Embracing Microservices

Grzegorz Piwowarek

@pivovarit

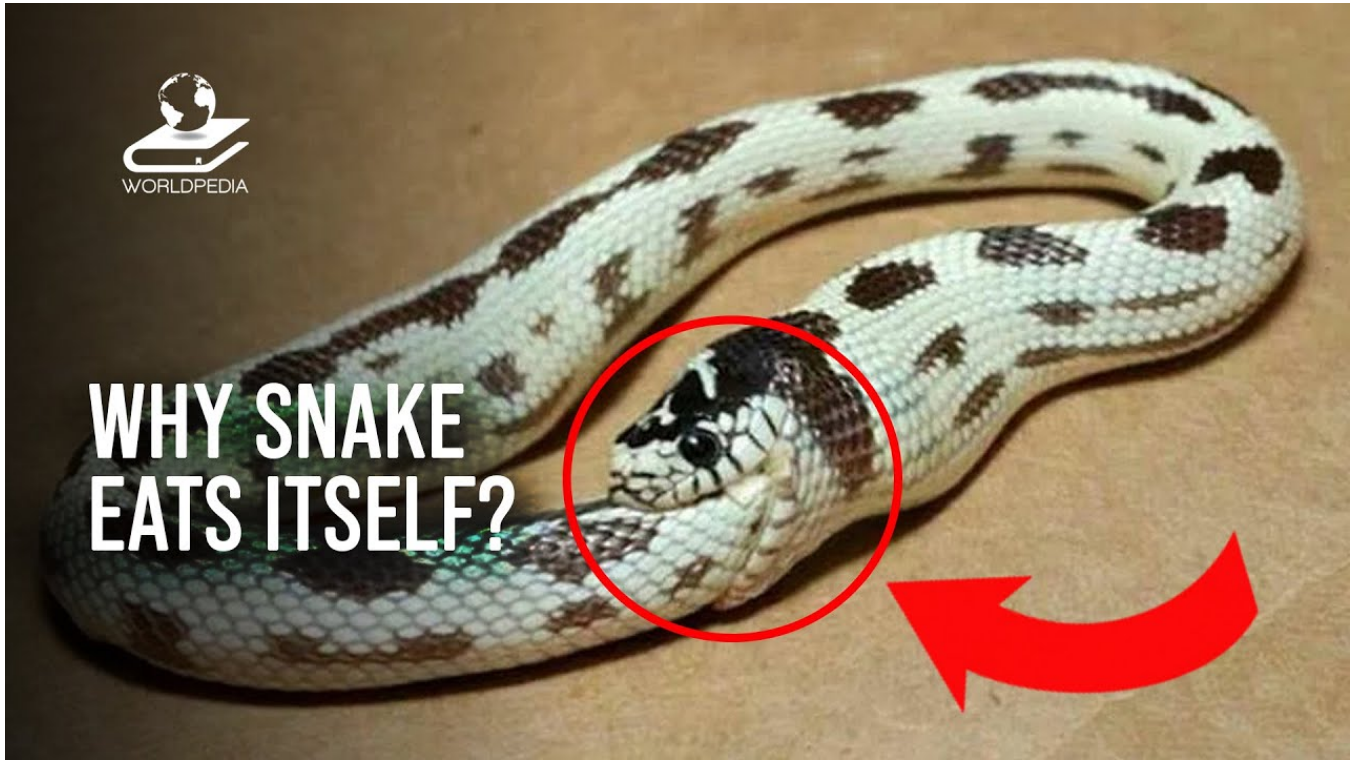
{ 4comprehension.com }

Founding Engineer @ Quesma

Owner @ 4Comprehension

WarsawJUG Leader

@pivovarit


WORLDPEDIA
WHY SNAKE
EATS ITSELF?

Monolith

Microservices

https://twitter.com/ddprrt/status/1425418538257428488

Monolith

Microservices

https://twitter.com/ddprrt/status/1425418538257428488

"It was the invention of the null reference in 1965...I call it my billion-dollar mistake."

Tony Hoare


WORLD WITHOUT NULL
imgflip.com

Another *"billion-dollar mistake":*

Another *"billion-dollar mistake":*

Microservices

Another *”billion-dollar mistake”:*

”Micro”

Antipattern #1: Size Obsession

Antipattern #1: Size Obsession

Microservice vs service?

Antipattern #1: Size Obsession

Microservice vs service?

How small is ”micro”?

Antipattern #1: Size Obsession

Microservice vs service?

How small is ”micro”?

N lines of code?

Antipattern #1: Size Obsession

Microservice vs service?

How small is ”micro”?

N lines of code?

N classes?

Antipattern #1: Size Obsession

Microservice vs service?

How small is ”micro”?

N lines of code?

N classes?

N MBs?

Antipattern #1: Size Obsession

Microservice vs service?

How small is ”micro”?

N lines of code?

N classes?

N MBs?

