中国科学技术大学
University of Science and Technology of China

Software Architecture

SSE USTC Qing Ding
dingqing@ustc.edu.cn
http://staff.ustc.edu.cn/~dingqing

Microservice Architecture

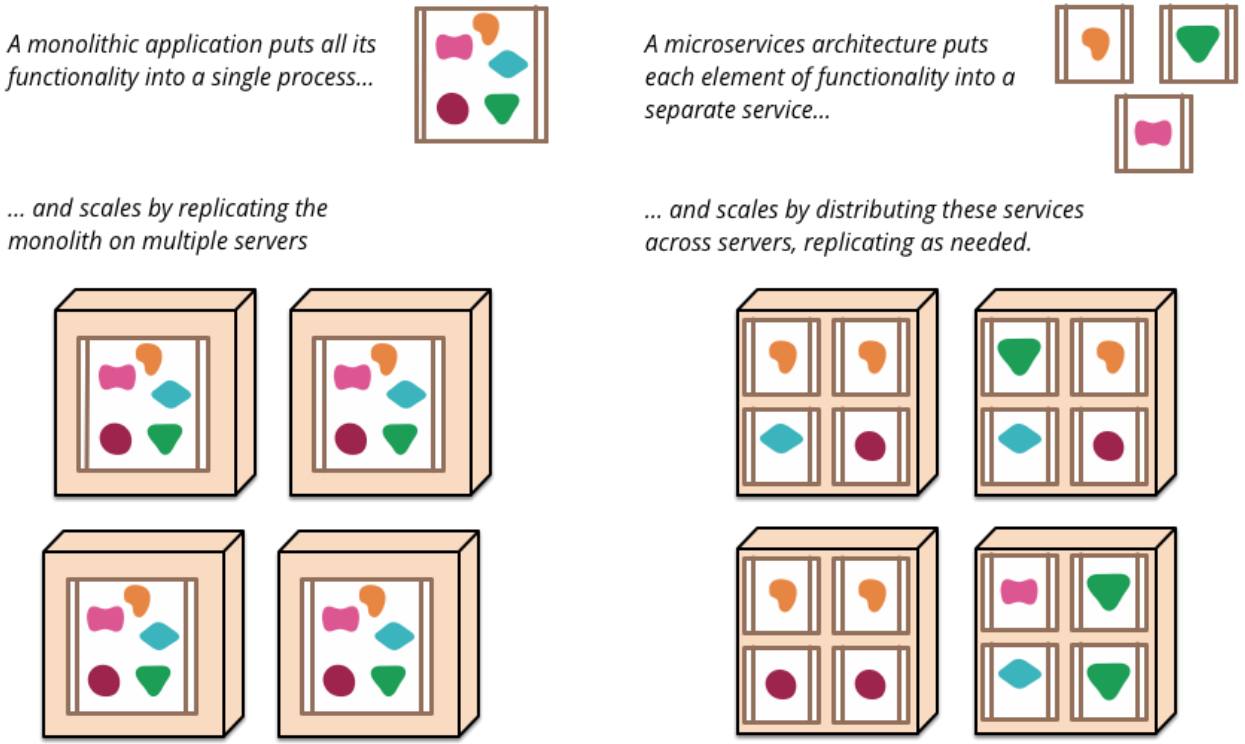
outline

- We’ve come a long way already
- Why Micro-services?
- What is a micro-service?
- Comparisons with Precursors
- MicroService Implemention
- MicroService Challenges
- Conclusion
- Examples about Microservices

Microservices:
we've come a long way already

2014: Martin Fowler and James Lewis

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



碎片化、扁平式

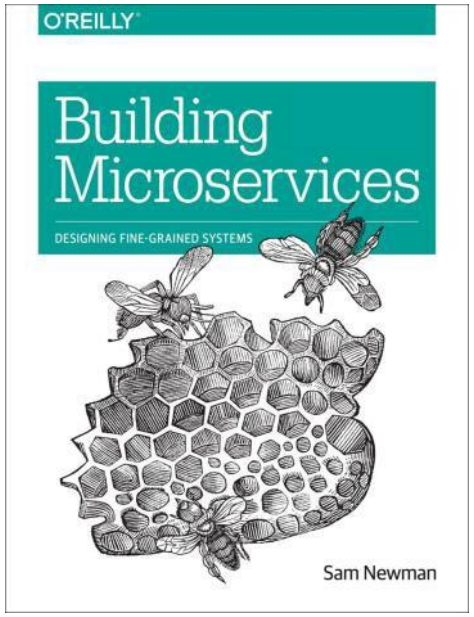
2015: Sam Newman

在我看来，整合是与微服务相关的技术中最重要的一个方面。做得好，你的微服务就会保留它们的自主权，允许你独立于整体进行更改和发布。如果做错了，灾难就来了

"Getting integration right is the single most important aspect of the technology associated with microservices in my opinion. Do it well, and your microservices retain their autonomy, allowing you to change and release them independent of the whole. Get it wrong, and disaster awaits."

- **Sam Newman**, *Building Microservices*

http://shop.oreilly.com/product/0636920033158.do

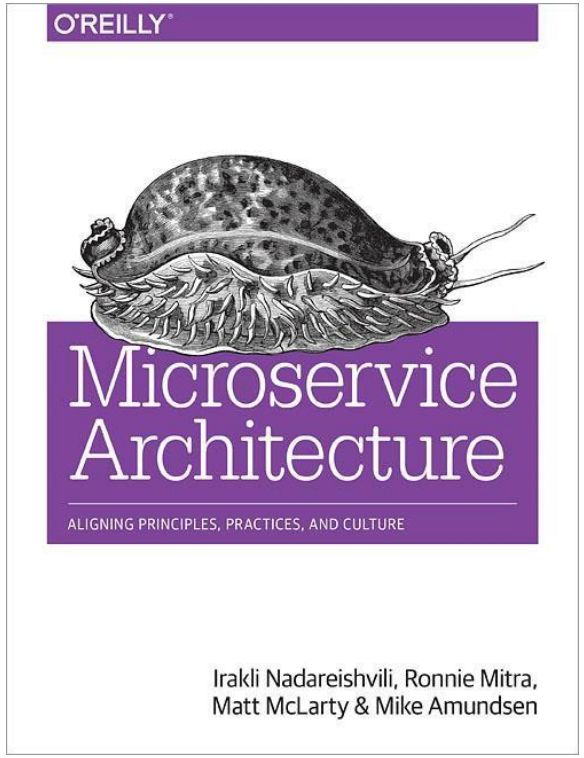


2016: Microservice Architecture

"A ***microservice*** is an independently deployable component of bounded scope that supports interoperability through message-based communication.

Microservice architecture is a style of engineering highly-automated, evolvable software systems made up of capability-aligned microservices."

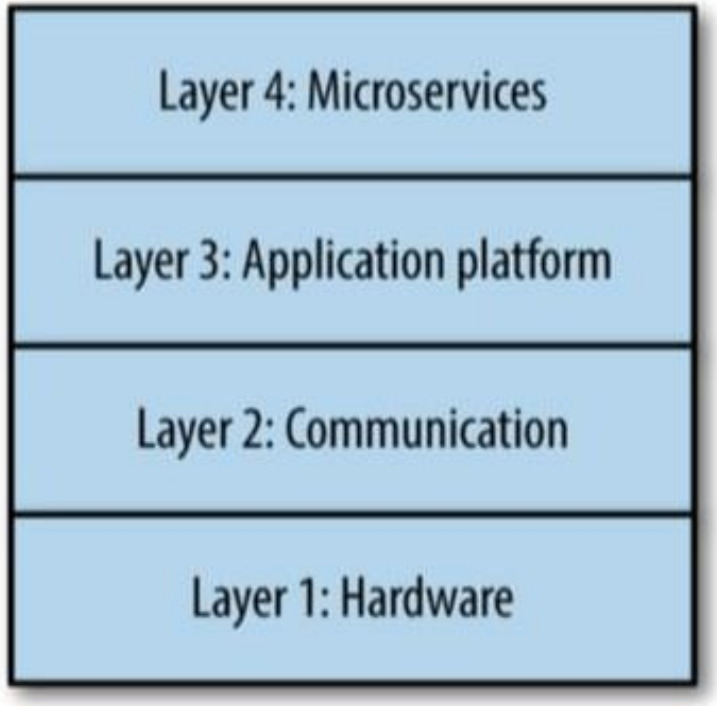
http://shop.oreilly.com/product/0636920050308.do

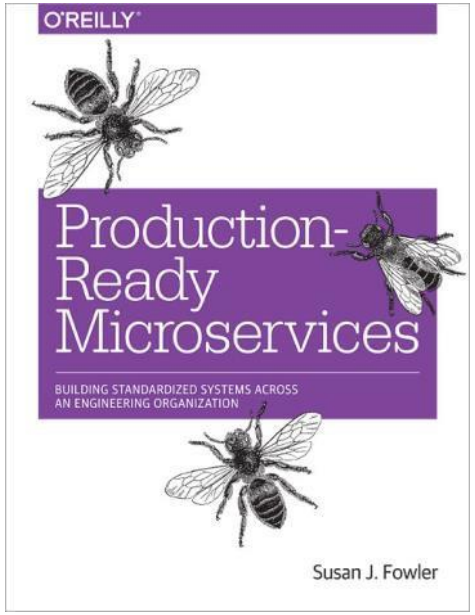


2016: Susan Fowler

The Four Layers of Microservice Architecture

http://shop.oreilly.com/product/0636920053675.do





Why Microservices?

Why Microservices?

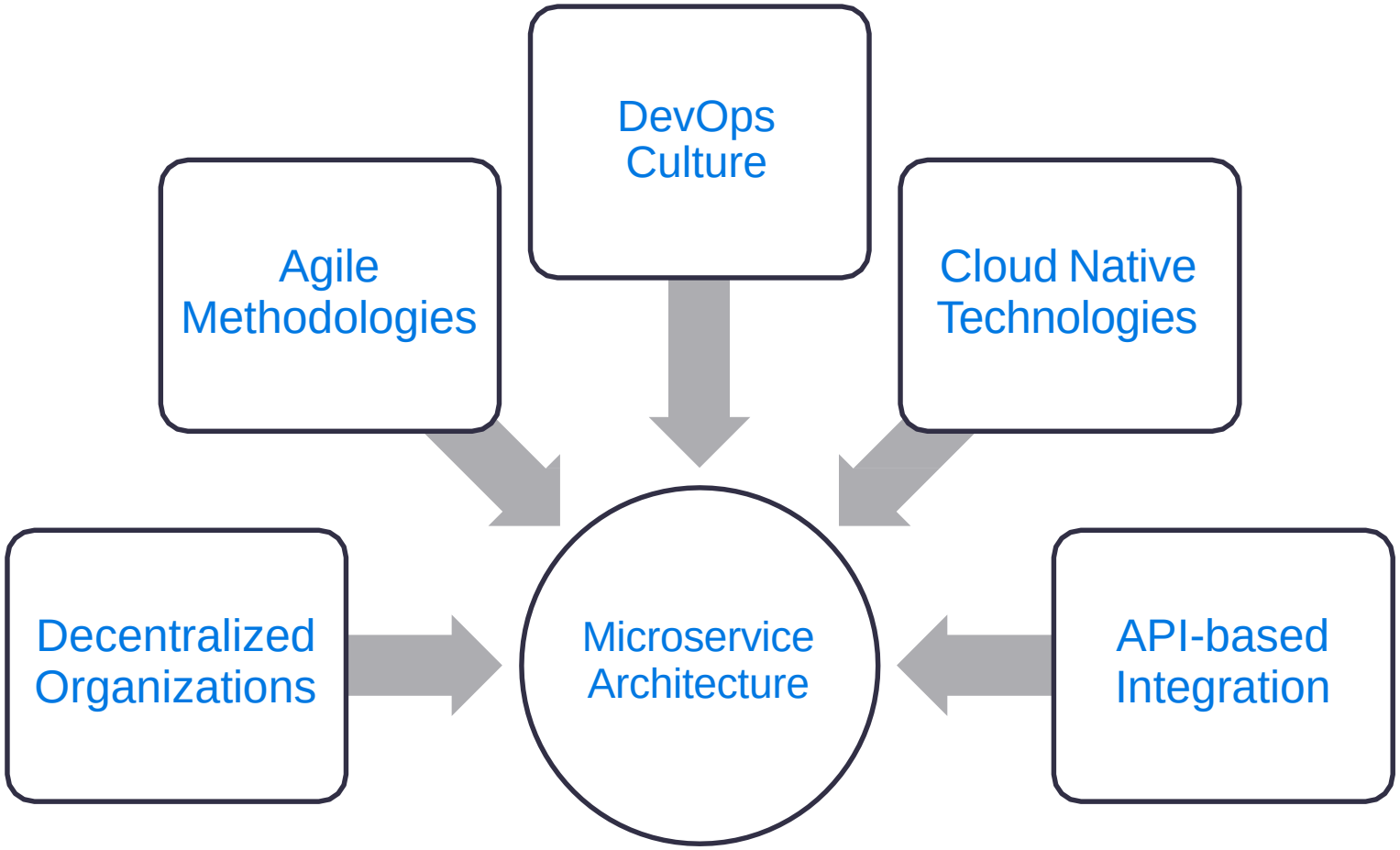
- Gilt: “From Monolith Ruby App to Distributed Scala Micro-Services” (*NYC Tech Talks*) [Link]
- Nike: “Nike’s Journey to Microservices” (*AWS Re:Invent 2014*) [Link]
- SoundCloud: ” Building Products at SoundCloud - Part III: Microservices in Scala and Finagle” [Link]
- Capital One: “Lack Of Legacy Lets Capital One Build Nimble
- Infrastructure” [Link]
- Hailo: “A Journey into Microservices” [Link]
- Autoscout24: “Why Autoscout24 changes its technology” [Link]
- Zalando: “From Monolith to Microservices” [Link]

Why?

- Faster and simpler deployments and rollbacks
  - Independent Speed of Delivery (by different teams) 模块间独立，先做完先发布
- Right framework/tool/language for each domain
  - Recommendation component using Python?, Catalog Service in Java ..
- Greater Resiliency 容错性好
  - Fault Isolation
- Better Availability
  - If architected right ☺

云计算隔离性最好，第1层是硬件，第2层时虚拟机（虚拟机上可有多个操作系统），第3层是操作系统，第4层是应用，将操作系统和应用打包成磁盘映像就可部署，应用的崩溃不会影响另一个操作系统中的应用

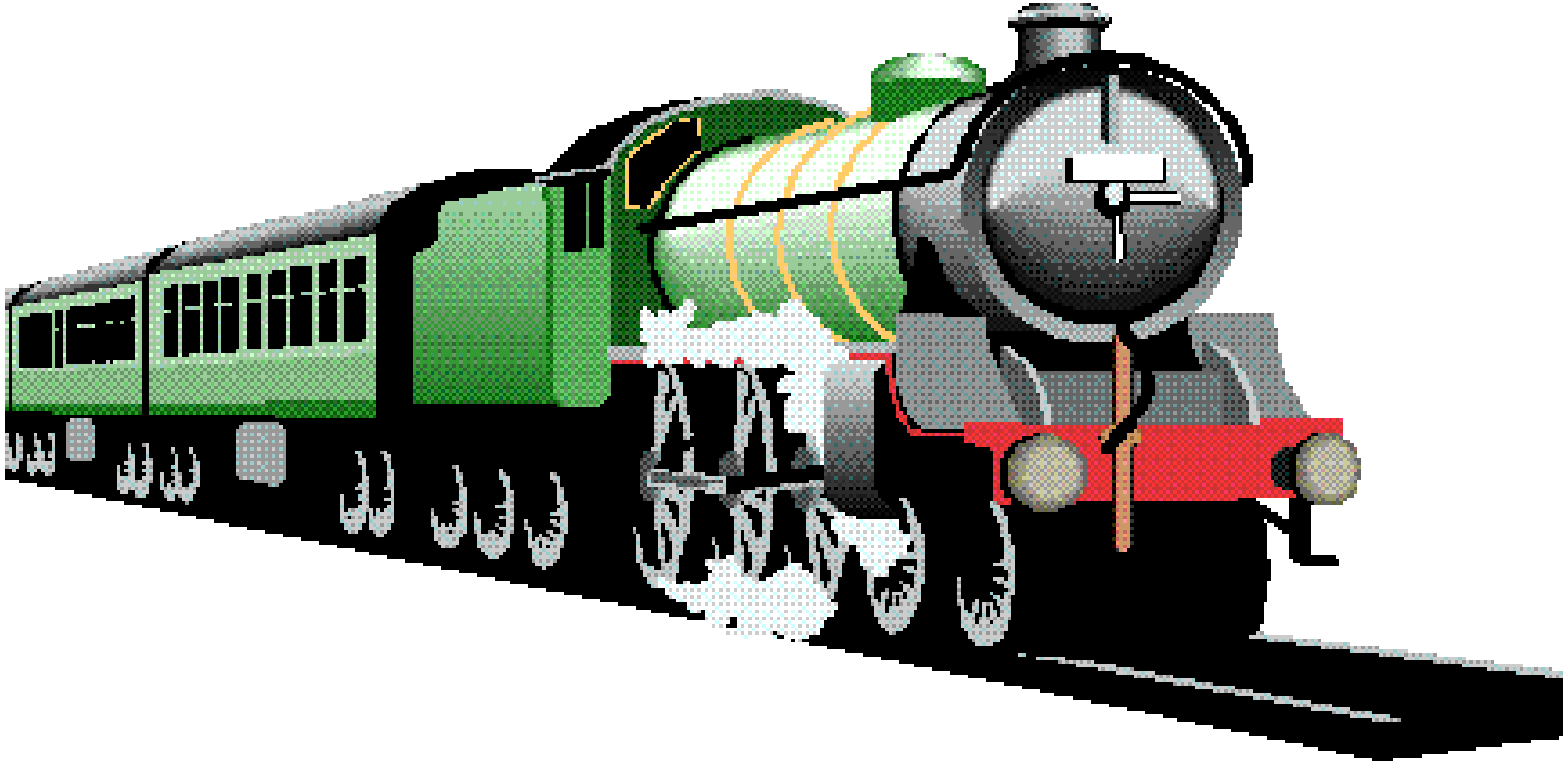
Forces driving microservices



Observed problems

- Area of consideration
  - Web systems
  - Built collaboratively by several development teams
  - With traffic load that requires horizontal scaling (i.e. load balancing across multiple copies of the system)

- Observation
  - Such systems are often built as *monoliths* or *layered* systems (JEE)



