Microservices for Scalability

Joe Yoder – joe@refactory.com
Twitter: @metayoda
https://refactory.com

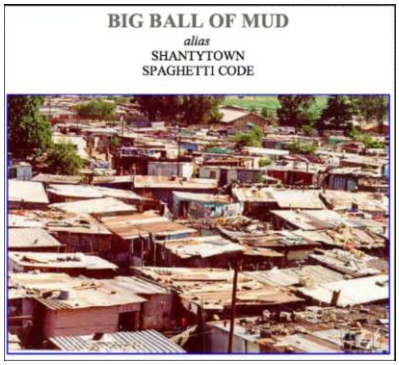
Refactory

Copyright 2021 Joseph Yoder, The Refactory, Inc.

If it's not microservices, it is…

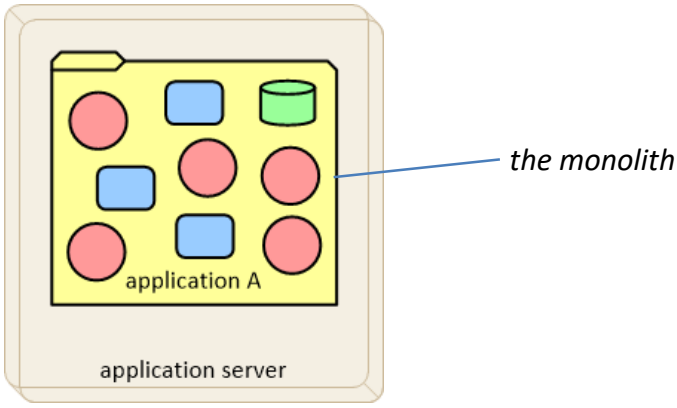
To understand the new we need to look back to the old:

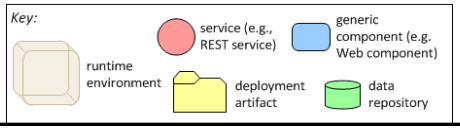
The monolithic model



3

Monolithic Style Illustrated





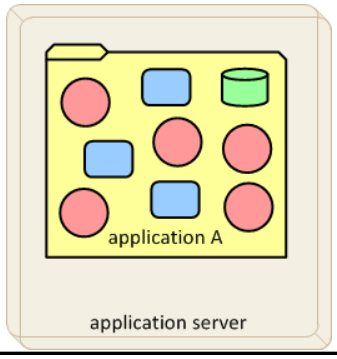
Consequences of the Monolith style

Benefits

- Performance and Security
- Simple to develop, test, deploy, scale
- Everything is in one place, usually single database
- Easier to deal with cross-cutting concerns

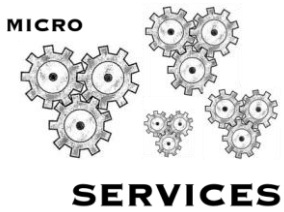
Costs

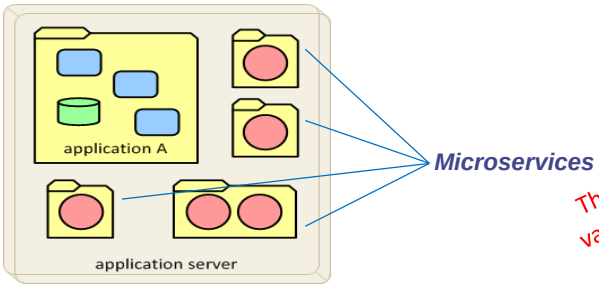
- As applications grow, monolith can be large and muddy, hard to change
- Agile development and quick delivery becomes difficult if not impossible
- Technology stack is less flexible and tends to become obsolete
- Rewrites are impossible
- Continuous deployment is cumbersome



Microservices

- **Microservices** is an architectural style that structures an application as a collection of loosely coupled services. In a **microservices** architecture, services should be fine-grained and the protocols should be lightweight. Lewis, J. & Fowler, M. "Microservices." 2014 martinfowler.com/articles/microservices.html