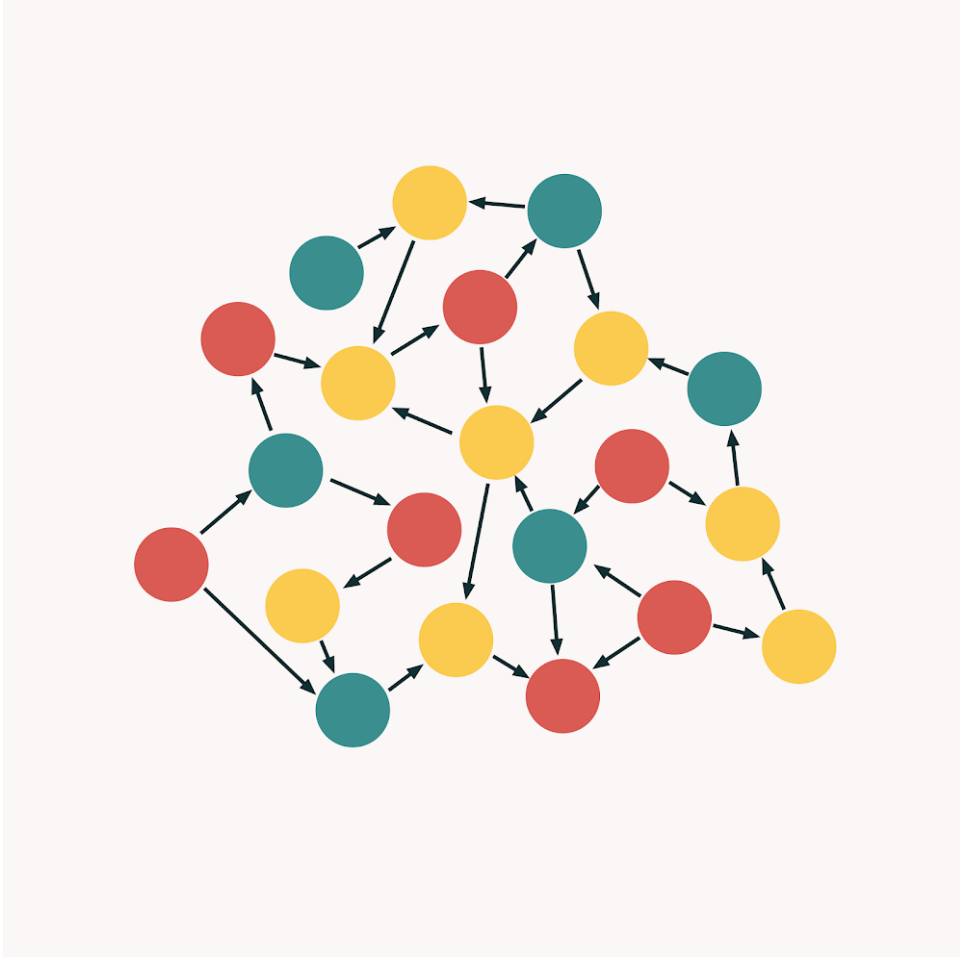
Rewritable in X time?

Microservices: the Main Idea

Microservices: the Main Idea

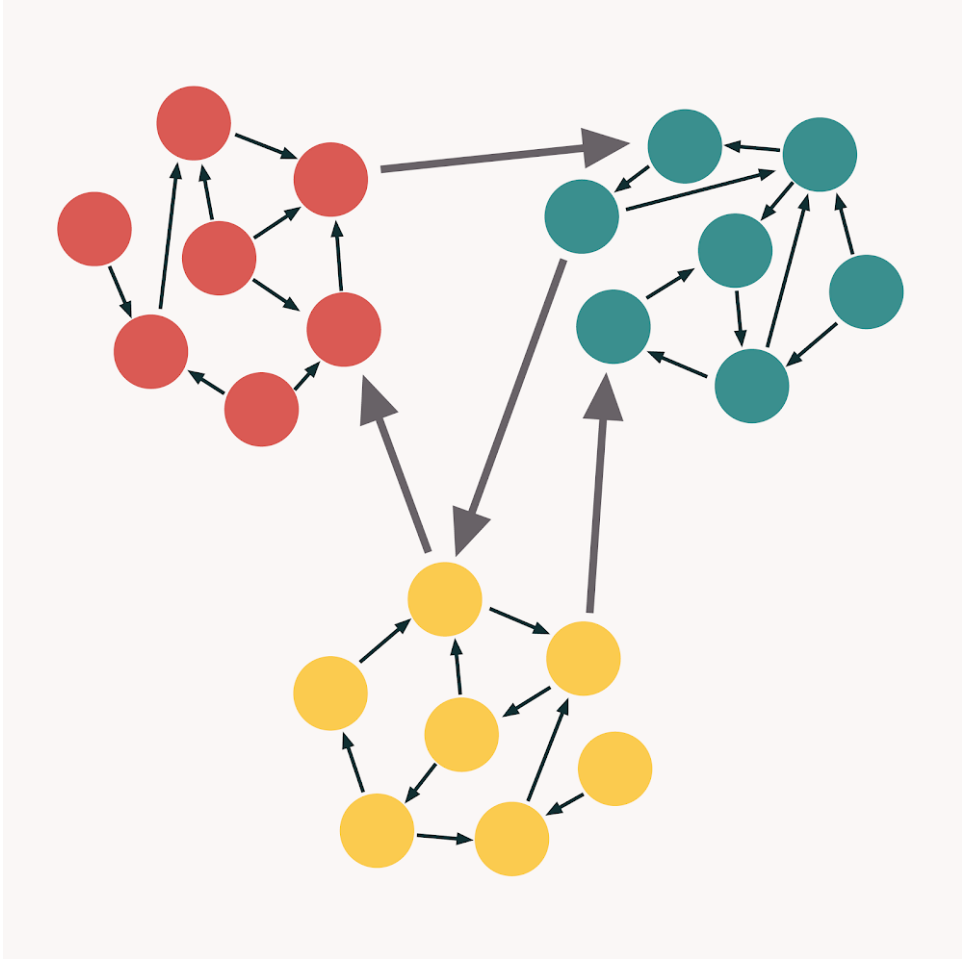
Enable scalability through independence and modularity