Monolithic App

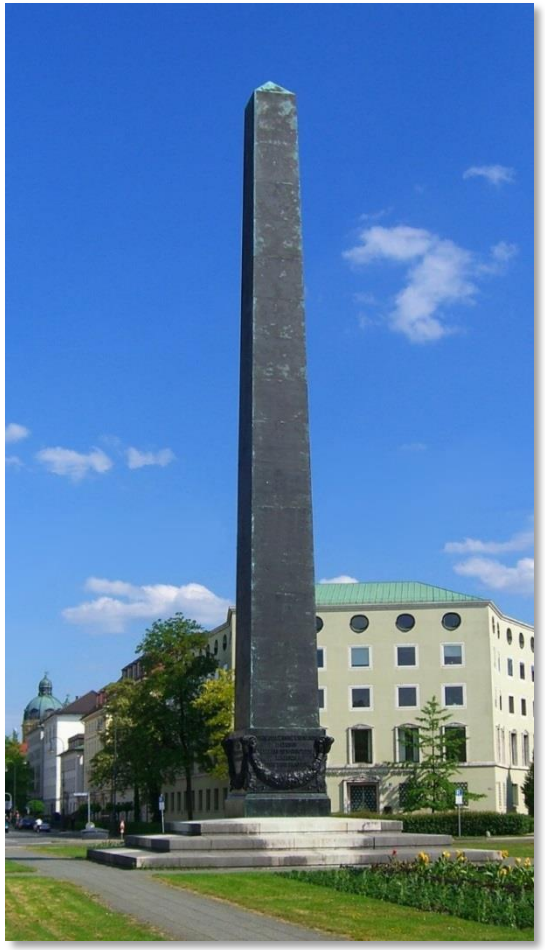
Software Monolith

A Software Monolith

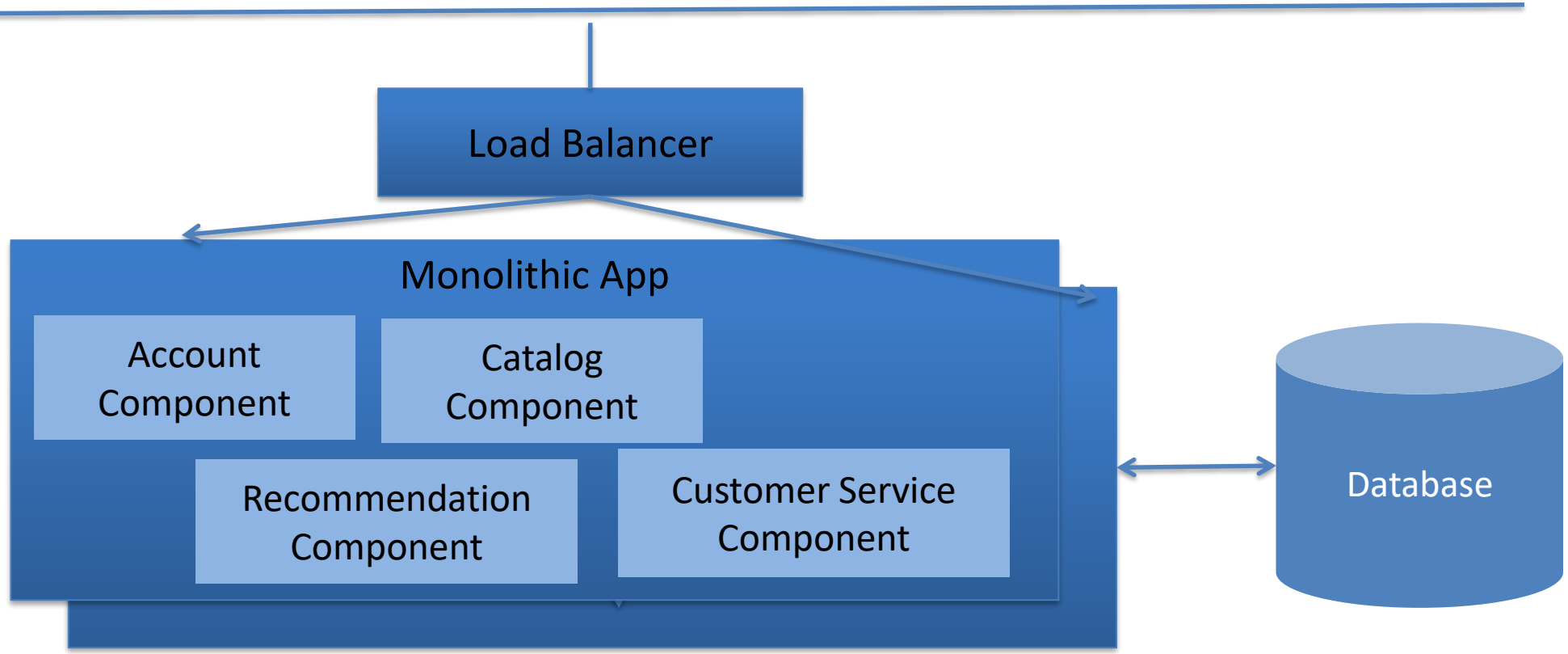
- One build and deployment unit
- One code base
- One technology stack (Linux, JVM, Tomcat, Libraries)

Benefits

- Simple mental model for developers
  - one unit of access for coding, building, and deploying
- Simple scaling model for operations
  - just run multiple copies behind a load balancer



Monolithic Architecture



Characteristics

- Large Codebase
- Many Components, no clear ownership
- Long deployment cycles

Monolithic App – Evolution

- As codebase increases …
  - Tends to increase “tight coupling” between components
    - Just like the cars of a train
  - All components have to be coded in the same language

Evolution of a Monolithic App

Pros

- Single codebase
  - Easy to develop/debug/deploy
  - Good IDE support
- Easy to scale horizontally (but can only scale in an “un-differentiated” manner)
- A Central Ops team can efficiently handle

Problems of Monolithic Architectures

- Code complexity and maintainability
  - Huge and intimidating code base for developers
- Deployment becomes the bottleneck
  - Re-deploying means halting the whole system
  - Re-deployments will fail and increase the perceived risk of deployment
- Fear to change
- Lack of ownership

Problems of Monolithic Architectures

- Failure dependencies
- One size doesn't fit all (ex: relational DB)
- Hard to scale out
  - Running a copy of the whole system is resource-intense
  - It doesn't scale with the data volume out-of-the-box
- Development tools get overburdened
  - refactorings take minutes
  - builds take hours
  - testing in continuous integration takes days

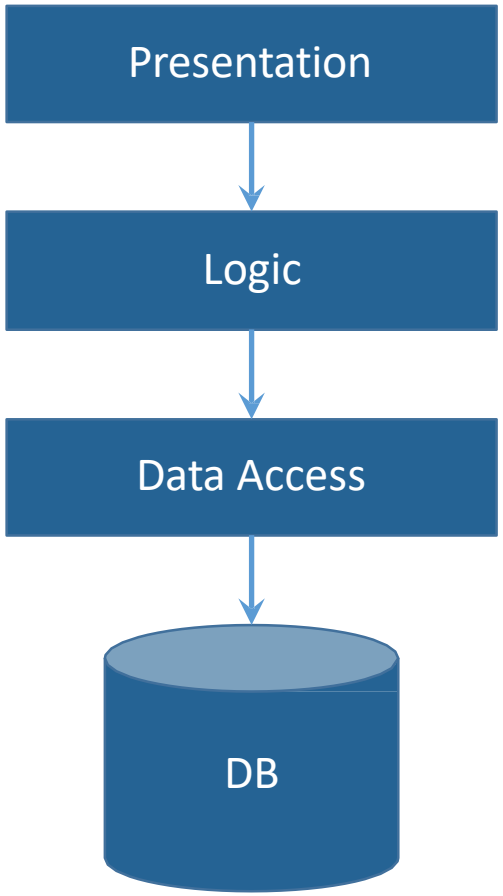
Layered Systems

A layered system decomposes a monolith into layers

- Usually: presentation, logic, data access
- At most one technology stack per layer
  - Presentation: Linux, JVM, Tomcat, Libs, EJB client, JavaScript
  - Logic: Linux, JVM, EJB container, Libs
  - Data Access: Linux, JVM, EJB JPA, EJB container, Libs

Benefits

- Simple mental model, simple dependencies
- Simple deployment and scaling model



Problems of Layered Systems

- Still huge codebases (one per layer)
- … with the same impact on development, building, and deployment
- Scaling works better, but still limited
- Staff growth is limited: roughly speaking, one team per layer works well
  - Developers become specialists on their layer
  - Communication between teams is biased by layer experience (or lack thereof)

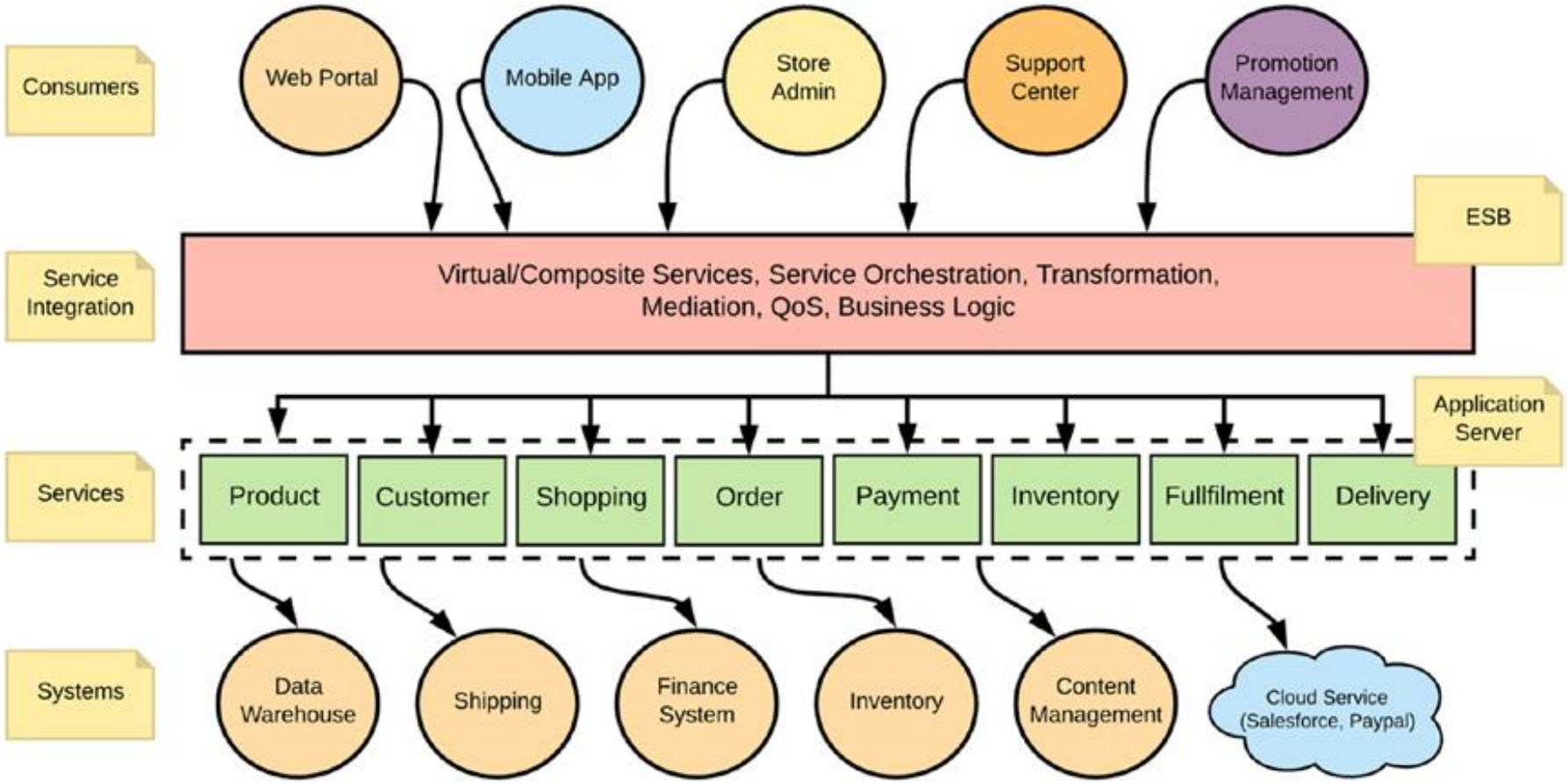
Growing systems beyond the limits

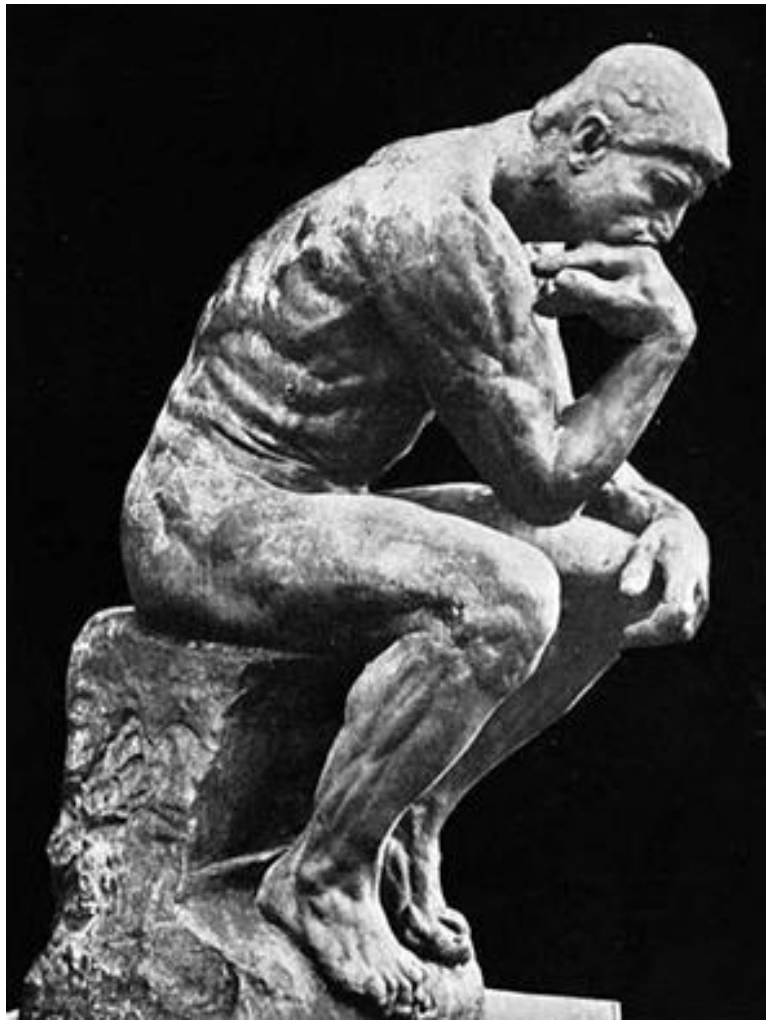
- Applications and teams need to grow beyond the limits imposed by monoliths and layered systems, and they do - in an uncontrolled way.
- Large companies end up with landscapes of layered systems that often interoperate in undocumented ways.
- These landscapes then often break in unexpected ways.

How can a company grow and still have a working IT architecture and vision?

- Observing and documenting successful companies (e.g. Amazon, Netflix) lead to the definition of microservice architecture principles.

SOA/ESB style based online retail system





MicroServices

You Think??

Benefits of Microservices

Speed

- Faster development and deployment

Innovation

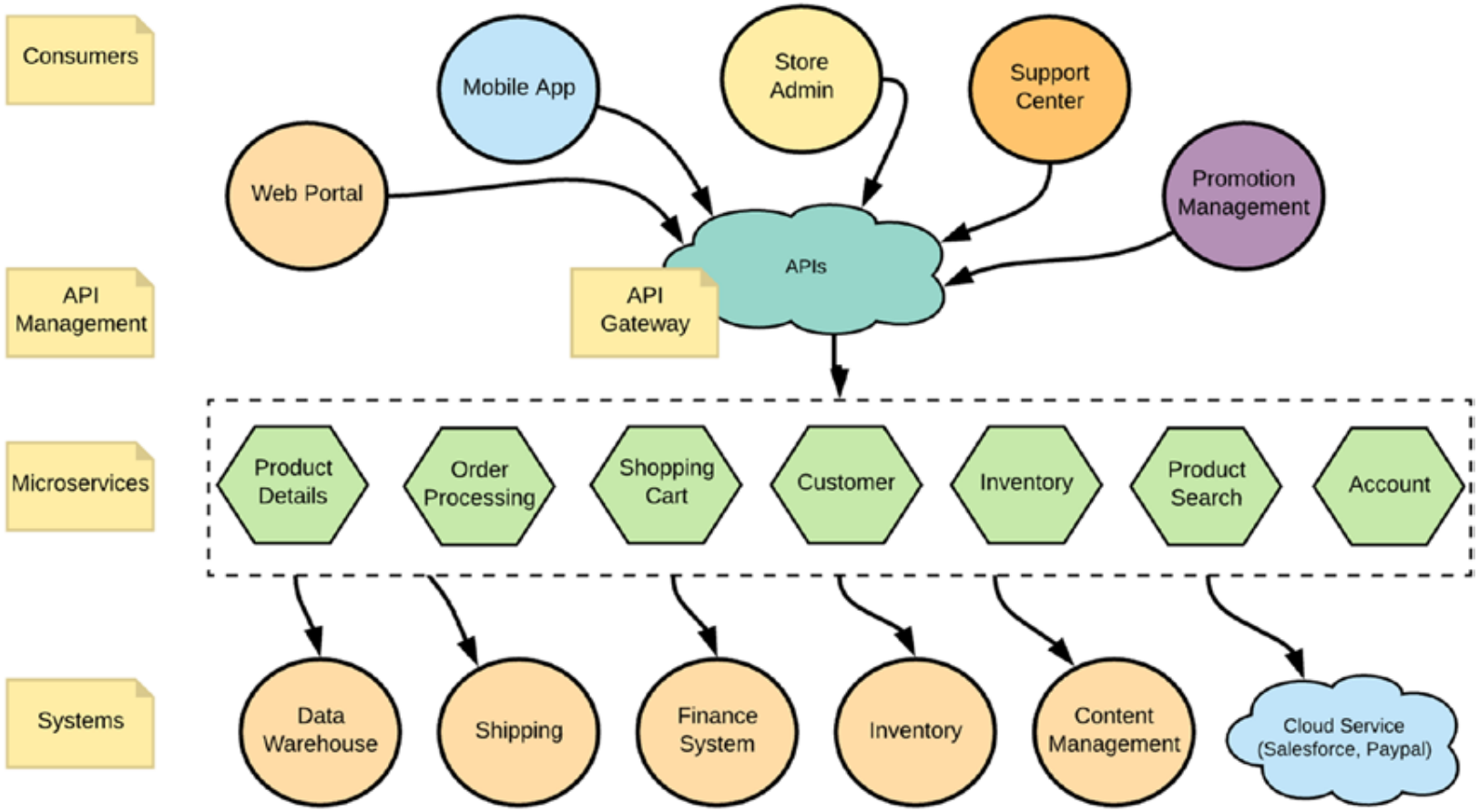
- Autonomy of teams, culture of change
- Ownership and DevOps culture

Quality

- Composability and reusability
- More maintainable code
- Better scaling and optimizations
- Failure Isolation and Resiliency

What is a MicroService?