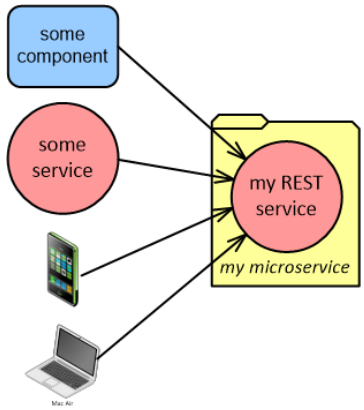


There are several possible variations. For example, each microservice could run on a separate application server

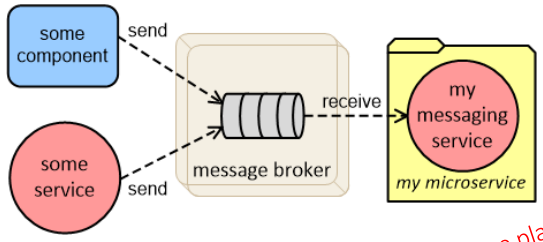
Microservices communication

Microservices typically communicate via

HTTP (following the REST style), or

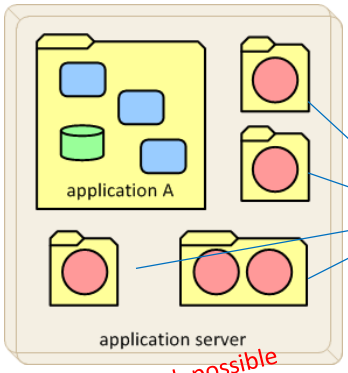


asynchronous messages/events



There are platform specific options, such as gRPC and Apache Thrift

Microservices Consequences



There are several possible variations. For example, each microservice could run on a separate application server

- **Potential Benefits**
  - **Smaller** and **Modular**
  - Easier to **Scale**
  - Easy to **change**
  - Easier to **deploy**
  - Less coupling
  - **Faster** to **build**, **test**, and **deploy**

Microservices

- **Potential Challenges**
  - **Cross-cutting** concerns
  - **Complexity** (many services)
  - **Distributed** System problems
  - Multi-databases (**consistency**)
  - **Transaction** management issues
  - **Testing** can be hard

Continuous Delivery is key!!!

How can Microservices help with Scalability???

What do we want to scale?

- Increased
  - ~~performance~~
  - ~~throughput~~
  - **adaptability/flexibility**
  - **rate of supporting new requirements**
  - **quicker time to deploy**
  - **support for configuration options**
  - **growing and evolving teams**

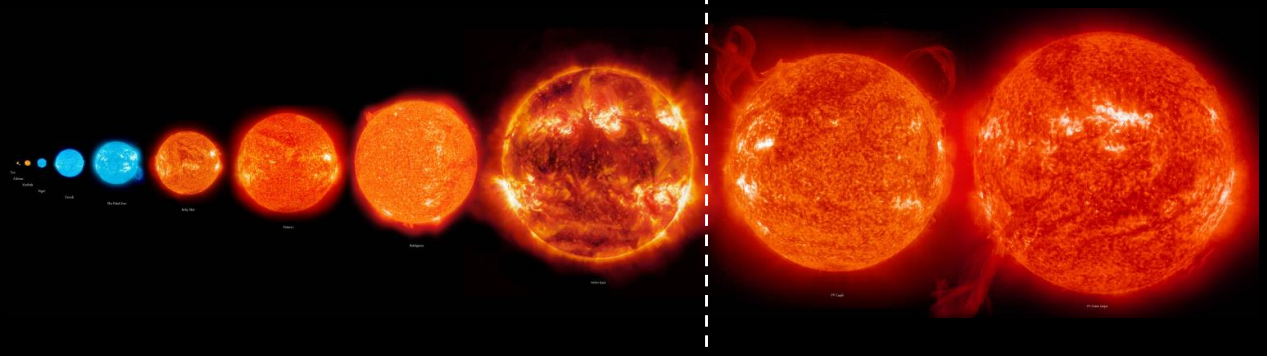
© Can Stock Photo / rukanoga

What about doing microservices

Serves you well if:

