Java EE Microservices Architecture

Migrating Monolithic Systems for Sustainable Enterprise Development

@myfear
blog.eisele.net

Red Hat Developer Advocate

redhat.

monolith

/ ˈmɒn(ə)lɪθ/

noun

noun: **monolith**; plural noun: **monoliths**

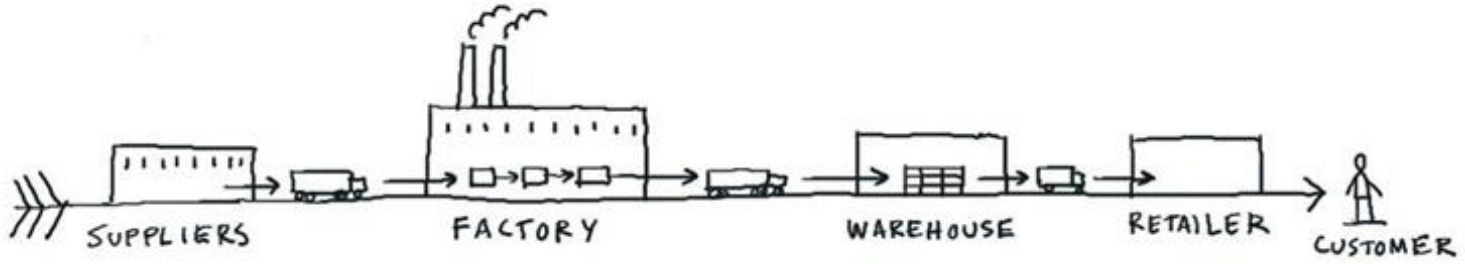
The Curse Of The Monolith

We know how to operate them

- We know how to develop
- We know how to deploy
- We know how to scale

but there is a price to pay

- Large code-bases
- Hard to understand and modify
- Complex configuration



service oriented

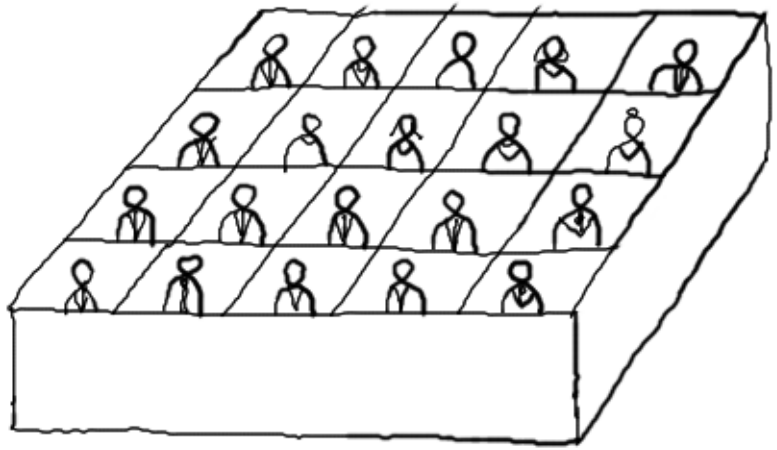
/ˈsəːvɪs/

noun

The Curse Of SOA

Increased everything

- Interoperability
- Federation
- Business and technology alignment



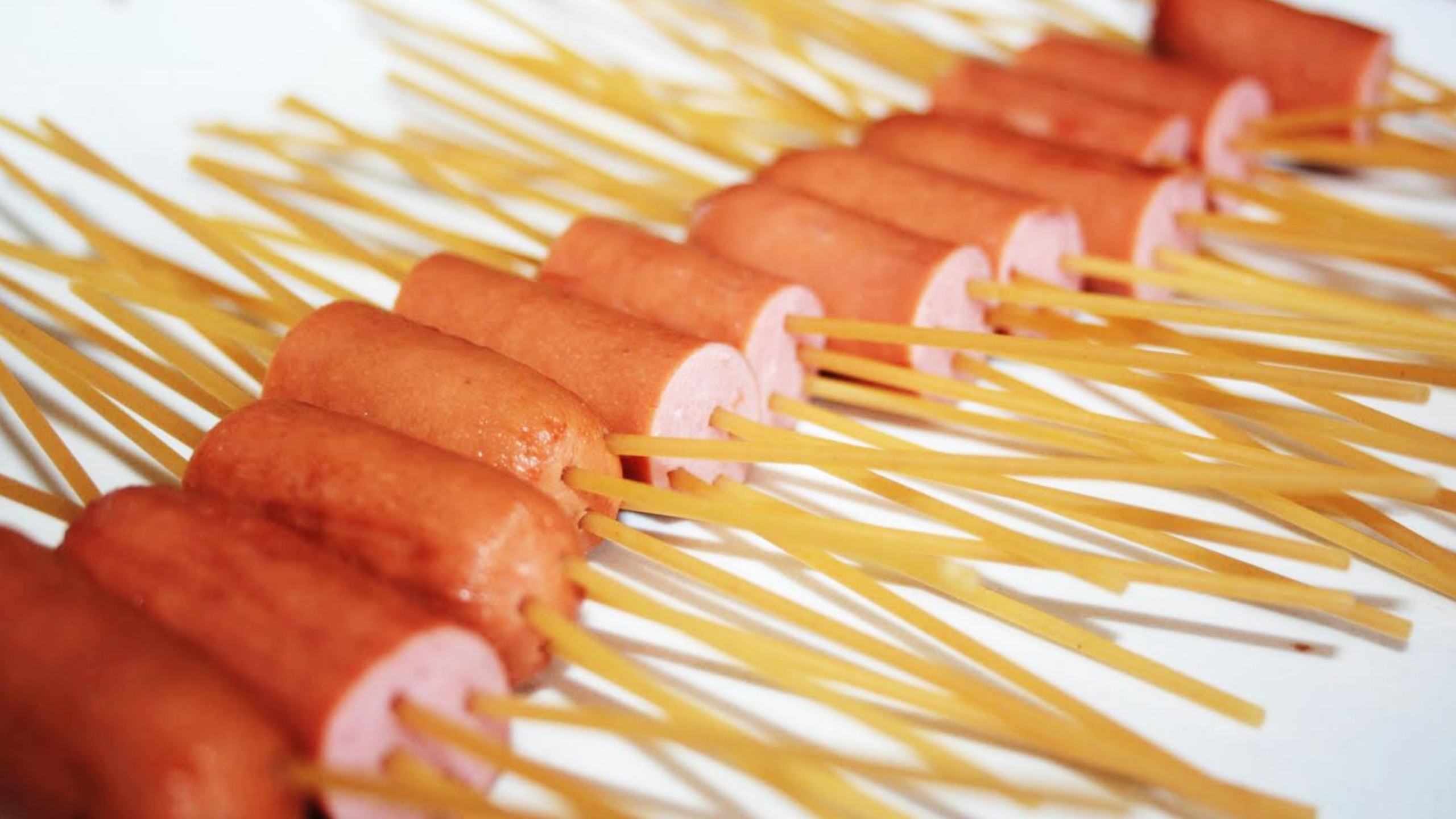
but there is a price to pay

- Centralized infrastructures
- Restricted communication protocols
- Vendor driven movement

micro services

/ˈmʌɪkrəʊ/

noun



What Are Microservices?

SOA for DevOps

- Single, self-contained, autonomous
- Easy(er) to understand individually
- Scalability
- Testing independently
- Individually deployed, has own lifecycle
- Single service going down should not impact other services
- Right technology stack for the problem (language, databases, etc)
- Fail fast
- Faster innovation, iteration

> *We want flexible systems and organizations that can adapt to their complex environments, make changes without rigid dependencies and coordination, can learn, experiment, and exhibit emergent behavior.*

Why now?

Why me?

"What if we don't change at all ...
and something magical just happens?"

Enterprise Goals and Objectives

Resistant to Change and Economically Efficient

Developers Left Alone

Technology-Centric Versus Business-Centric

Single Vendor Platform decisions increasingly unattractive

OpenSource moves quickly into this direction

Platform as a Service offerings mature

Increasing need for business value from software

Quicker turnaround cycles for changes required

We need to build systems
for **flexibility** and **resiliency**, not
just **efficiency** and **robustness.**

And we need to start building them today with what we have.

