How to handle the complexity of migrate to Microservices from Monolithic 10 years of code

Reasons to being or not involved in migrating to Microservices

Alberto Salazar,
CTO Advance Latam

4th May 2017
Who am I?

Alberto Salazar

- Developer -> Architect -> Speaker
- Founder and CTO: AdvanceLatam & Cloudbanco
- Involved on a C level for the last 10 years
- Working 17 years with monolithic & continue writing code
- Founder and Leader EcuadorJUG
- JCP Associate Member & JSR’s early adopter


asalazar@advlatam.com  @betoSalazar  @EcuadorJUG
Agenda

- The use case - The evolution of 10 years of code
- The motivation to move from Monolith to Microservices
- The fact - 10 years of code
- The path - Tips, tricks, pros & cons of Microservices
- Summary
The evolution of 10 years of code

The use case

JEE 5 (2005)
JBOSS 4.2.2.GA
JSF, EJB3, EIP & others
(Facade, DAO, DTO
Services Delegate, etc).

SOA (2007)
Industry approach
ESB
BPM
BAM
(-) Not succeed
external influence

OSGi (2011)
Any App Server
SpringDM
Apache Camel
Apache ActiveMQ
Html 5 Css3 Mobile

OSGi Blueprint
(2013)
Apache Karaf
AMQ
(Split the Front-end
from the back-end)

Microservices
(2014)
-> Split the monolith
Elasticsearch
Logstash
Hazelcast

Microservices
(2016 - 2017)
+ Vert.x | Spring boot
+ Apache Cassandra
+ NodeJS, ReactJS & React Native
The business
You have to understand the sponsor

- First -> Multichannel
- Lately -> Omnichannel
- Now -> Digital
The motivation to migrate from Monolith to Microservices
Why microservices?

The buzz

Microservices is the architectural approach that everybody talks about and everybody wants it, but be careful ....
The goal that organizations need is to increasing velocity & agility;
Get into production as soon as possible;
Deploy new features as soon as possible.
The reasons of this talk

The plan of move forward (Microservices ?)

Every body are talking about the result (microservices architectural style)
but nobody are showing the path
The fact - 10 years of code
The Monolithic
The Monolith

Characteristics

- Attachment to language, platform & OS
- Single logical executable, deploy everything at once or nothing at all
- Failure of part == failure of whole
- Take months even years getting into production
- Centralized authority slows the delivery process (DBA, OPS, QA)
- Coordinated releases are hard, because brings many changes together from different teams
1 Year of code

Always the goal was to keep the modularization.

12 ejb-jars, 8 wars, 15 developers, JEE 5, build by 2 Teams (The framework & The Business)
5 Years of code

Modularization -> Osgi bundles

Around 50 osgi-modules, 3 wars, 1 ear, 40 developers, JEE + SpringDM, organized in around 5 Teams (The framework, Mobile, The Business, 3 Customization team’s)
More than 100 osgi-bundles + more than 50 jars modules, 1 ear + 5 wars, Modularize JS resources as Jars, >100 Developers, JEE + Spring + Apache Karaf, Camel, AMQ, 10 to 15 Teams (framework, Mobile, Business App, Customer’s team)
10 Years of code

Application Metrics

# Lines of Code
2 169 233
0 (Not:MyCode)

# Types
60 225
278 Projects
4 961 Packages
539 523 Methods
198 640 Fields
0 Source Files
Migrate 10 years of code, It will be Easy?

The challenge

- > 2MM lines of highly coupled code
- Build one microservice is, easy but what about a complete microservice architecture based on 2MM lines of code
- Time to delivery features are between: 6 to 12 months & QA overhead $$
- Several customers using the system on production environment
The path - Tips, tricks, pros, cons
Microservices
Microservices

Characteristics

- Independently deployable & executable
- Based on services
- High cohesion, low coupling
- Failure is isolated
- Model driven design
- Effective & efficient scaling
- Polyglot “Plus”

https://martinfowler.com/articles/microservices.html
https://martinfowler.com/bliki/MicroservicePrerequisites.html
The path
Microservices

- Split the frontend from the backend
- Split the backend & keep the centralized data
- Rest endpoints & the API Gateway
- Split the data
- Don’t forget the frontend & the agility
The path
Split the frontend from the backend (1/5)
The path

Split the backend (2/5)

JEE application server (JBoss or WL or WAS)

Oracle
The path
Split the backend (2/5)

Apache Karaf with a custom distribution
install just what you need

https://karaf.apache.org/manual/latest/#_custom_distributions
The path

Split the backend (2/5) - **Logging, trace & Monitoring**

1) Use Mapped Diagnostic Context (MDC)
2) Introduce a correlationId
3) Collect the logs
4) Search by rest API or use Kibana

http://camel.apache.org/mdc-logging.html
https://www.elastic.co/products/elasticsearch
https://www.elastic.co/products/logstash
https://www.elastic.co/products/kibana
The path
Rest endpoints & API Gateway (3/5)

POST bank.com/api/v1/accounts

JEE application server (JBoss or WL or WAS)
- ENGINE.war
- ENGINE-JS.jar
- wizard.war
- services.war
- scripts-builder.war

Osgi container
- engine-orchestrator.jar
- dynamic-camel-routes.jar
- batch.jar
- services.jar
- business-module1.jar
- business-module-n.jar

Service
- some api

Osgi container
- api-gateway.jar

Oracle

hazelcast
logstash
elastic
The path
Split de database (4/5)
The path
Don’t forget the frontend & the agility (5/5)
Summary & Code blueprints
Summary
Microservices Architecture Losses

- Transactions
- Single data repository
- Better architects needed
- Greater complexity - it bears repeating
- The system and services have to deal with network communications, failures, rebalances, splits.
- Monolithic apps are far easier to develop and debug (when viewing the platform as a whole)
Summary

Recommendations

- Design your application modular (either monolith, OSGi or microservices)
- Continuously refactor your modules or micro services to achieve optimal boundaries
- Define your remote and async APIs carefully, design remote calls for failure
- Monolithic apps only look simple from the outside, but you just open the box
- A lot of help using Event Driven Architecture (decoupled, scalable, Competing Consumers Patter)
- Care about logs, monitoring and always use a CORRELATIONID and MDC
Summary

Recommendations

- Each team will be able to explore and test new technology
- Automate the deployment and delivery process -> CI & CD
- Split the database
- Use lightweight frameworks or java containers (Karaf, Vert.x, Spring boot)
- Microservices are not everyone, be careful
- Design for failover, Service load balancing and automatic scaling, data Separation, Integrity, Performance
code blueprints

https://github.com/lasalazarr/fastdev
THANK YOU

plus.google.com/+RedHat
linkedin.com/company/red-hat
youtube.com/user/RedHatVideos
facebook.com/redhatinc
twitter.com/RedHatNews
LEARN. NETWORK. EXPERIENCE OPEN SOURCE.