- large number of teams
- in a phase of rapid growth
- have a culture of agile and devops
- evolutionary architecture development
- have good tooling and infrastucture

You must be this big to ride



Design IDEALS for microservices

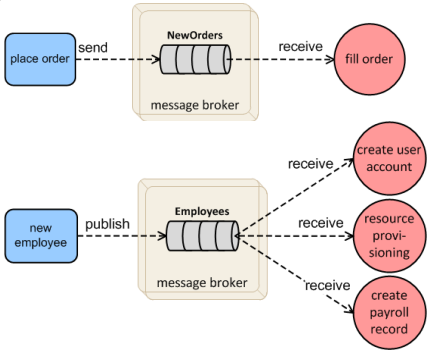
I	Interface segregation
D	Deployability
E	Event-driven
A	Availability over consistency
L	Loose Coupling
S	Single responsibility

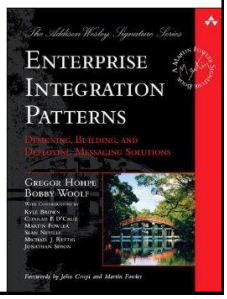
Evolved IDEALS collaborating with Paulo Merson
many personal discussion, talks, and workshops

Service interaction strategies

For each service, we make a design decision between

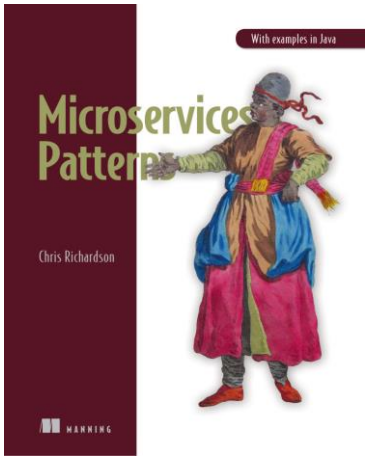
- synchronous (call-return) interaction, typically REST
- asynchronous messaging
  - ✓point-to-point channel
  - ✓pub-sub channel





Transactional Messaging Ideas and Patterns

- Sync over async
- Push vs Pull
- Store and forward
- Transactional outbox
- Dead Letter channel
- …

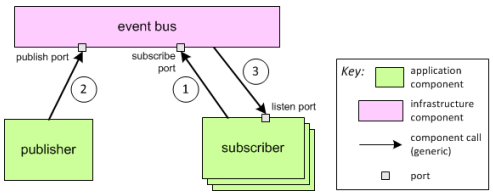


Chris Richardson Microservices Patterns https://microservices.io

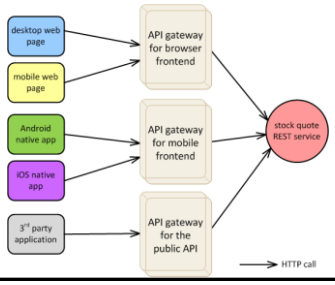
Service interaction strategies

In addition, we can choose to employ these two patterns for the overall interaction with services:

- EDA



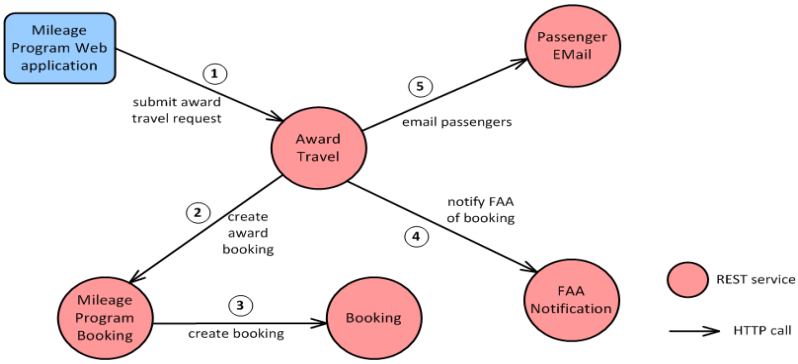
- API Gateway (BFF)



