# Scalable Web Systems

How to Model Services

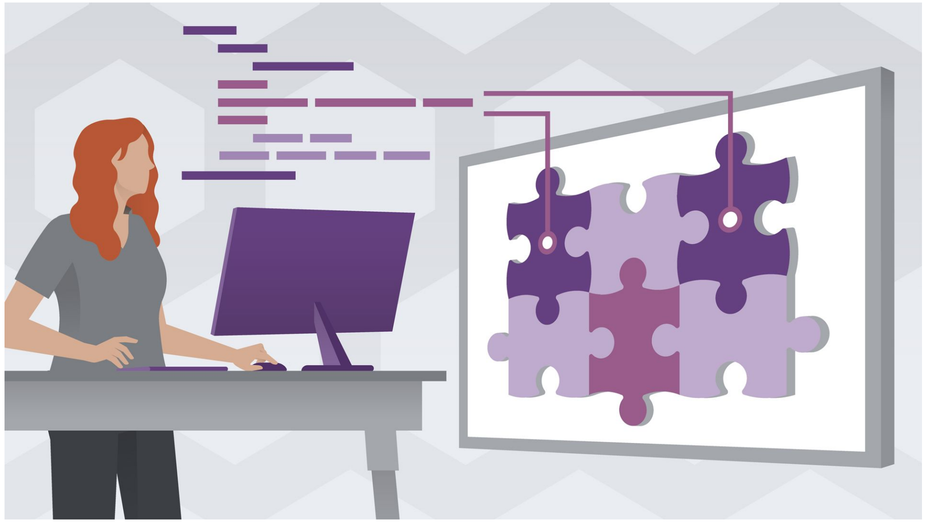
## What makes a good service?

**Loose Coupling**

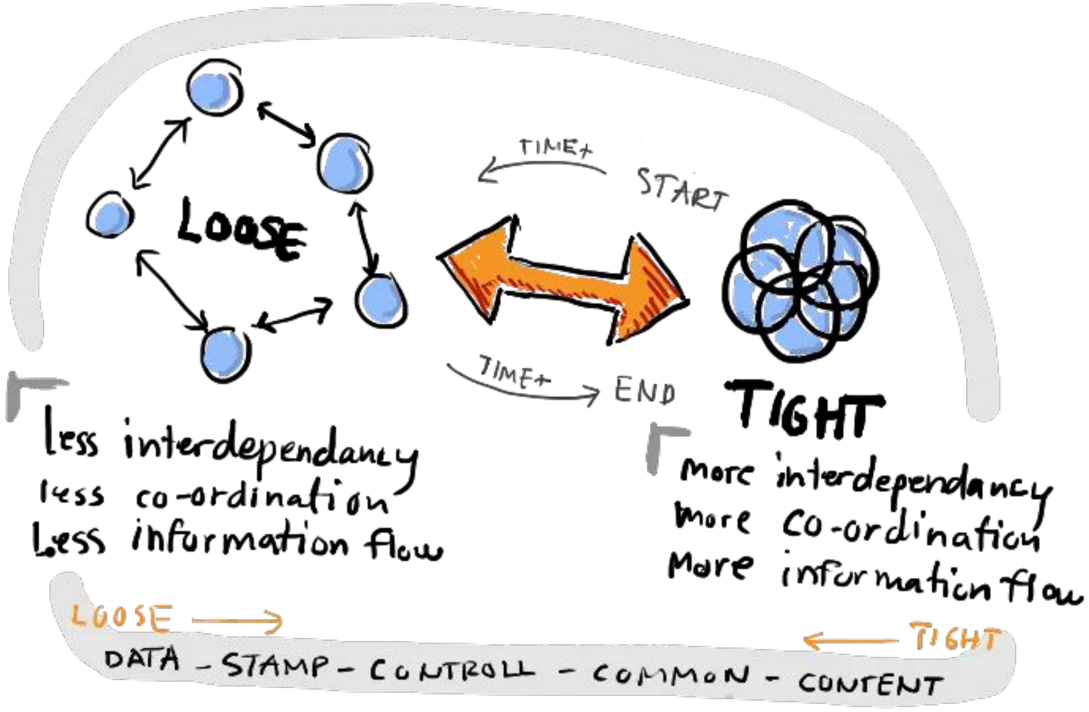
A change to one service should not require a change to another.

