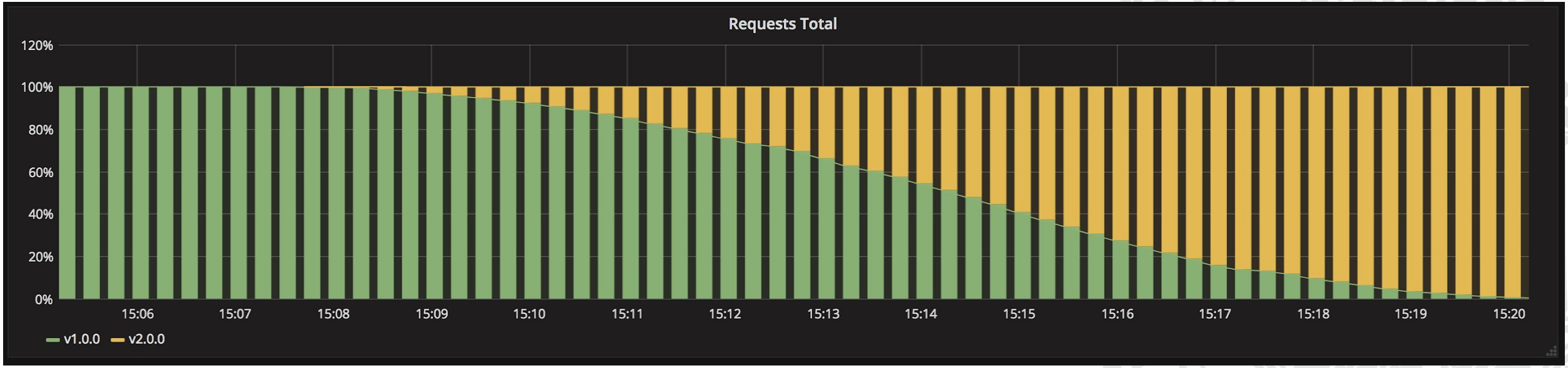
- Fundamental service is critical to the application quality, just like electricity and water.
- Naive strategy: "Deploy the application at mid-night because users are sleeping"
  ⇒ False for a globalized application and critical application (e.g. bank)
- In production, it is rare to shutdown an application out of service, then upgrade to a newer version

  ⇒ Requires "zero down-time"
- Two common and easy to use strategies
  - **Ramped strategy (aka Rolling Update)**
  - Blue/green strategy



Ramped Strategy in K8s

- Version B is slowly rolled out and replacing version A.



https://github.com/ContainerSolutions/k8s-deployment-strategies/tree/master/ramped

Zero Downtime Deployment in K8s

- Create a continuous access pattern
  `$ sh request.sh`
- Change image version from
  `mimuc/helloworld:v0.0.1`
  to
  `mimuc/helloworld:v0.0.2`
- Define strategy in the config file

```
 # type: use rolling update strategy
 # maxSurge: define how many additional
pods can be started
 # maxUnavailable: define how many pods can
be stopped from the current number of
replicas
 strategy:
   type: RollingUpdate
   rollingUpdate:
     maxSurge: 0
     maxUnavailable: 1
```