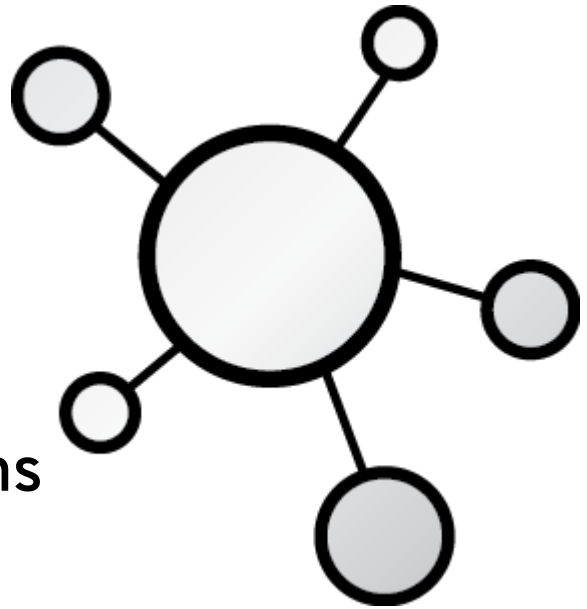
From monolith to microservice architecture

?

Technology alone
won’t solve your problem.

Outer Architecture

Software Design



Methodology and Organization

Distributed Systems

Platform As A Service

Organisation

- Autonomous, self-directed teams
- Transparency
- Small (2-pizza rule)
- Purpose, Trust, Empathy driven
- Feedback
- Experimentation
- Respond quickly to change
- Own services, delivery, operations
- Build it, you own it

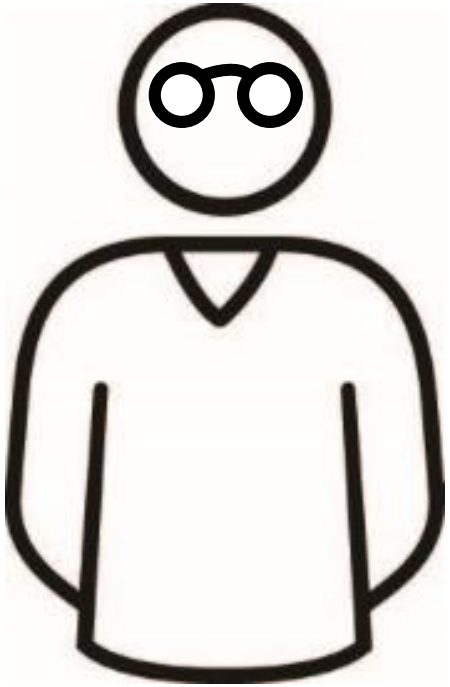
> Model culture after open source organizations: **meritocracy, shared consciousness, transparency, network, platforms.**
> (Christian Posta, Red Hat)

DEV

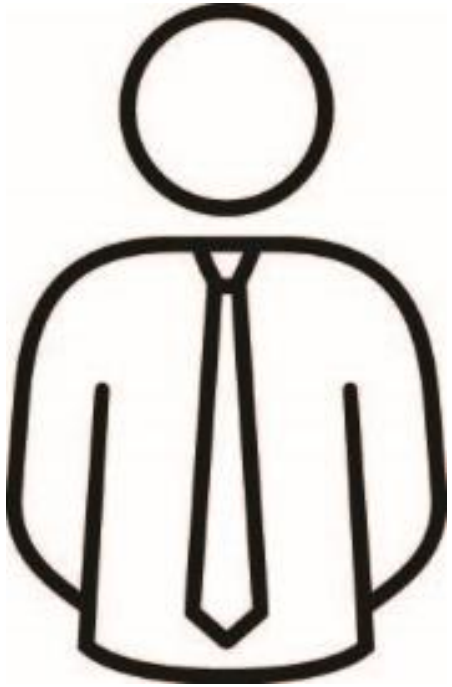
Continuously deliver software

Focus on adding value, not maintenance

Deliver features faster



OPS



Minimize manual, repetitive work

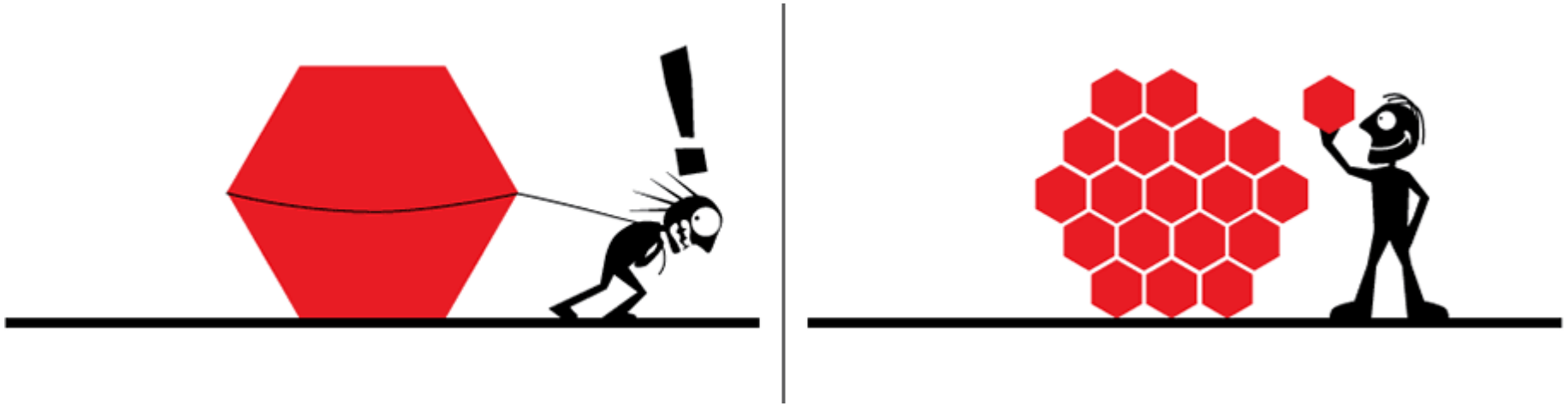
Stabilize operating environments

Encounter issues of reduced complexity

Resolve problems quicker

WATERFALL

AGILE



http://blog.procademysoftware.com/agile-is-waterfall/

Software Design

Architecture Principles

- Single Responsible Principle
- Service Oriented Architecture
  - Encapsulation
  - Separation of Concern
  - Loose Coupling
- Hexagonal Architecture

Design Patterns

- Domain-driven Design
- Bounded Contexts
- Event Sourcing
- CQRS
- Eventual Consistency
- Context Maps

Design Constraints

- Availability
- Scalability
- Performance
- Usability
- Flexibility

Best Practices

- Design for Automation
- Designed for failure
- Service load balancing and automatic scaling
- Design for Data Separation
- Design for Integrity
- Design for Performance

