

# Web Infrastructure

RES, Lecture 5

---

Olivier Liechti

heig-vd

Haute Ecole d'Ingénierie et de Gestion  
du Canton de Vaud

# Systemic Qualities

# The Notion of Systemic Quality

---

- How do we specify the **requirements** of a system?
  - We always build systems to deliver some sort of "**service**" (web, messaging, access control, customer management, etc.).
  - We first have to specify "**what**" the system should do. In other words, we have to specify **functional requirements**.
  - We also have to specify "**how**" the system should behave, i.e. what qualities it should exhibit. These are the **non-functional requirements**.
- Non-functional requirements characterize **systemic qualities, or "-ilities"**:
  - There are lots of different systemic qualities. Depending on the system, some are more important than others.
  - Choices made when defining the system architecture have a large impact on the systemic qualities.
  - Your life as an architect will be to deal with **trade-offs** in addressing conflicting systemic qualities.

**System**  
Vehicle

**Functional requirements**  
Move people around

**Non-functional requirements**  
Performance  
Capacity  
Reliability  
Cost  
Aesthetics  
Ease of use





Different systemic qualities often create **opposing forces**.

Defining the “right” architecture for a system means finding the right **balance** between these forces.





# Some Systemic Qualities...

---

- **Response time**
  - Measures the time required to present a result to the user
  - Important for the end-user
- **Throughput**
  - Measures the number of requests that can be processed in a given time frame
  - Important for the service provider
- **Scalability**
  - Measures how easy/costly it is to adapt the system in order to handle additional load
  - Ideally: linear scalability. "2 x more users => 2 x more servers"
- **Availability**
  - Measures the percentage of time during which the system can be used
  - 99% availability = average unavailability of 3.65 jours per year, i.e 1 hour and 41 minutes per week.
- **Security**
- **Manageability**
  - Measures the ease of operating the system: how easy is it to monitor it, to detect issues, to upgrade, etc.

