

# How would ESBs look like If They Were Done Today?

Markus Eisele, @myfear  
Developer Advocate  
March, 2015

**“What’s right isn’t always  
popular. What’s popular isn’t  
always right”**

Howard Cosell

# Large Java EE / J2EE based applications

LoadBalancer

EAM

Application Server

<?>

EAR

WAR

JAR

JAR

EAR

JAR

JAR

JAR

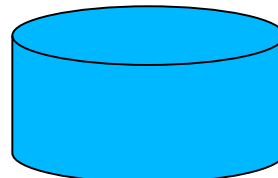
JAR

EAR

WAR

JAR

JAR



# Technical Implications

- Monolithic application – everything is package into a single .ear
- Reuse primarily by sharing .jars
- A “big” push to production once or twice a year
- Single database schema for the entire application
- $\geq 500k$  loc
- $\geq$  Heavyweight Infrastructure
- Thousands of Testcases
- Barely New Testcases

# Team and QA Implications

- $\geq 20$  Team Member
- The single .ear requiring a multi-month test cycle /
- Huge bug and feature databases
- User Acceptance Undefined
- Technical Design Approach
- Barely Business Components or Domains
- Requiring multiple team involvement & significant oversight

# And even **now** ...

- Still changing requirements.
- New features tend to be HUGE!
- Cross-cutting concerns nearly impossible to implement.

# Why?

**Technical Dept!**

**Inexperienced!**

**Grown application**

**We're lazy!**

**We always did it like that.**

**No education!**

**Outdated Infrastructure!**

**Outdated Designpattern?**



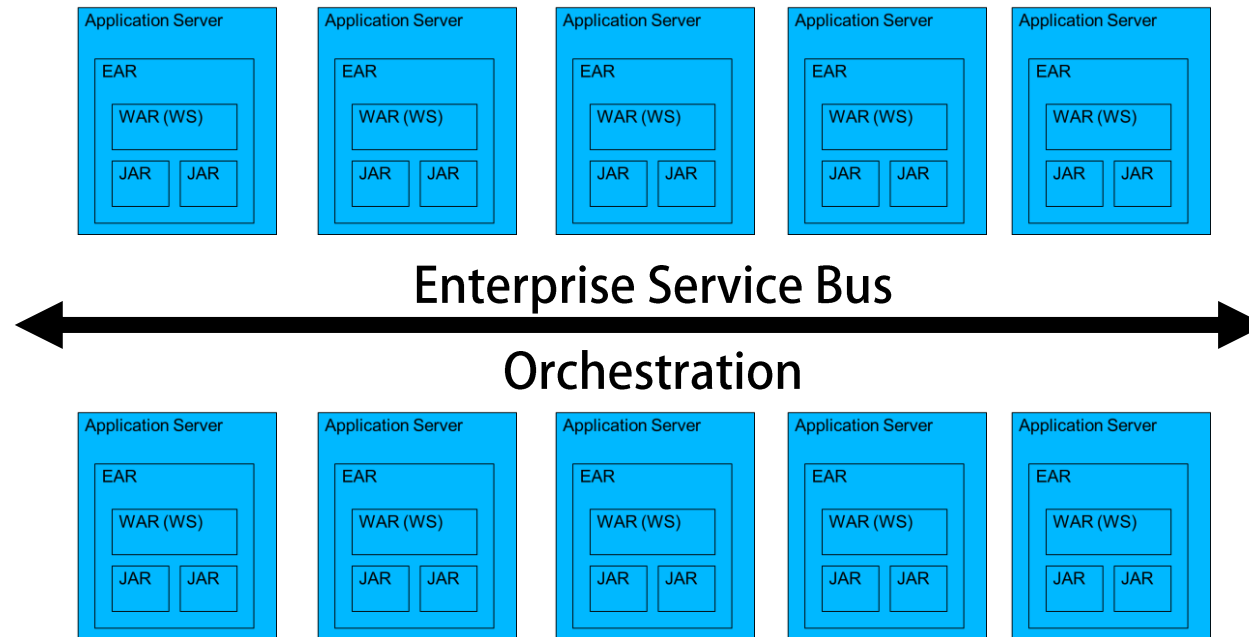
**Where did we go  
from here?**

**We treated  
everything as a  
legacy system and  
try to solve  
integration  
problems.**

STANDARDS!