Adapted from workshop developed by Paulo Merson and Joseph Yoder

Event-Driven

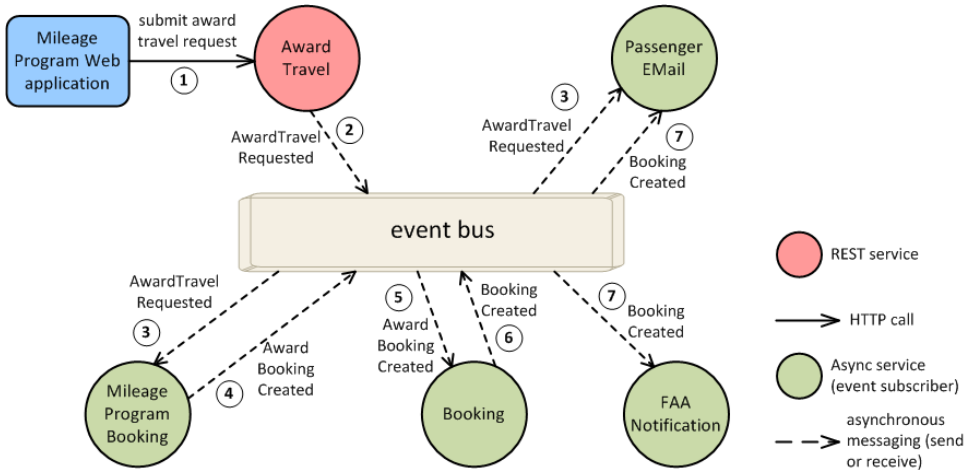
- Synchronous request-response calls are still everywhere



- But today's scalability and performance requirements pose a challenge that calls for events processed asynchronously

Adapted from workshop developed by Paulo Merson and Joseph Yoder

Event-driven example



Adapted from workshop developed by Paulo Merson and Joseph Yoder

Benefits of EDA

Maintainability

- Publishers and subscribers are independent and hence loosely coupled
- There's more flexibility to add functionality by simply adding subscribers or events

Scalability and throughput

- Publishers are not blocked, and events can be consumed by multiple subscribers in parallel

Availability and reliability:

- Temporary failures in one service are less likely to affect the others

Adapted from workshop developed by Paulo Merson and Joseph Yoder

Challenges of EDA

Maintainability

- The event-based programming model is more complex:
  - Some of the processing happens in parallel and may require synchronization points
  - Correction events, and mechanisms to prevent lost messages may be needed
  - Correlation identifiers may be needed

Testability

- Testing and monitoring the overall solution is more difficult

Interoperability and portability

- The event bus may be platform specific and cause vendor lock-in

Good UX is harder if end user needs to keep track of events

Adapted from workshop developed by Paulo Merson and Joseph Yoder

Managing data strategies

- Saga pattern
- Avoiding Distributed Transactions
- Database per Microservice Pattern
- Service Data Replication Pattern
- CQRS pattern

…

Examples can be found at: Microservices Patterns https://microservices.io

Model services to avoid distributed transactions

Saga still involves service calls that may fail

The best approach with respect to autonomy is to avoid the need for distributed transactions altogether

It can be done by modeling (or remodeling) services so that atomic data changes are collocated

- "Collocated" means they can use the same database connection

The premise is that

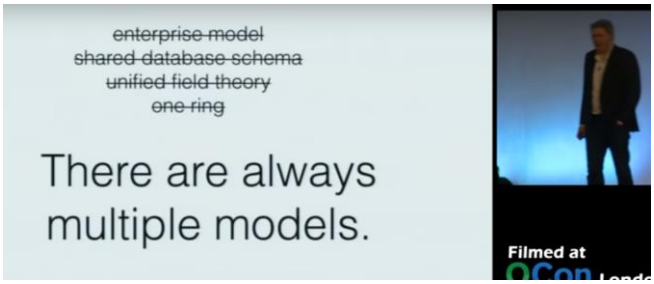
- a transaction is more likely to change data entities that are closely associated
- so we model associated data entities within the same service