Tight coupling - Low cohesion



source: https://enterprisecraftsmanship.com/posts/cohesion-coupling-difference/

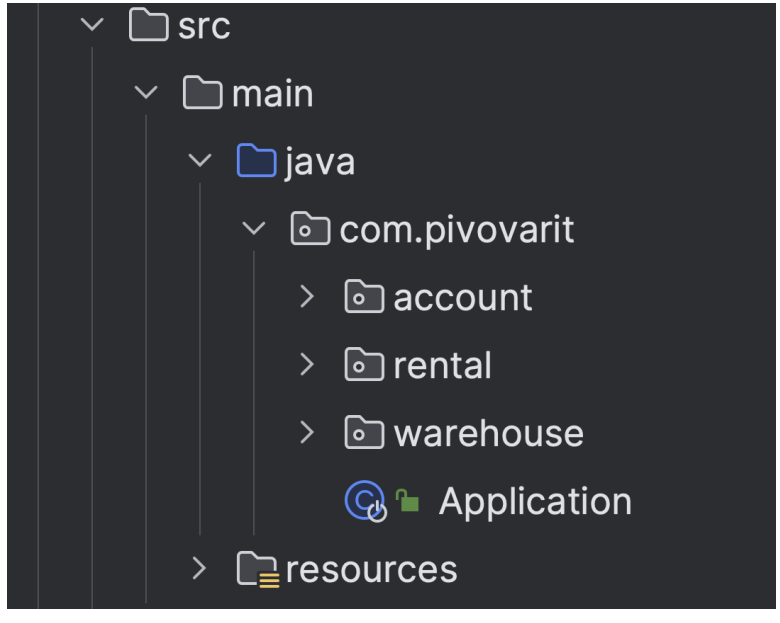
Low coupling - High cohesion



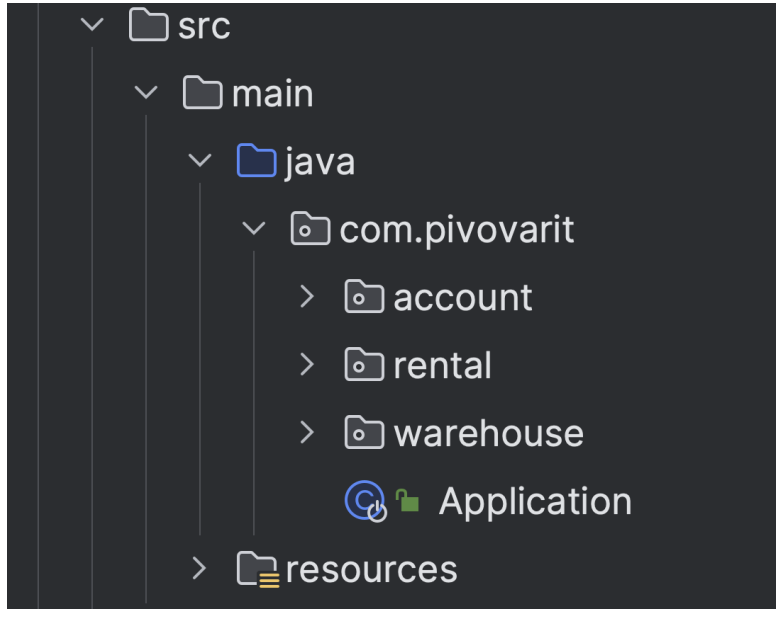
source: https://enterprisecraftsmanship.com/posts/cohesion-coupling-difference/