ENTERPRISE!

INTEGRATION?



LICENSES!

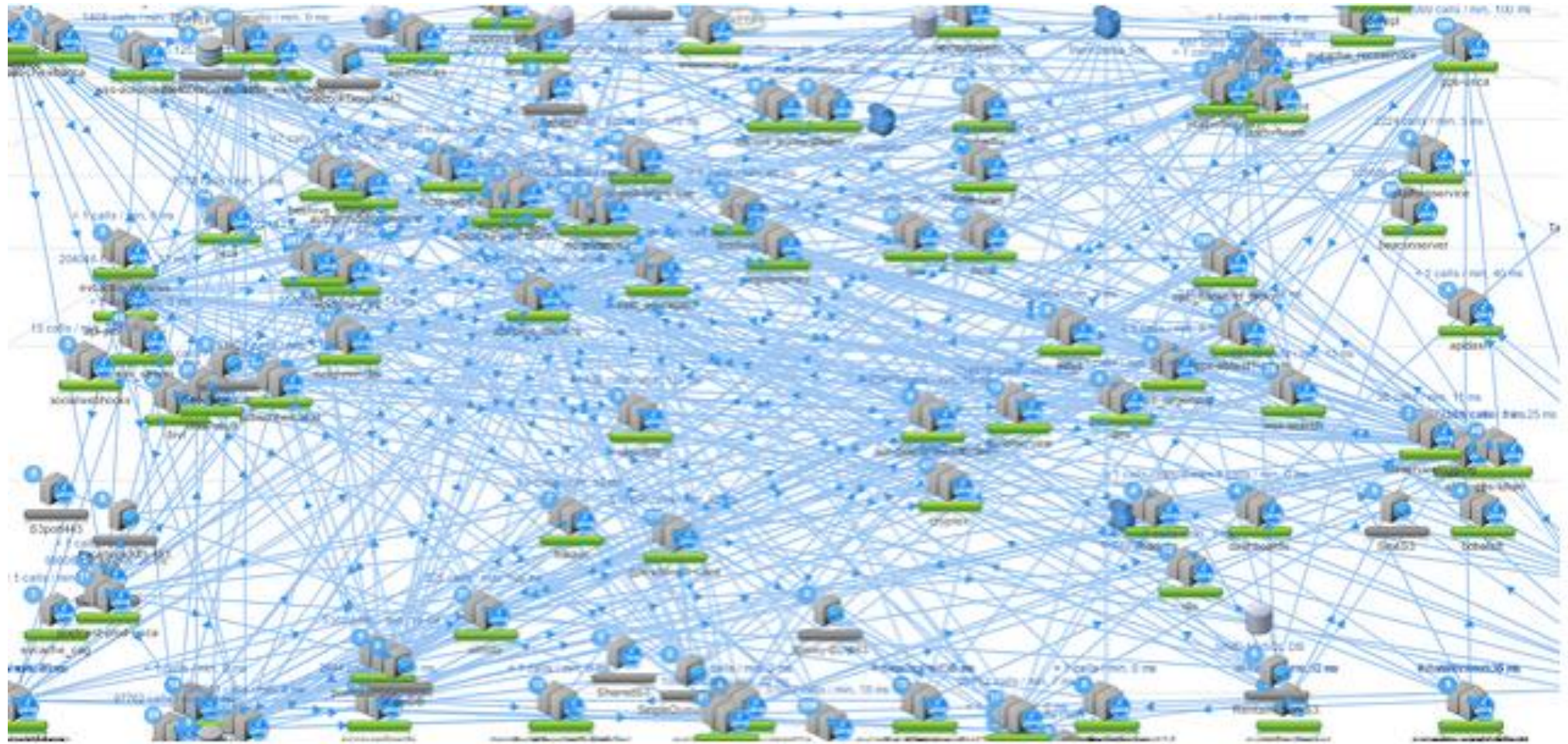
CENTRALIZE!