# Manageability

---

heig-vd

Haute Ecole d'Ingénierie et de Gestion  
du Canton de Vaud

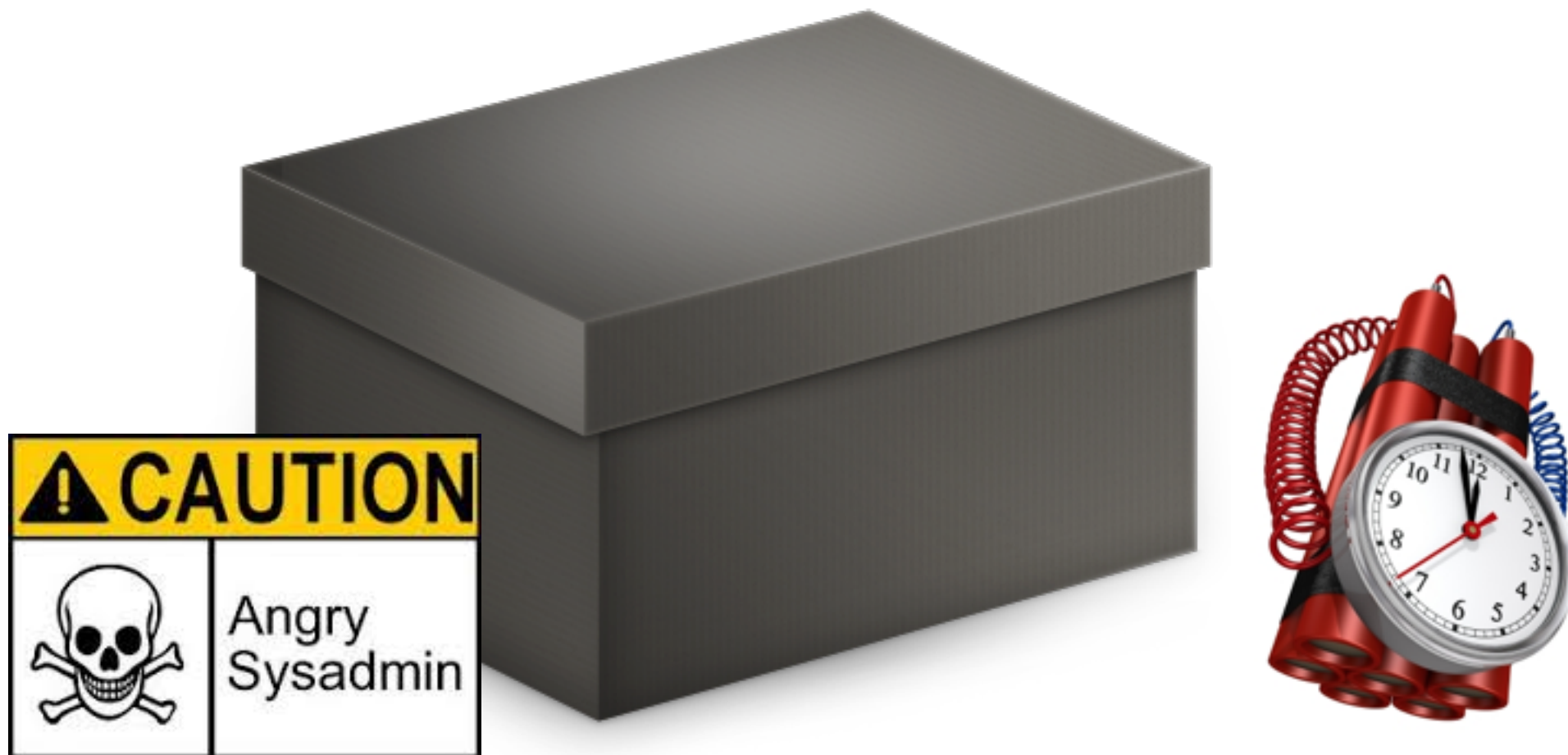


# Manageability

---

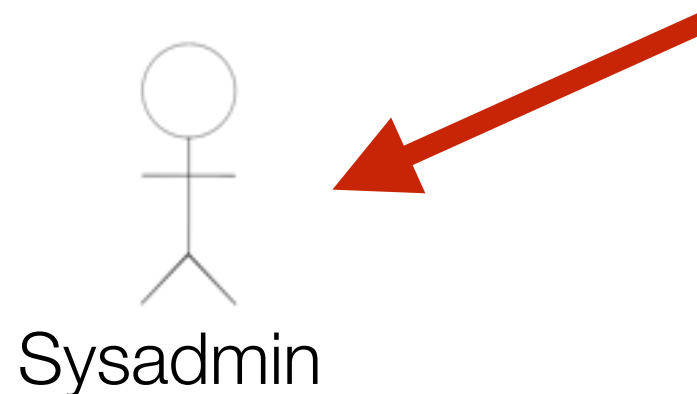
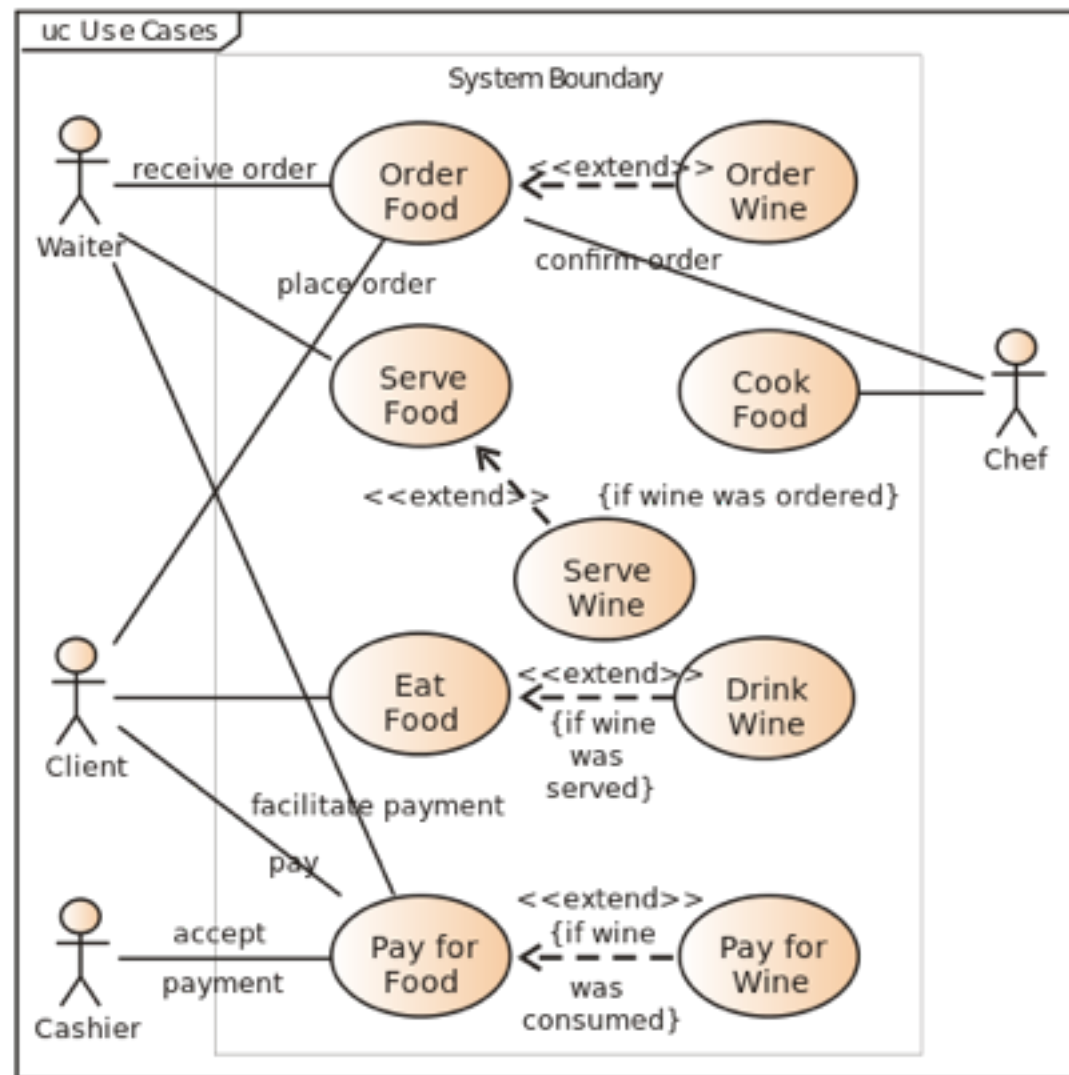
heig-vd