Strategies For Decomposing

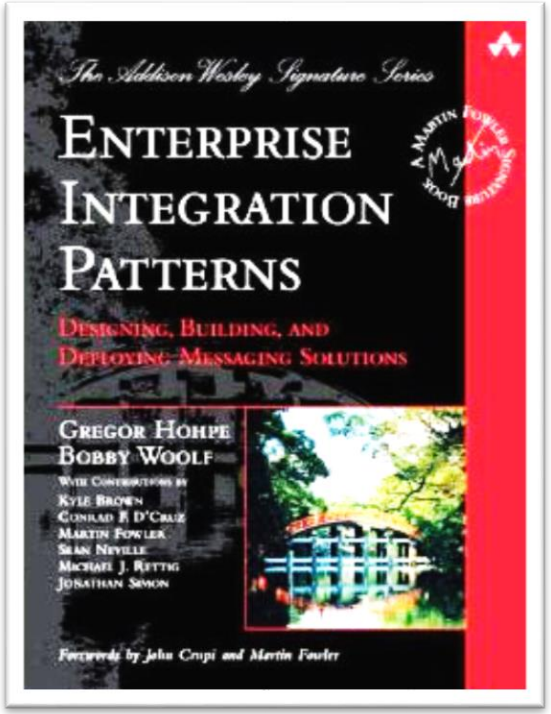
Verb or Use Case
e.g. Checkout UI

Noun
e.g. Catalog product service

Single Responsible Principle
e.g. Unix utilities

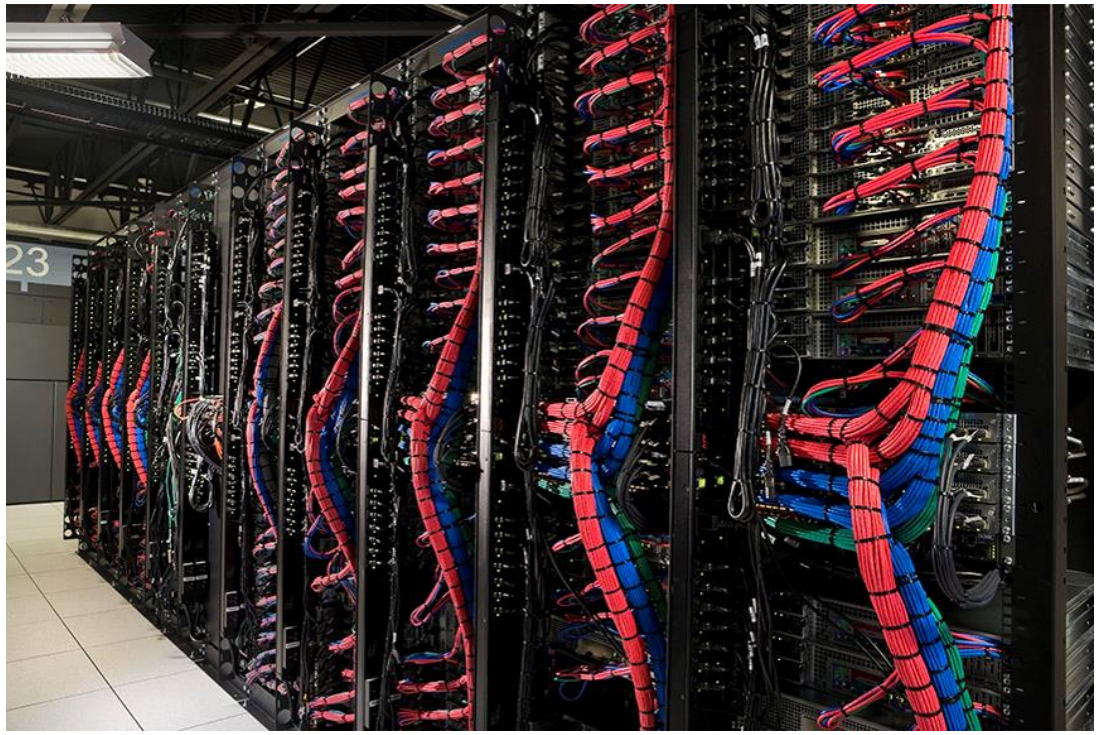
Distributed Systems

- The network is unreliable
- Design time coupling
- Unintended, run-time coupling
- Components will fail
- Design for resilience, not just robustness



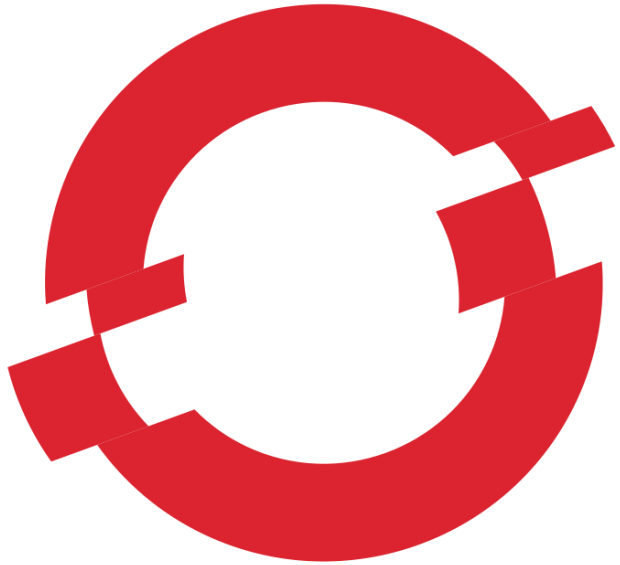
Control Dependencies

- What components depend on the others
- Which teams need to engage to make a change
- What services need to be changed if one changes
- Coordination, contention, synchronization, blocking
- Hidden dependencies



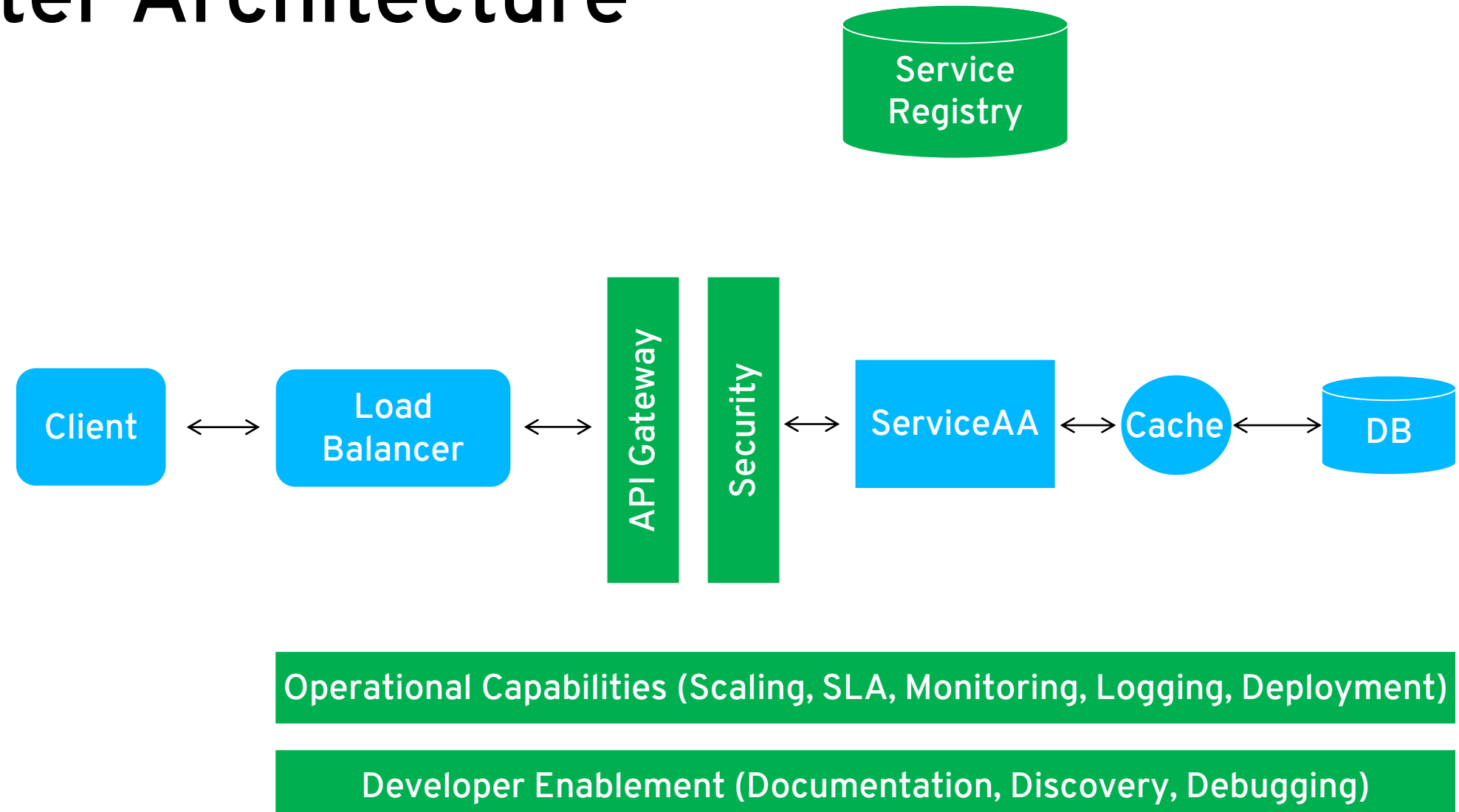
Platform as a Service

- Docker, Kubernetes
- Developer focused workflow
- Source 2 Image builds
- Build as first-class citizen
- Deployments as first-class citizen
- Software Defined Networking (SDN)
- Docker native format/packaging
- Run docker images
- CLI/Web based tooling