An online retail application built using a microservices architecture

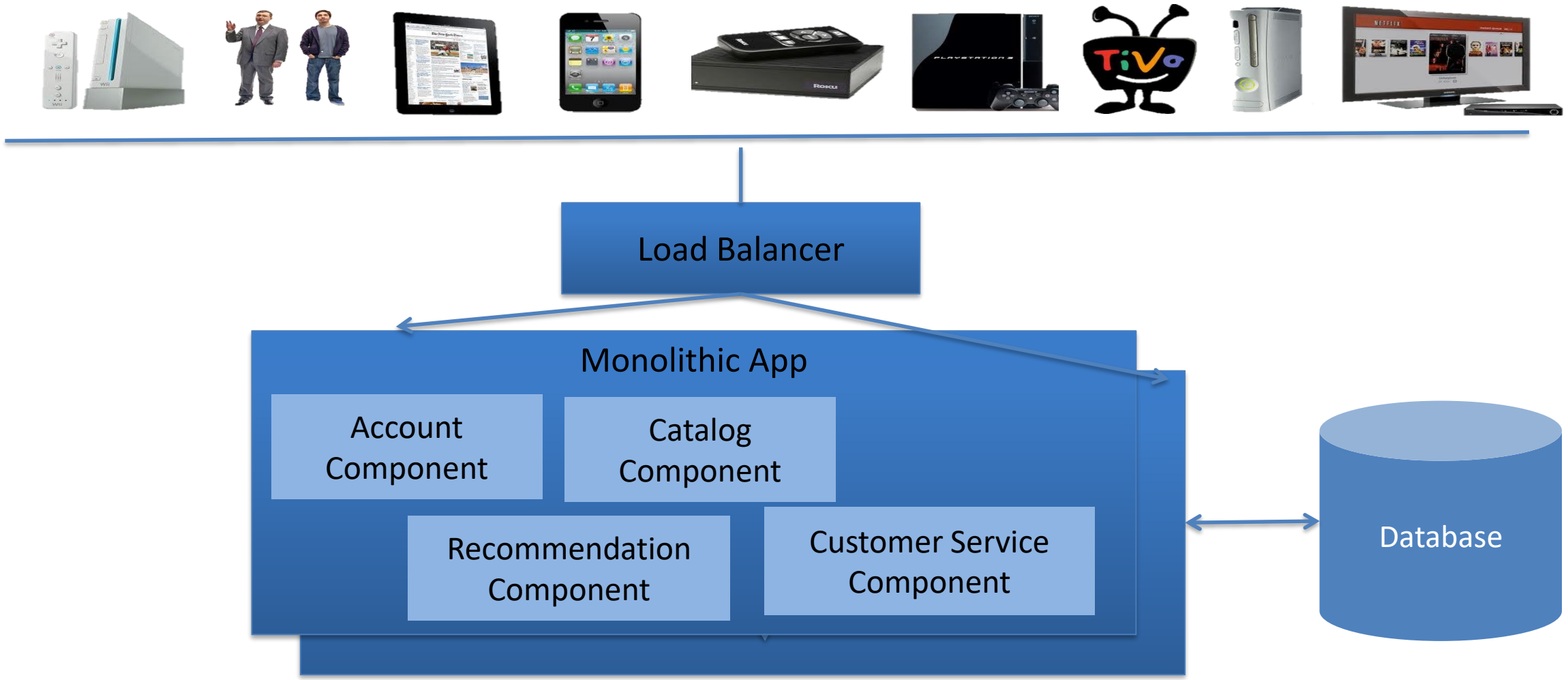


Comparing Monolithic to MicroServices

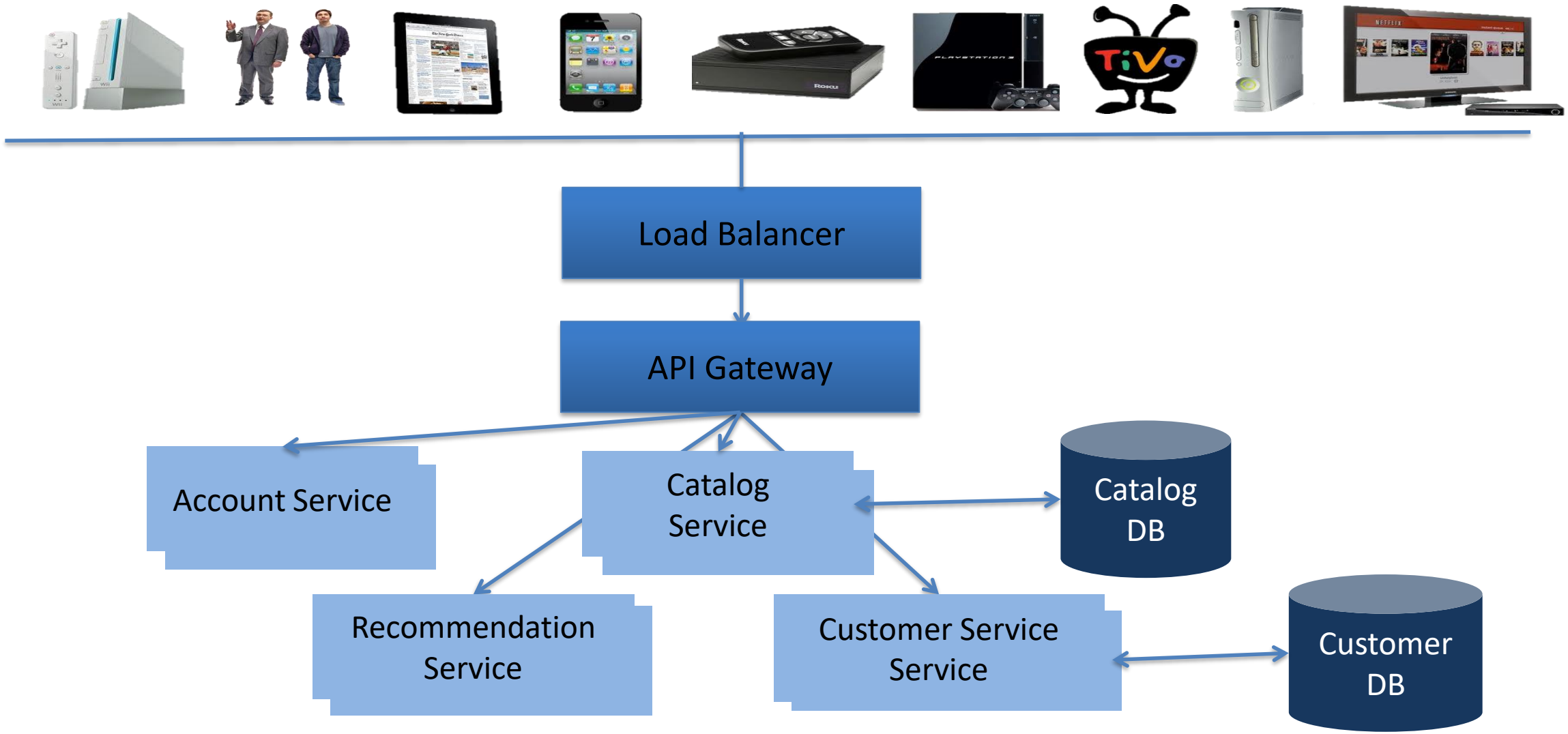
Monolithic App (Various Components linked together)

MicroServices – separate single purpose services

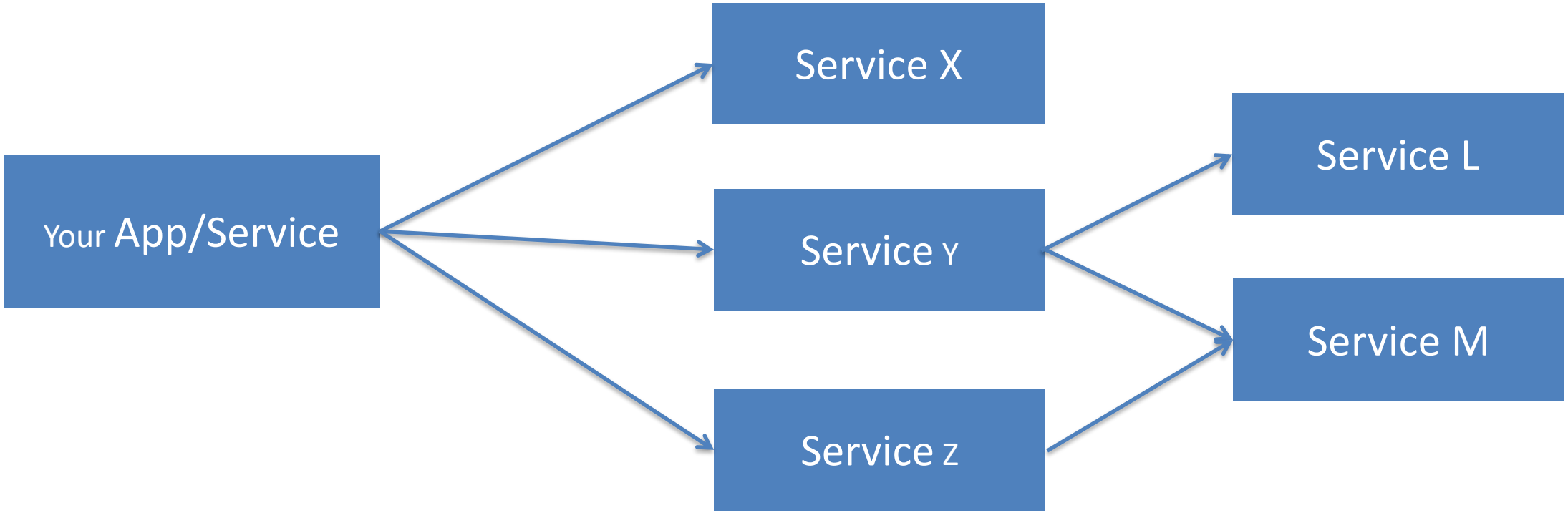
Monolithic Architecture (Revisiting)



Microservices Architecture



Concept -> Service Dependency Graph

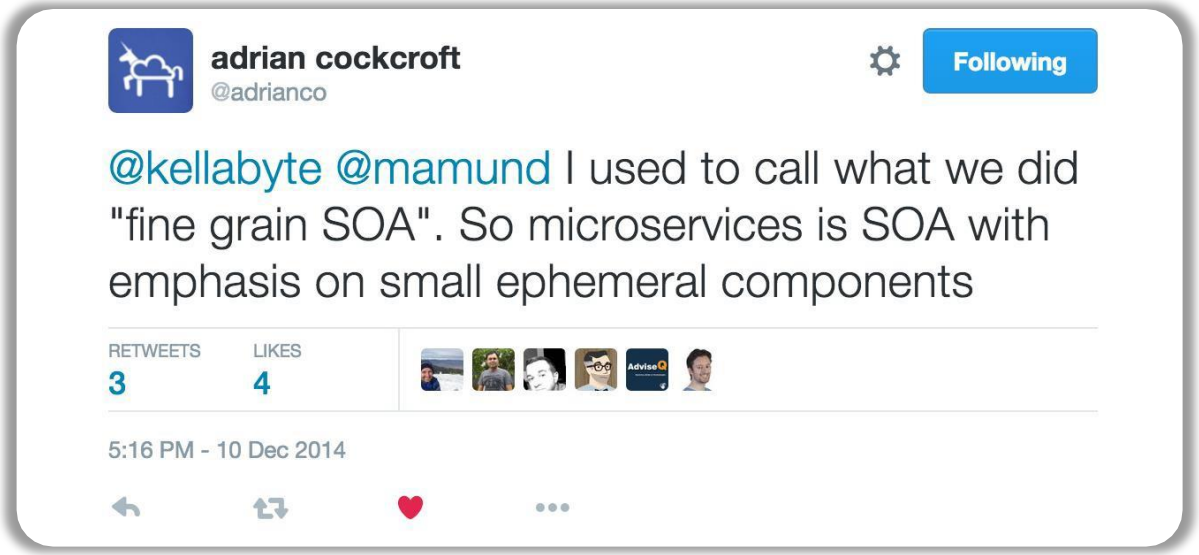


数据表之间的关联比较多的不适合微服务架构

What is a Microservice?

“Loosely coupled service oriented architecture with bounded context”,

– Adrian Cockcroft, April 2015



Functional decomposition of systems into manageable and independently deployable components,

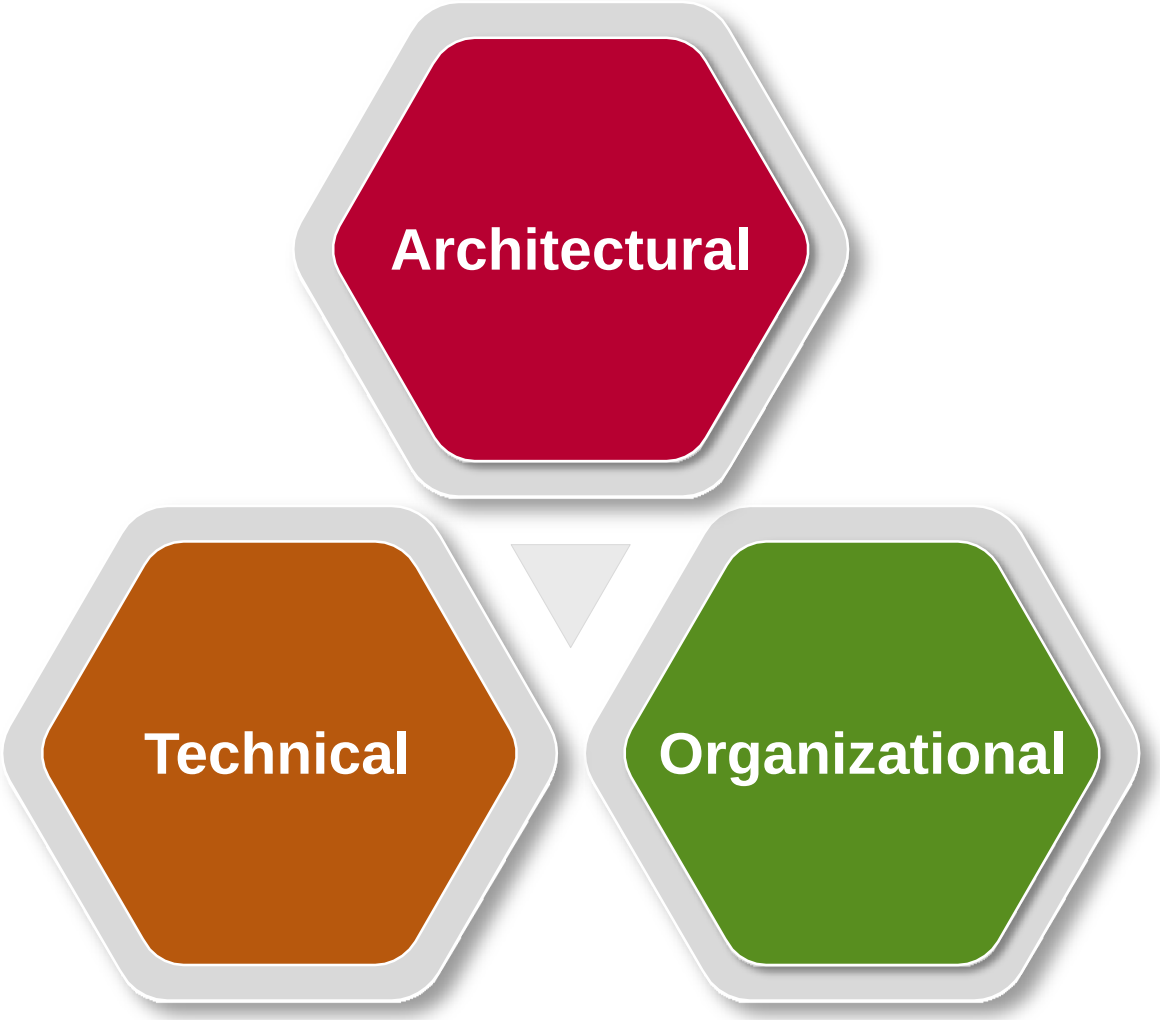
Microservice Architectures by Dr. Andreas Schroeder (http://bit.ly/1TOGZK8)

What is a Microservice?