# What did ESBs do?

- Monitor and control routing of message exchange between services
- Resolve contention between communicating service components
- Control deployment and versioning of services
- Marshal use of redundant services
- Cater for commodity services like
  - event handling,
  - data transformation and mapping,
  - message and event queuing and sequencing,
  - security or exception handling,
  - protocol conversion and
  - enforcing proper quality of communication service

**Hmmm ... and  
where are we  
today?**

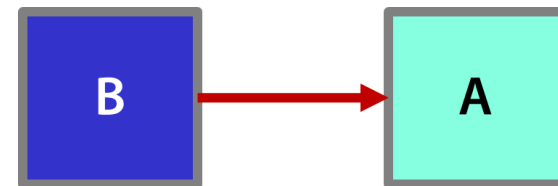
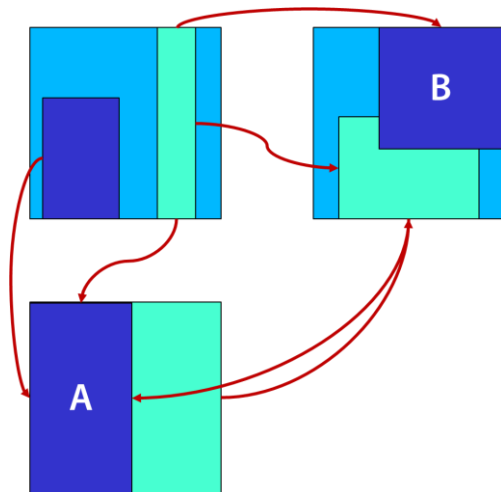


**Name it whatever  
you like.**

**We're decomposing  
monoliths  
and evolve them into  
microservices  
architectures.**



# Reduce Impact of Change by Encapsulating Source of Change



**Domain Driven Design**

**Bounded contexts**

**Designed For Automation**

**Designed for Failure**

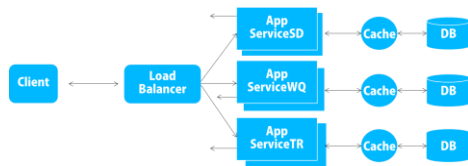
**Independently Deployable**

**Let's deconstruct the  
\$hit.**

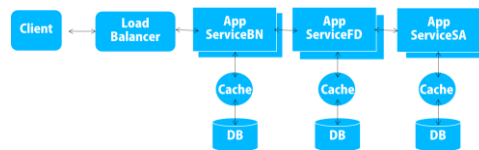
**“Monitor and control routing of message exchange between services”**

- Not really anymore.
- “Services do one thing well”
- Bunch of different approaches to service design and interaction.
- No centralized point of “control”

## Aggregator Pattern w or w/o Proxy

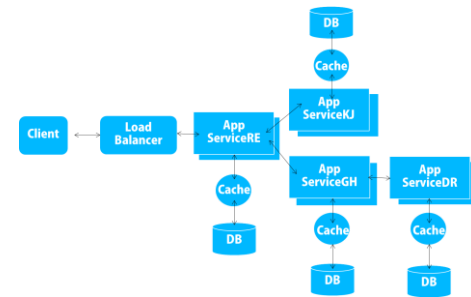


## Chained Pattern



## Branch Pattern

.....



**“Resolve contention between  
communicating service components”**

# **“Smart endpoints and dumb pipes”**

**- Martin Fowler**

<http://martinfowler.com/articles/microservices.html>



**“Control deployment and  
versioning of services”**

- **Deployment**
- **Configuration**
- **Profiles / App Packaging**
- **Service Discovery**
- **Versions**
- **Monitoring**
- **Governance**