Adapted from workshop developed by Paulo Merson and Joseph Yoder

Domain Modeling

The key to creating right-grained microservices is domain modeling

- Engage domain experts
- Avoid enterprise-wide models
- Establish well-defined borders for independent domain models
- Allow for flexible integration between domain models



Domain-Driven Design is a popular technique for domain modeling

But domain modeling can be done in an ad-hoc fashion as well

Adapted from workshop developed by Paulo Merson and Joseph Yoder

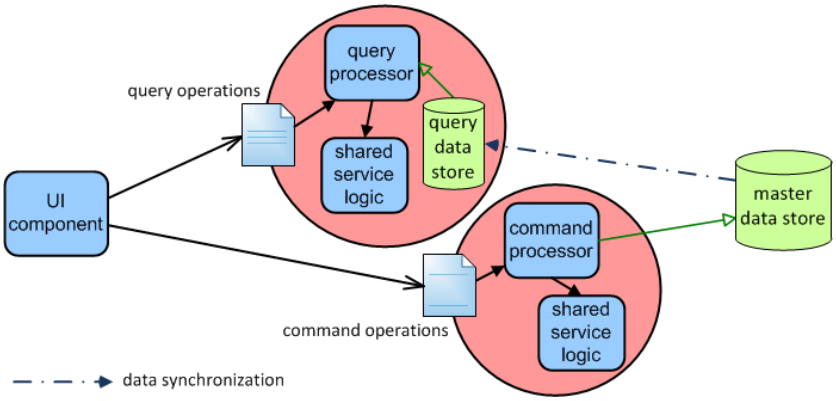
Availability over consistency

- The CAP theorem gives you two options: availability xor consistency
- We see enormous effort in industry to provide mechanisms to enable you to choose availability, ergo embrace *eventual* consistency
- Why? Users won't put up with lack of availability!

Adapted from workshop developed by Paulo Merson and Joseph Yoder

Availability over consistency in practice

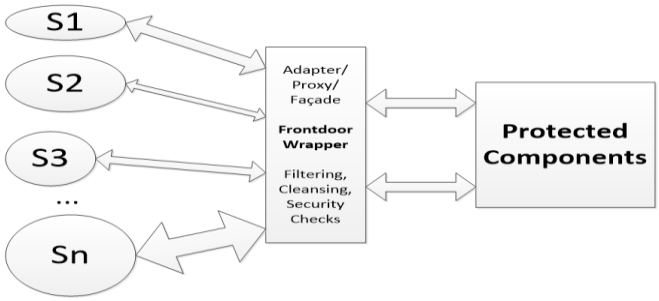
- CQRS and Service Data Replication



Adapted from workshop developed by Paulo Merson and Joseph Yoder

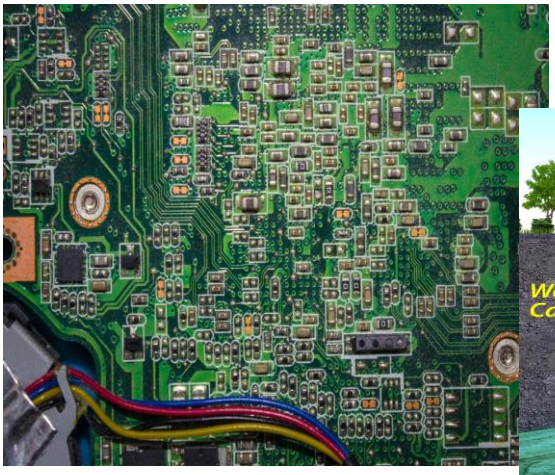
Loose coupling for microservices

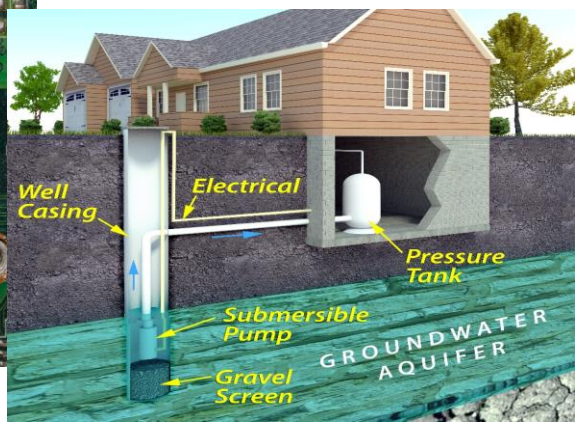
- Model around the business domain (DDD)
- Carefully design the contract
- Use wrapper patterns (adapter, façade, decorator, proxy)
- Use EDA, API Gateway, Asynchronous Messaging, Hypermedia, …



Adapted from workshop developed by Paulo Merson and Joseph Yoder

How big should your Microservices Be?