**High Cohesion**

We want related behavior to sit together, and unrelated behavior to sit elsewhere.



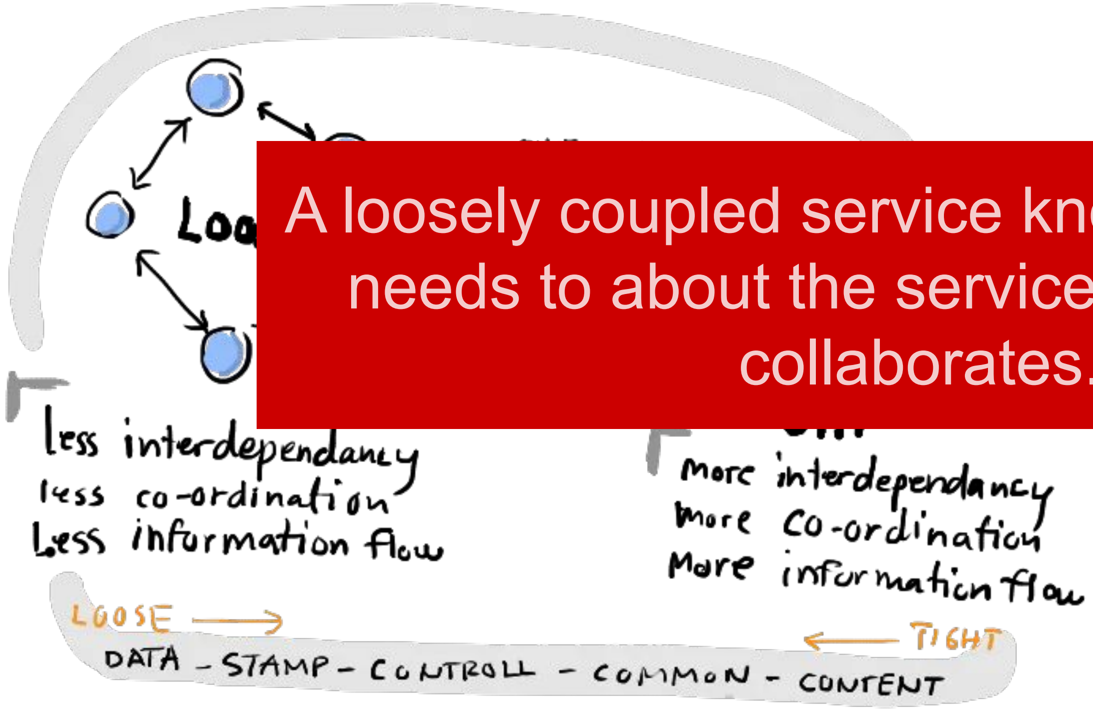
## Loose Coupling



When services are loosely coupled, a change to one service should not require a change to another.

The whole point of a microservice is being able to make a change to one service and deploy it, without needing to change any other part of the system.

## Loose Coupling



When services are loosely coupled, a change to one

quire a

able to make a change to one service and deploy it, without needing to change any other part of the system.