Haute Ecole d'Ingénierie et de Gestion  
du Canton de Vaud





# Manageability



# Manageability

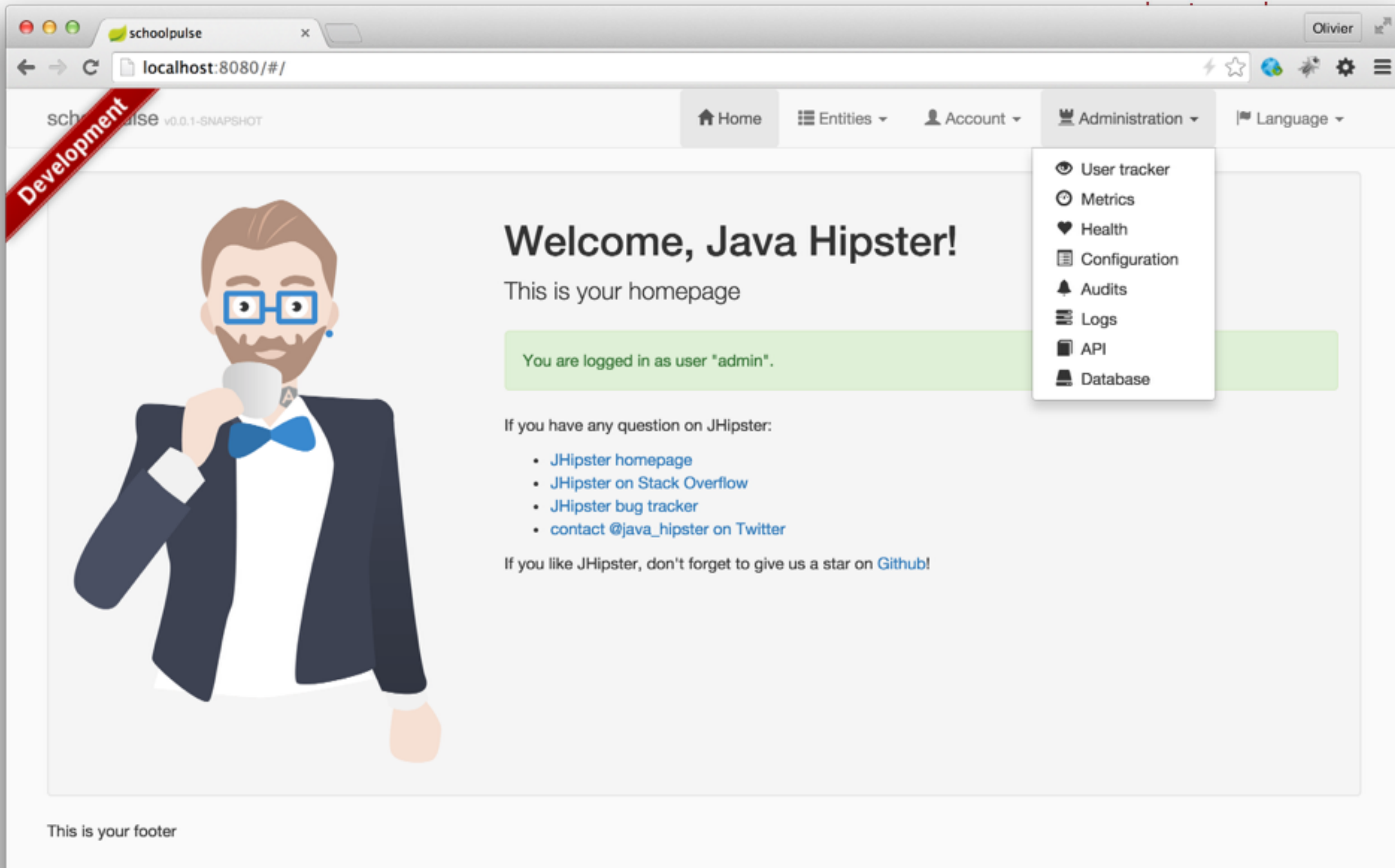
---

heig-vd

Haute Ecole d'Ingénierie et de Gestion  
du Canton de Vaud

*Greetings, Java Hipster!*







# Manageability

heig-vd

Haute Ecole d'Ingénierie et de Gestion  
du Canton de Vaud

## Application Metrics

Refresh

### JVM Metrics

#### Memory

Total Memory (259M / 3,818M)

7%

Heap Memory (141M / 3,818M)

4%

Non-Heap Memory (118M / 121M)

98%

#### Threads (Total: 48)

Runnable 8

17%

Timed waiting (21)

44%

Waiting (19)

40%

Blocked (0)

0%

#### Garbage collections

Mark Sweep count	5
Mark Sweep time	644ms
Scavenge count	33
Scavenge time	146ms

## Health checks

Refresh

Service name	Status	Details
Database	UP	
Email	DOWN	
Disk space	UP	

## HTTP requests (events per second)

Active requests: 1 - Total requests: 146

Code	Count	Mean	Average (1 min)	Average (5 min)	Average (15 min)
Ok	145	0.15	0.03	0.07	0.08
Not found		0.00	0.00	0.00	0.00
Server Error		0.00	0.00	0.00	0.00

HTTP is a Stateless  
Request-Reply Transfer Protocol

# Stateless Protocol... but Stateful Applications!

---





# Managing State on Top of HTTP

---

- **Approach 1: moving the state back-and-forth**
  - One way to do it is to use **hidden fields** in HTML forms
- **Approach 2: maintaining state on the backend, transfer session IDs**
  - One way to do it is to use parameters in the **query string** (security...)
  - One way is to use **cookies**



# Passing State Back and Forth

C: Hello, I am new here. My name is Bob.

S: Welcome, let's have a chat [You told me that "My name is Bob"].

C: [My name is Bob]. What's the time?

S: Hi again Bob. It's 10:45 AM. [You told me that "My name is Bob". You asked me what is the time]

The mockup displays a horizontal process flow with four steps: 1 Account Data, 2 Profile, 3 Confirm, and 4 Finish. Step 2 is the active step. Below the flow is a table with three columns: #, Name, and Description. The table contains two rows of data. At the bottom, there are 'previous' and 'next' navigation buttons.

#	Name	Description
1	Mark	Otto
2	Jacob	Thornton

# Passing Session ID Back and Forth

C: Hello, I am new here. My name is Bob.