Related concepts

- Service Oriented Architectures
- API First
- Agile Software Development
- Continuous Delivery
- DevOps

The three aspects of Microservices Architecture

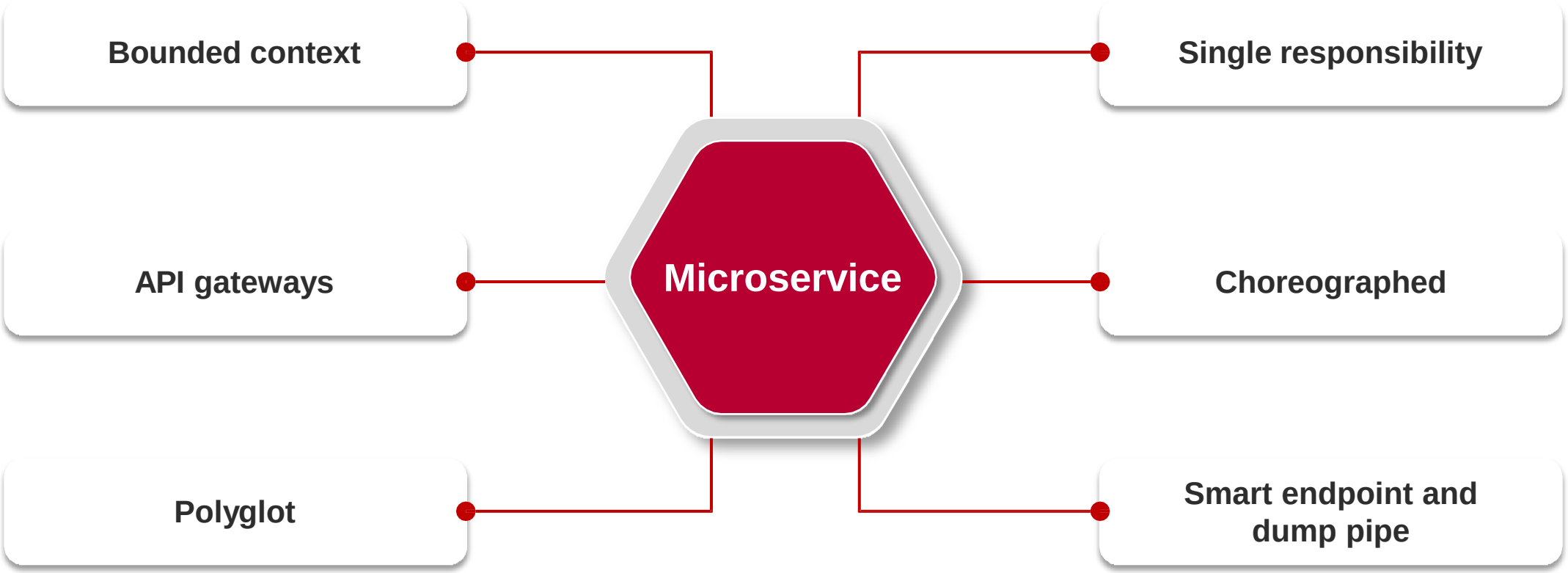


Technical

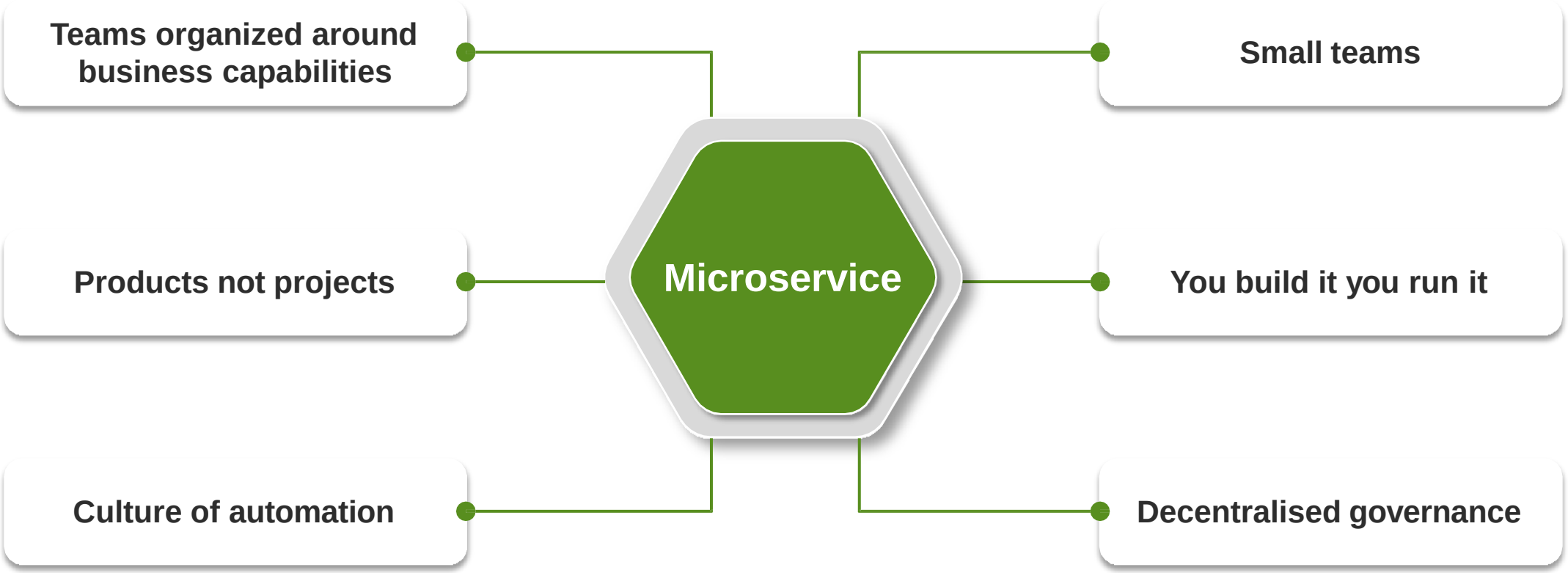
Microservice

- Run on its own process
- Isolates faults
- Is stateless
- Deployed independently
- Scales independently
- Owns its data

Architectural



Organisational

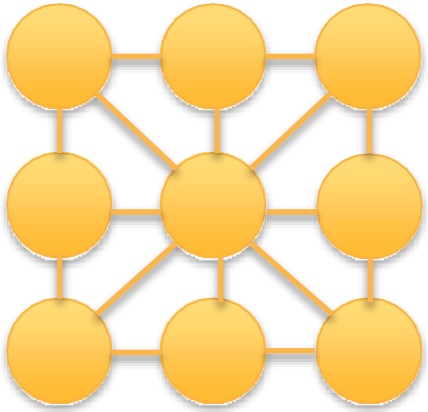


Characteristics of Microservice Architectures

Decentralized

Independent

You build it, you run it



Black Box

Polyglot

Do one thing well

Conway's law

Conway's Law

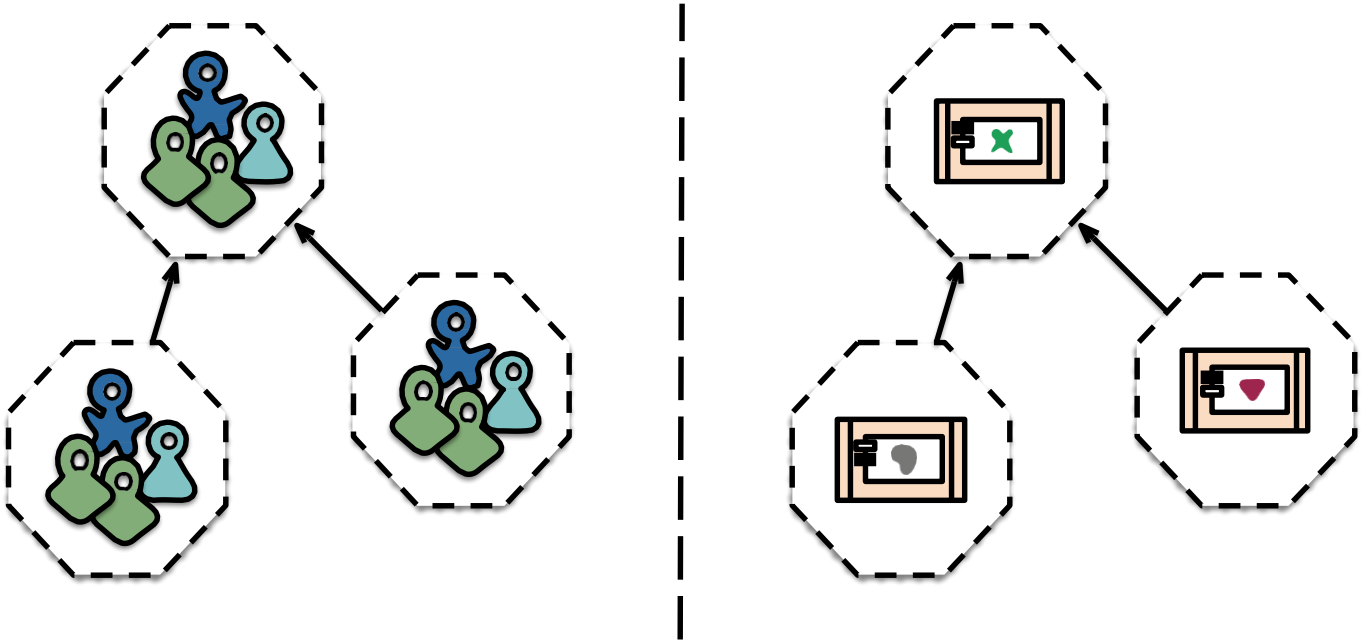
organizations which design systems (in the broad sense used here) are constrained to **produce designs which are copies of the communication structures** of these organizations

Melvin Conway, Datamation, **1968**

http://www.melconway.com/Home/Conways_Law.html

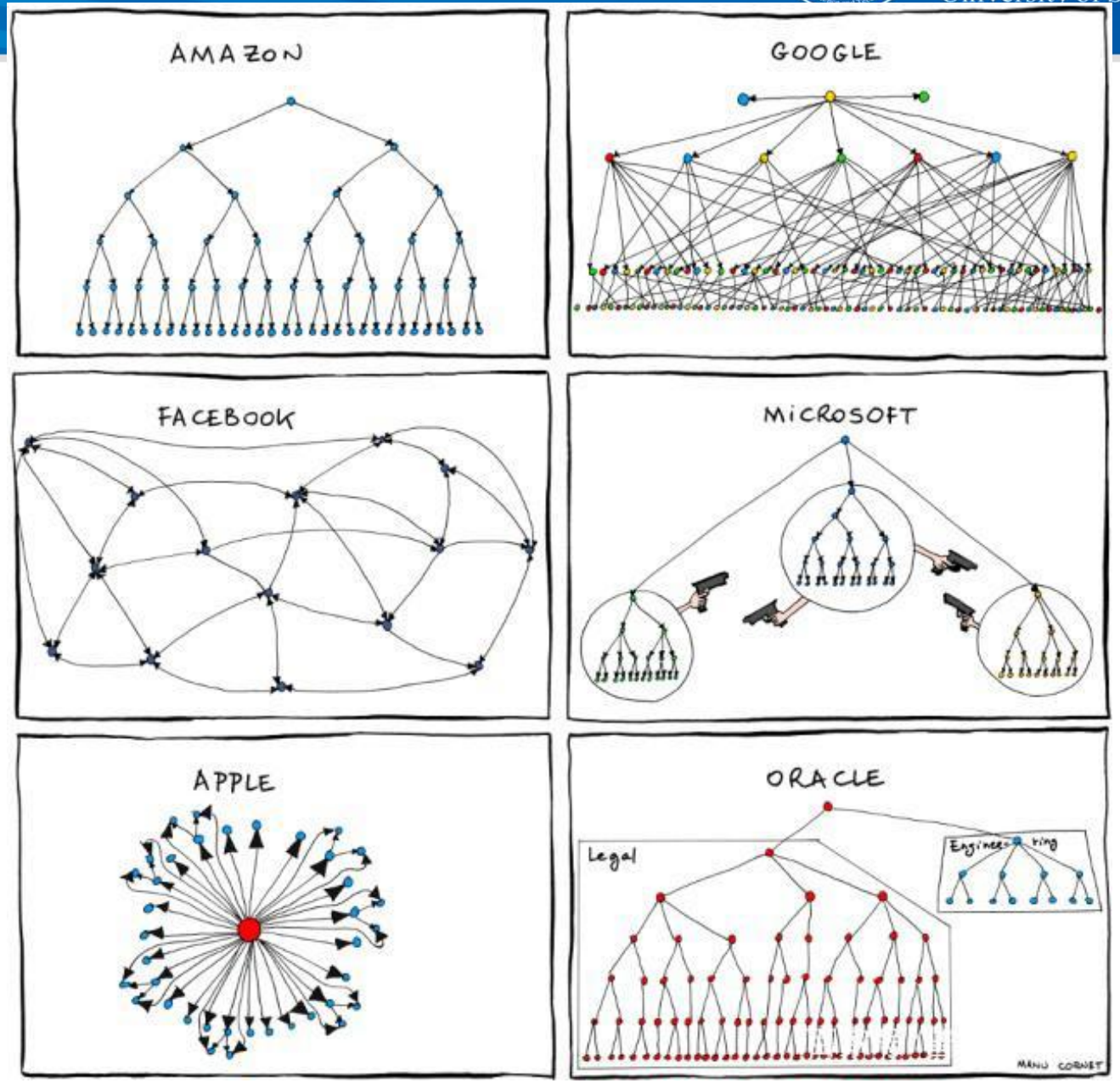
选择的架构与团队的组织有关系

Conway's law



Cross-functional teams...

... organised around capabilities Because Conway's Law


AMAZON
GOOGLE
FACEBOOK
MICROSOFT
APPLE
ORACLE
Legal
Engineering
MANU CORNET

Underlying principle

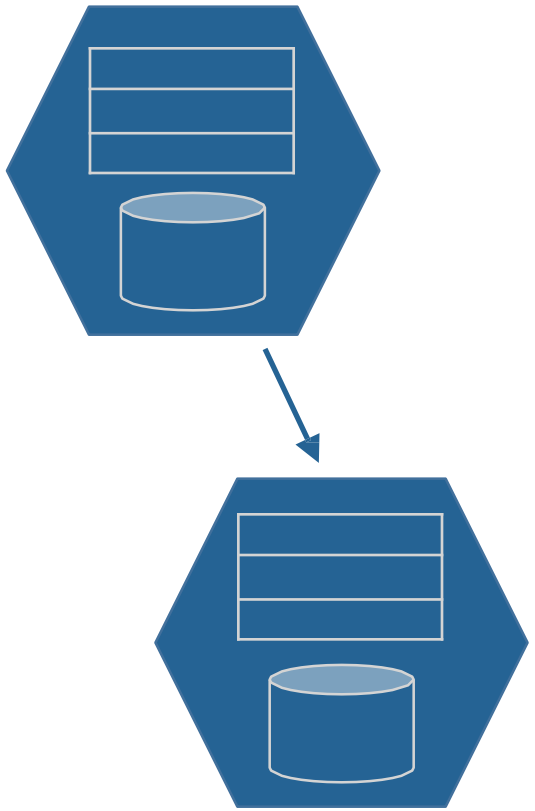
On the logical level, microservice architectures are defined by a

functional system decomposition into manageable and independently deployable components

- The term "micro" refers to the sizing: a microservice must be manageable by a single development team (5-9 developers)
- Functional system decomposition means vertical slicing (in contrast to horizontal slicing through layers)
- Independent deployability implies no shared state and inter-process communication (often via HTTP REST-ish interfaces)

More specifically

- Each microservice is functionally complete with
  - Resource representation
  - Data management
- Each microservice handles one resource (or verb), e.g.
  - Clients
  - Shop Items
  - Carts
  - Checkout



Microservices are *fun-sized* services, as in "still fun to develop and deploy"

Independent Deployability is key

- It enables separation and independent evolution of
  - code base
  - technology stacks
  - scaling
  - and features, too

Independent code base

- Each service has its own software repository
  - Codebase is maintainable for developers - it fits into their brain
  - Tools work fast - building, testing, refactoring code takes seconds
  - Service startup only takes seconds
  - No accidental cross-dependencies between code bases

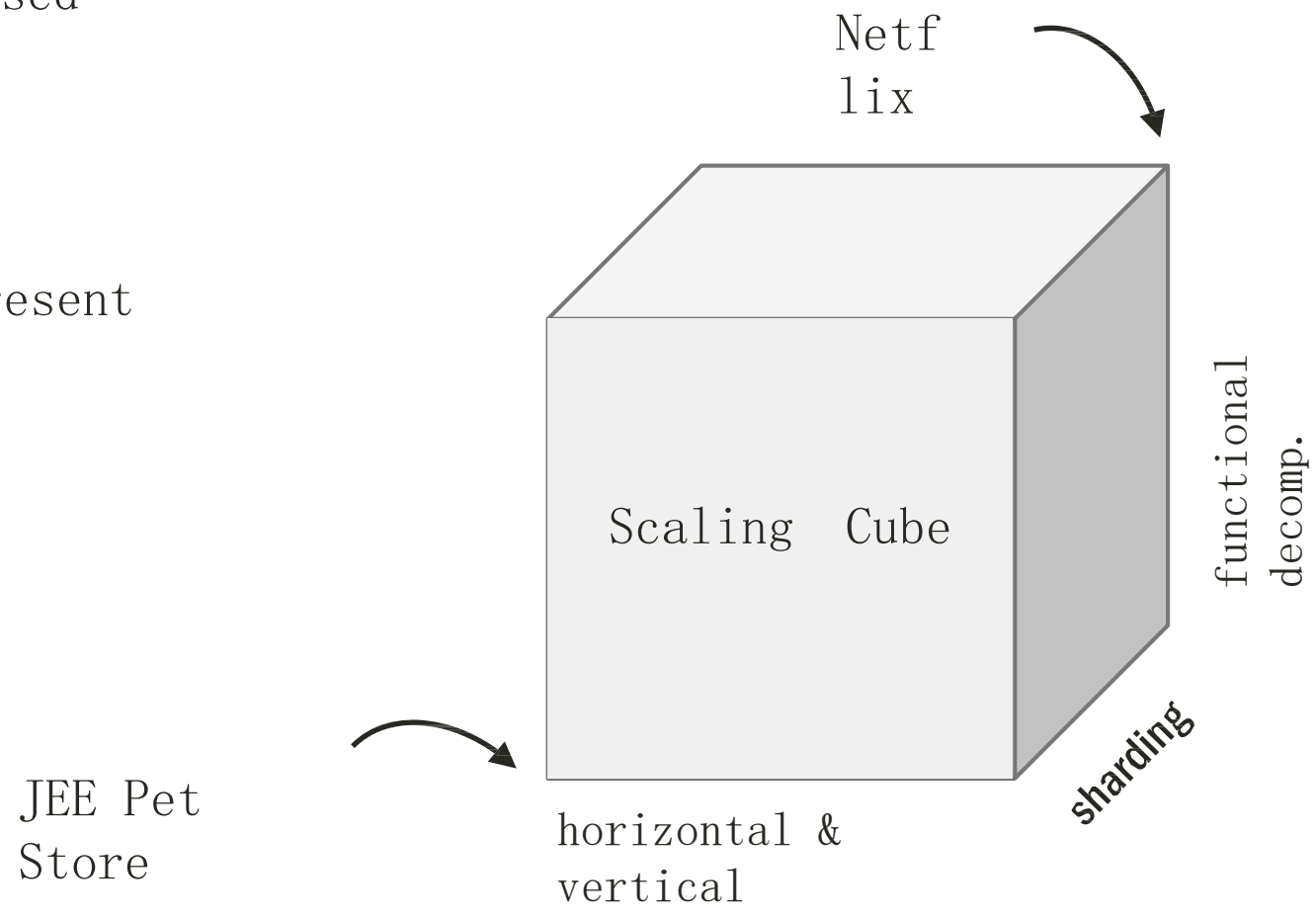
Independent technology stacks

- Each service is implemented on its own technology stacks
  - The technology stack can be selected to fit the task best
  - Teams can also experiment with new technologies within a single microservice
- No system-wide standardized technology stack also means
  - No struggle to get your technology introduced to the canon
  - No piggy-pack dependencies to unnecessary technologies or libraries
  - It ‘s only your own dependency hell you need to struggle with ☺
- Selected technology stacks are often very lightweight
  - A microservice is often just a single process that is started via command line, and not code and configuration that is deployed to a container.