OPENSHIFT

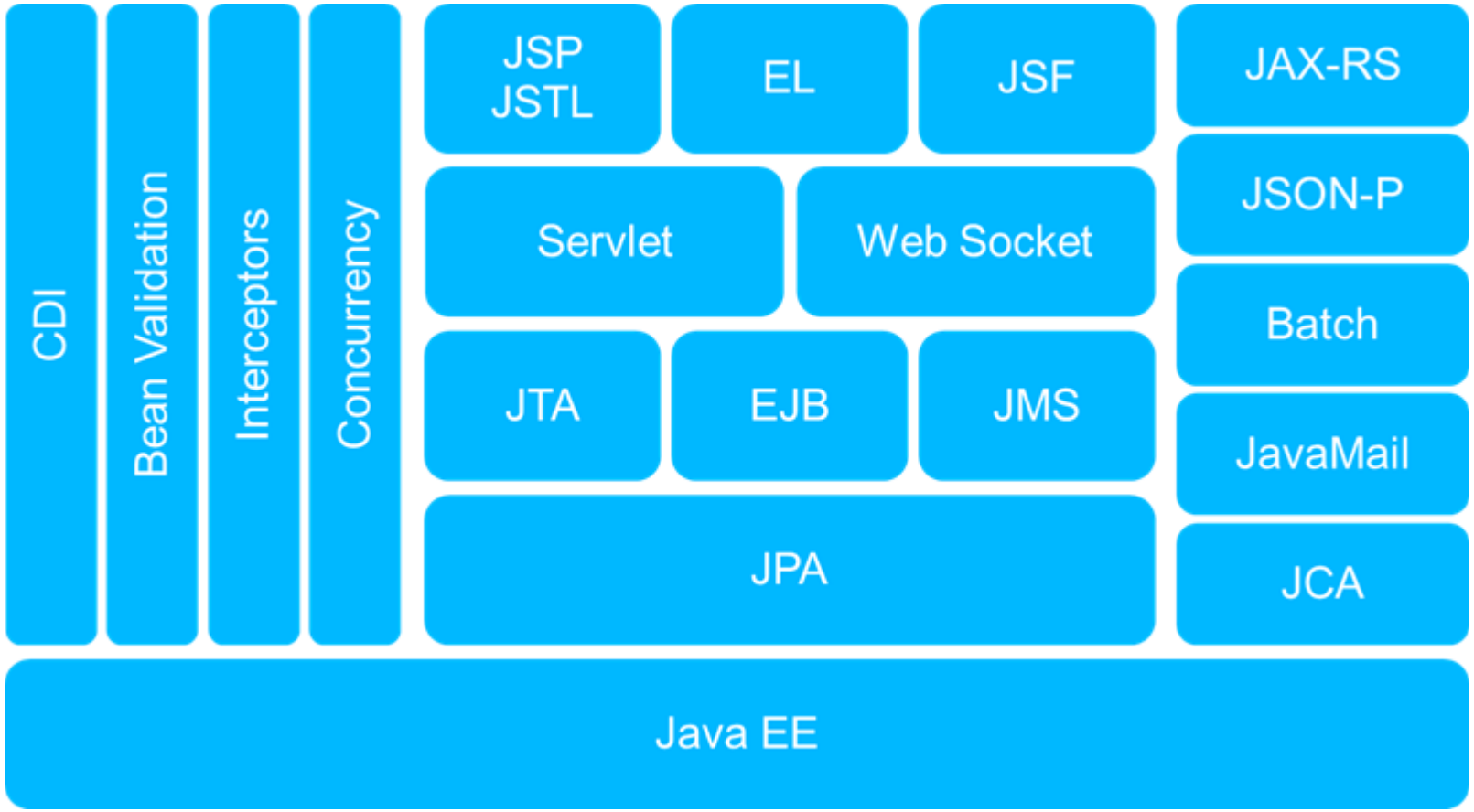
Outer Architecture



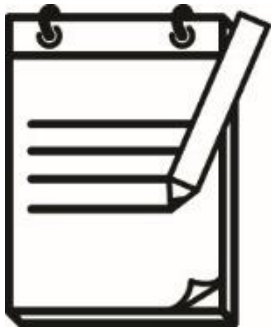
Why now?

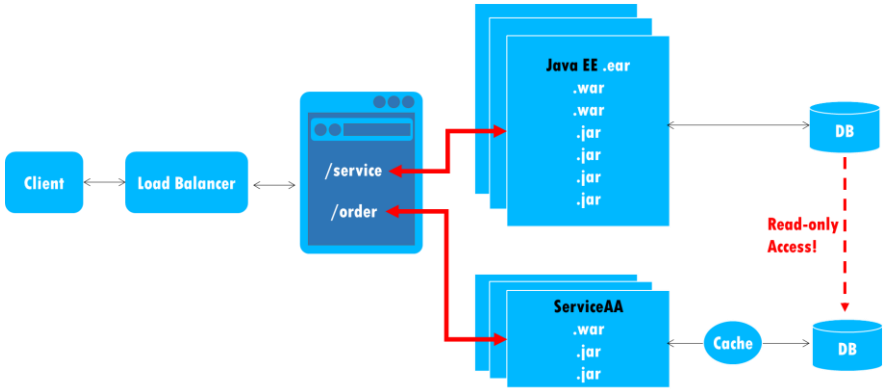
Why me?

