# How I learned to stop worrying and love the monolith

Nikola Zivkov
@nikolazivkov
Seavus

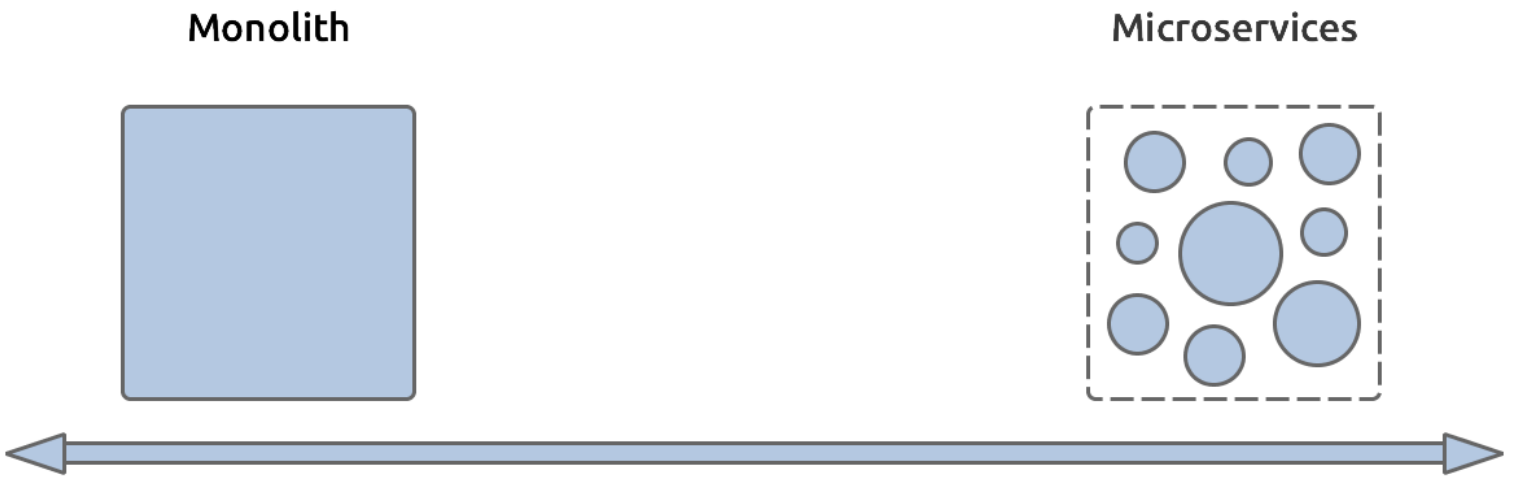
# Why monolith?

- Well known to developers
- Supported by vendors
- IDE-friendly
- Easy to develop / debug / test / deploy / operate
- Low technical complexity
  - In-process communication
  - Transactions and consistency

# Why microservices?

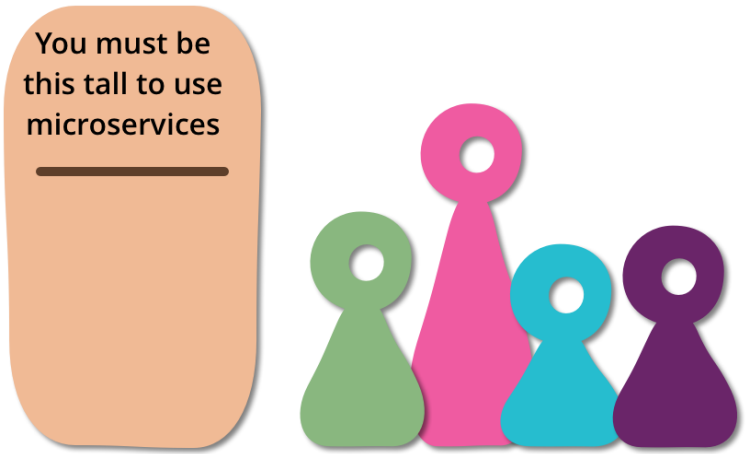
- Individually deployable / upgradeable / replaceable / scalable
- Availability / design for failure
- Polyglot programming / heterogeneous technology stack