You can have modularity without microservices

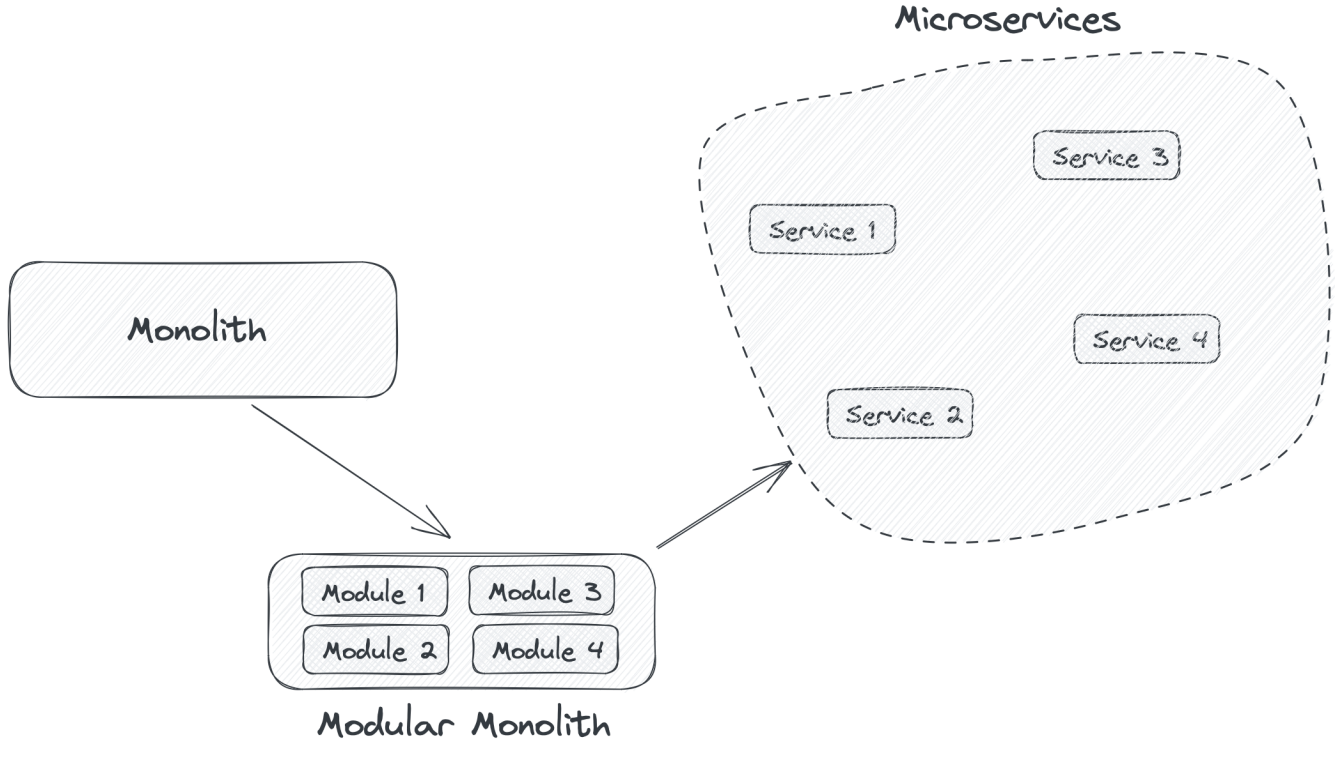
You can have modularity without microservices



You can have modularity without microservices



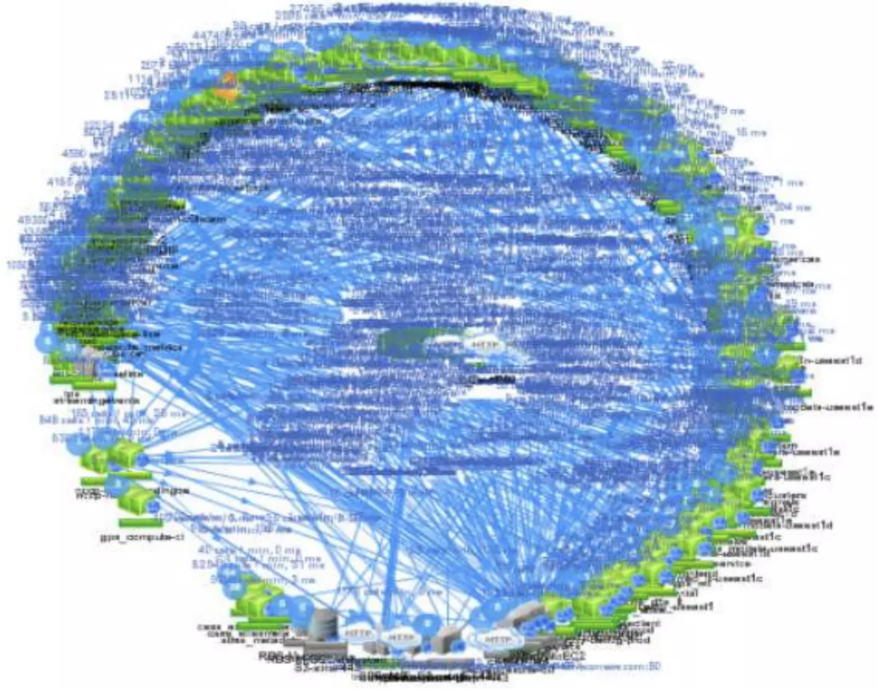
Naturally, you don't get all the benefits, but it's not a demanding investment