**“Marshal use of redundant services”**

# **“Decentralized Governance”**

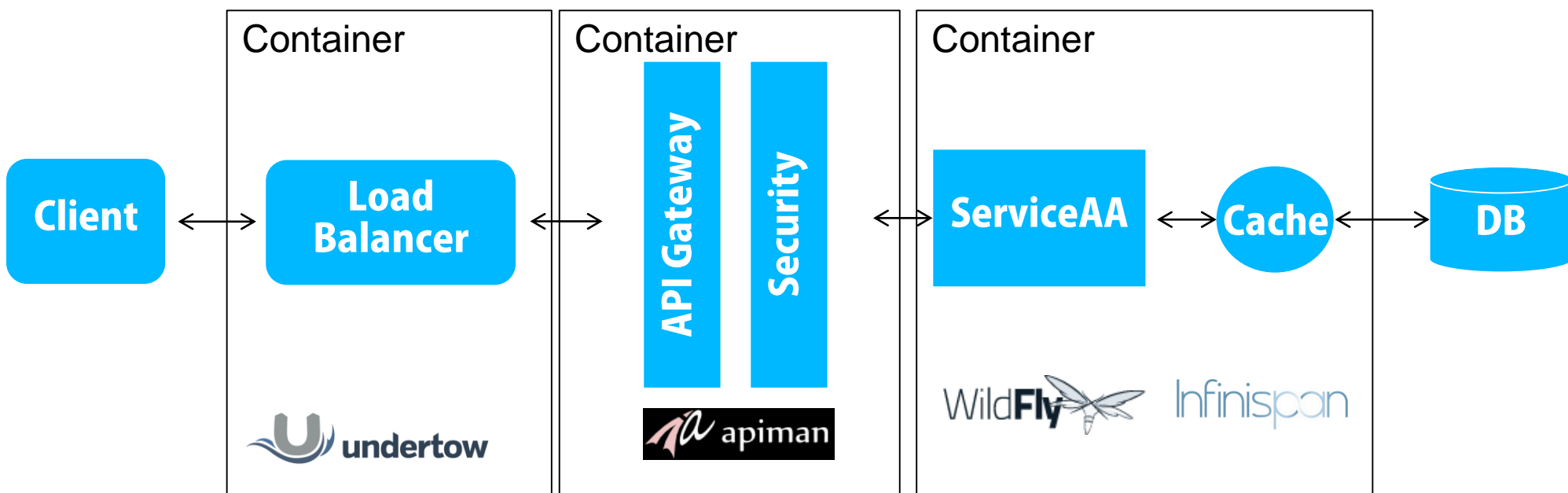
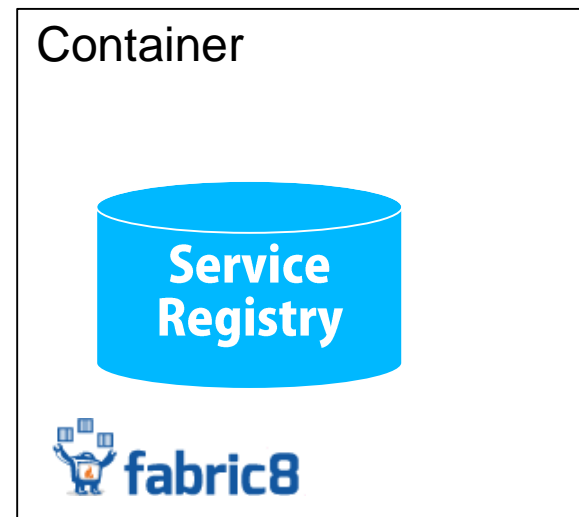
**- Martin Fowler**