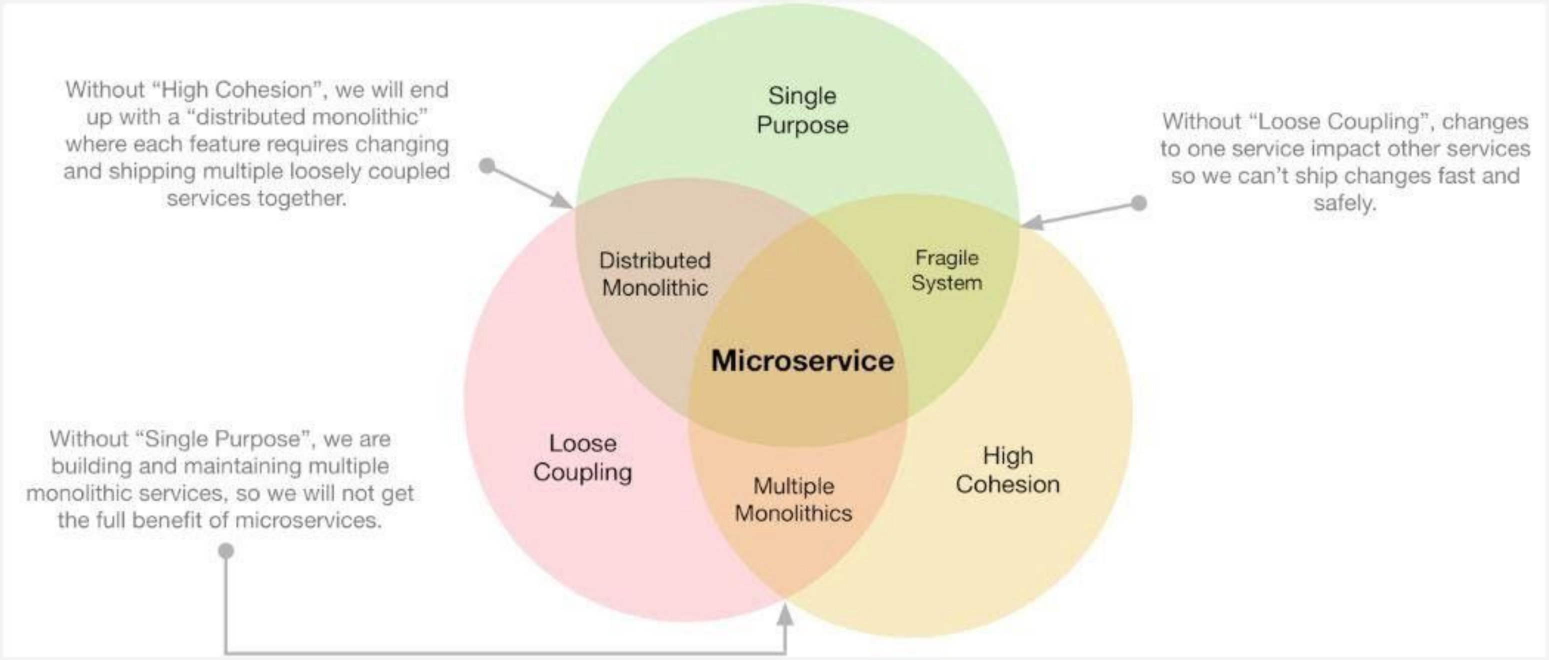
A loosely coupled service knows as little as it needs to about the services with which it collaborates.

## High Cohesion

We want related behavior to sit together, and unrelated behavior to sit elsewhere.

So we want to find boundaries within our problem domain that help ensure that related behavior is in one place, and that communicate with other boundaries as loosely as possible.

Three principles for well-architected microservices. Source: Ma 2018.*


Without "High Cohesion", we will end up with a "distributed monolithic" where each feature requires changing and shipping multiple loosely coupled services together.
Single Purpose
Without "Loose Coupling", changes to one service impact other services so we can't ship changes fast and safely.
Distributed Monolithic
Fragile System
Microservice
Loose Coupling
Multiple Monolithics
High Cohesion
Without "Single Purpose", we are building and maintaining multiple monolithic services, so we will not get the full benefit of microservices.

## The Bounded Context

Any given domain consists of multiple **bounded contexts**.

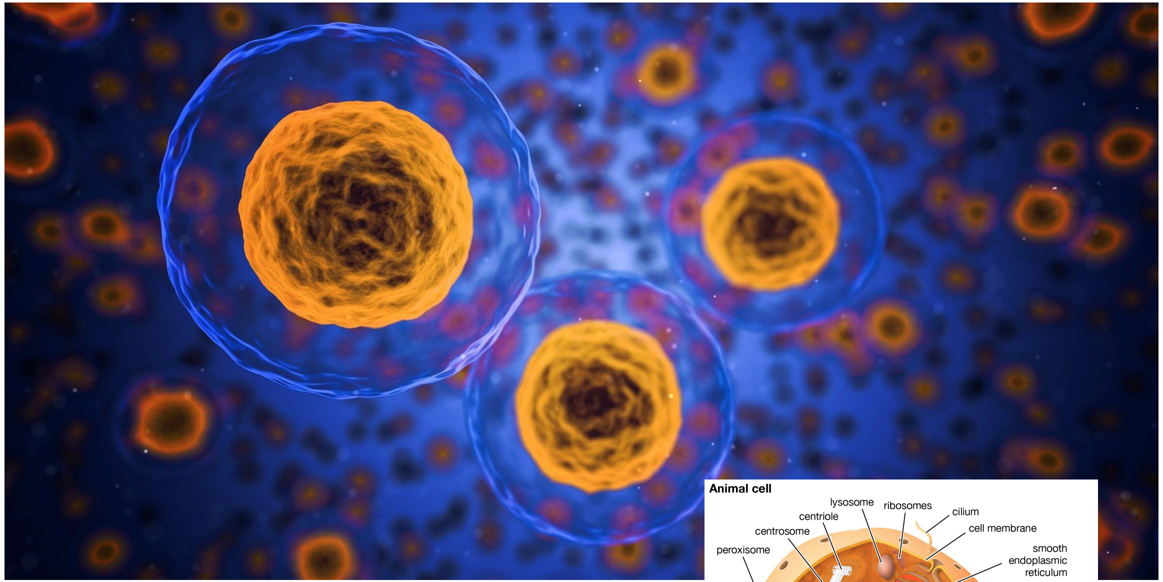
Residing within each are things that do not need to be communicated outside as well as things that are shared externally with other bounded contexts.