Microservices
Service 3
Service 1
Service 4
Service 2
Monolith
Module 1
Module 3
Module 2
Module 4
Modular Monolith

"The smaller the service, the more you maximize the benefits and downsides of microservice architecture."

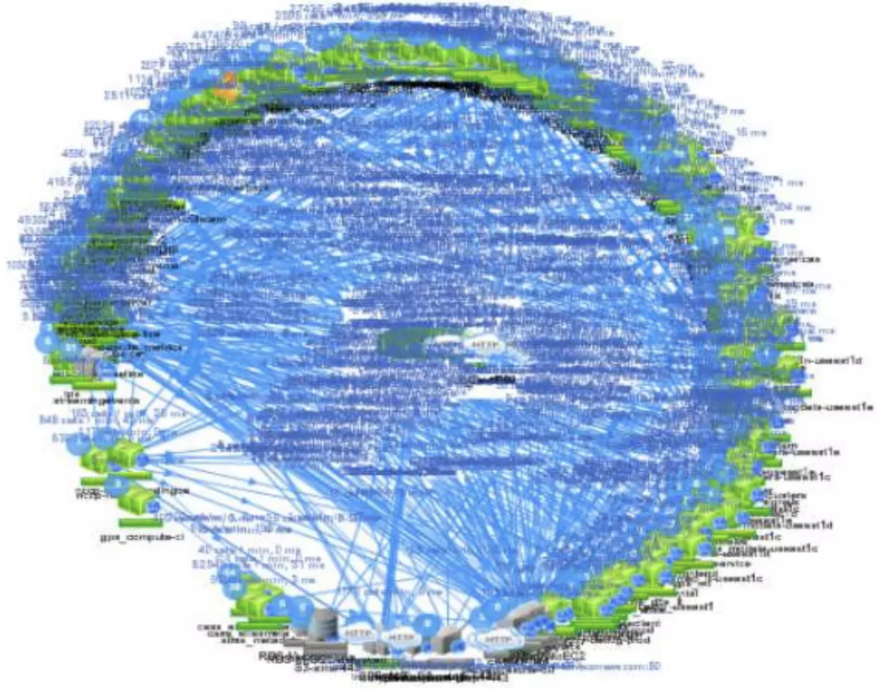
Sam Newman

Netflix & Micro-Services



author: Bruce Wong

Netflix & Micro-Services



author: Bruce Wong

Netflix: 2000 engineers

Applying the high scalability tips from Google and Netflix software architectures to your trivial project.

Do not ask about the max size, but when to split

So... when do we split?

So... when do we split?

When it hurts too much

So... when do we split?

When it hurts too much

The #1 rule of distributed systems: don't do it until you have to

The Pragmatic Default

https://twitter.com/dhh/status/1247522358908215296

Antipattern #2: Old Habits Die Hard

Antipattern #2: Old Habits Die Hard

Single version for all services

Antipattern #2: Old Habits Die Hard

Single version for all services

All-or-nothing deployments

Antipattern #2: Old Habits Die Hard

Single version for all services

All-or-nothing deployments

Dedicated ”DevOps” bottleneck teams

Antipattern #2: Old Habits Die Hard

Single version for all services

All-or-nothing deployments

Dedicated ”DevOps” bottleneck teams

End-to-end Acceptance tests

"Can you make a change to a service and deploy it by itself without changing anything else? If the answer is no, then many of the advantages we discuss throughout this book will be hard for you to achieve."

Sam Newman

Monorepos

Monorepos

make it easy to stick to old habits

Monorepos

make it easy to stick to old habits

- require extra discipline

Monorepos

make it easy to stick to old habits

- require extra discipline

- require extra tooling

Antipattern #3

Not Embracing Eventual Consistency

So you have 150 services and they all need to be up and responsive at the same time for the whole system to work?