<http://martinfowler.com/articles/microservices.html#DecentralizedGovernance>

**“Cater for commodity services”**

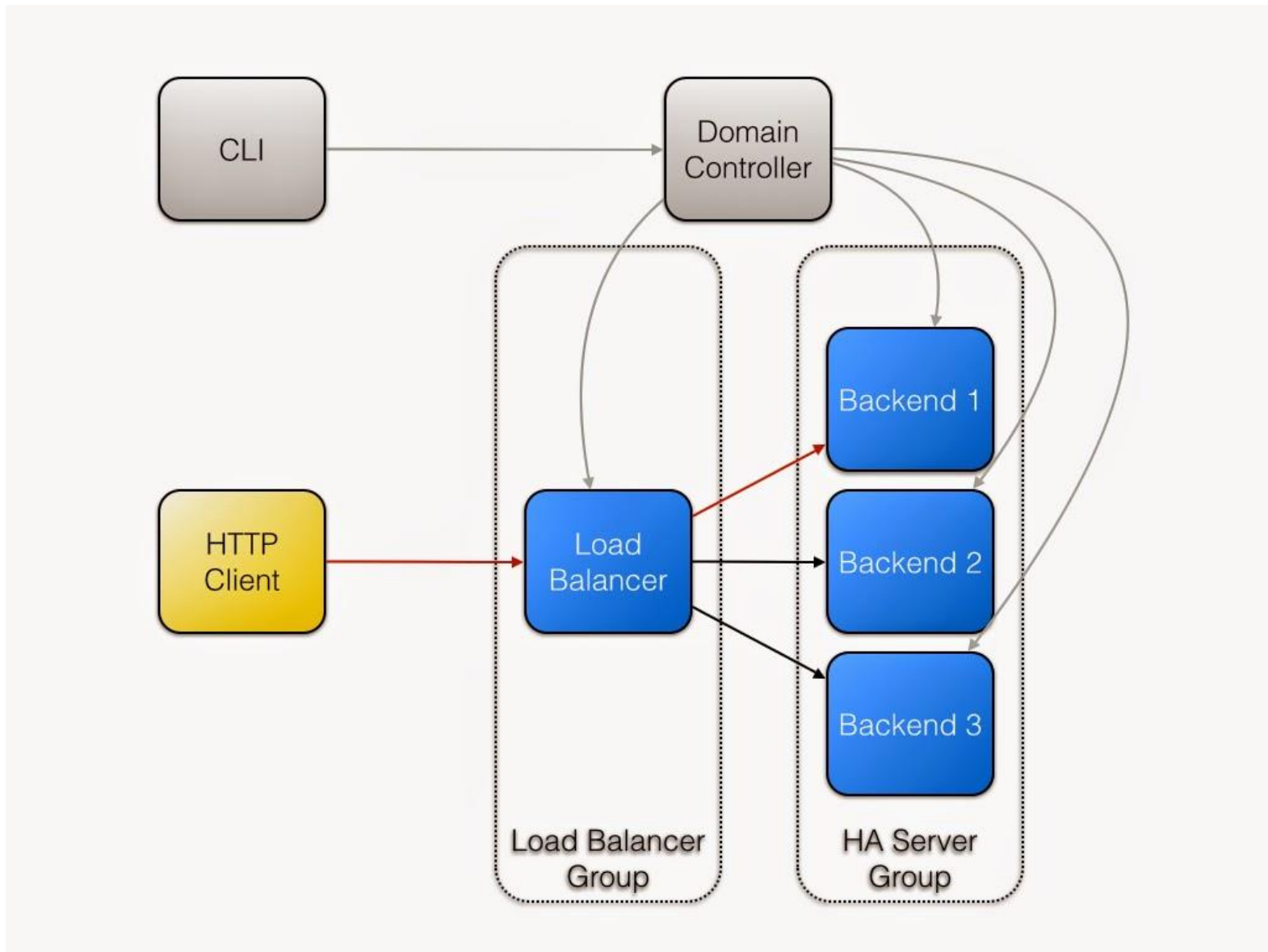
- **a lightweight service runtime**
- **Cross – Service Security**
- **Transaction Management**
- **Service Scaling**
- **Load Balancing**
- **Deployment**
- **Configuration**
- **Profiles / App Packaging**
- **Service Discovery**
- **Versions**
- **Monitoring**
- **Governance**
- **Failure Handling**
- **Asynchronous vs. Synchronous**
- **Cross – Service Logging**
- **...**

**An approach.**





# The Pieces



[Home](#) » [Public Library](#) » [BookListing](#)

## BookListing

Version: 1.0 ▾

[New Version](#)

Overview

Implementation

Plans

Policies

Activity

### Service

All Books of the library

⌚ Created on 2014-09-10

👤 Created by [admin](#)

### Version

Version: **1.0**

Status: **READY**

⌚ Created on 2014-09-10

👤 Created by [admin](#)

### Things To Do

- [Link my Application to this Service \(New Contract\)](#)
- [Create a new version of this Service \(New Version\)](#)

[Publish](#)

[Home](#) » [Public Library](#) » [BookListing](#)

## BookListing

Version: 1.0 ▾

[New Version](#)

Overview

Implementation

Plans

Policies

Activity

### Service Implementation

API Endpoint:

API Type:

▾



Infinispan

RED HAT JBOSS FUSE Management Console

Fabric admin

Runtime Wiki Dashboard Health

Containers Profiles Manage MQ APIs EIPs Registry

Filter... + Create

devnexus / 1.0

RED HAT JBOSS FUSE Management Console

Camel Connect Dashboard Health JMX Logs OSGi Threads

Camel Tree

Filter...