Independent Scaling

Each microservice can be scaled independently

- Identified bottlenecks can be addressed directly
- Data sharding can be applied to microservices as needed
- Parts of the system that do not represent bottlenecks can remain simple and un-scaled



Independent evolution of Features

- Microservices can be extended without affecting other services
  - For example, you can deploy a new version of (a part of) the UI without re-deploying the whole system
  - You can also go so far as to replace the service by a complete rewrite

    But you have to ensure that the service interface remains stable

Comparisons with Precursors

~~Microservices vs SOA~~ Wrong comparison… The difference lies in the realisation style to implement SOA

"The value of the term microservices is that it allows to put a label on a useful subset of the SOA terminology" ,

Martin Fowler (minute 14), GOTO conference, Berlin November 2014

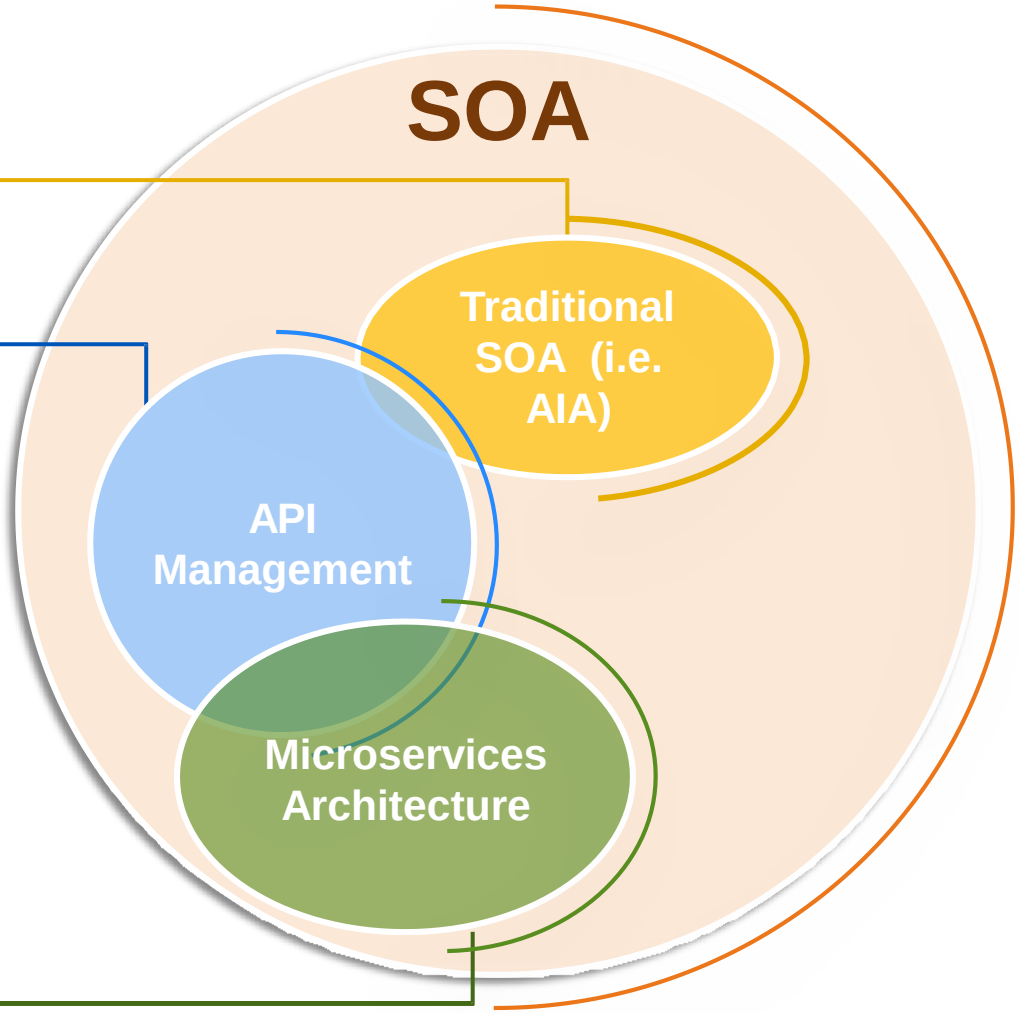
Inspiration from Martin Fowler's Microservices presentation at GOTO conference, Berlin November 2014 (minute 14)

https://www.youtube.com/watch?v=wgdBVIX9ifA

Typically adopted to deliver horizontal integrations

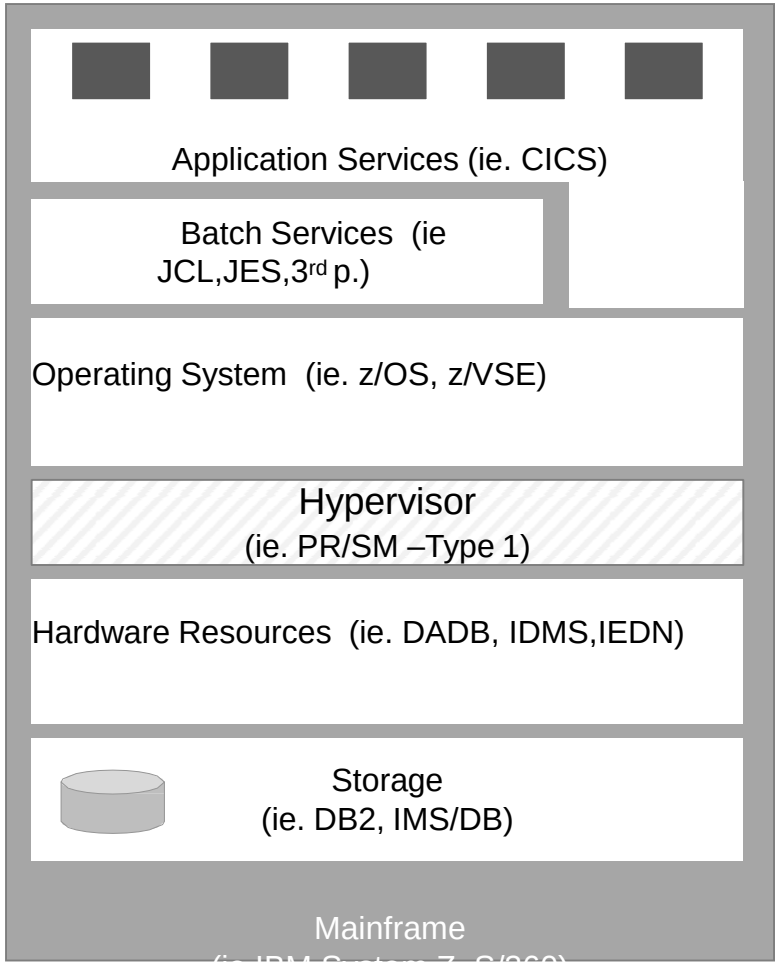
Best for vertical integrations

Not for integration. Best for building modern systems

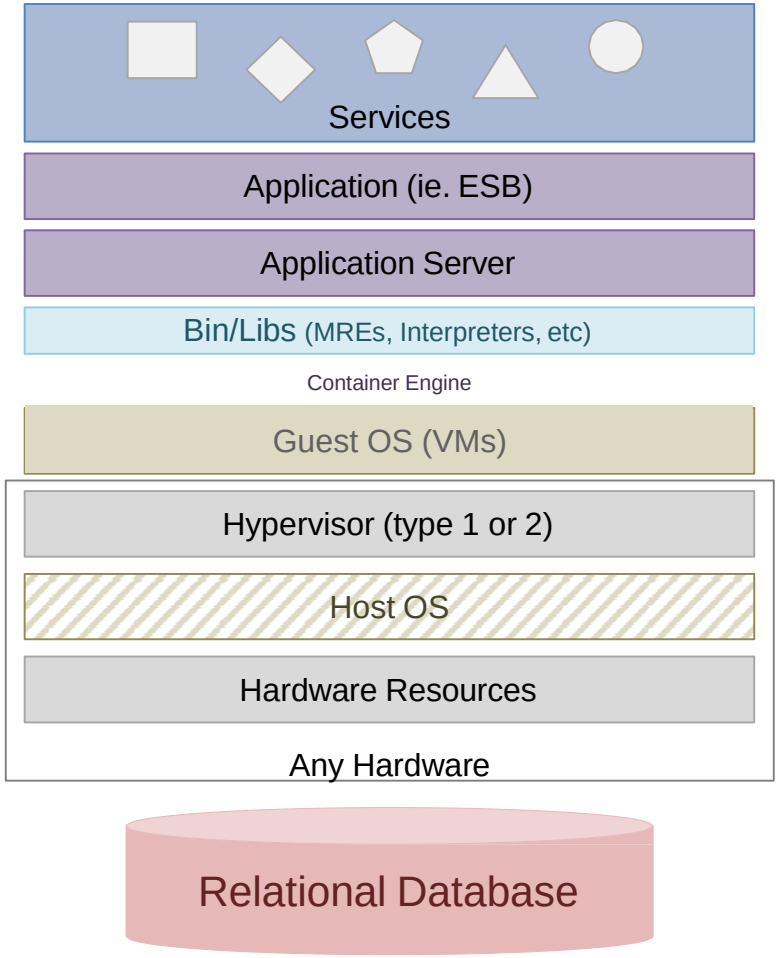


Microservices vs SOA – Technical Stack

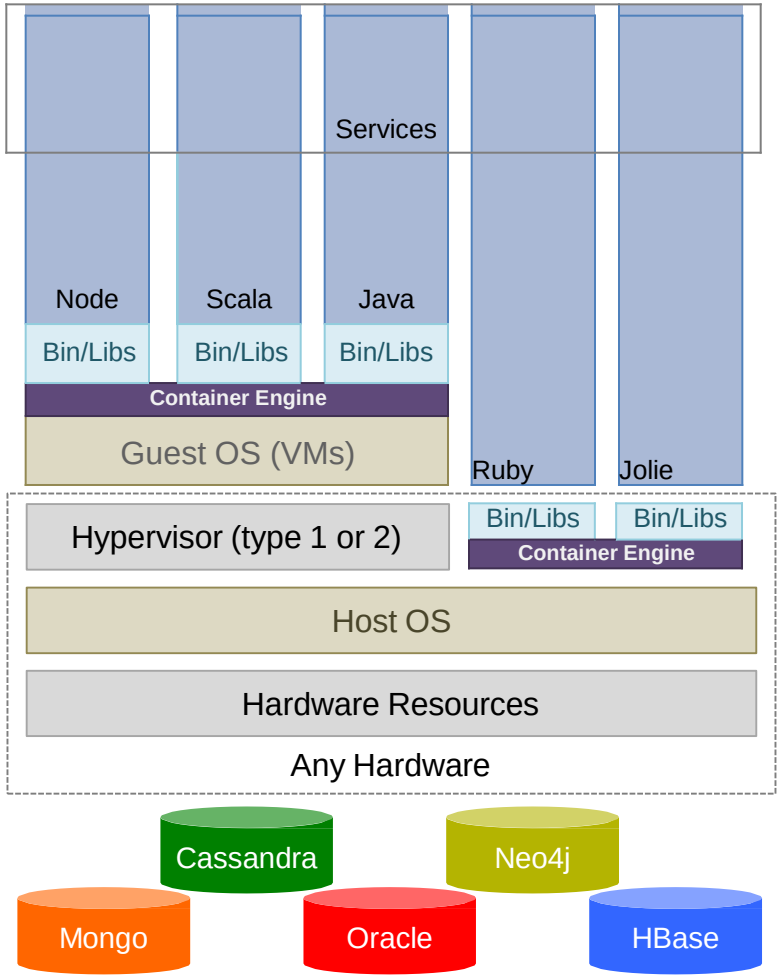
Legacy Monolith



Modern Monolith



Microservice Architecture

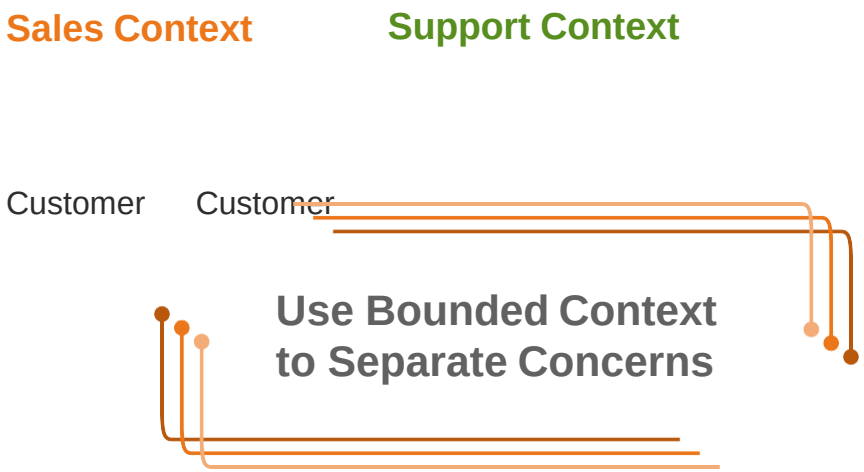


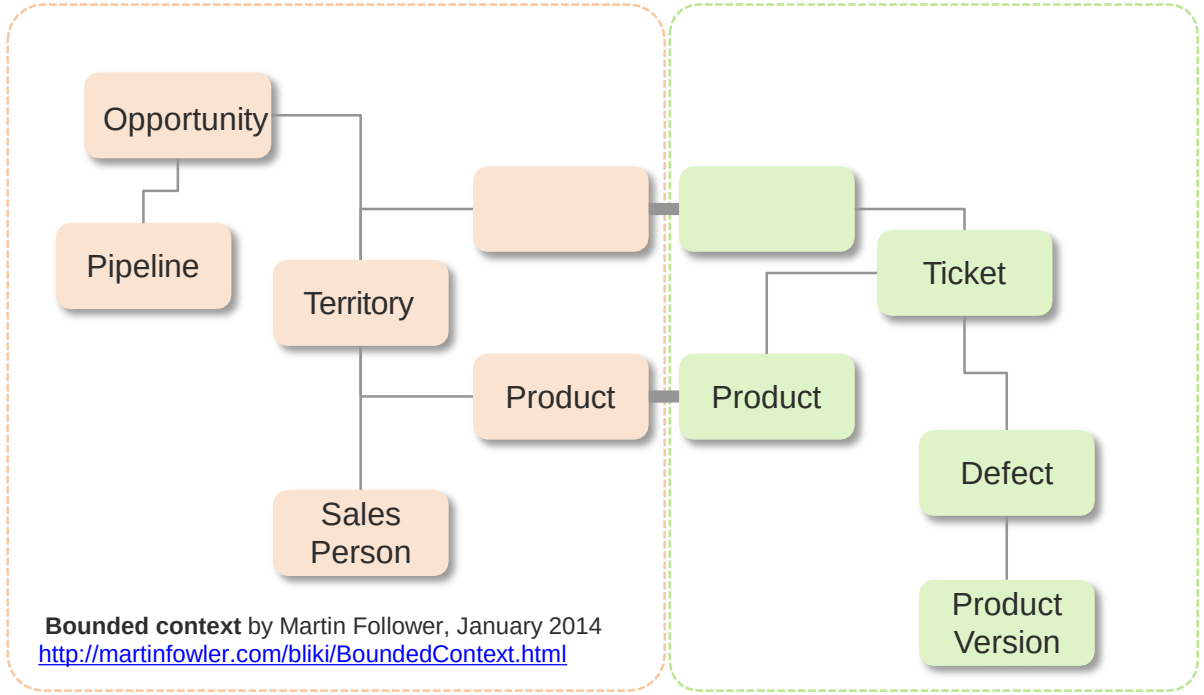
Architectural

Pattern	Traditional SOA	MSA
Monolith pattern (http://bit.ly/1Gjr2Y0)	Yes	No
Polyglot Programming & Persistence (http://bit.ly/18BvDIj & http://bit.ly/1XYiak2)	Not traditionally (use of Suites)	Yes
API gateway pattern (http://bit.ly/1WTyNLJ)	Yes	Yes
Orchestration (http://bit.ly/1U0SWil)	Yes	No
Choreography (http://bit.ly/1ssALZQ)	No	Yes
Event Collaboration (http://bit.ly/25Dk7oE)	Yes	Yes
Canonical Schema (http://bit.ly/1r6KkfK)	Very common	No
Schema centralization (http://bit.ly/1sVlqkc)	Very common	No
Decouple Contract (http://bit.ly/1O8mVpm)	Yes	Could be….
Bounded Context (http://bit.ly/1o7AK8B)	Some times	Yes
Ubiquitous Language (http://bit.ly/1c8nXQe)	Some times	Yes
Bulkhead (http://bit.ly/1c8nXQe)	Not really…	Yes
Tolerant Reader (http://bit.ly/1aa4mr9)	Some times	Yes
Client-side Service Discovery (http://bit.ly/1OunUyq)	Initially only (service registry)	Recommended
Server-side Service Discovery (http://bit.ly/1X3RmzA)	Yes	Yes
ESB Pattern (http://bit.ly/1ZlSKeT)	Yes	Across bounded contexts (dump pipe)

Bounded Context

"Gather together those things that change for the same reason, and separate those things that change for different reasons" – The single responsibility principle by Robert C. Martin, November 2009, http://bit.ly/1VDgw79

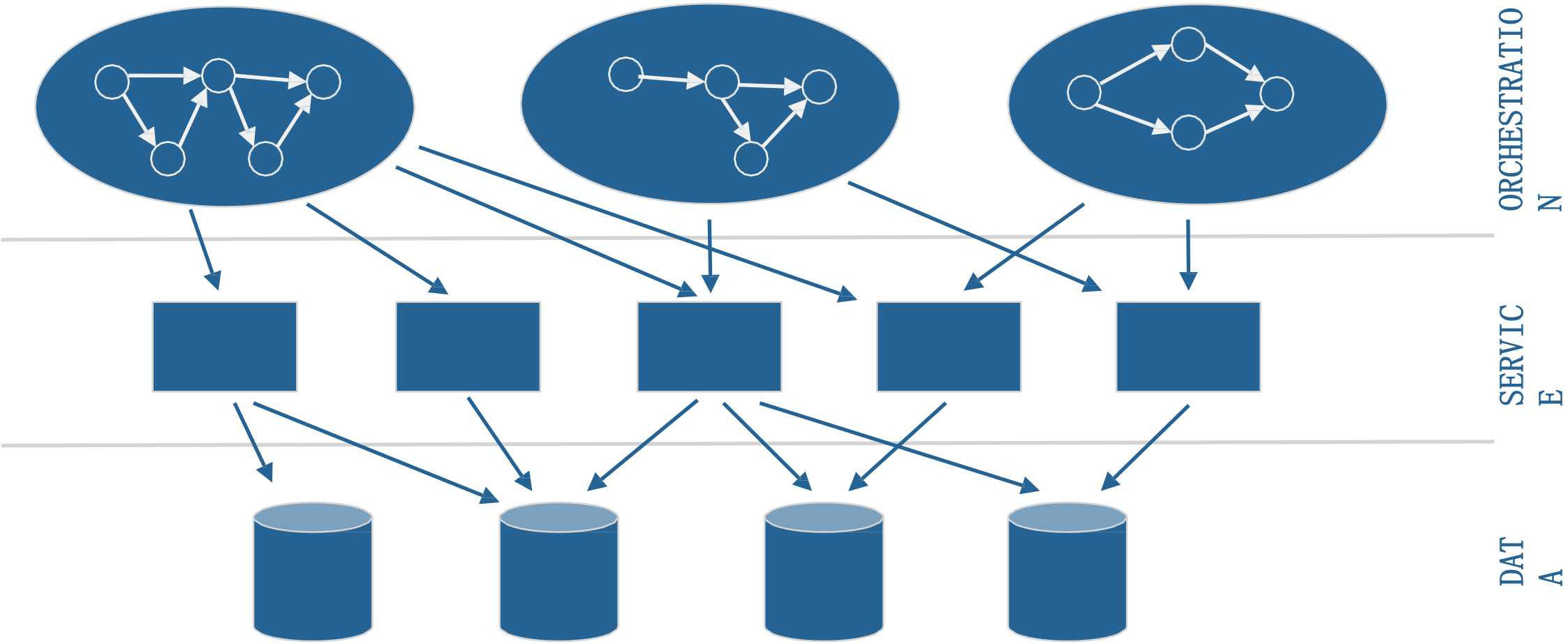




Bounded context by Martin Follower, January 2014
http://martinfowler.com/bliki/BoundedContext.html

"Domain driven design (DDD) divides up a large system into Bounded Contexts, each of which can have a unified model – essentially a way of structuring Multiple Canonical Models."