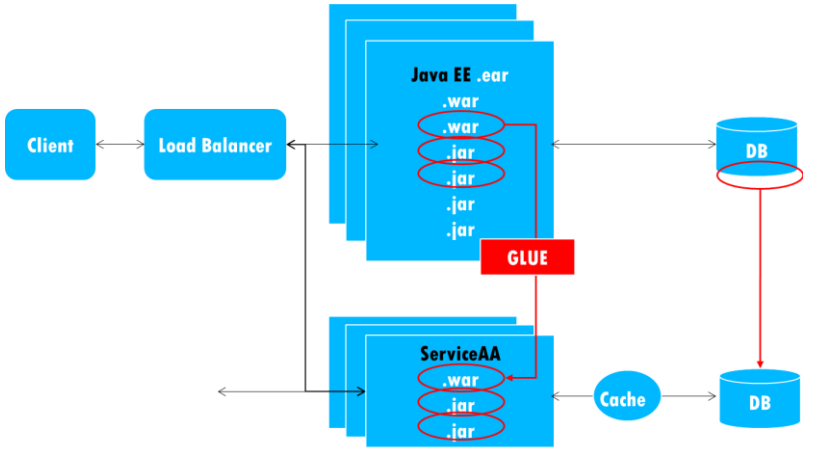
Java EE 7 Features

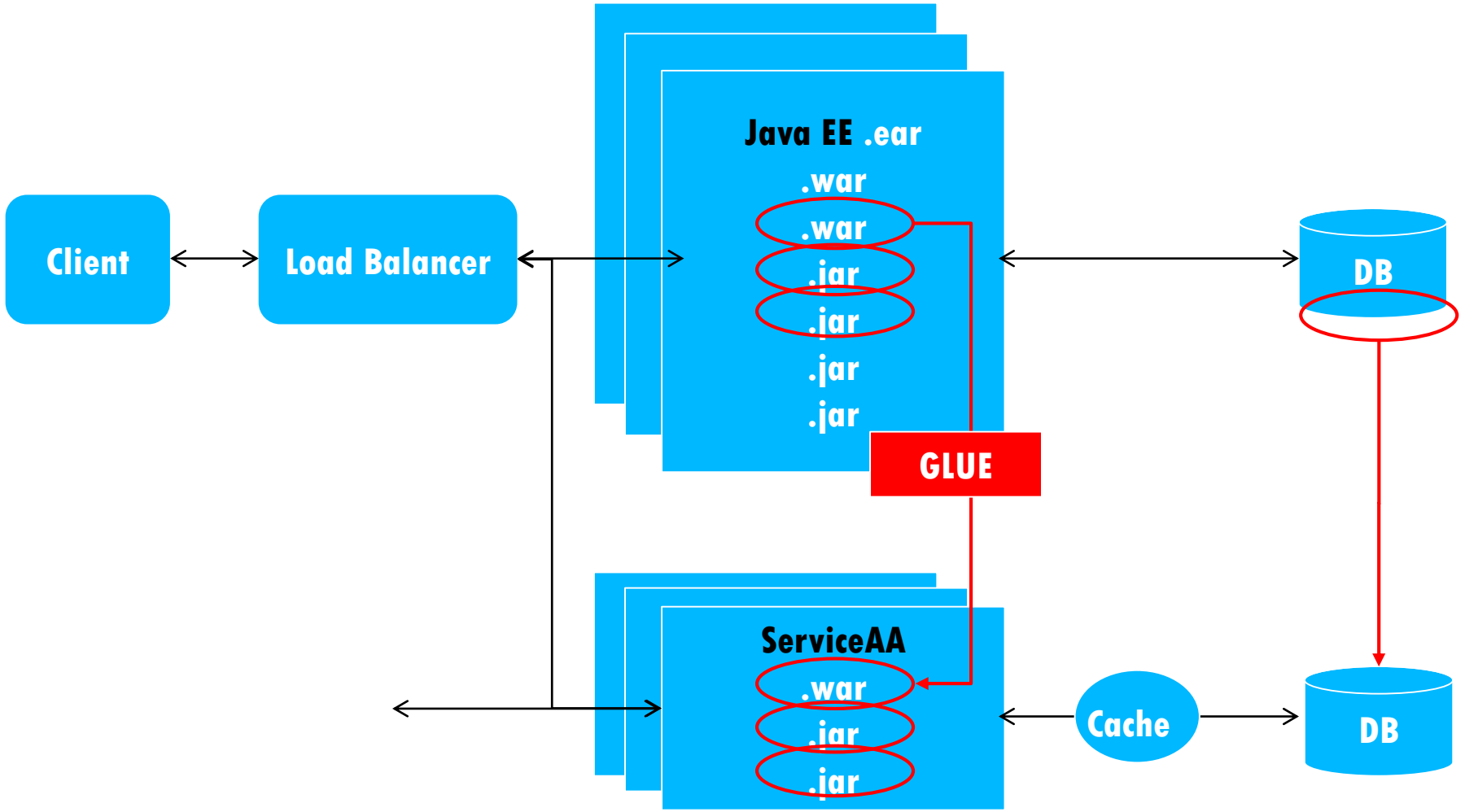


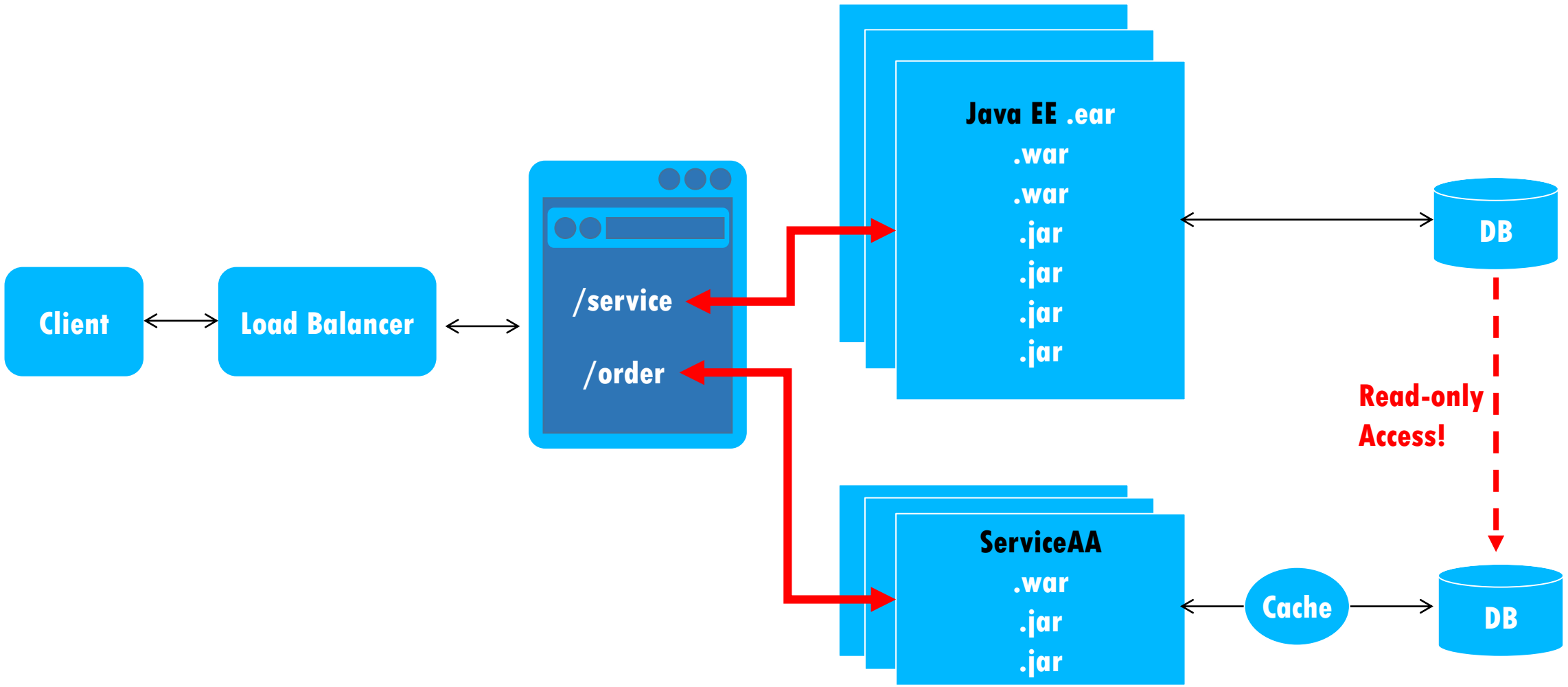
Migration Approaches



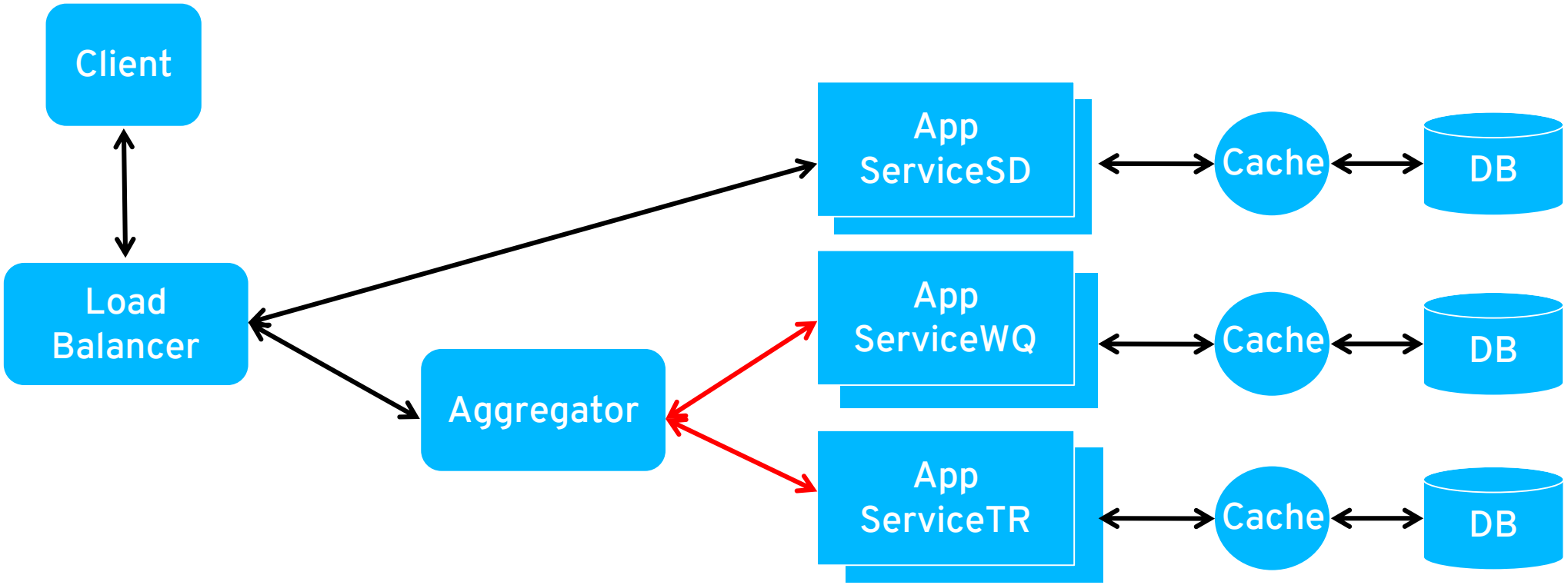


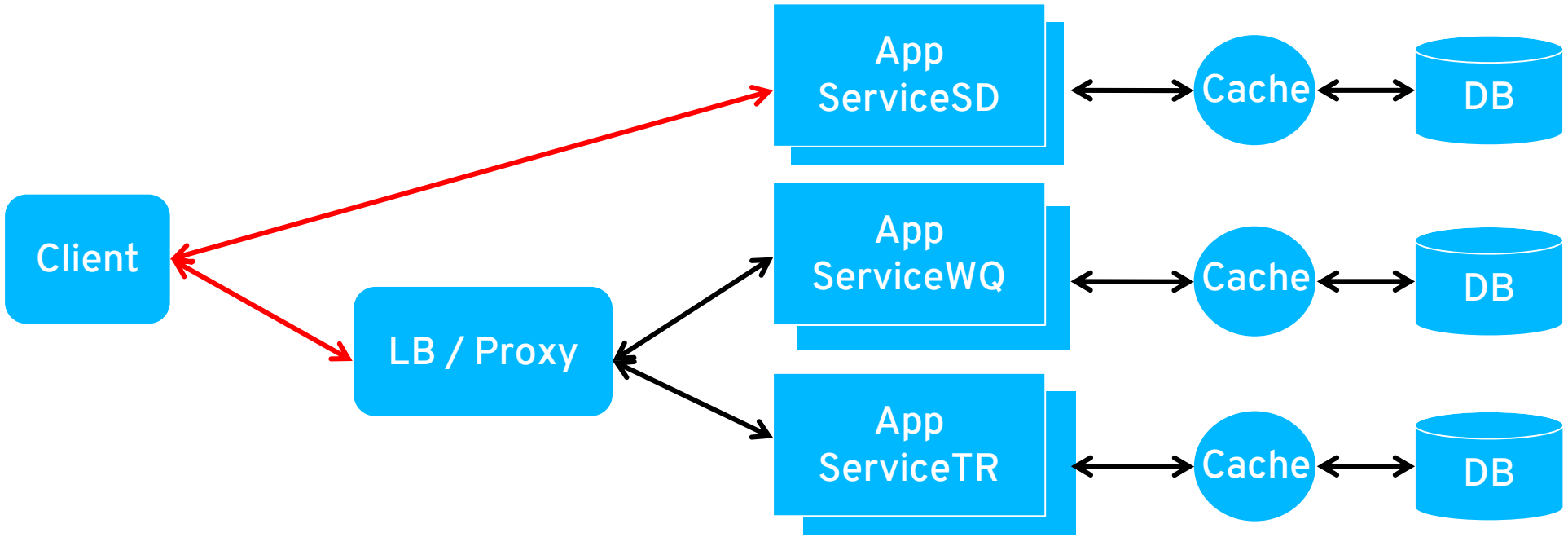


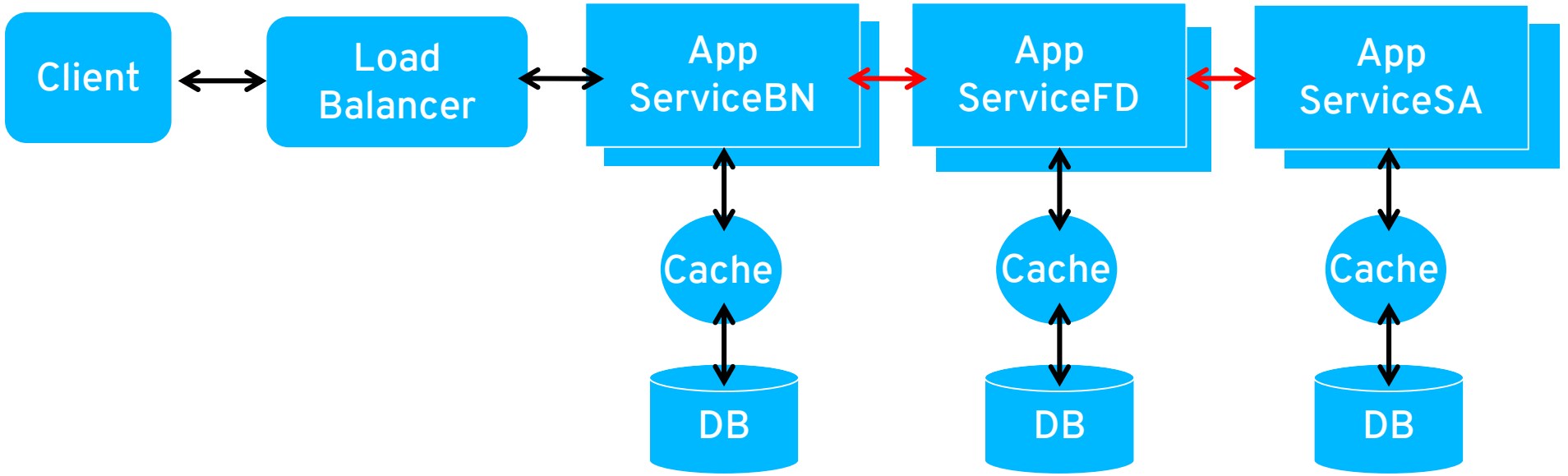