S: Welcome Bob, let's have a chat. Your session id is 42.

C: My session id is 42. What's the time?

S: -- checking my notes... hum... ok, I found what I remember about session 42...

S: Hi again Bob. It's 10:45 AM.

C: My session id is 42. How do you do?

S: -- checking my notes... hum... ok, I found what I remember about session 42...

S: I am fine, Bob, thank you.

C: My session id is 42. How do you do?

S: -- checking my notes... hum... ok, I found what I remember about session 42...

S: I told you I am fine... are you stupid or what?

C: My session id is 42. If you take it like that, I am gone.  
Forever.

S: -- checking my notes... hum... ok, I found what I remember about session 42...

S: -- putting 42 file into trash...

S: Bye Bob.



# Passing Session ID in URL

---

GET /login HTTP/1.1  
Host: intra.heig-vd.ch

HTTP/1.1 302 Found  
Location: http://intra.heig-vd.ch/home?sessionId=83939

GET /home?sessionId=83939 HTTP/1.1  
Host: intra.heig-vd.ch

GET /page2?sessionId=83939 HTTP/1.1  
Host: intra.heig-vd.ch

GET /logout?sessionId=83939 HTTP/1.1  
Host: intra.heig-vd.ch

# Passing Session ID in Cookies

---

- The client **sends a request to the server for the first time**. It cannot send any cookie (it does not have any!).
- The server **understands that it is a new client**. It creates a session and generates a unique session ID.
- In the HTTP response, the server sends the unique session ID in a **Set-Cookie header** (“Here is a token... next time you come to see me, show me the token and I will recognize you!”).
- The **client keeps track of the cookie** either in memory or on disk (it has a cookie store, i.e. a list of cookies sent by servers).
- Before the client sends a second request to the server, it lookups the cookie store: “I am about to visit a server... did I receive any cookie from it?”. If yes, it sends the cookie in a **Cookie header**.

# Passing Session ID in Cookies

---

- The server sends a cookie (token):

**Set-Cookie:** GAPSSessionID=4rm?????na; path=/; domain=gaps.heig-vd.ch

- The client sends it each time it returns to the server:

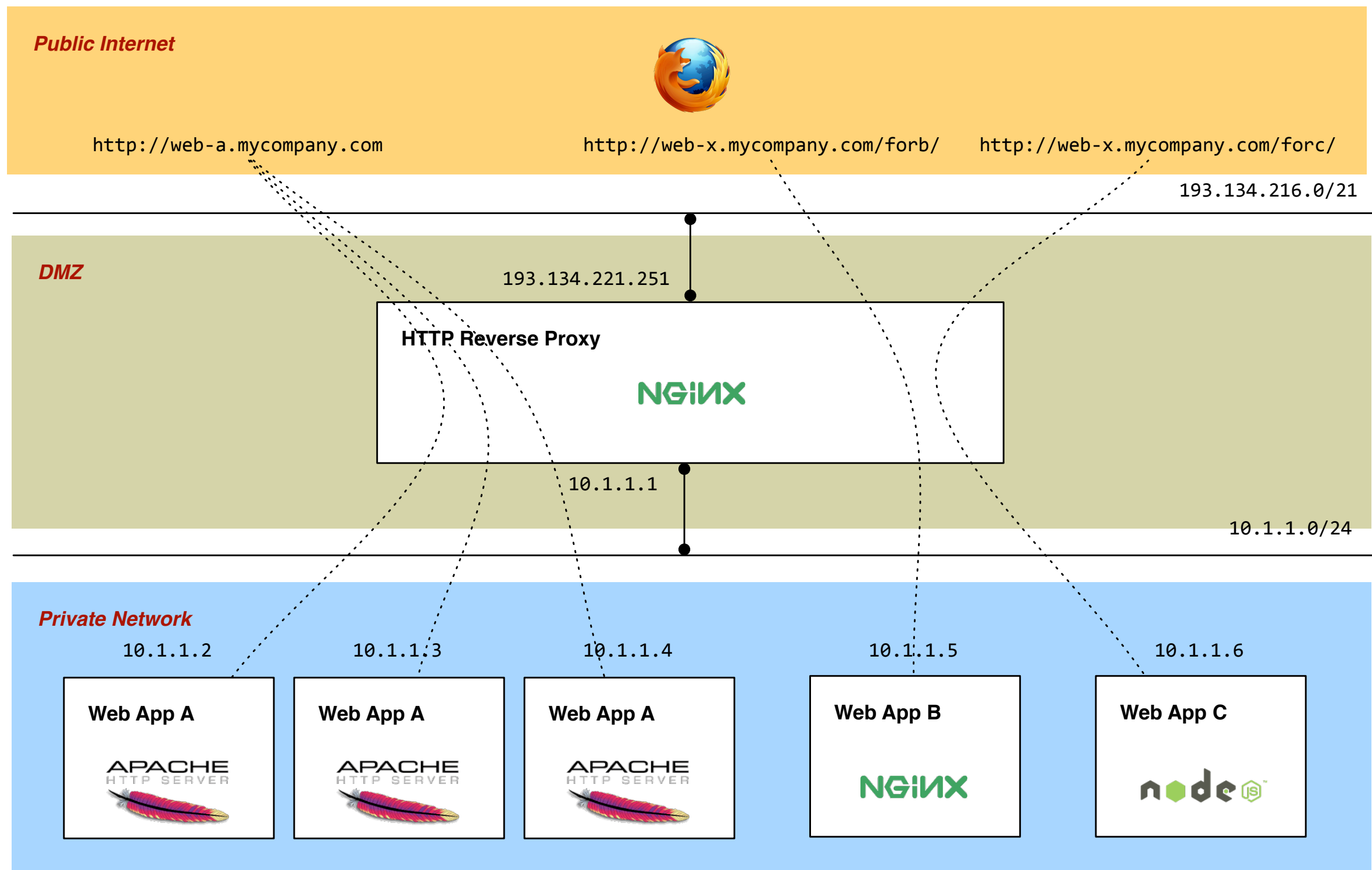
**Cookie:** GAPSSessionID=4rm?????na



# Where does the server store the state?

- What is transferred in the cookie is **only a session ID**, not session data (and BTW not user credentials!).
- The **data associated to the session** typically includes the user identity (and roles), the previous actions performed by the user, the preferences, etc.
- **This data can be stored in various places.** Where you store it can have a big impact on performance, scalability and fault-tolerance. There is no universal “best solution” - the choice depends on many factors.
- Choice 1: **store everything in the DB.** One advantage is that if a web server crashes, it does not bring down the state. One disadvantage is that performance may
- Choice 2: **keep in the RAM of the web server.** One advantage is the performance. One disadvantage is that you have to use sticky sessions. Another disadvantage is that RAM consumption could be high.
- Choice 3: **use a distributed caching layer** (e.g. redis, memcached). One advantage is fault tolerance and scalability. One disadvantage is higher complexity and cost.

# HTTP Infrastructure



# The Role of the Reverse Proxy

- An HTTP proxy that is “close” to the server
- Forwarding requests to the “appropriate” server
- Balancing requests between several “equivalent” servers (load balancing)
- **Sticky sessions!**

```
ProxyRequests Off
```

```
<Proxy *>  
Order deny,allow  
Allow from all  
</Proxy>
```

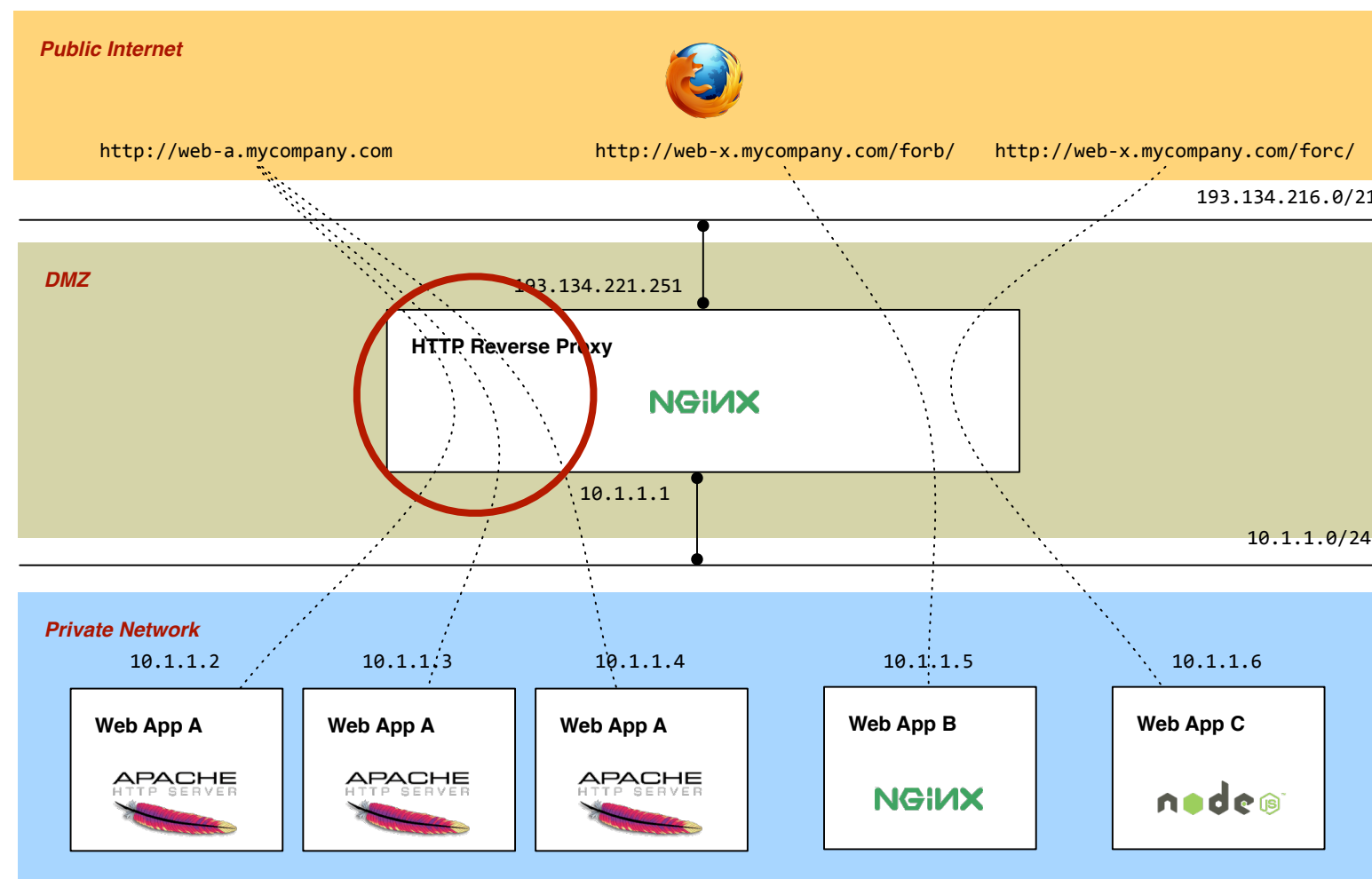
```
ProxyPass /foo http://192.168.1.2:8080/bar  
ProxyPassReverse /foo http://192.168.1.2:8080/bar
```

```
Header add Set-Cookie "ROUTEID=.%{BALANCER_WORKER_ROUTE}e;  
path=/" env=BALANCER_ROUTE_CHANGED  
<Proxy balancer://mycluster>  
BalancerMember http://192.168.1.50:80 route=1  
BalancerMember http://192.168.1.51:80 route=2  
ProxySet stickysession=ROUTEID  
</Proxy>  
ProxyPass /test balancer://mycluster
```