05-k8s-zerodown/kubernetes/deployments.yml

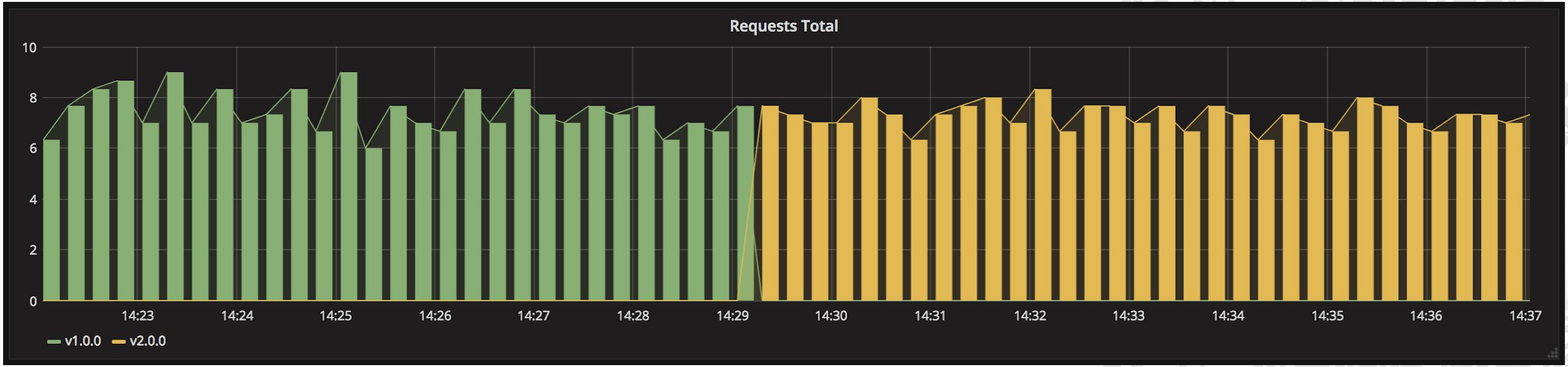
Result

- No request failure
  - Zero down-time
- `m76vY3mmFF` is upgraded firstly
- Then `aXltxWTnRQ` is upgraded

```
$  sh request.sh
{"hello":"world","name":"pe2ZSnvemO"}
...
{"hello":"world","name":"ktegmG7iLJ"}
{"hello":"world","name":"m76vY3mmFF","counter":1}
{"hello":"world","name":"pe2ZSnvemO"}
{"hello":"world","name":"m76vY3mmFF","counter":2}
{"hello":"world","name":"pe2ZSnvemO"}
{"hello":"world","name":"m76vY3mmFF","counter":3}
{"hello":"world","name":"pe2ZSnvemO"}
{"hello":"world","name":"m76vY3mmFF","counter":4}
{"hello":"world","name":"pe2ZSnvemO"}
{"hello":"world","name":"pe2ZSnvemO"}
{"hello":"world","name":"m76vY3mmFF","counter":5}
{"hello":"world","name":"pe2ZSnvemO"}
{"hello":"world","name":"m76vY3mmFF","counter":6}
{"hello":"world","name":"aXltxWTnRQ","counter":1}
{"hello":"world","name":"m76vY3mmFF","counter":7}
{"hello":"world","name":"aXltxWTnRQ","counter":2}
...
```

Blue/Green Strategy Update in K8s

- Version B is released alongside version A, then the traffic is switched to version B.



https://github.com/ContainerSolutions/k8s-deployment-strategies/tree/master/blue-green

Breakout #3

Discuss:

1. What front-end issue could be caused by rolling upgrade?
2. What are the pros and cons of **blue/green** deployment strategy?

Timeframe: 5 Minutes

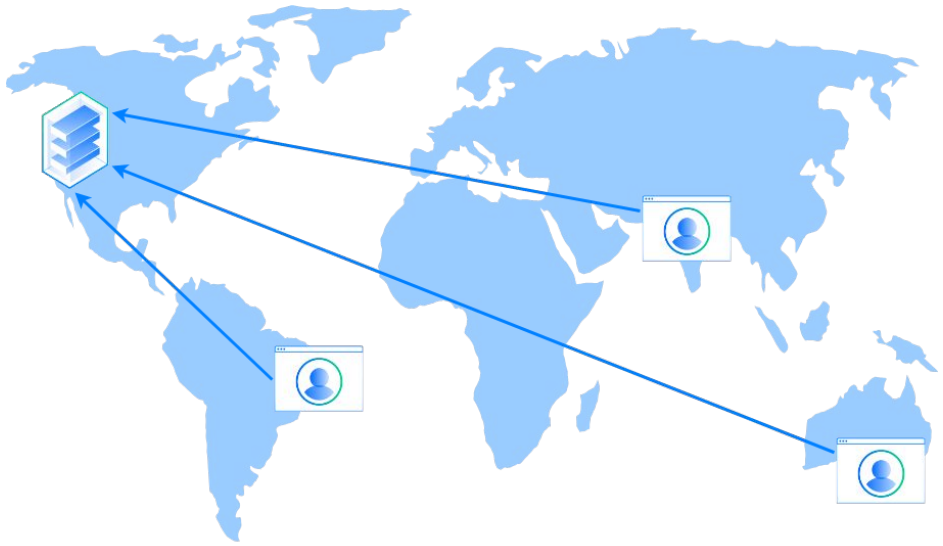
Breakout #3

Discuss:

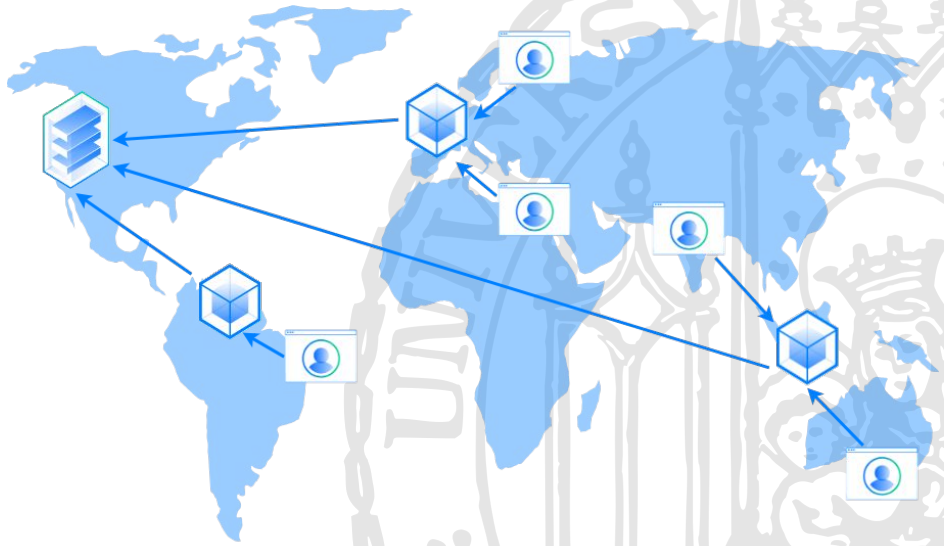
1. What front-end issue could be caused by rolling upgrade?
    - Front-end users may get different versioned assets while updating
2. What are the pros and cons of **blue/green** deployment strategy?
    - Pros:
        - a good fit for front-end that load versioned assets from the same server
    - Cons:
        - can be expensive because it requires doubled deployments

Infrastructure: CDN

- CDN distributes static contents multi-regional to accelerating access speed.



Without CDN

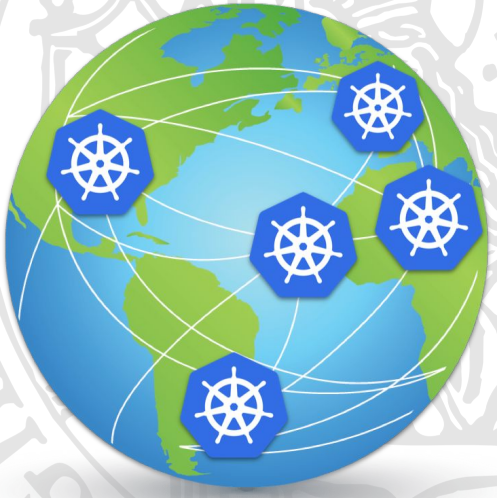


Content Delivery Network (CDN)

https://www.digitalocean.com/community/tutorials/using-a-cdn-to-speed-up-static-content-delivery

Infrastructure: Building CDN with K8s

- K8s is able to managing worker nodes in multi-regions
  - https://kubernetes.io/docs/setup/best-practices/multiple-zones/
- K8s is able to scale pods directly in multiple geo-regions
- Basic Idea:
  - Trigger deployment globally for front-end pods works like CDNs



Thanks!
What are your questions?

Uncovered Topics in Infrastructures

- Auto scaling: Horizontal scaling and Vertical scaling
- Networking: DNS
- Container Storage: Volumes
- ...
- Read more: https://kubernetes.io