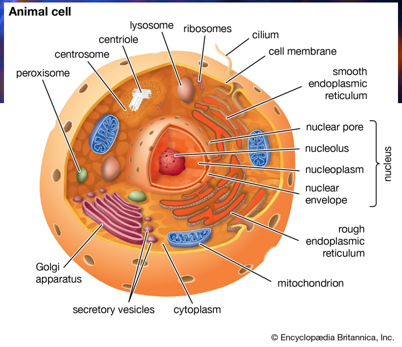
“a specific responsibility enforced by explicit boundaries.”

Each bounded context has an explicit interface, where it decides what models to share with other contexts.

## The Bounded Context

If you want information from a bounded context, or want to make requests of functionality within a bounded context, you communicate with its explicit boundary using models.

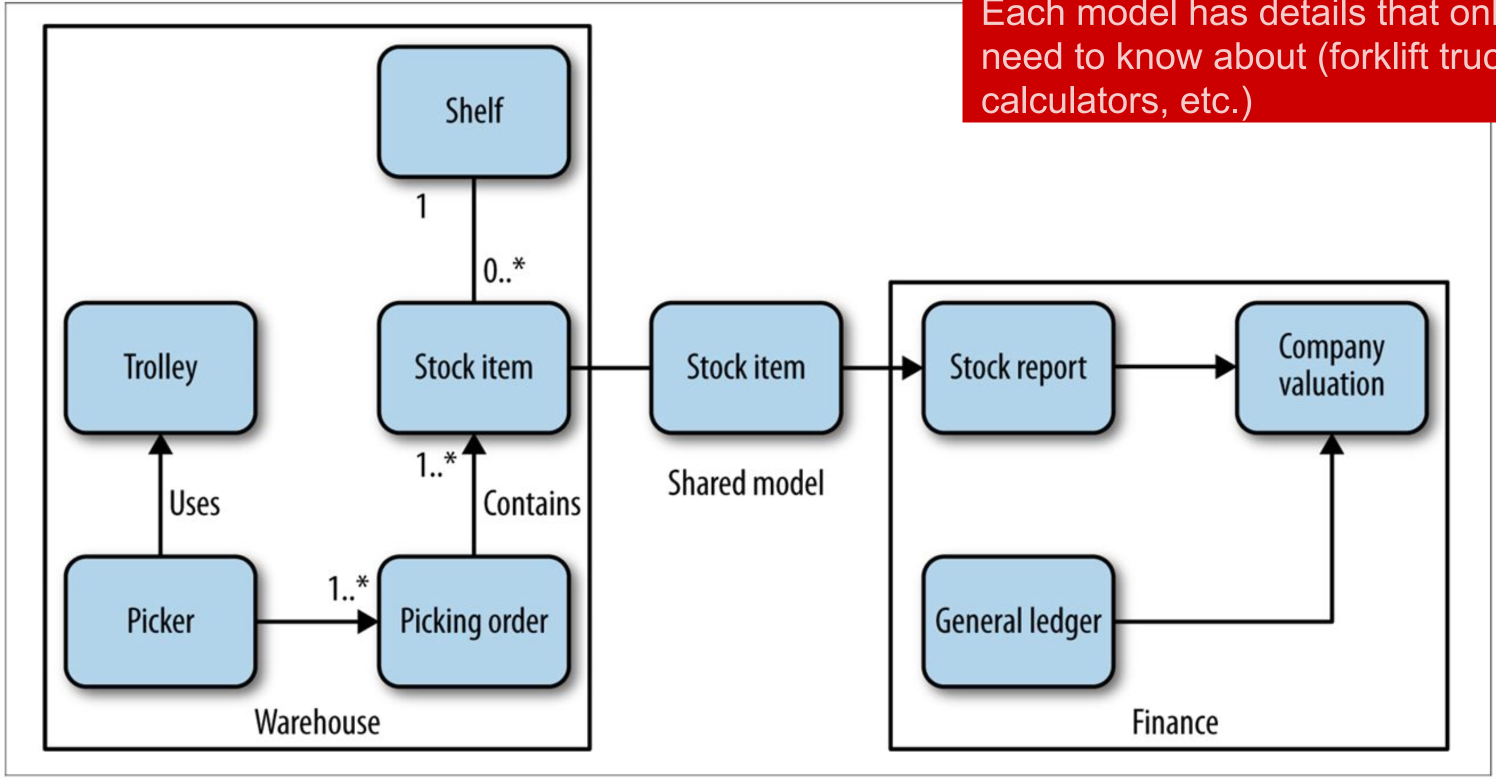




# Shared and Hidden Models

Each model has an explicit interface to the outside world (in terms of inventory reports, pay slips, etc.)

Each model has details that only they need to know about (forklift trucks, calculators, etc.)



## Modules and Services

if our service boundaries align to the bounded contexts in our domain, and our microservices represent those bounded contexts, we are off to an excellent start in ensuring that our microservices are loosely coupled and strongly cohesive.

We have the option of using modules within a process boundary to keep related code together and attempt to reduce the coupling to other modules in the system.

When you’re starting out on a new codebase, this is probably a good place to begin. So once you have found your bounded contexts in your domain, make sure they are modeled within your codebase as modules, with shared and hidden models.