Sync vs Async

Sync vs Async

Both are good choice if you understand trade-offs

Sync vs Async

Both are good choice if you understand trade-offs

Consistency vs Availability

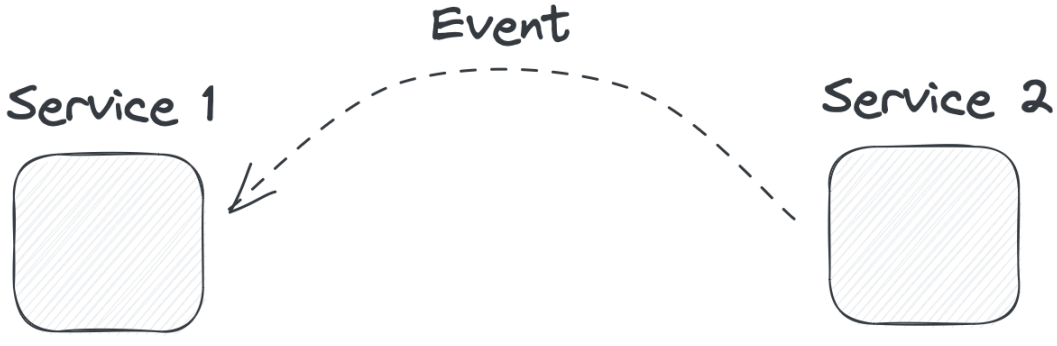
Sync vs Async

Both are good choice if you understand trade-offs

Consistency vs Availability

Want to be fully independent? Forget about Strict Consistency

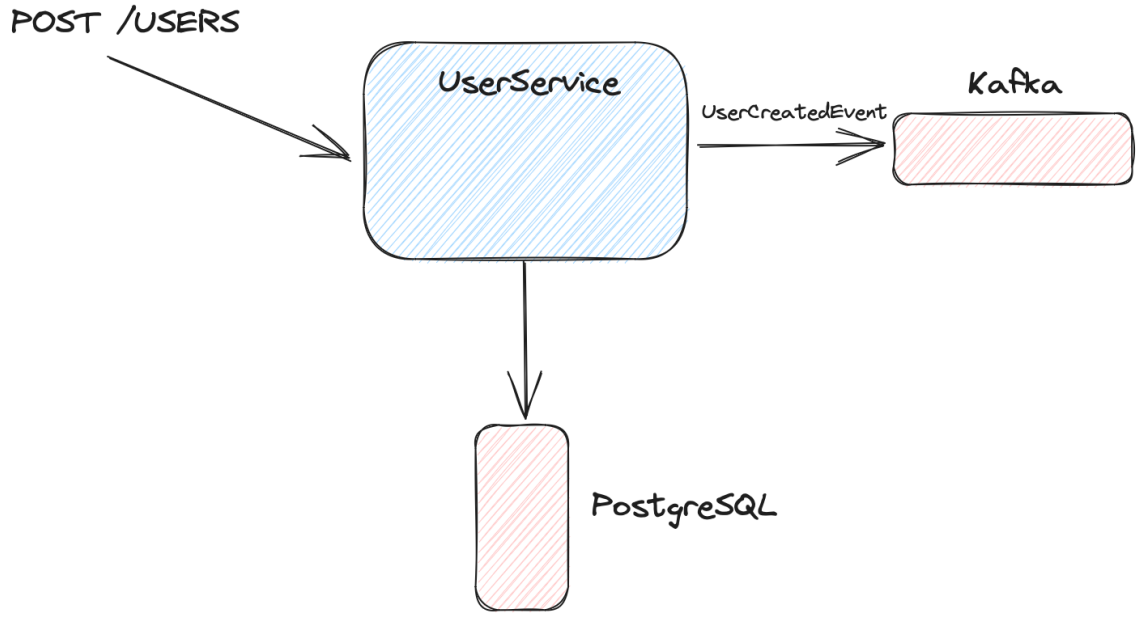

Service 1
HTTP GET
Service 2

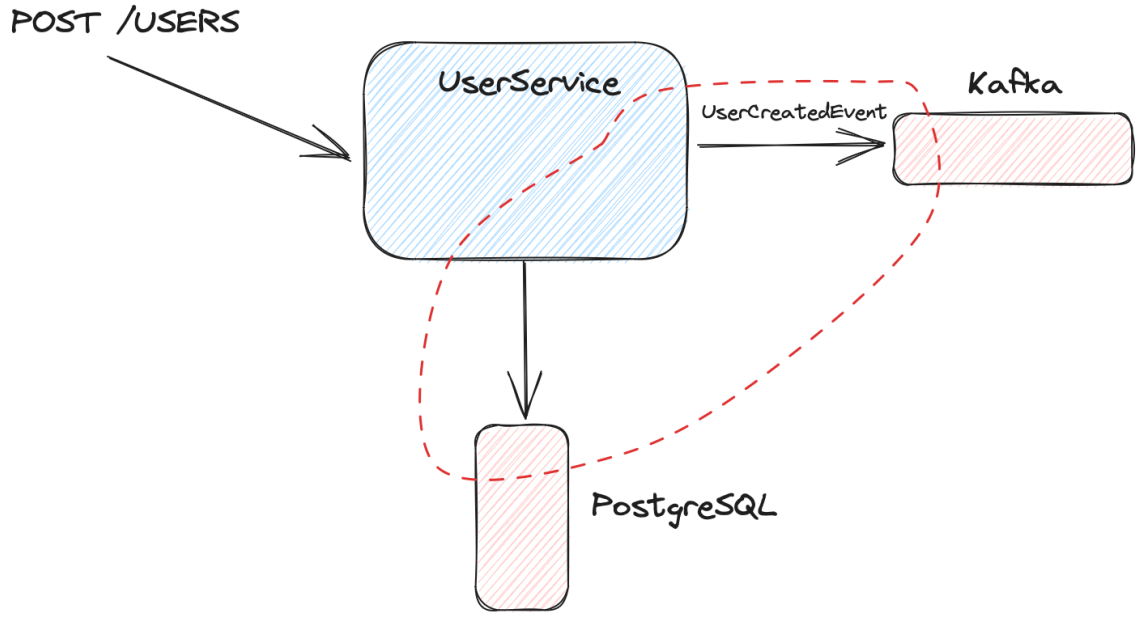

Event
Service 1
Service 2

Antipattern #4:

Accidental Consistency