Client
Load Balancer
Java EE .ear
.war
.war
.jar
.jar
.jar
.jar
GLUE
DB
ServiceAA
.war
.jar
.jar
Cache
DB

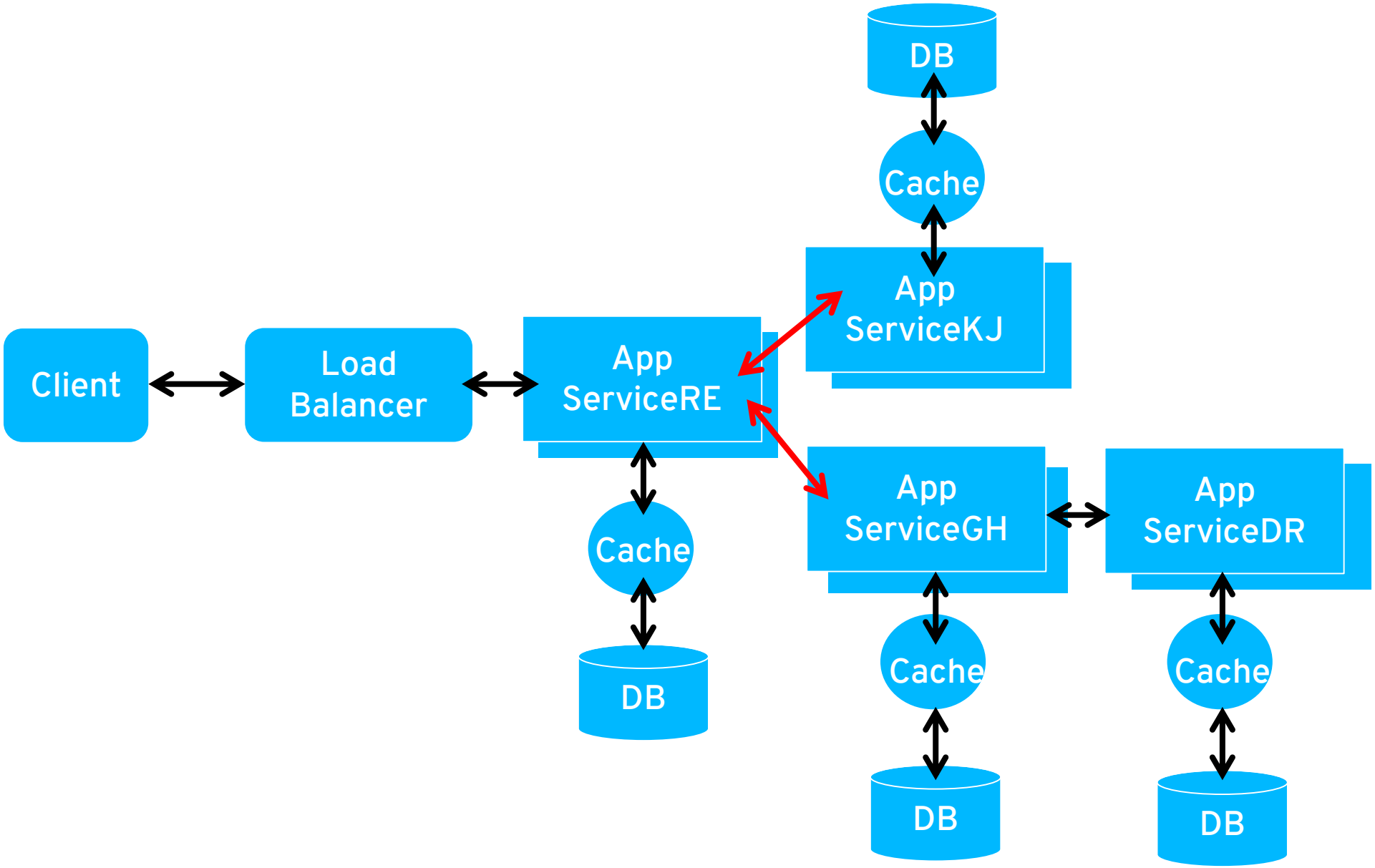

Client
Load Balancer
/service
/order
Java EE .ear
.war
.war
.jar
.jar
.jar
.jar
DB
Read-only
Access!
ServiceAA
.war
.jar
.jar
Cache
DB

