Microservices with OpenShift
Experience from the fields
Campfire & Battlefield stories

Sébastien Pasche
Security Architect
22.06.2016
Welcome
Who am I?

twitter @braoru
@Leshop since 2012

I ’am a computer security enthusiast
  Security contests, Malware, Web architecture

Open source contributor since 2001

Cinema, Music, Beer, Whisky, Food, Wine
Presentation plan

Not so technical

Who & what is LeShop

Why we dramatically needed micro-services

Battlefield & campfire story
  Jobs, Load Balancing, PAAS, Distributed architecture & more
“The corollary of constant change is ignorance. This is not often talked about: we computer experts barely know what we’re doing. We’re good at fussing and figuring out. We function well in a sea of unknowns. Our experience has only prepared us to deal with confusion. A programmer who denies this is probably lying, or else is densely unaware of himself.”

Ellen Ullman, Close to the Machine: Technophilia and Its Discontents
Before we start

This talk is not a set of absolute truth

There are plenty of amazing microservices related talks

I will try to focus on what we found useful in our case

Our choice could not be the best at all, but at least, it worked for us
Who is leshop.ch?
Private customers & B2B

Between 10% and 12% of everything is sold online

Our customer want the ease of use of amazon with the quality and flexibility of a local retail shop

Small country, and relatively wide area to cover (for our company size)
A lot of boxes

700,000 orders/year

45,000,000 items/year to deliver

Most of them are fresh (perishable) product

2,900,000 boxes/year to deliver

180 ton/day to deliver
Where it started
<table>
<thead>
<tr>
<th>Leshop</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Internal ERP, Internal website, Internal supply chain management,</td>
<td></td>
</tr>
<tr>
<td>Internal warehouse management and so on..</td>
<td></td>
</tr>
<tr>
<td>15 years old massive &amp; organically grown applications started with</td>
<td></td>
</tr>
<tr>
<td>java 1.0</td>
<td></td>
</tr>
<tr>
<td>More than 250 various recurrent jobs</td>
<td></td>
</tr>
<tr>
<td>Everything distributed on 5 Data center</td>
<td></td>
</tr>
<tr>
<td>308 collaborators</td>
<td></td>
</tr>
<tr>
<td>20 IT people (Dev, Design, Operation &amp; Infrastructure, Business</td>
<td></td>
</tr>
<tr>
<td>intelligence)</td>
<td></td>
</tr>
<tr>
<td>12 Devs</td>
<td></td>
</tr>
<tr>
<td>5 Operations people</td>
<td></td>
</tr>
</tbody>
</table>
The Great divide

Migros (main shareholder) ask us to follow group guidelines and centralize all our IT within 2 new Data-Center

Several technical requirements must be meet
- **Active-Active** (Connection between DC can be loss at anytime without any warning)
- Spread users across both DC (internal and external)
- No DC stickiness

And there is internal requirements
- Must be manageable by small team of people
- 2 years from start time to production full launch
Critical business requirements

Time to market
- Being able to release often (hourly) without any downtime

Compliance & security
- Being constantly up to date with software update (Systems, software and dependencies)
- Being easier to review and analyses

Reproducible & cost efficient
- Unit testable infrastructure and systems configurations
People
People

People tend to react badly to violent change

Always keep in mind the human advantages of technology changes (ex: less midnight call)
DevOps is not a dream, you just have to make it happen

Make people working in peer (cross team)
- Software packaging
- Bug solving is a good source of accomplishment feeling

Involve everyone on software requirements
People

1 change at a time, and don't forget mind adaptation time

Avoid the 30 minutes barrier

2 minutes is the maximum length it should take to explain a change
- If it's longer, split it
Does *Never* apply a change in production until you are sure *everyone* knows what you are doing.

Never forget the “midnight call effect”
How we did it
Reminder

Avoid overengineering

Avoid massive and non-flexible frameworks

Prefer open and well-maintained frameworks

Workaround is not a new dev methodology

If something doesn’t work change it

Sometime you need to create

Do not hesitate to learn
Reminder

Follow RFC
- Don’t forget drafts

Apply web best practices

Don’t ignore OWASP

Be critic about what do you read

Understand what you do

Google coding is not a new dev methodology
LeShop OpenShift dev guidelines

Every request is authenticated
- Client – Service
- Service – Service

Input validation & Output validation within each service
- JSON format, fields content, object creation and so on...

Input business validation & output business validation
- Can this item be in this cart? Can this item be added?
LeShop OpenShift dev guidelines

Applications must be stateless and scalable (known limits)

No Client-Service and Service-Service session

Independent and idempotent messages

“Current state” never expire

Work with functional programming approach as much as possible
LeShop OpenShift dev guidelines

No local logs

Split business logs from technical logs

HTTPS only

Application only speak JSON

Application only expose Data
LeShop OpenShift dev guidelines

Security context (user + roles) used at every steps

Input & Output request/s must be throttled and measurable

DB & Service access must be throttled and measurable
PAAS & LeShop 2014/2016

We run on Openshift 2

2 clusters of 25 nodes

Approximately 100 micro services in production

Docker Kubernetes dedicated clusters
- Non Heterogeneous services
- Databases (ElasticSearch, Riak, Redis, Mongo, InfluxDB, Graphite, LevelDB)
- Services that require high resources
One rule
Independent Immutable Idempotent
Design a container ready micro services
Elementary concept

1 service 1 business functionality aka “the unix way”

Services are responsible of their own data

Data processing lead to schema then schema lead to storage selection

If service(s) need to communicate with other service(s) then use messages

A database is not a communication channel
Run a self contained services
Self contained container
Advices