Eventual Consistency != Accidental Consistency

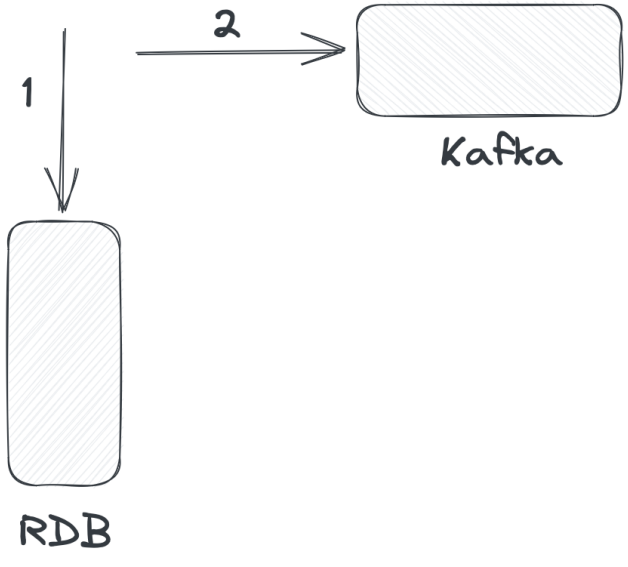

POST /USERS
UserService
UserCreatedEvent
Kafka
PostgreSQL


POST /USERS
UserService
UserCreatedEvent
Kafka
PostgreSQL

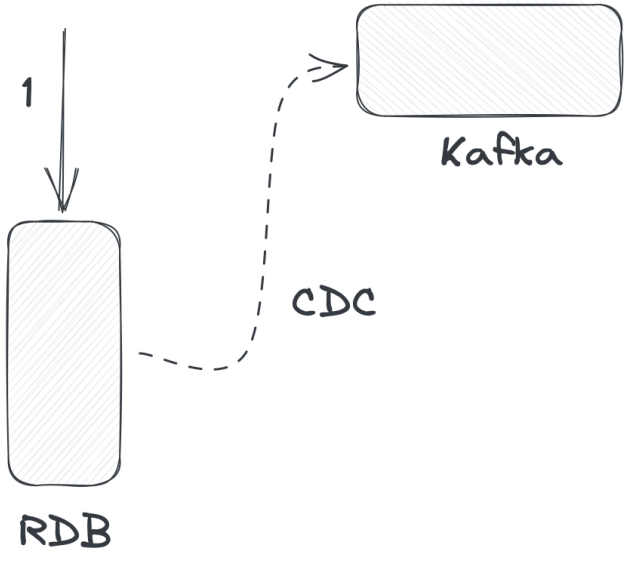
```java
@Transactional
public void createUser(CreateUserCommand command) {
    var user = from(command);

    persist(user);                          // 1
    send(UserCreatedEvent.from(user)); // 2
}
```

`@Transactional` won't save you in the distributed world

Dual-Write (distributed transaction)