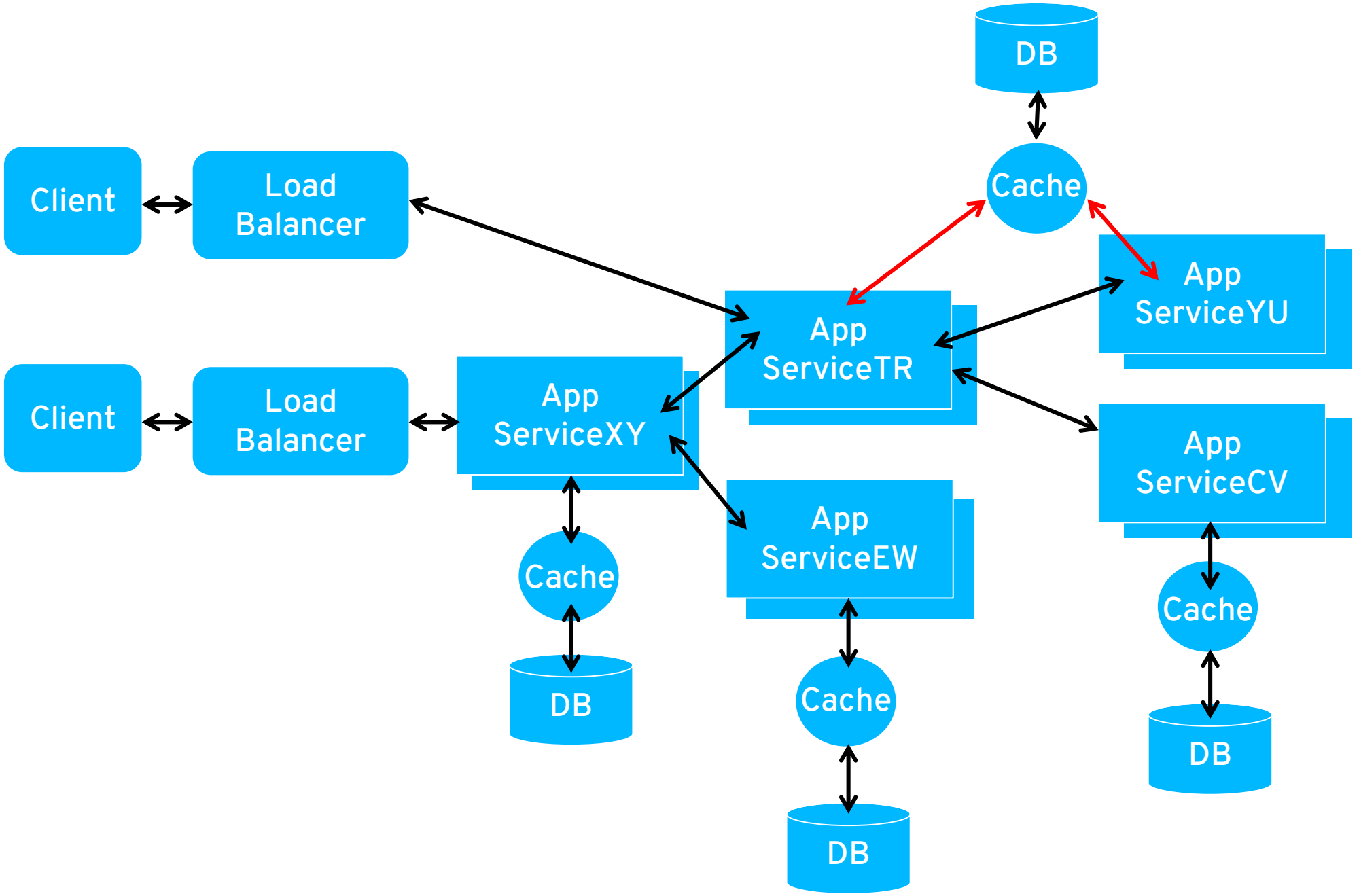
Let’s look at some patterns


Client
Load Balancer
Aggregator
App ServiceSD
App ServiceWQ
App ServiceTR
Cache
Cache
Cache
DB
DB
DB


Client
LB / Proxy
App
ServiceSD
App
ServiceWQ
App
ServiceTR
Cache
Cache
Cache
DB
DB
DB


Client
Load Balancer
App ServiceBN
App ServiceFD
App ServiceSA
Cache
Cache
Cache
DB
DB
DB

