# Docker & Mesos/Marathon in production at OVH

Balthazar Rouberol

https://ovh.to/6bRrkAn

# About Docker at OVH

- 2014-2015: Home-made container orchestrator, Sailabove, based on LXC
- 2016: Switch to Docker & Mesos/Marathon
- 6 (soon 7) Mesos clusters:
  - Internal production: 2 (soon 3)
  - External production: 2
  - External gamma: 2
- At our peak:
  - 800 hosts
  - 3000 cores
  - 12TB RAM
  - 200TB disk
- 60 teams, ~2500 production containers

# Problems we faced

- Docker instabilities and crashes
- Traceability of all network accesses established by containers
- Security rules enforcing
- No baked-in multi-tenancy in Marathon
- Incoming connections dropped due to marathon-lb/HAProxy reload stuck
- Partial network outages impacting production due to LB misconfiguration
- And **many** more, but I only have 30 minutes :)

# What UnionFS to choose? The land of BUTs.

- **devicemapper** in loop file (default): works fine on dev machine, BUT catastrophic performances in production
- **AUFS**: abandoned
- **overlay**: faster than devicemapper BUT high inode consumption
- **overlay2**: lower inode consumption BUT kernel > 4.0
- **ZoL**: Few production feedback that I know of. Good reputation BUT hard to install on Linux. Will test.

We currently run **overlay2**, on kernel 4.3.0 without noticeable issues, except regular image cleanup (which has an impact on docker).

# Traceability of network accesses

- Each packet is marked by the kernel with a class id
- A class id defines a cluster / team / app
- Iptables rules with classid filters can be written where appropriate (u32)
- **Prototype**: Log all incoming/outgoing SYN packets with https://github.com/google/gopacket

# Security rules enforcing

Home made mesos-docker-executor:

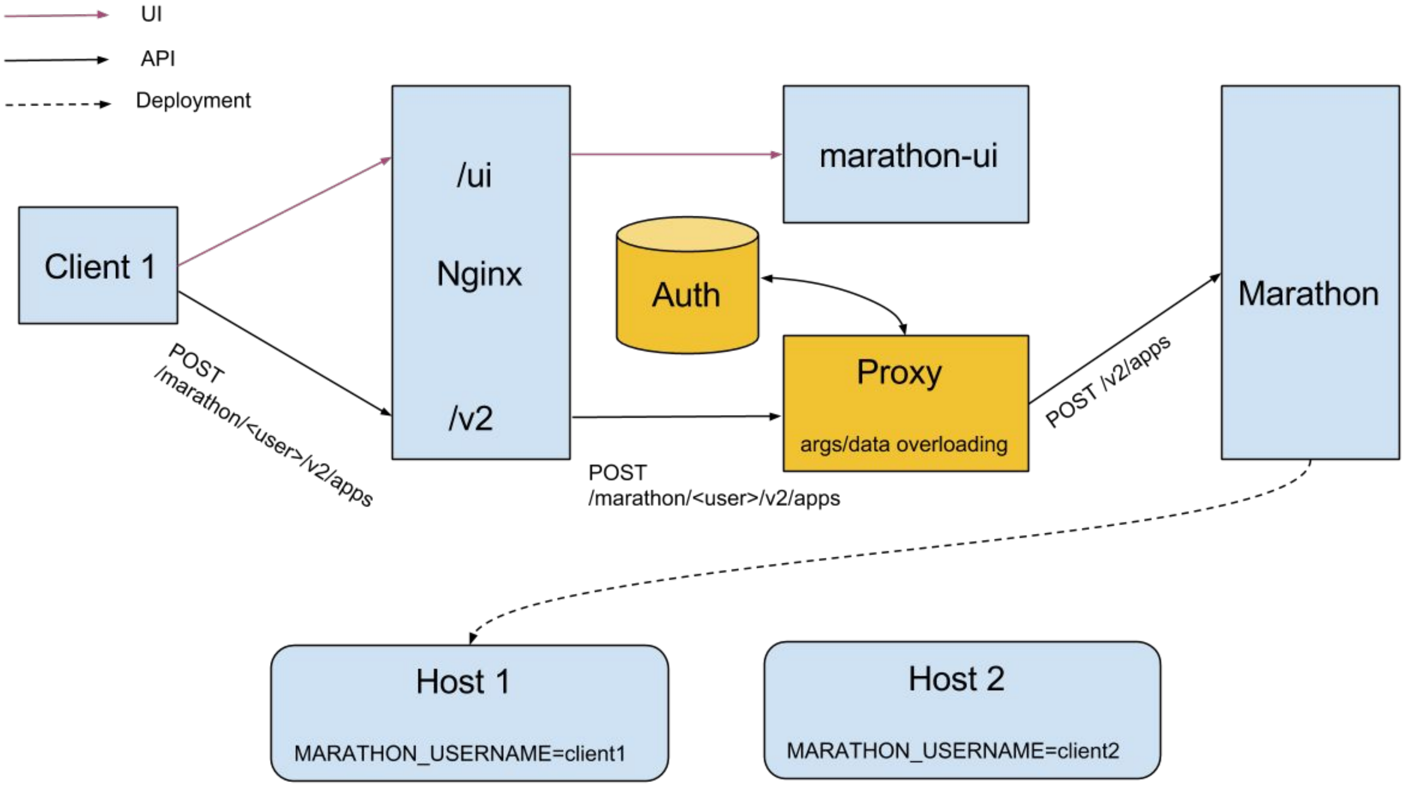
- No privileged mode
- Limited default CAPs
- Class ID injection