Singe-Write with async propagation

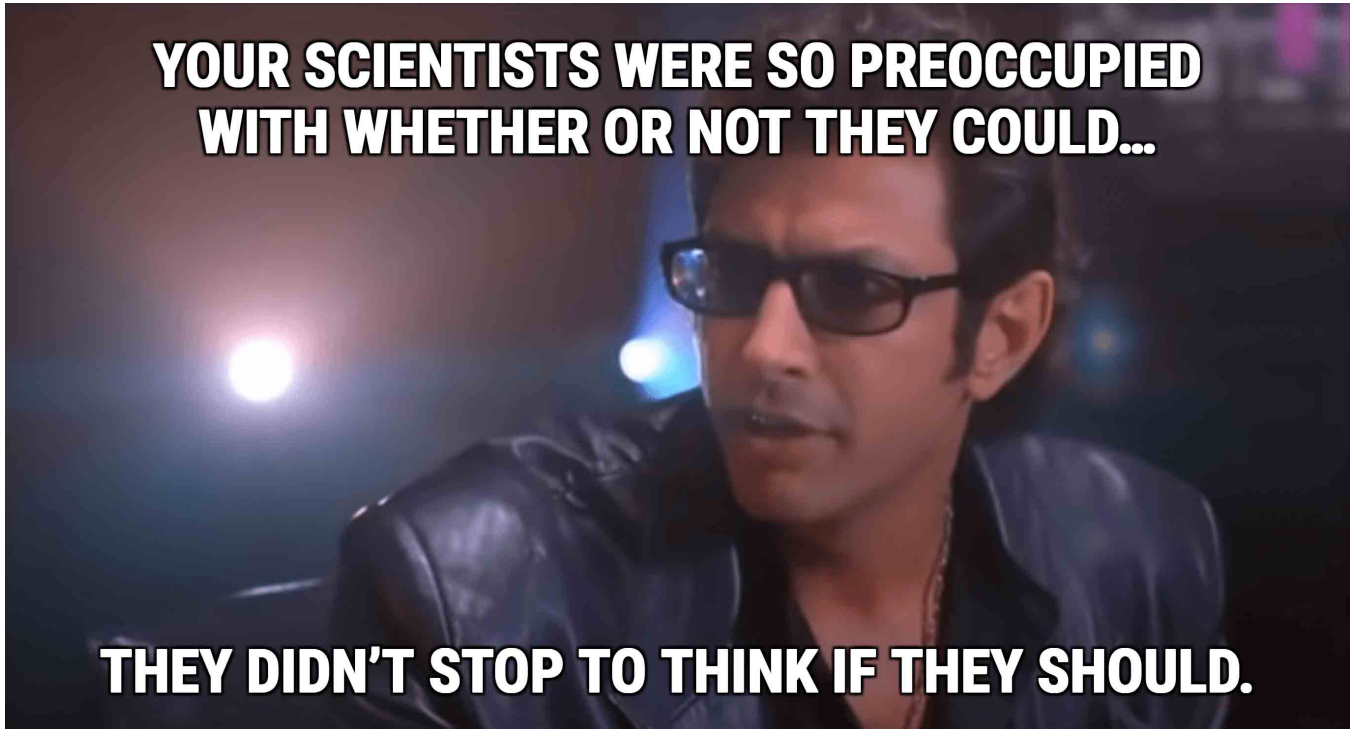


Transactional Outbox Pattern

Change Data Capture

Antipattern #5:

Technology Heterogeneity



Sounds great on paper

Sounds great on paper

Don’t throw in new tech unless there’s a good reason for it

Sounds great on paper

Don’t throw in new tech unless there’s a good reason for it

Most companies will benefit more from standardized tech stacks with occasional experiments

Antipattern #6:

DRY abuse and Shared Libraries

Antipattern #6:

DRY abuse and Shared Libraries

Sound like a noble effort

Antipattern #6:

DRY abuse and Shared Libraries

Sound like a noble effort

Can contribute to dependency hell

Antipattern #6:

DRY abuse and Shared Libraries

Sound like a noble effort

Can contribute to dependency hell

A small change can result in N deployments

Antipattern #6:

DRY abuse and Shared Libraries

Sound like a noble effort

Can contribute to dependency hell

A small change can result in N deployments

Need to be treated like libraries and not bags with code

Antipattern #6:

DRY abuse and Shared Libraries

Sound like a noble effort

Can contribute to dependency hell

A small change can result in N deployments

Need to be treated like libraries and not bags with code

Dangerous when business logic sneaks in

I need to consume REST API of another service. Do I create my own DTOs?

I need to consume REST API of another service. Do I create my own DTOs?

Yes. You want to share contract and not Java classes

I need to consume REST API of another service. Do I create my own DTOs?

Yes. You want to share contract and not Java classes

You get independence at the expense of writing boring code

Summary

Summary

Do not start with microservices, split existing services when absolutely necessary

Summary

Do not start with microservices, split existing services when absolutely necessary

You're now working with N independent systems, stop treating them like they are one

Summary

Do not start with microservices, split existing services when absolutely necessary

You're now working with N independent systems, stop treating them like they are one

You're now prone to Fallacies of Distributed Computing - embrace Eventual Consistency

Summary

Do not start with microservices, split existing services when absolutely necessary

You're now working with N independent systems, stop treating them like they are one

You're now prone to Fallacies of Distributed Computing - embrace Eventual Consistency

Pay attention to guarantees of your system - At-Most-Once and Accidental Consistency might not be good enough

Summary

Do not start with microservices, split existing services when absolutely necessary

You're now working with N independent systems, stop treating them like they are one

You're now prone to Fallacies of Distributed Computing - embrace Eventual Consistency

Pay attention to guarantees of your system - At-Most-Once and Accidental Consistency might not be good enough

Add new tech only when there's a good reason to do it

Thank You!

@pivovarit

4comprehension.com

https://pivovarit.github.io/talks/embracing-microservices