- Camel Contexts
  - camel-example-twitter
    - Routes
      - twitter-demo
      - Endpoints
      - MBeans

Diagram Source Debug

From twitter://search 23

Log 23

OPENSIFT ONLINE

Applications Settings Support Add-ons

ipaas-meisele.rhcloud.com Started 1

Created about 1 hour ago in domain meisele and the aws-us-east-1 region

Cartridges

	Status	Gears	Storage
JBoss Fuse 6.1.0 EA	Started	1 large	1 GB

Source Code

ssh://543501234382ec67cc0004d3@ipaas-meisele.rh

Pass this URL to 'git clone' to copy the repository locally.

Remote Access

Want to log in to your application?

Delete this application...

Databases

- Add MongoDB 2.4
- Add MySQL 5.5
- Add PostgreSQL 9.2

Continuous Integration

- Enable Jenkins

Browse the Marketplace, or see the list of cartridges you can add

RED HAT JBOSS FUSE Management Console

Runtime Wiki Dashboard Health

Welcome

Java Heap Memory

7.4% (20634728) Used

92.6% (132619888) Free

Java Non Heap Memory

14.2% (25289328) Used

85.8% (127391976) Free

Loaded Classes

0.0% (0) UnloadedClassCount

100.0% (20634728) LoadedClassCount

Process CPU Load

Frequency

Containers

Filter... + Create

ipaaS / 1.0

Active Profiles

Filter... + Container Target Count

- fabric / 1.0
- hawtio / 1.0
- openshift / 1.0

	12:34	12:35	12:36	12:37	12:38	12:39	12:40	12:41	12:42	12:43	12:44	12:45	12:46
Peak thread count													
Current thread cpu time													
Daemon thread count													
Total started thread count													
Thread count													
Current thread user time													

Logs

2014-10-08 12:44:26	INFO	com.openshift.internal.client.RestService Requesting GET with protocol 1.2 on https://openshift.redhat.com/broker/rest/api
2014-10-08 12:44:26	INFO	com.openshift.internal.client.RestService Requesting GET with protocol 1.2 on https://openshift.redhat.com/broker/rest/user
2014-10-08 12:44:27	INFO	com.openshift.internal.client.RestService Requesting GET with protocol 1.2 on https://openshift.redhat.com/broker/rest/domains
2014-10-08 12:44:27	INFO	com.openshift.internal.client.RestService Requesting GET with protocol 1.2 on https://openshift.redhat.com/broker/rest/domain/meisele/applications?include=cartridges

# Fabric8 V2



- Implemented with Docker and Kubernetes
- Use any JVM (or any technology)
- Docker images, encourage static, well-defined, well-tested deployments
- Provides networking, JVM isolation, orchestration, auto-scaling, health checks, cloud deployments
- Still in community!
- Will support OpenShift v3

**And keep in mind....**

**DevOps .. Is a culture.**



## **WARNING: Challenges ahead!**

- No silver bullet; distributed systems are \*hard\*
- Dependency hell, custom shared libraries
- Fragmented and inconsistent management
- Team communication challenges
- Health checking, monitoring, liveness
- Over architecting, performance concerns, things spiraling out of control fast

# **WARNING: Challenges ahead!**

- Complex Runtime: many moving parts
- Distributed Systems are inherently complex
- Services are deployed on multiple instances
- Decentralized Data (Distributed Transactions vs eventual consistency)
- Communication between services (Network and Configuration)
- Synchronous vs. Asynchronous vs. Messaging Communication
- Communication overhead ( $n^2n$ )
- Failure Handling (Circuit Breaker)
- Service-/Metadata Registry

**Take with you today:**

- **There is no single successor to ESBs.**
- **The whole turned into pieces.**
- **We're still evolving them.**



**<http://bit.ly/virtualJBUG>  
@vJBUG**





<http://www.lordofthejars.com/2014/07/rxjava-java8-java-ee-7-arquillian-bliss.html>

<http://www.lordofthejars.com/2014/09/defend-your-application-with-hystrix.html>

<http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html>

<http://martinfowler.com/articles/microservices.html>

<http://microservices.io/patterns/microservices.html>

<http://techblog.netflix.com/2013/01/optimizing-netflix-api.html>

<http://www.infoq.com/articles/microservices-intro>