Service-Oriented Architecture



Service-Oriented Architecture

- SOA systems also focus on functional decomposition, but
  - services are not required to be self-contained with data and UI, most of the time the contrary is pictured.
  - It is often thought as decomposition within tiers, and introducing another tier - the service orchestration tier
- In comparison to microservices
  - SOA is focused on enabling business-level programming through business processing engines and languages such as BPEL and BPMN
  - SOA does not focus on independent deployment units and its consequences
  - Microservices can be seen as "SOA - the good parts"

Component-Based Software Engineering

- Underlying functional decomposition principle of microservices is basically the same. Additionally, the following similarities and differences exist:

- State model
  - Many theoretical component models follow the share-nothing model
- Communication model
  - Component technologies often focus on simulating in-process communication across processes (e.g. Java RPC, OSGi, EJB)
  - Microservice communication is intra-process, serialization-based
- Code separation model
  - Component technologies do require code separation
  - Components are often developed in a common code repository
- Deployment model
  - Components are often thought as being deployed into a uniform container

MicroService Implemention

Sample Microservice Design Canvas

Service Name: Transaction Search Service

Description: The Transaction Search Service allows consumers to find specific transactions that fit a set of specified criteria. This criteria can range from date/time, to spending category, to amounts, and more.

Consumer Tasks

Banking Customer using Online Banking Web or Mobile App
- Search transactions
- Get transaction details

Banking CSR using Branch Banking or Call Center App
- Search transactions
- Get transaction details

Interface

Queries
- Query customer transactions
- Get transaction details

Commands
- Remove transaction

Event Subscriptions
- Add transaction
- Add transaction set

Dependencies

Event Publications

Product-specific Transaction Service
- Query transactions

Qualities

- Mostly read only, except for transaction removal
- Medium volume service , not mission critical
- Direct customer and delegated authentication
- Not transactional

Logic/Rules

- Populate aggregated transaction store through event listening (event sourcing)
- Filter or adjust data query based on input parameters
- Offer idempotent transaction removal command

Data

- Aggregated transaction store

More here: http://www.apiacademy.co/the-microservice-design-canvas/

Partition of Micro Services

- The principle of partition
  - After micro-service partitioning, the system must be stable.
  - Microservices must be small enough to be independently developed and tested by small development teams.
  - Micro-services must have a single function. Thus, the new functions and requirements of the application layer of the system almost affect the single micro-services, which reduces the inefficiency caused by multi-team collaborative development.
  - Compliance with Closure Principle in OOD.

Partition of Micro Services

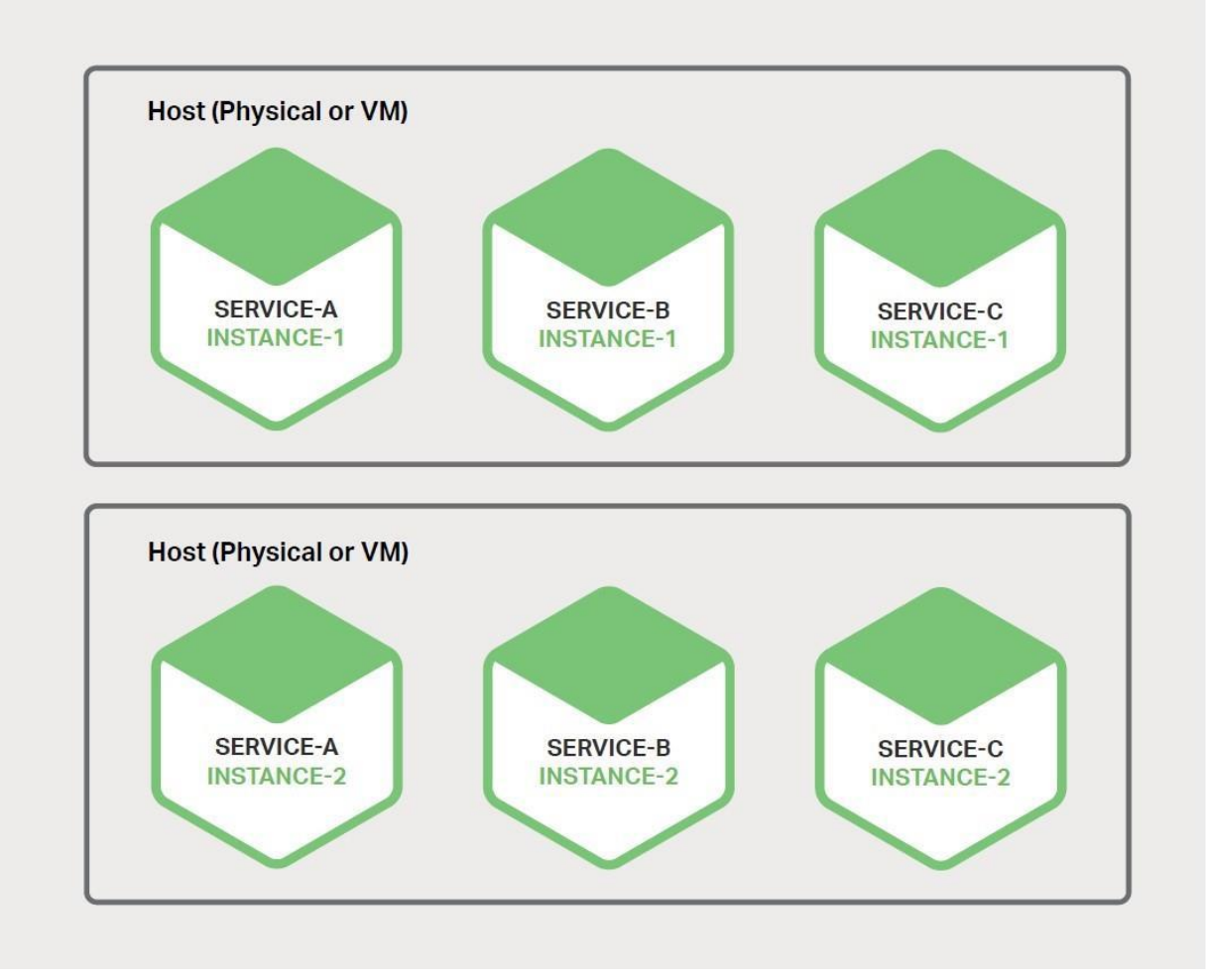
- Partition granularity
  - Monotony
  - integrity

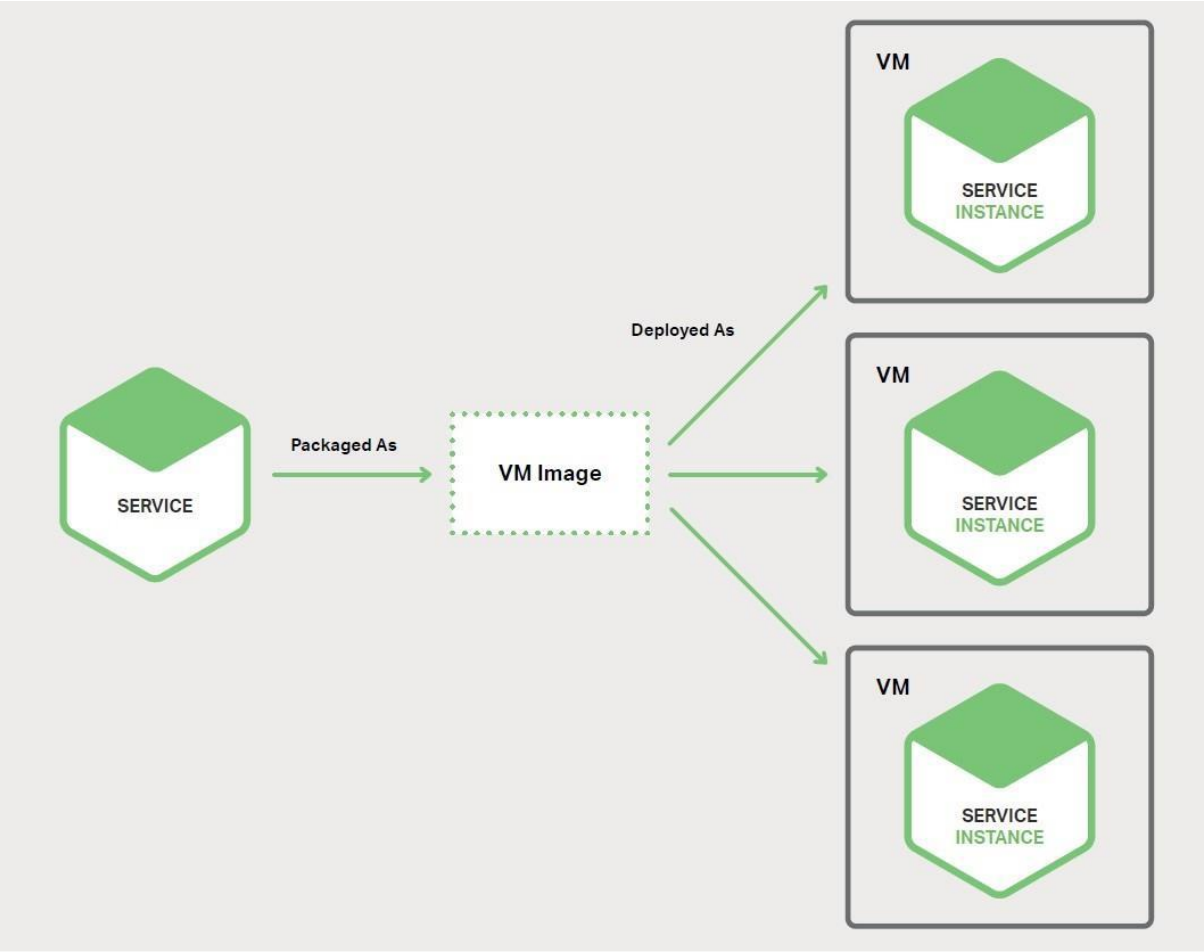
Partition of Micro Services

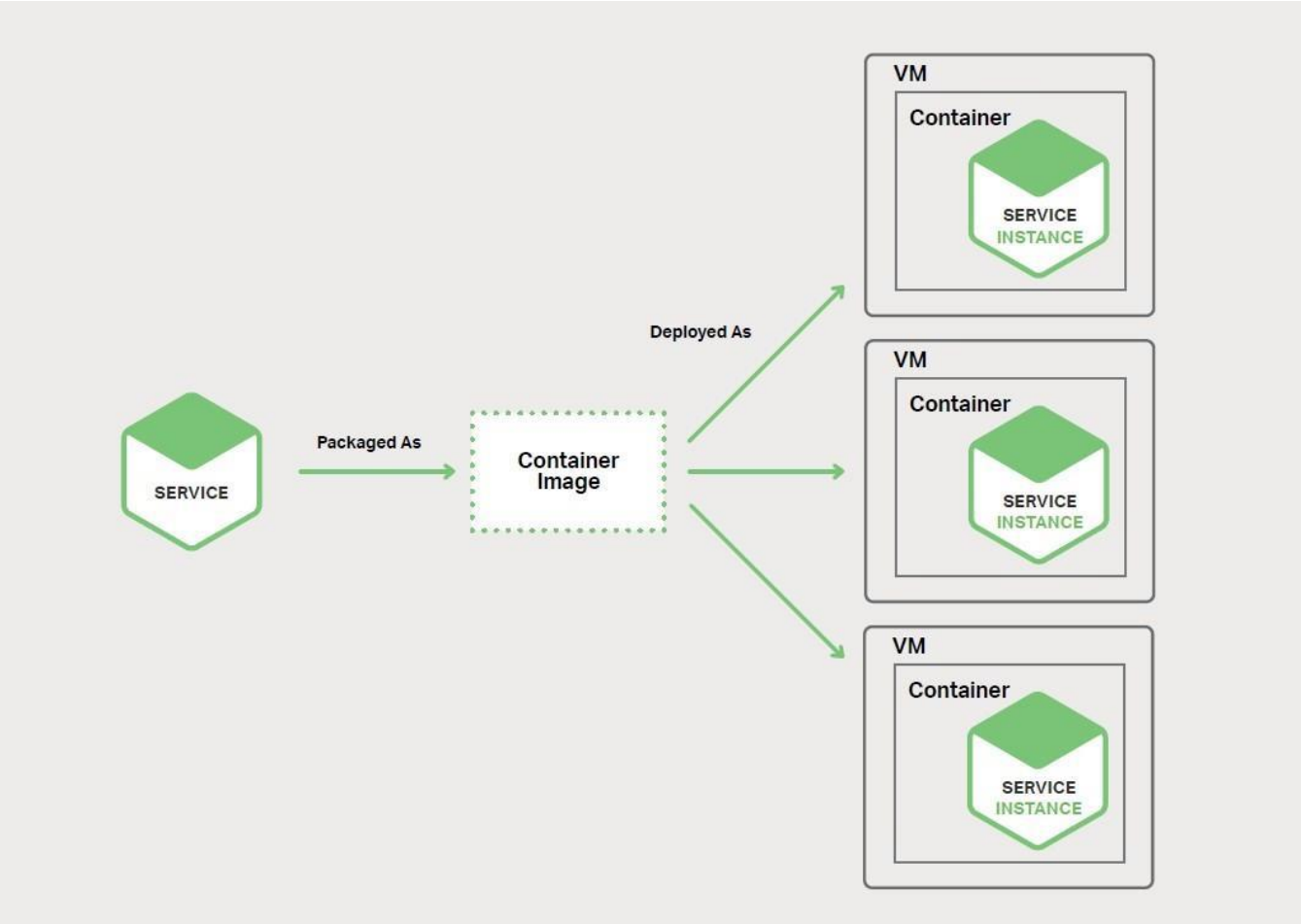
- Partition method
  - According to business capability. Thus, Micro-services divided by business capability correspond to each functional module in the program.
  - According to business subdomains. The sub-domain can be understood as all the problem sub-domains that the system needs to solve.
  - In particular cases, business capabilities and sub-domains are basically the same, representing different system partitioning perspectives.

Deploying Micro Services

- Multiple Service Instances per Host




SERVICE
Packaged As
VM Image
Deployed As
VM
SERVICE
INSTANCE
VM
SERVICE
INSTANCE
VM
SERVICE
INSTANCE


SERVICE
Packaged As
Container Image
Deployed As
VM
Container
SERVICE INSTANCE
VM
Container
SERVICE INSTANCE
VM
Container
SERVICE INSTANCE

Focus on Migration Strategies

From a Monolithic to a Microservices Architecture

Migration Strategy: Use Incremental Refactoring

- The process of transforming a monolithic EIP into a series of microservices is a modernization effort for both the application and its governance.
- This migration is a perfect fit for incremental refactoring. It should not be approached as a complete application rewrite, often called “big bang.”
- Incremental refactoring enables the development team to build experience with microservices
- Incremental refactoring allows teams to build experience with the service extraction process

“Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code, yet improves its internal structure."

Martin Fowler

Migration – Choose a Few Services to Extract from the Monolithic Application

- Monolithic applications often consist of hundreds of modules
- Determine which modules to extract first. Choose modules that are
  - **Easy to extract**
  - **change frequently**
  - **have significantly different resource requirements compared to other modules**
  - **Do not share resource requirements**
  - **implement computationally expensive algorithms.**

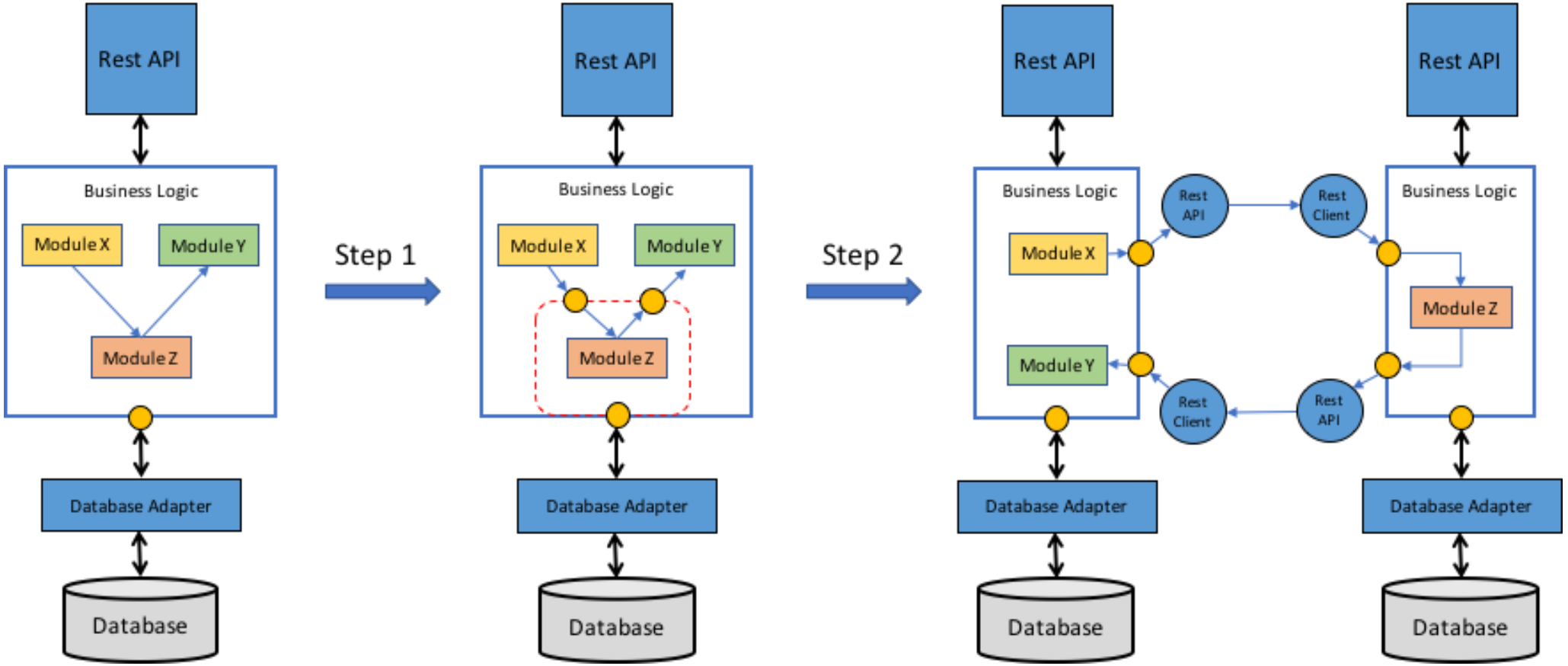
When migrating from monolithic to microservices, using incremental refactoring is akin to safely servicing your car when driving 70mph on the highway, and stopping the car is not an option.