# Modules and Services

if our service boundaries align to the bounded contexts in our domain, and our microservices represent those bounded contexts, we are off to an excellent start in ensuring t e.

We have t related code toge e system.

When you ce to begin. So make sure they are modeled within your codebase as modules, with shared and hidden models.

These modular boundaries then become excellent candidates for microservices.

Microservices should cleanly align to bounded contexts.

## Business Capabilities

“When you start to think about the bounded contexts that exist in your organization, you should be thinking not in terms of data that is shared, but about the capabilities those contexts provide the rest of the domain.”

“These capabilities may require the interchange of information—shared models—but I have seen too often that thinking about data leads to anemic, CRUD-based (create, read, update, delete) services.”

So ask first “What does this context do?”, and then “So what data does it need to do that?”

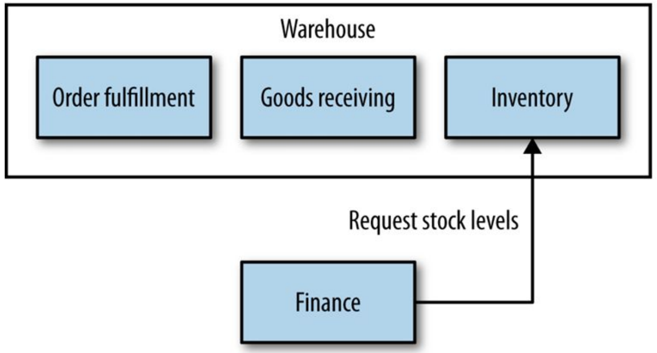
Think about capabilities first!

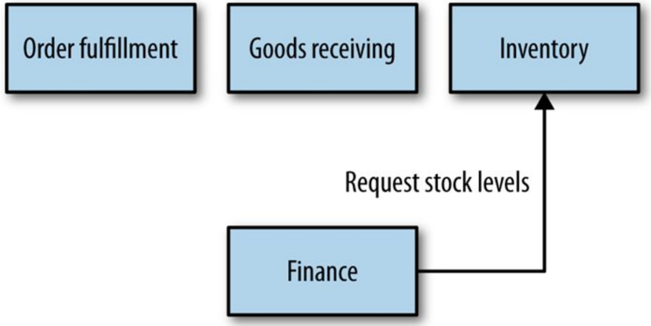
## Turtles all the way down...

At the start, you will probably identify a number of coarse-grained bounded contexts. But these bounded contexts can in turn contain further bounded contexts.