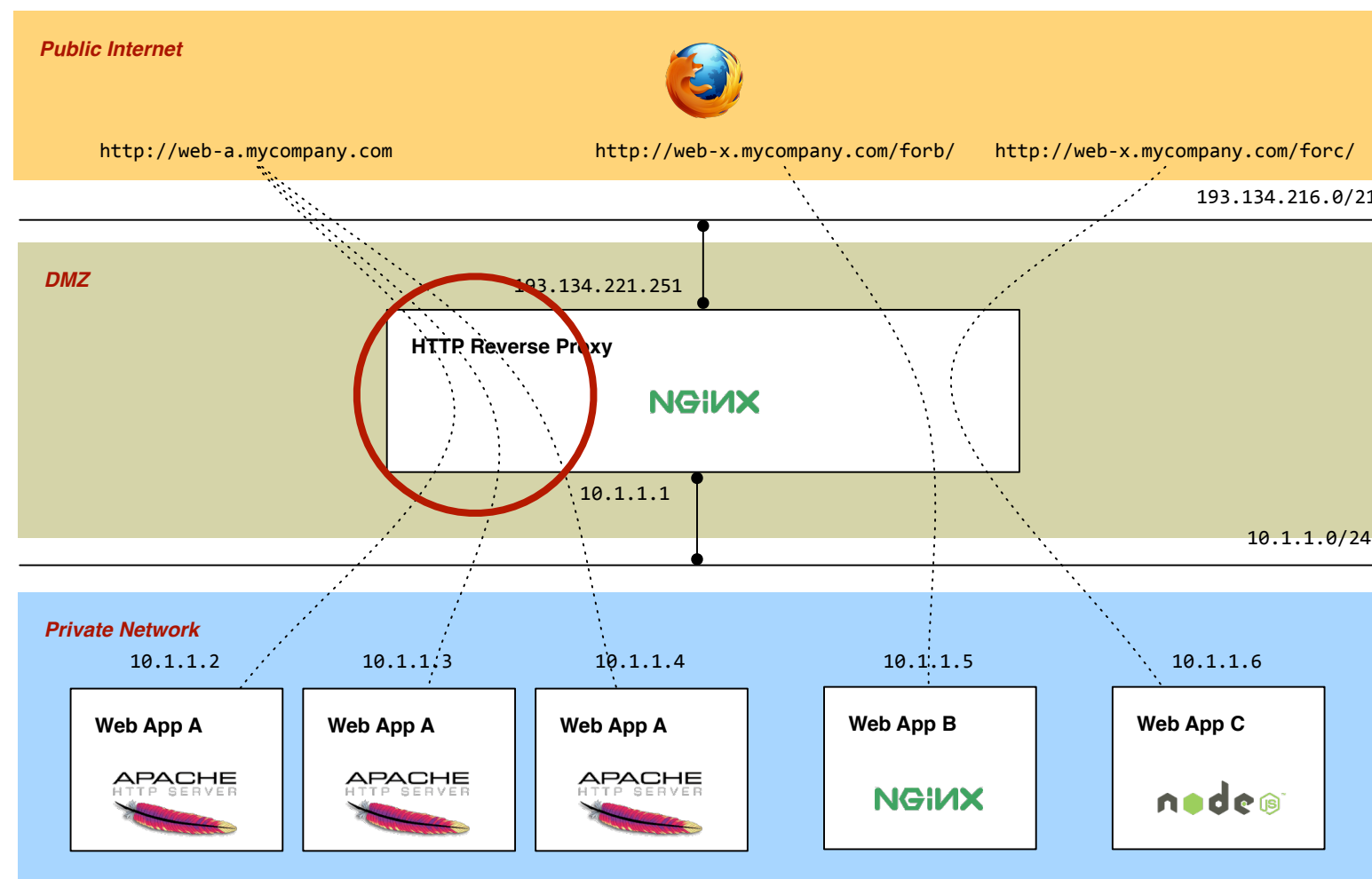
# Availability

- What happens if a server (or server component) fails?