Migration – How to Extract a Module

- Step 1 – define a coarse-grained interface between the module and the monolith
  - Consider creating a bidirectional API
  - Pay close attention to the complexities of business logic refactoring. Significant code changes may be needed to break dependencies
- Step 2 – Turn the module into a free-standing service
  - Write code for the monolith and service to communicate through an API mechanism
  - Combine module with a microservice framework that handles cross-cutting concerns such as service discovery

Over time, as modules continue to be extracted into services, the amount of functionality implemented by the monolithic application shrinks until either it disappears entirely or it becomes just another microservice

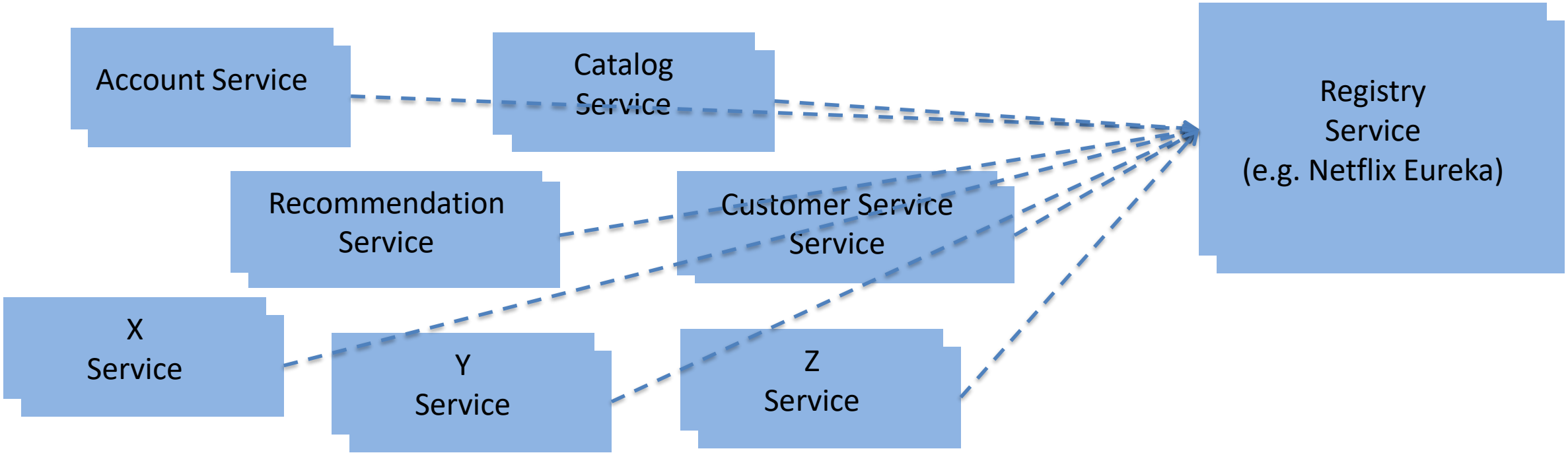
Illustration of Monolithic Module Refactoring



MicroServices - Challenges

Service Discovery

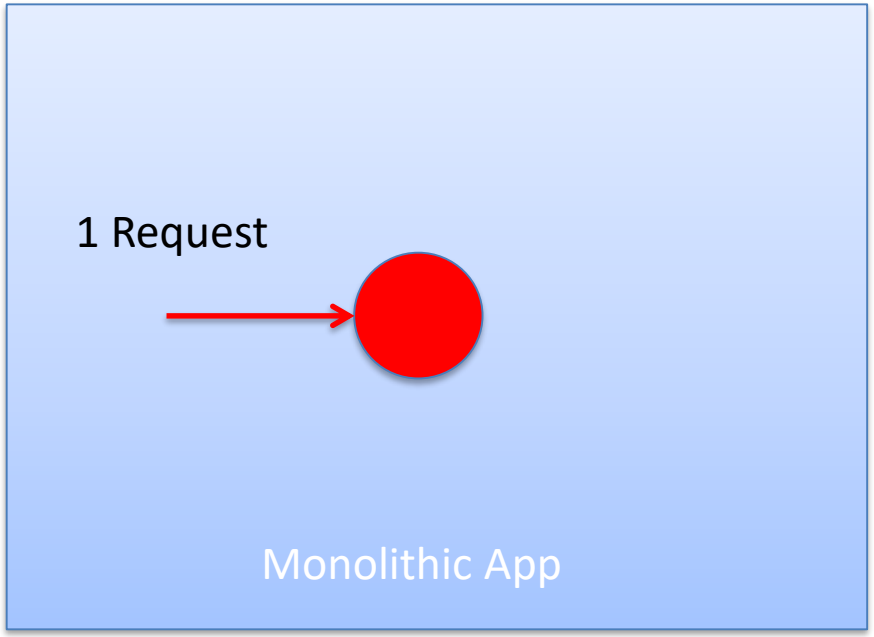
- 100s of MicroServices
  - Need a Service Metadata Registry (Discovery Service)

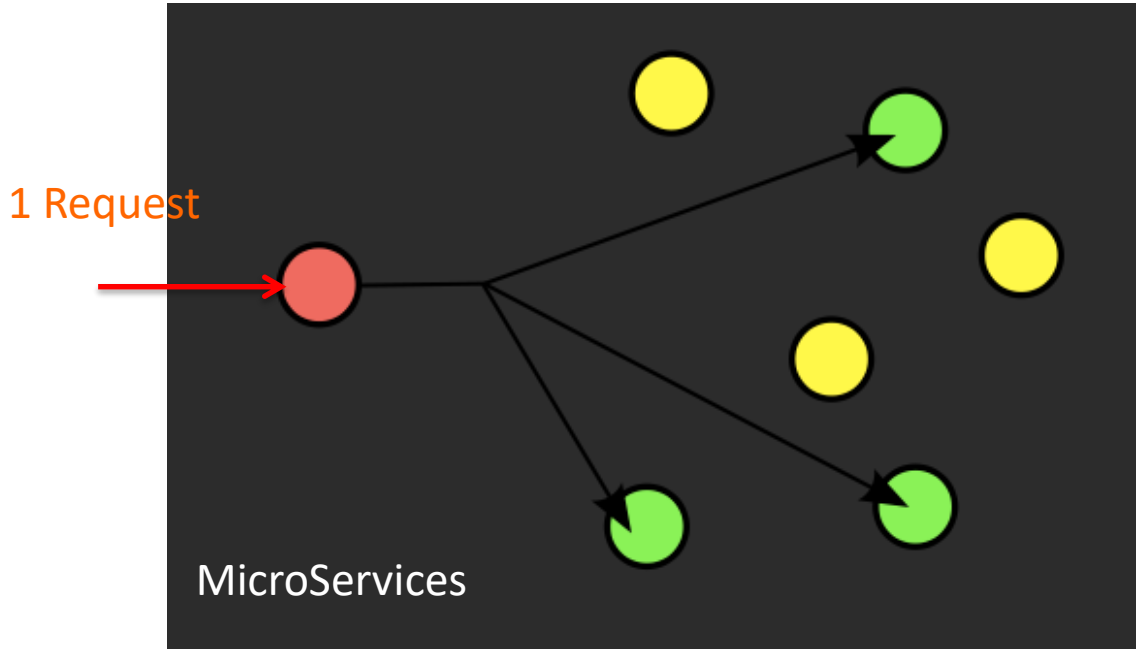


Chattiness (and Fan Out)

~2 Billion Requests per day on Edge Service

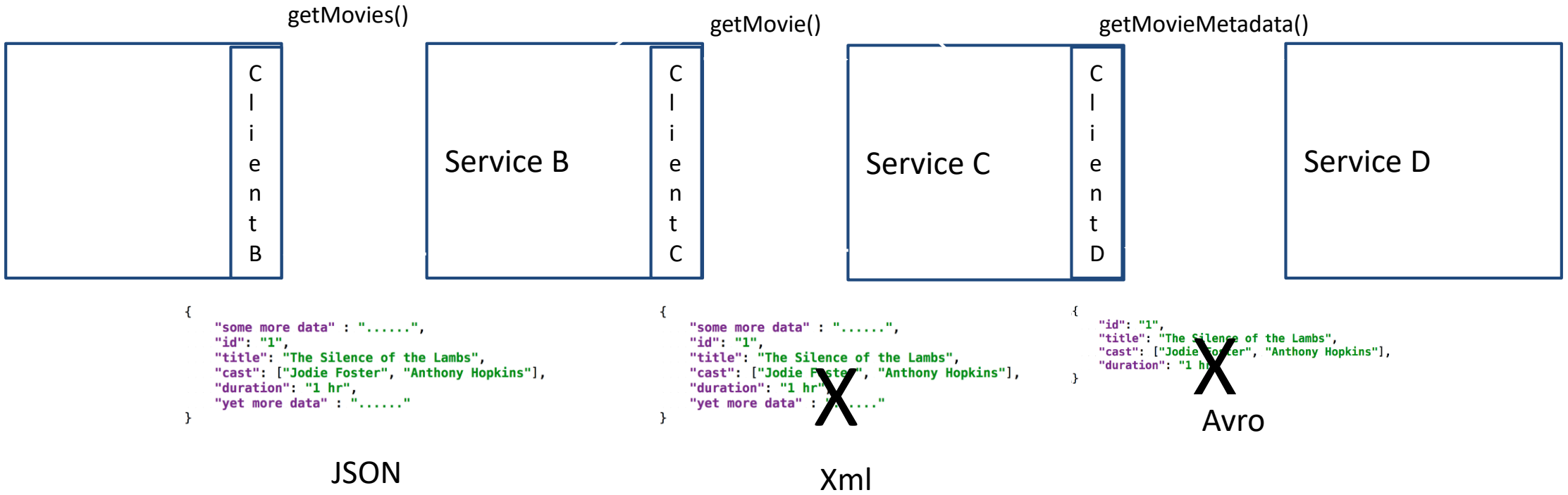
Results in ~20 Billion Fan out requests in ~100 MicroServices





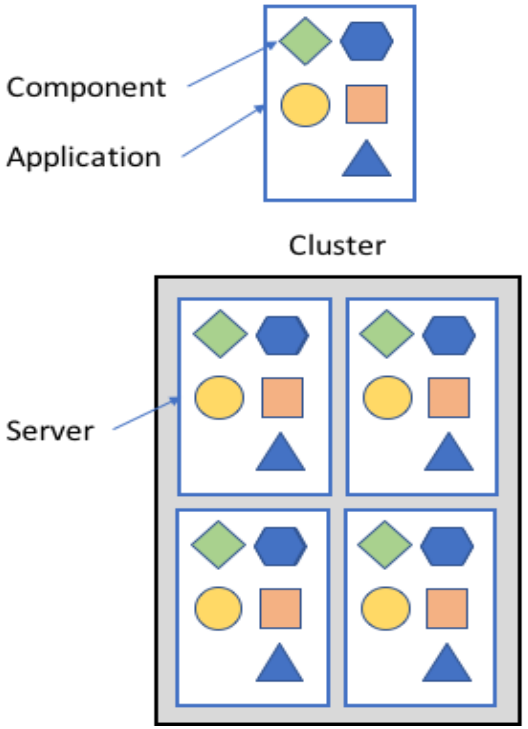
Data Serialization Overhead

Data transformation

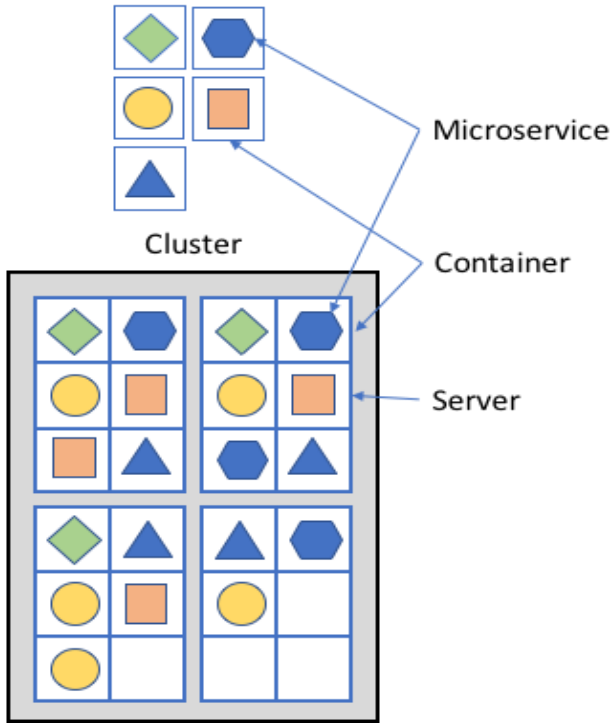


Visual Interpretation of Architecture Patterns

Traditional Monolithic Architecture Pattern



Microservice Architecture Pattern



"Monoliths and microservices are not a simple binary choice.

Both are fuzzy definitions that mean many systems would lie in a blurred boundary area among the two"

-Martin Fowler

Martinfowler.com

Microservices Prerequisites

Before applying microservices, you should have in place

- Rapid provisioning
  - Dev teams should be able to automatically provision new infrastructure
- Basic monitoring
  - Essential to detect problems in the complex system landscape
- Rapid application deployment
  - Service deployments must be controlled and traceable
  - Rollbacks of deployments must be easy

Source
http://martinfowler.com/bliki/MicroservicePrerequisites.html

Evolving interfaces correctly

- Microservice architectures enable independent evolution of services – but how is this done without breaking existing clients?
- There are two answers
  - Version service APIs on incompatible API changes
  - Using JSON and REST limits versioning needs of service APIs
- Versioning is key
  - Service interfaces are like programmer APIs – you need to know which version you program against
  - As service provider, you need to keep old versions of your interface operational while delivering new versions
- But first, let’s recap compatibility

API Compatibility

There are two types of compatibility

- Forward Compatibility
  - Upgrading the service in the future will not break existing clients
  - Requires some agreements on future design features, and the design of new versions to respect old
    interfaces
- Backward Compatibility
  - Newly created service is compatible with old clients
  - Requires the design of new versions to respect old interfaces

The hard type of compatibility is forward compatibility!

Forward compatibility through REST and JSON

REST and JSON have a set of inherent agreements that benefit forward compatibility

- JSON: only validate for what you really need, and ignore unknown object fields (i.e. newly introduced ones)
- REST: HATEOAS links introduce server-controlled indirection between operations and their URIs

```
{ "number" : 12345,
  ...
  "links" : [ {
    "rel" : "deposit",
    "href" : "https://bank.com/account/12345/deposit"
  } ]
}
```

```
"https://accounts.bank.com/12345/deposit"
```

Compatibility and Versioning

- Compatibility can't be always guaranteed, therefore versioning schemes (major.minor.point) are introduced
  - Major version change: breaking API change
  - Minor version change: compatible API change
- Note that versioning a service imposes work on the service provider
  - Services need to exist in their old versions as long as they are used by clients
  - The service provider has to deal with the mapping from old API to new API as long as old clients exist

Conclusion

Microservices: just …?

- Just adopt?
  - No. Microservices are a possible design alternative for new web systems and an evolution path for existing web systems.
  - There are considerable amounts of warnings about challenges, complexities and prerequisites of microservices architectures from the community.

- Just the new fad?
  - Yes and no. Microservices is a new term, and an evolution of long-known architectural principles applied in

a specific way to a specific type of systems.

  - The term is dev and ops-heavy, not so much managerial.
  - The tech landscape is open source and vendor-free at the moment.