Of course, no SSH access on hosts running the containers

# Marathon & Multi-tenancy

- No built-in support for multitenancy in marathon
- Possible Scala plugin integration, but poorly documenter
- 1 marathon / team (or client) → extreme load on Mesos

# Multi-tenancy by API Proxy


UI
API
Deployment
Client 1
POST /marathon/<user>/v2/apps
/ui
Nginx
/v2
marathon-ui
Auth
Proxy
args/data overloading
POST /marathon/<user>/v2/apps
POST /v2/apps
Marathon
Host 1
MARATHON_USERNAME=client1
Host 2
MARATHON_USERNAME=client2

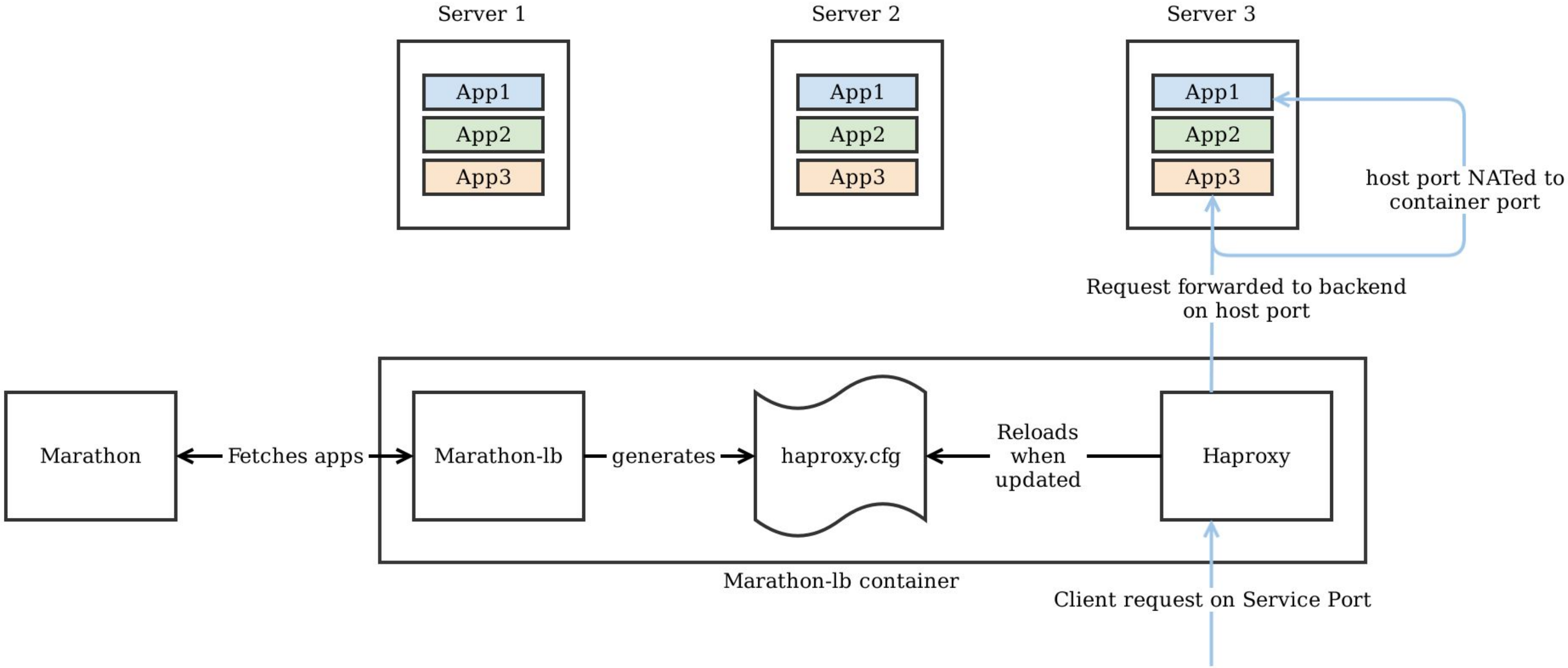
# Multi-tenancy by API Proxy, in a nutshell