# Scalability

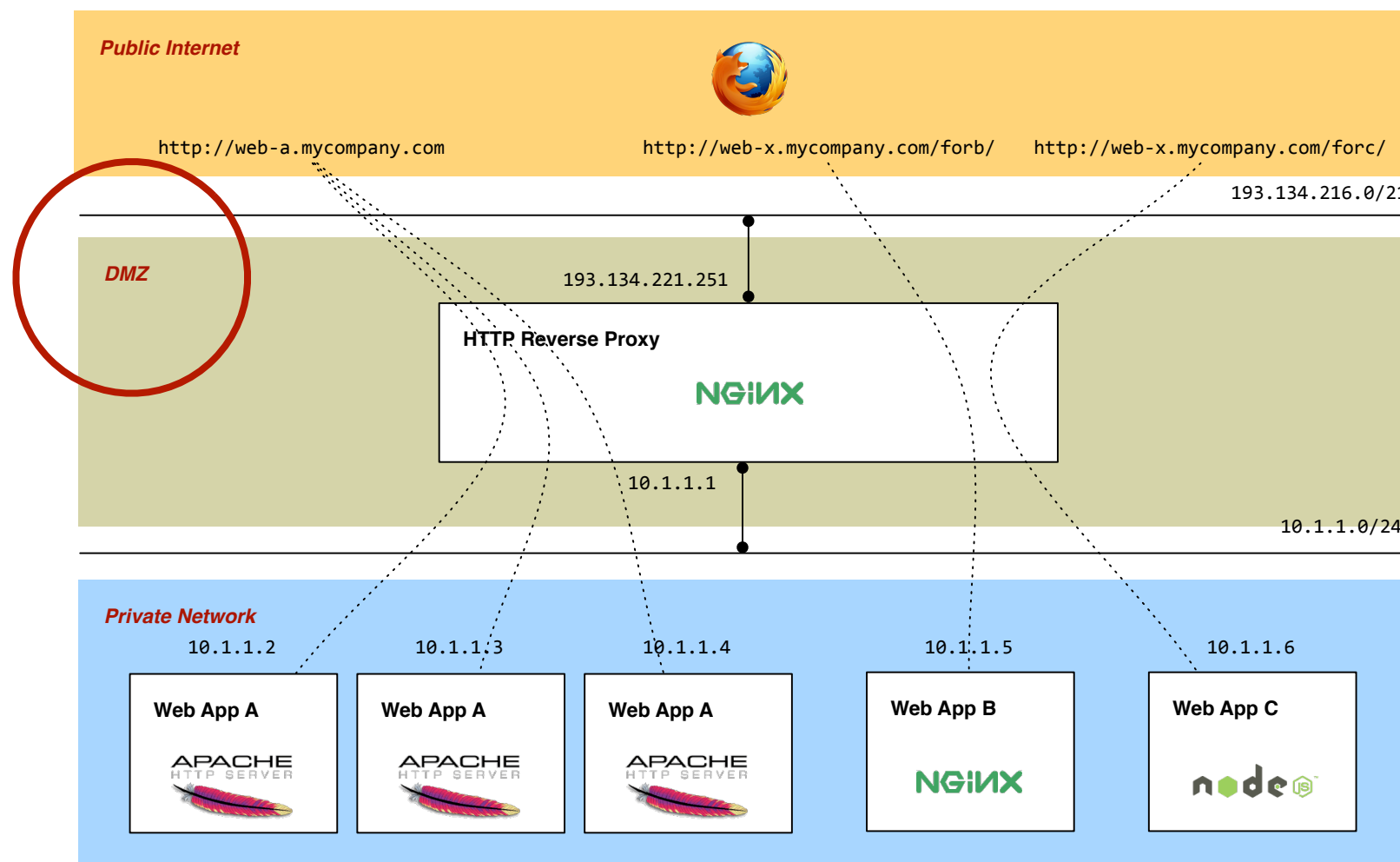
- Ability to evolve in order to sustain a bigger load, in quick and economical manner.
- Horizontal vs Vertical Scalability
- Elasticity