Do input & Output throttling
- We use Nginx as input throttling within each container
- We use actors-based programming for output throttling

Each micro-service is tested and a request/s value is setted
- Scaling and resources planning

I I I
- Each micro-service can run by itself everywhere (independent)
- Each container is completely immutable and reproducible (idempotent)
Let’s make them speak together
Client based approach

Each service is released with a corresponding client library
- Business related Dev should only use business object and never deal with HTTP layers

Don’t forget to use version within your HTTP endpoints to allows easier and smooth internal library upgrade
Client based approach

Service 1

HTTP

Module 1A  Module 1B  Client 1

Data 1A  Data 1B

Service 2

HTTP

Client 1  Module 2A  Module 2B

Import

Data 2A  Data 2B
Databases and MicroServices
Load Balancing
Load balancing

Avoid single point of failure

III

Place LB as close as possible from client

Internal & External client Facing : FrontEnd LB

Services & Tools client : Load balancer on each node (Independent)
- More complicated but no single point of failure
- All fqdn to 127.0.0.1 (Allow SSL passthroug)
Jobs
Jobs

The best way to manage jobs is to avoid recurrent Jobs
-But it’s almost never possible

Use reactive programming instead of batch programming

Minimal requirements
- Jobs must have a single “effective” running instance
- Detect and handle job failures
Jobs – distributed the heavy way

Distributed jobs engine
- Mesosphere
- Chronos + mesos

At least 5 different kinds of services to manage

Not very compliant with small teams and low maintenance times
Jobs – leShop way

Each job is owned by a micro-service

Allow each job to be idempotent and independent instances

Each job decided to stop/start himself
- This require each micro-service to use an internal scheduler
- Quartz is an internal scheduler, cron/systemd.timer are not :-(
Jobs – leShop way

Cleaning phase: detects jobs that are running for too long or are hung.

Reservation phase: place a reservation for the job

Execution phase: If the instance owns the reservation, it executes and handles the job.

Place easy to use milestone before critical operation within the execution phase

You need atomic operations on “quorum”
- You have to use a fully consistent write-oriented database (this is exactly the SQL pattern)
- But this imply active-passive setup (Break III)
Reservation phase
1. Read status of job
2. If `<status>` is START/RUN → ABORT
3. Update job status (Start, worker, now)

Execution phase
1. Read status of job
2. If `<status>` != START || not my worker id → ABORT
3. Update job status : (Run, worker, now)
4. Perform job:
   a. Regularly update job status
   b. For critical action, check if worker still owns the job
5. Finish job (Term, worker, now)
Monitoring
Monitoring

Each component runs multiple instances, each instance deals with jobs
- You may not know the current number of instances at a time $T$

Monitor business state instead of detailed views

Base your monitoring on business requirements
- A customer can create an account
- A retailer order can be send
- And so on ...
Monitoring

Require deep knowledge of business and inter-services communication
- Luckily, It’s exactly the job of the acceptance phase in continuous delivery
- Just reuse a subset of your acceptance tasks/tests
- Dedicated special HTTP route to run services self-business test

Create a service in charge of running end to end business case
- Use it for continuous integration/delivery

To follow III each test must be self contained and leave do track
Monitoring jobs & instances

We store micro-services status within a distributed databases

Same for jobs

Services instances periodically update their status (from business internal test)
Job periodically update their status (last completion time)

Check if number of alive instances are able to support business
Check if jobs have run in a business reliable time-span
Migrate and survive the great divide
Yesterday

- website (legacy)
- web service
- glass

Shopping module (legacy)

- ShoppingService
- ShoppingUserProxy
- UserService

INTERNAL

- Leshop-db

Today

- website (legacy)
- web service
- glass

Shopping module (legacy)

- ShoppingService
- ShoppingUserProxy
- UserService

INTERNAL

- Leshop-db

Shopping component

- AccountService
- UserService

Customer component

- REST

INTERNAL

- Leshop-db

Tomorrow

Requirements: all components must be productive

- website (new)
- web service (new)
- panel (new)

Shopping module (legacy)

- ShoppingService
- ShoppingUserProxy

INTERNAL

- Leshop-db

Shopping component

- AccountService
- UserService

Customer component

- REST

INTERNAL

- Leshop-db

New impl

- New DB
Where we are and where we go
PAAS & LeShop 2016/2017

We run on Openshift 2

Approximately 100 micro services in production

Docker Kubernetes dedicated clusters

Plan to move on Openshift 3 within 2017

We will not add Database within Openshift

We plan to move some services from dedicated dockers cluster to OpenShift 3
Wrap up
Don’t forget people

Never forget the human natural Fear of Changes
- Keep change smooth and make them natural and part of your software life-cycle
- Release often and water down perception of “Breaking change”

Never forget to try on the battlefield
- Use less important services to effectively test your changes

III :-)
Thank you
LEARN. NETWORK. EXPERIENCE OPEN SOURCE.
Authentication
Security context, authentication, AC and no session

JSON Web Token (JWT) “pronounced JOT”
-draft-ietf-oauth-json-web-token-32

Contain user role & authorizations (Called claims in literature)

Non-cookie based Token
Not automatically sent by browser
Easier to manipulate from non-browser apps
Short and Hard lifetime
Signed and enciphered
Allow message to be independent
Security context, authentication, AC and no session

Authenticate

Get a JWT

Authentication

Cart

Request + jwt

response

Can user do this action?
Is this action business correct?
Security Lib
From a token:
- check authentication
- check the roles
- get user infos

A token is valid for a given period of time -> No need to contact the authentication component