- Override ~ all Marathon API calls to perform a virtual isolation
- `VERB /marathon/<user>/v2/<path>` + Basic Auth
- `POST /marathon/<user>/v2/apps`
  - `/<app_id> → /<user>/<app_id>`
  - `Add label MARATHON_USERNAME=<user>`
- `GET /marathon/<user>/v2/apps`
  - `Add Label selector MARATHON_USERNAME==<user>`
  - `/<user>/<app_id> → /<app_id>`
  - `Hide MARATHON_USERNAME label`
- `GET /marathon/<user>/v2/apps/<app_id>`
  - `/<user>/<app_id> → /<app_id>`
  - `Hide MARATHON_USERNAME label`
- `...`

# Multi-tenancy by API Proxy, limitations

- All apps are deployed, scaled, checked, etc, by a **single** Marathon cluster
- Global & progressive performance degradation
- Horizontal scaling to the rescue!
    - Deploy multiple Marathon clusters
    - Limit the number of different teams/users per cluster
    - We’ve yet to measure our limit

# Load Balancer reload: marathon-LB



# Load Balancer reload: marathon-LB’s approach

1. Block SYN for all bound ports (80, 443, 9000, service ports), one by one
2. Reload
3. Wait
4. Remove SYN drop rules

# Load Balancer reload: marathon-LB’s approach

Problems:

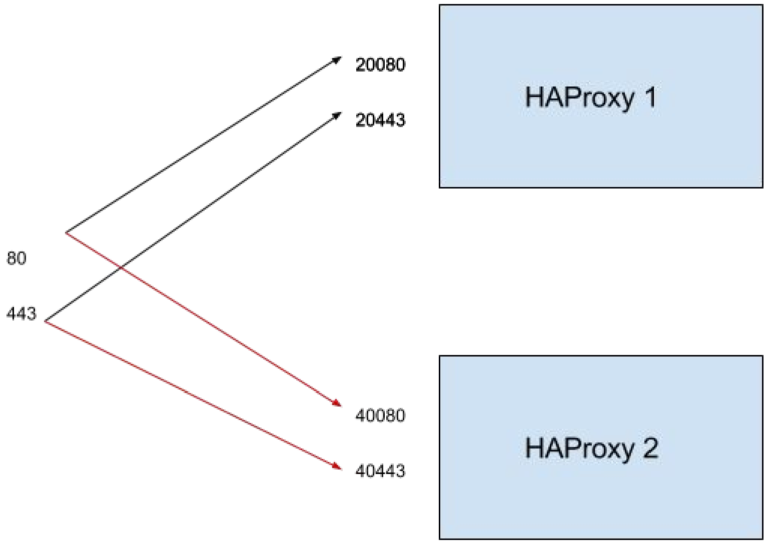
- Incoming connections are dropped for a while
- Reload is not atomic (2 iptables rules/port/reload)
- SYN DROP/ACCEPT is blocking, for each port → can lead to **catastrophic** situations

# Load Balancer reload: enters sprint-LB

Same architecture than marathon-LB but:

- Supports multiple orchestrators
- Supports multiple LB (nginx & UDP, wink wink)
- Atomic and non-locking reload
- Soon to be open-sourced

# Load Balancer reload: sprint-LB’s approach



- Start 2 HAProxy side by side
- **Transactional** NAT of each port (or range)
- Old HAProxy only handles previously open connexions (conntrack), then dies (SIGTTOU)
- New HAProxy handles new connections

Benefits:

- No connection drop
- No locking

# Load balancing configuration

Goals of a load balancer:

- Balance traffic between multiple healthy applications
- Perform health checks to detect unhealthy applications
- Remove unhealthy applications from the backend
- Bring back healthy applications into the backend

Your SLI depends on a good load balancer configuration!

# Guaranteeing a good SLI

- Quickly detect unhealthy applications: minimize errors
- Quickly detect healthy applications: spread load across applications

Health checks: regular checks performed on each application

- L4 (TCP): connection attempt
- L7 (HTTP/..): request and response analysis

# Guaranteeing a good SLI

HAProxy configuration values

- `redispatch=1:` try a new application at each retry
- `rise=1:` one OK is enough for an app to be seen as healthy
- `fall=1:` one KO is enough for an app to be seen as unhealthy
- `observe layer 4:` each L4 connection is considered as a health-check

# Thanks!
# Questions?