Size is Key

Macroliths (Mini-monoliths)

Evolutionary Approach - right sized services

It's not about being fashionable by just developing microservices, It's finding the right size MS and modeled around the domain

Evolutionary process allows the team find the boundaries of the system

Lots of communication and coordination between systems

DDD to the rescue

DDD can help you define the size of your microservice

- Not size in terms of lines of code (LOC)
- The size in terms of functional scope and/or business capabilities

Model Microservices around the Domain

A single responsibility, a well designed microservice must have!

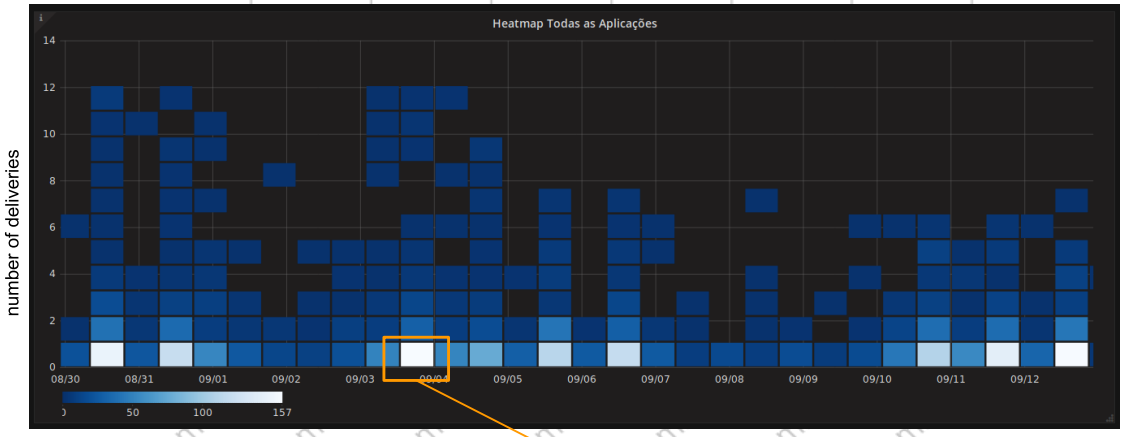
Microservices for Agility

Architecture Evolution

Architecture is rarely definitive:

pagseguro UOL	twitter	ebay	amazon.com
3º GERAÇÃO	3º GERAÇÃO	4º GERAÇÃO	Nth GERAÇÃO
Monolithic C#	Monolithic Rails	Monolithic Perl	Monolithic C++
↓	↓	↓	↓
Monolithic Java	JS/Rails/Scala	Monolithic C++	Java ... Scala
↓	↓	↓	↓
Microservices	**Microservices**	Java	**Microservices**
		↓	
		Microservices	

PagSeguro Success by using Microservices helped to evolve more quickly and stay lean/agile



You might have an agile elephant, but that is different of an agile cat.

One day previously, total of 157 different services did one delivery to production including new features, configuration changes and so on, impossible before!!!

It is a Journey

- Values drive practice (use design IDEALS)
- Deal with Technical Debt, Delivery Size,
- Model around Domain and Testing
- Continues Improvement and Learning
- Automation, DevOps, Cloud Computing
- Microservices are not a silver bullet but can help with agility and scalability

© Can Stock Photo Inc. / jefras

Obrigado!

yodamann

joe@refactory.com

@metayoda

"You can't fix what you can't see"
"If you think good architecture is expensive, try bad architecture"

© 2021 Joseph Yoder & The Refactory