Considering the boundaries of your microservices, first think in terms of the larger, coarser-grained contexts, and then subdivide along these nested contexts when you’re looking for the benefits of splitting out these seams.

# Turtles all the way down...





## Communication in Terms of Business Concepts

When you start to think about the bounded contexts that exist in your organization, you should be thinking not in terms of data that is shared, but about the **capabilities those contexts provide** the rest of the domain.

These capabilities may require the interchange of information (shared models).

---

So ask first “What does this context do?”,

and then “So what data does it need to do that?”

**Repeat: think about capabilities first!**

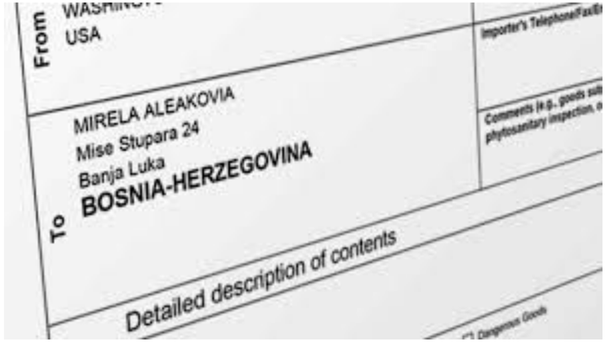
# Communication in Terms of Business Concepts

The changes we implement to our system are often about changes the business wants to make to how the system behaves.

It’s also important to think of the communication between these microservices in terms of the same business concepts.

The same terms and ideas that are shared between parts of an organization should be reflected in your interfaces.

**Good analogy:** It can be useful to think of forms being sent between these microservices, much as forms are sent around an organization.



From

WASHIN

USA

Importer's Telephone/Fax/E

To

MIRELA ALEAKOVIA
Mise Stupara 24
Banja Luka
BOSNIA-HERZEGOVINA

Comments (e.g. goods su
phytosanitary inspection, o

Detailed description of contents

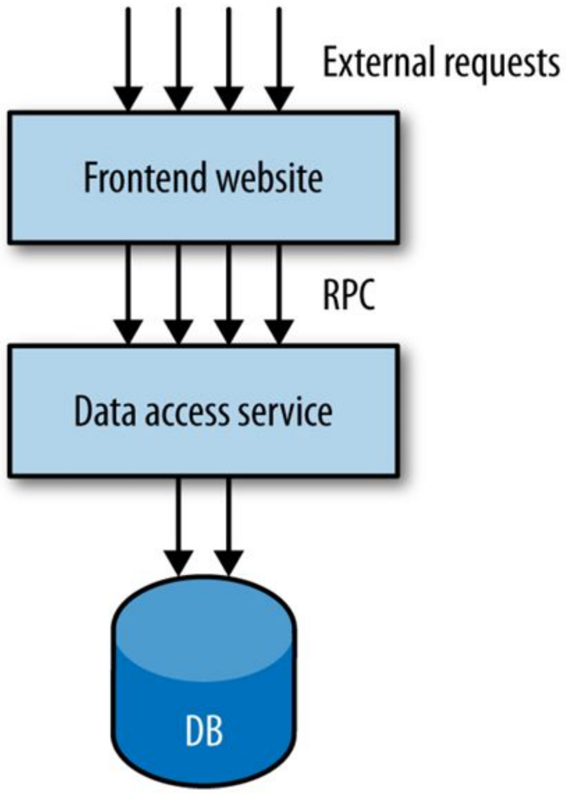
Dangerous Goods

## The technical boundary

It can be useful to look at what can go wrong when services are modeled incorrectly.

“The back half of the system was simply a remote procedure call (RPC) interface over a data store.”

“Making decisions to model service boundaries along technical seams isn’t always wrong. I have certainly seen this make lots of sense when an organization is looking to achieve certain performance objectives, for example. However, **it should be your secondary driver for finding these seams, not your primary one.**”



## The technical boundary

“This resulted in the need for elaborate RPC-batching mechanisms. I called this onion architecture, as it had lots of layers and made me cry when we had to cut through it.”

## The technical boundary

Rather than taking a **vertical**, *business-focused* slice through the stack, the team picked what was previously an in-process API and made a **horizontal** slice.

Again...

Making decisions to model service boundaries along technical seams isn't always wrong. However, it should be your secondary driver for finding these seams, not your primary one.