Summary

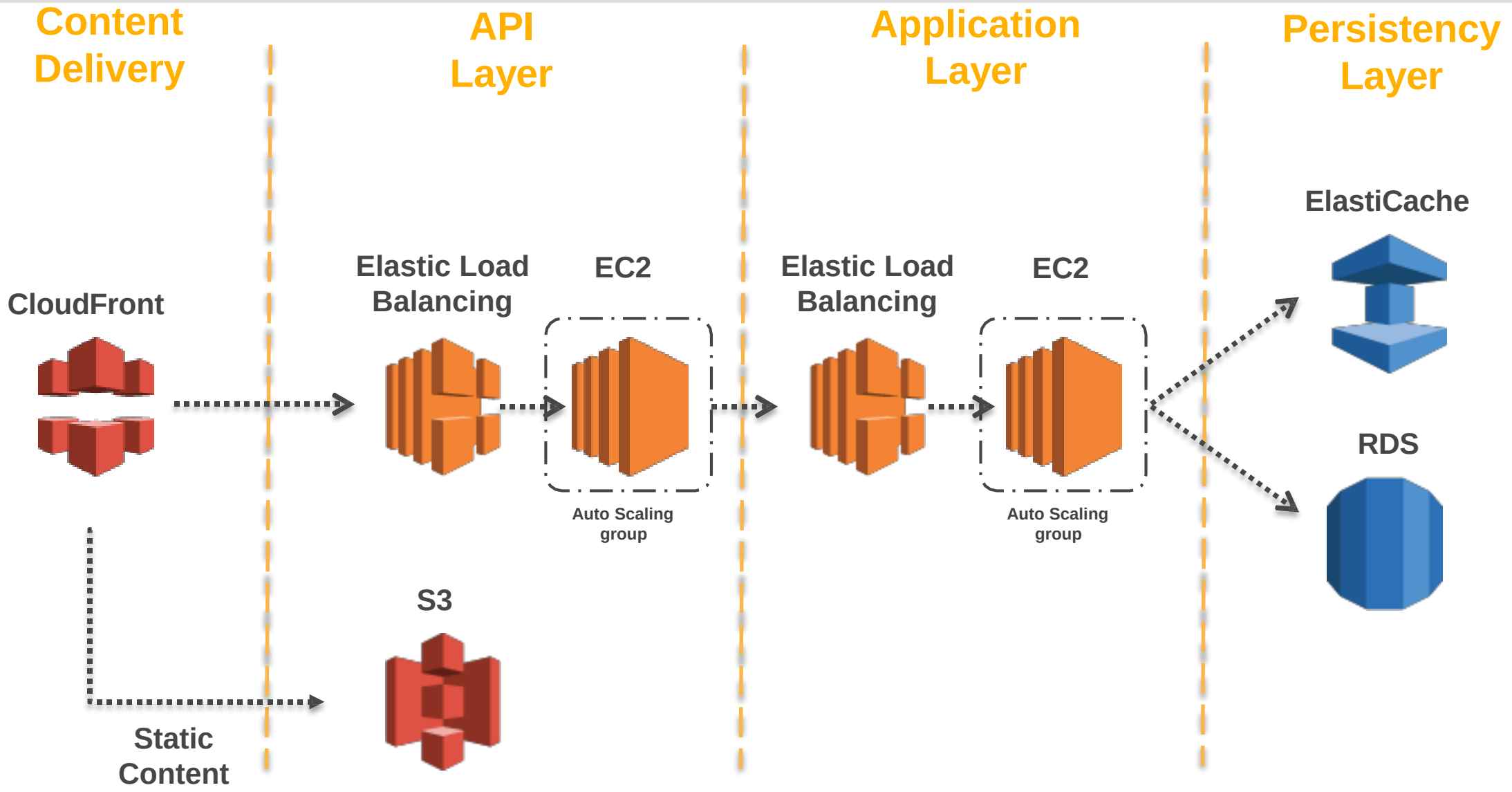
- There is an alternative to software monoliths
- Microservices: functional decomposition of systems into manageable and independently deployable services
- Microservice architectures means
  - Independence in code, technology, scaling, evolution
  - Using battle-tested infrastructure (HTTP, JSON, REST)
- Microservice architectures are challenging
  - Compatibility and versioning while changing service interfaces
  - … transactions, testing, deploying, monitoring, tracing is/are harder

Microservices are no silver bullet, but may be the best way forward for

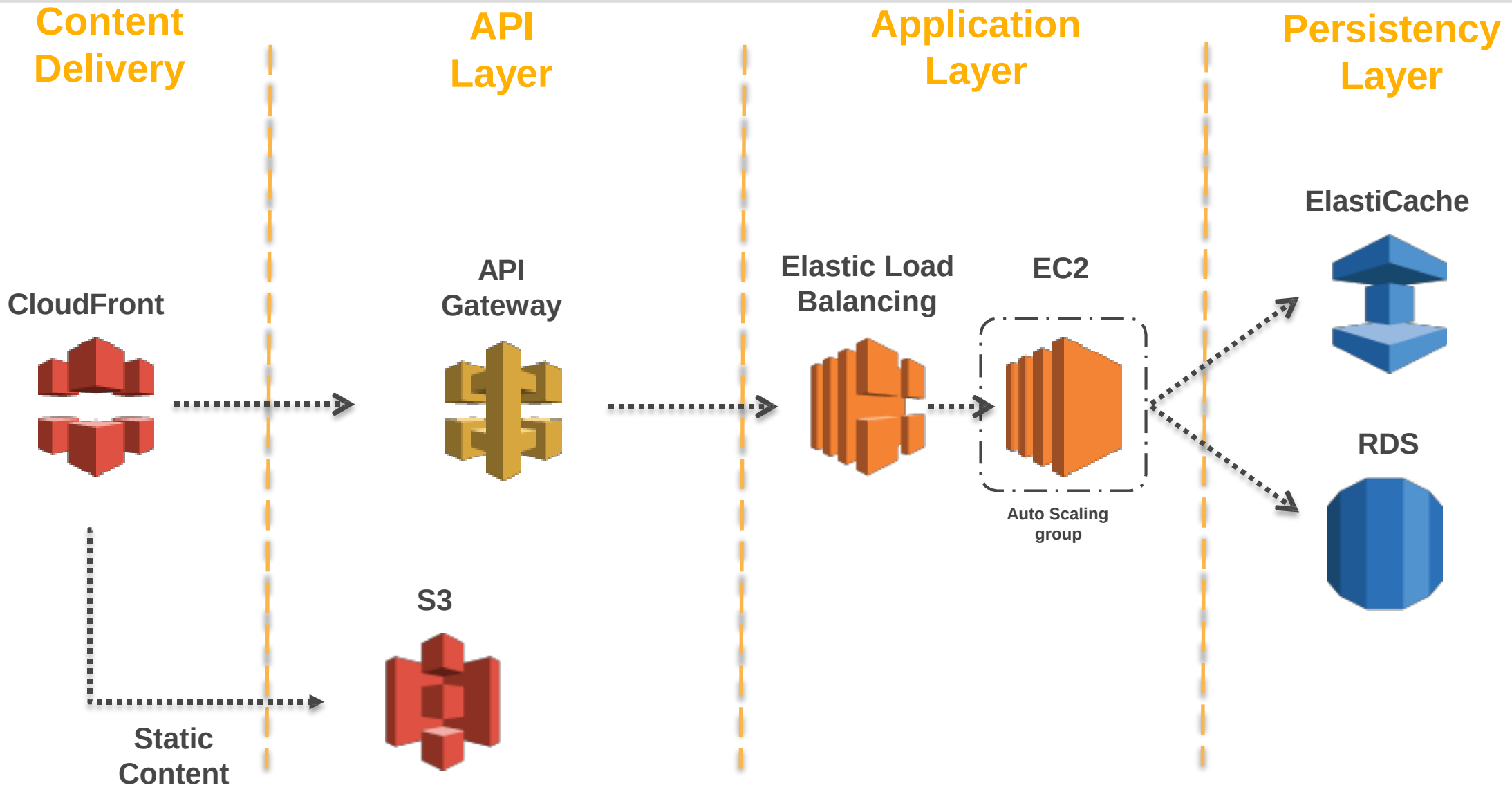
- large web systems
- built by professional software engineers

Microservice Architectures examples

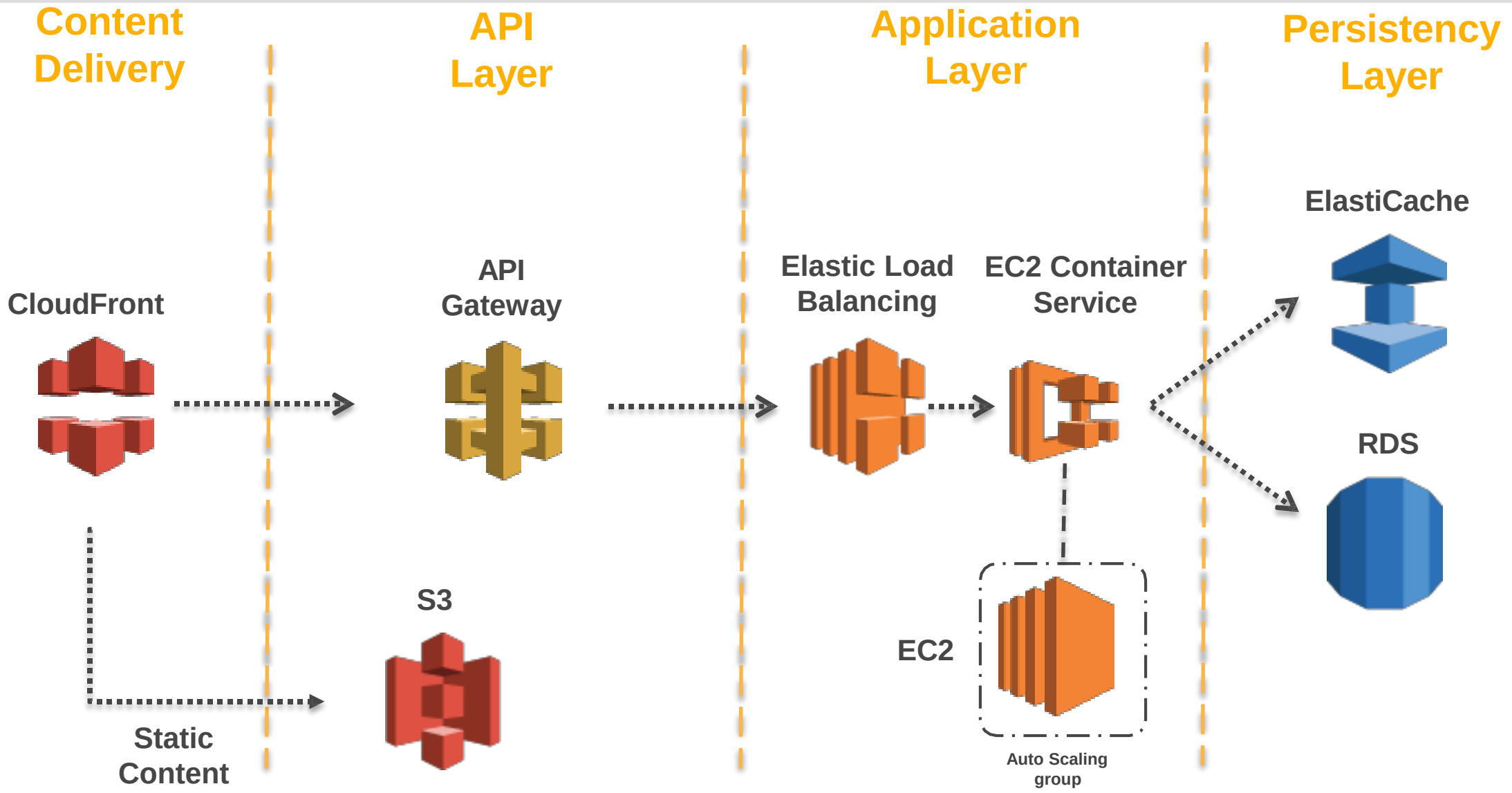
A Typical Microservice Architecture on AWS



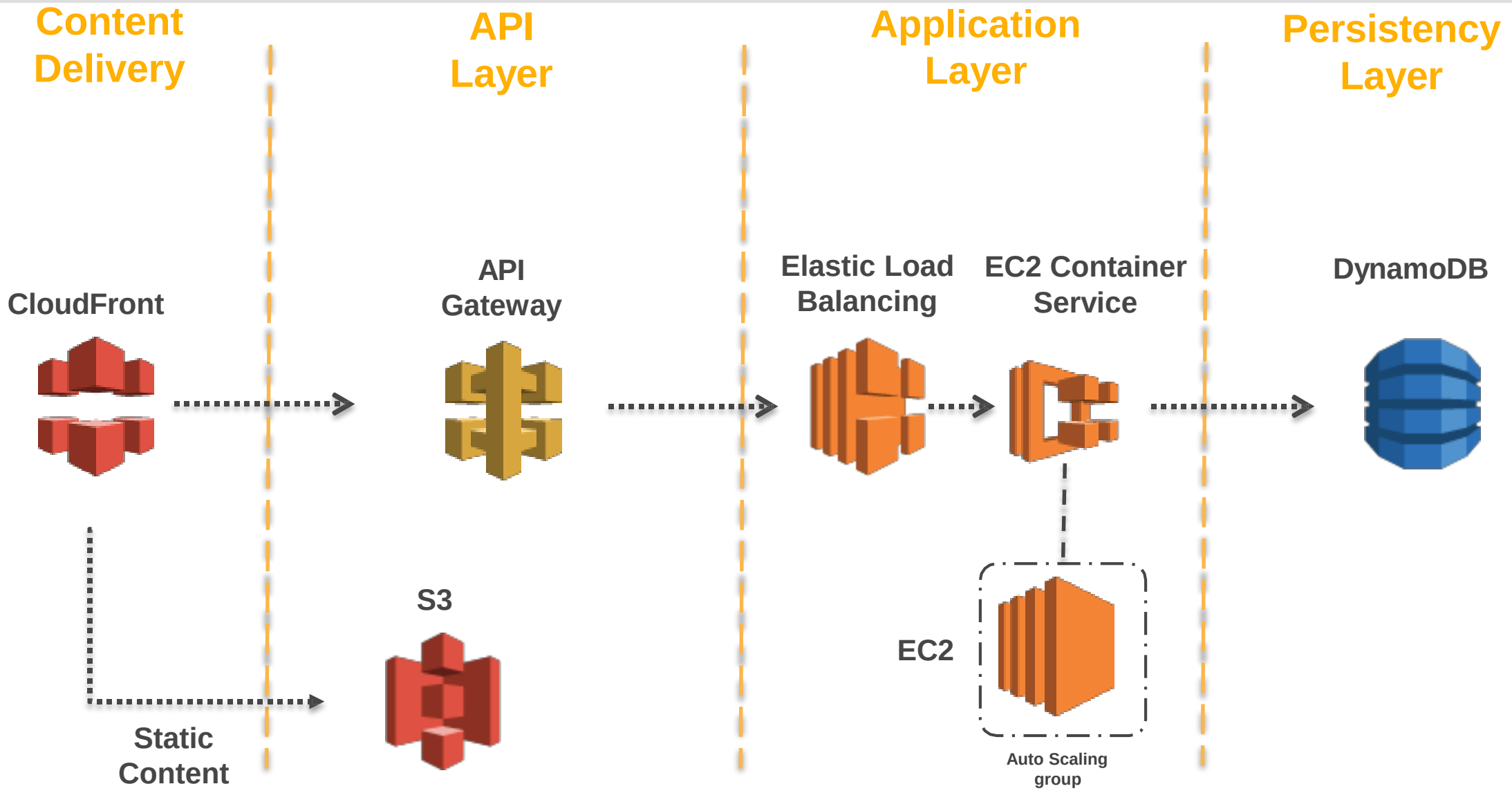
A Typical Microservice Architecture on AWS



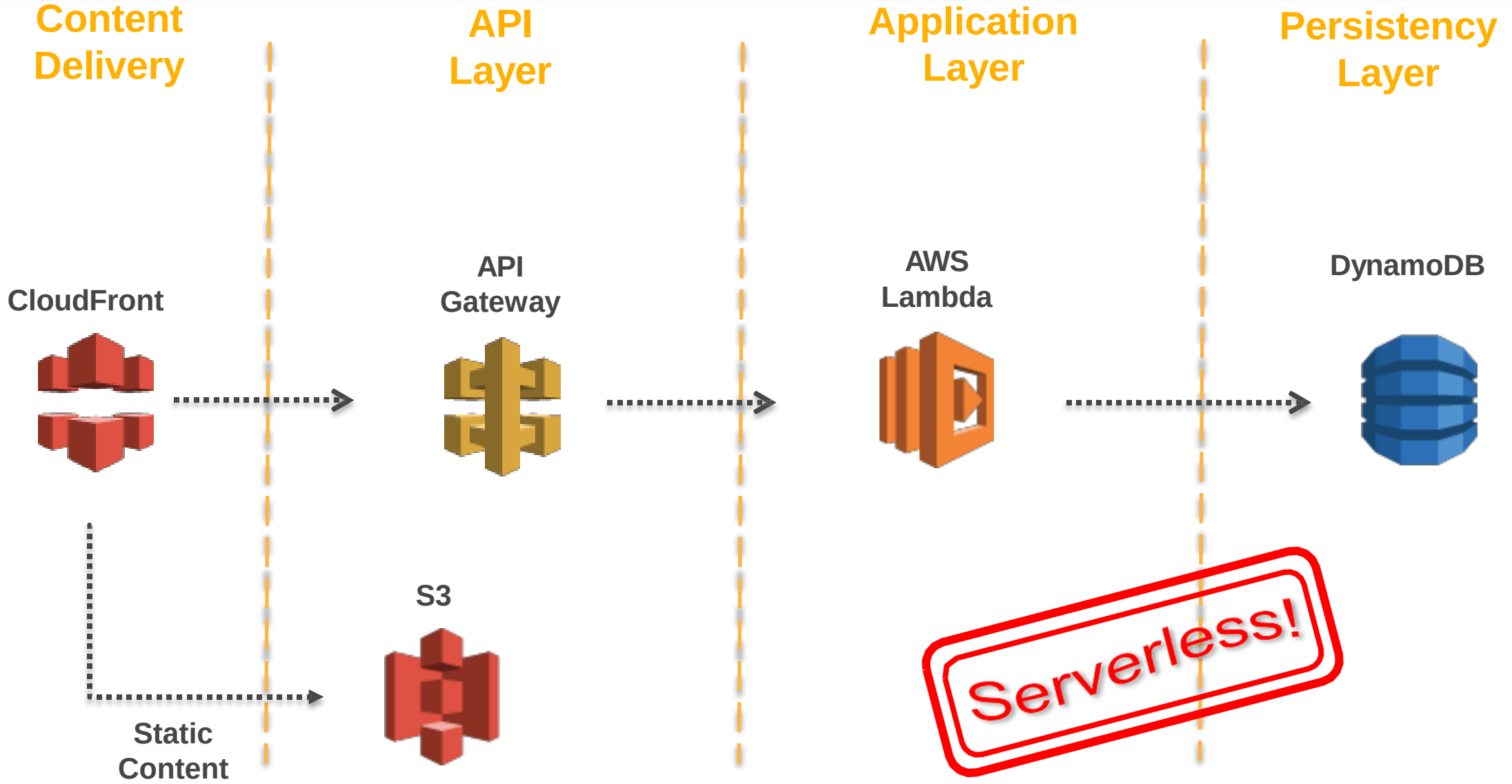
A Typical Microservice Architecture on AWS



A Typical Microservice Architecture on AWS



A Typical Microservice Architecture on AWS



Resources

Books:

- Continuous Delivery - Jez Humble, Dave Farley
- Working Effectively with Legacy Code - Michael Feathers
- Domain Driven Design - Eric Evans
- Your Brain at Work - David Rock
- Refactoring Databases - Scott W Ambler & Pramod Sadalage
- Building Microservices - Sam Newman

Articles/Blogs:

- Ball of Mud: http://www.laputan.org/mud/
- Demming - http://leanandkanban.wordpress.com/2011/07/15/demings-14-points/
- Coding Horror: http://www.codinghorror.com/blog/2007/11/the-big-ball-of-mud-and-other-architectural-disasters.html
- http://devlicio.us/blogs/casey/archive/2009/05/14/commercial-suicide-integration-at-the-database-level.aspx
- Evolutionary Architecture and Emergent Design: http://www.ibm.com/developerworks/java/library/j-eaed1/
index.html
- Microservices: http://www.infoq.com/presentations/Micro-Services and http://yobriefca.se/blog/2013/04/29/ micro-service-architecture/ and http://davidmorgantini.blogspot.co.uk/2013/08/micro-services-what-are-micro- services.html
- http://martinfowler.com/articles/microservices.html
- http://highscalability.com/blog/2014/4/8/microservices-not-a-free-lunch.html

Sources and Further Reading

- http://martinfowler.com/articles/microservices.html
- http://www.infoq.com/articles/microservices-intro
- http://brandur.org/microservices
- http://davidmorgantini.blogspot.de/2013/08/micro-services-what-are-micro-services.html
- http://12factor.net/
- http://microservices.io/
- https://rclayton.silvrback.com/failing-at-microservices
- http://www.activestate.com/blog/2014/09/microservices-and-paas-part-iii
- http://highscalability.com/blog/2014/7/28/the-great-microservices-vs-monolithic-apps-twitter-_melee.html
- http://capgemini.github.io/architecture/microservices-reality-check/