# Security

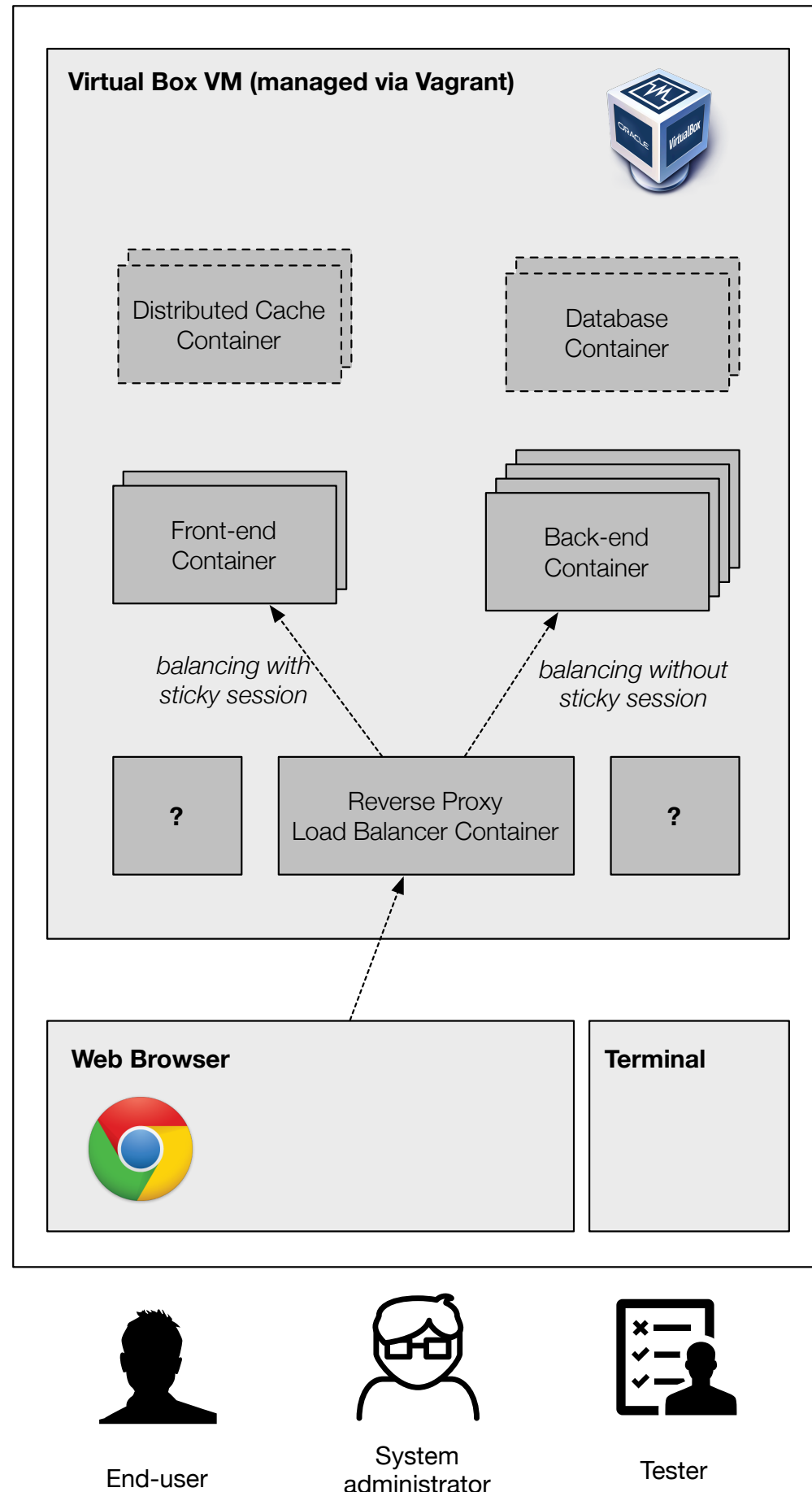
heig-vd

Haute Ecole d'Ingénierie et de Gestion  
du Canton de Vaud



# Web Infrastructure Lab





## Requirements

As a user, I can access a Web UI. The static content is served by one of the front-end containers. A Javascript script running in the browser makes AJAX calls to fetch dynamic data served by one of the back-end containers.

As a sysadmin, I can use a web browser to see the list of running Docker containers and see from which image they were created.

As a sysadmin, I can use a web browser to start and stop front-end and back-end containers.

As a tester, I can prove that HTTP requests are load-balanced between the back-end containers and that all containers receive requests to process.

As a tester, I can prove that load balancing is configured in a way that sticky sessions are enabled on the front-end containers.

As a tester, I can prove that load balancing is configured in a way that sticky sessions are disabled on the back-end containers.

As a tester, I can prove that when a new container (front-end or back-end) is started, the load balancer is reconfigured and starts sending some of the requests to it.

As a tester, I can prove that when a container is stopped (front-end or back-end), the load balancer is reconfigured and stops sending requests to it.

As a tester, I can prove that the dynamic reconfiguration of the load balancer uses a custom protocol based on UDP. I can provide the documentation of this protocol.

## Constraints

The front-end containers **MUST** use the apache httpd server and PHP.

The back-end containers **MUST** serve JSON resource representations under the `/api/` end-point.

The distributed cache and database containers are not mandatory. They **MAY** use technologies such as redis, memcached, mysql or mongodb.

The resources managed by the back-end containers must have a dynamic state, so that successive requests produce different representations.

1. How do we **decompose** in the project into (relatively) independent tasks?

2. **Who** takes care of the different tasks?

What are the main issues foreseen? What are the skills required? How do we estimate the relative effort?

3. How to do we **synchronize/coordinate** our work?

# Hint #1: Docker, services & processes

---

- Docker is often used to build systems based on the **micro-services architecture**.
- **A system is made of several services** (think of an online store with a user management service, a catalog service, a checkout service, a payment service).
- With Docker, the common approach is to **run one single logical service in every container**. However, a service can involve the execution of **several processes** (e.g. 2 server daemons, a watchdog, etc.).
- Typically, in the lab, you have a service where you need to **1) implement the web back-end server (Node.js) and 2) implement the UDP-based discovery protocol**. You may want to implement these two features in different programs (with possibly different languages).

# Hint #1: Docker, services & processes

- Even if you have several processes running in a container, Docker has the notion of “main process”.
- When you start a container, the “main process” starts. As soon as it finishes its execution, the container is stops.
- The main process can be specified either as a command line argument, when using docker run.

```
# Execute /bin/bash in a new container, created from the image  
$ docker run -i -t heigvd/res-demo /bin/bash
```



- It can also be defined in the Dockerfile with the CMD and/or the ENTRYPOINT instructions (and in that case, there is no need to provide an command line argument).



# Hint #1: Docker, services & processes

- Because the container stops as soon as the “main process” returns, you need to be careful with **daemon** programs.
- A daemon is usually a server that runs in the background, in its own process.

```
# I explicitly start the script in the background with the & character at the end  
$ ./start-my-server.sh &
```

```
# In this case, the script forks a new process  
$ ./start-my-server
```

- Be careful: if the main Docker process is a script that forks a new process and immediately returns, then **your Docker container will immediately stop!**
- This is the case for many network servers (e.g. apache). You need to know that there is very often a special flag (e.g. “**no-daemon**”) that you can specify when you start the network server. This will not fork a new process and the server will run in the foreground.

# Hint #1: Docker, services & processes

---

- We have spoken about two issues: 1) how do we run several processes in a Docker container and 2) how do we make sure that the main process runs in the foreground.
- In addition, the lab makes use of several Node.js components (at the least the Express.js app for the web back-end, and possibly the implementation of the UDP protocol).
- For these reasons, we suggest the use of the PM2 process manager.
- A process manager is responsible for managing a set of... processes:
  - Watching and restarting processes in case of crash.
  - Collecting logs in a central place
  - Generate init scripts executed at boot time
  - Etc.

# Hint #1: Docker, services & processes

---

- With PM2, you can start one or more scripts.
- You can execute a script in the background or in the foreground.
- You can execute Node.js scripts, but also other executables (with -x)

```
# pm2 start app.js
```

```
# pm2 start app.js --no-daemon
```

```
# pm2 start -x script.sh --no-daemon
```