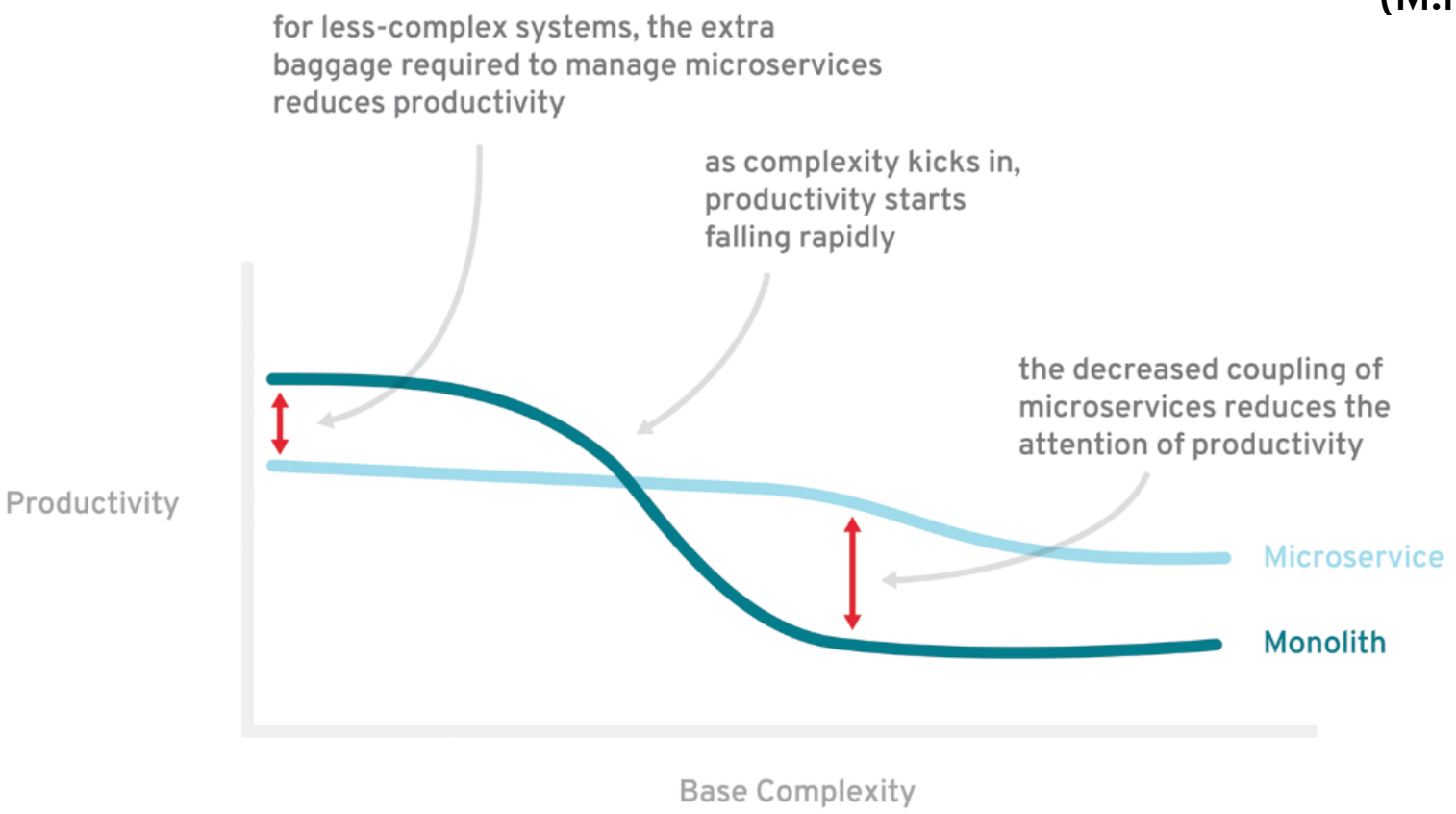

DB
Cache
App
ServiceKJ
Client
Load
Balancer
App
ServiceRE
Cache
DB
App
ServiceGH
App
ServiceDR
Cache
Cache
DB
DB


DB
Client
Load Balancer
Cache
App ServiceYU
App ServiceTR
Client
Load Balancer
App ServiceXY
App ServiceCV
Cache
App ServiceEW
Cache
DB
Cache
DB
DB

Respect The Challenge

- 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

> The fulcrum of whether or not to use microservices is the complexity of the system you're contemplating. (M.Fowler)

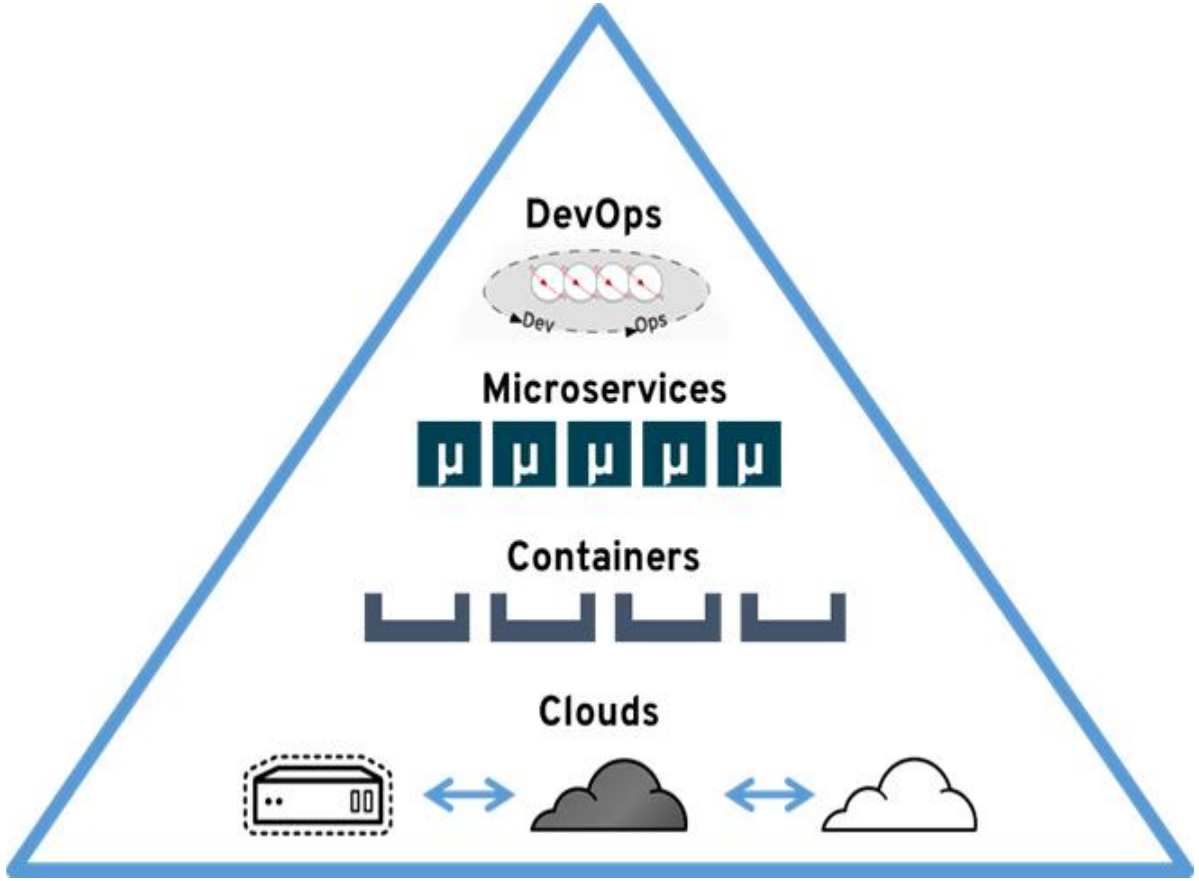


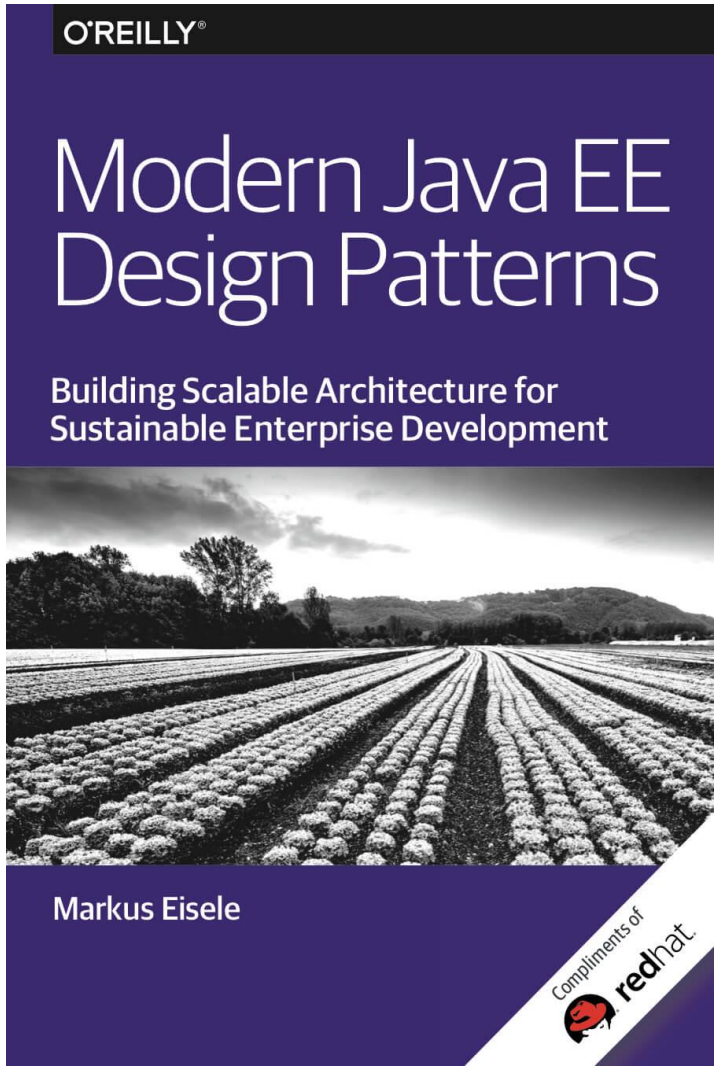
http://martinfowler.com/bliki/MicroservicePremium.html

Lessons Learned Today

- Correct functional decomposition is crucial for microservices:
  - pretty hard to get right from the start
  - a modular system can evolve to microservices
  - balance the needs with the costs
  - work on it evolutionary
- Java EE can be a platform for microservices
- You need a lot more than just technology

Fit all the pieces together





- Understand the challenges of starting a greenfield development vs tearing apart an existing brownfield application into services
- Examine your business domain to see if microservices would be a good fit
- Explore best practices for automation, high availability, data separation, and performance
- Align your development teams around business capabilities and responsibilities
- Inspect design patterns such as aggregator, proxy, pipeline, or shared resources to model service interactions

http://bit.ly/ModernJavaEE