# Monolith-first vs. microservices-first



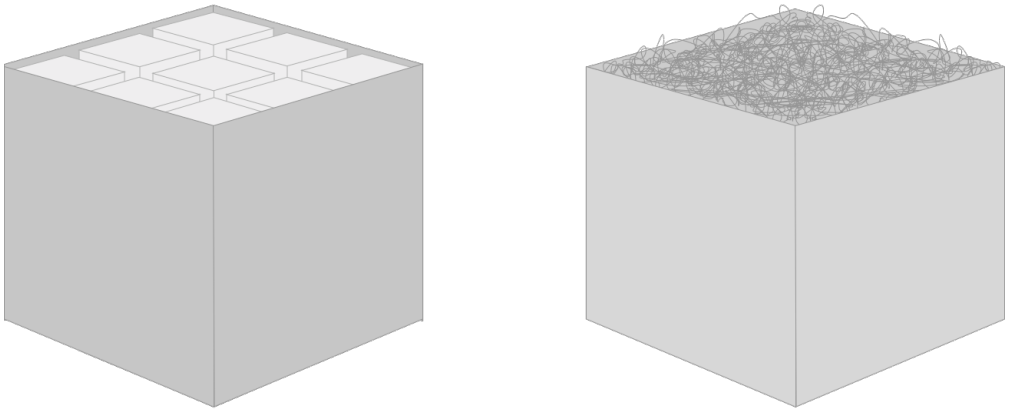
# Why monolith-first

- YAGNI – You Aren't Gonna Need It
- Service boundaries are hard to get at the beginning – refactoring is more expensive
- Microservices require DevOps:
  - Continuous delivery
  - Rapid provisioning
  - Rapid application deployment
  - Central logging
  - Monitoring
  - etc.



# Why microservices-first?

It’s hard to cut up a large (BBOM) monolith into microservices

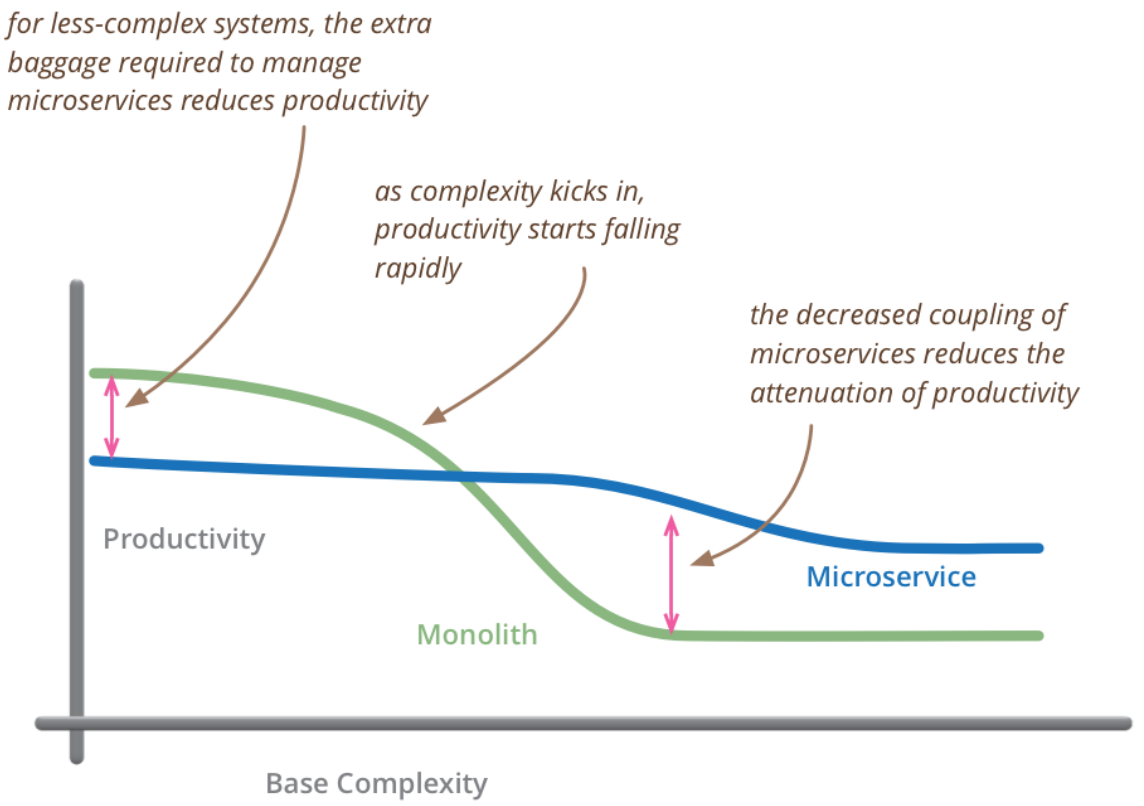


# Code entropy

*As a system is modified with the inclusion of new functionality, its disorder, or entropy, tends to increase.*

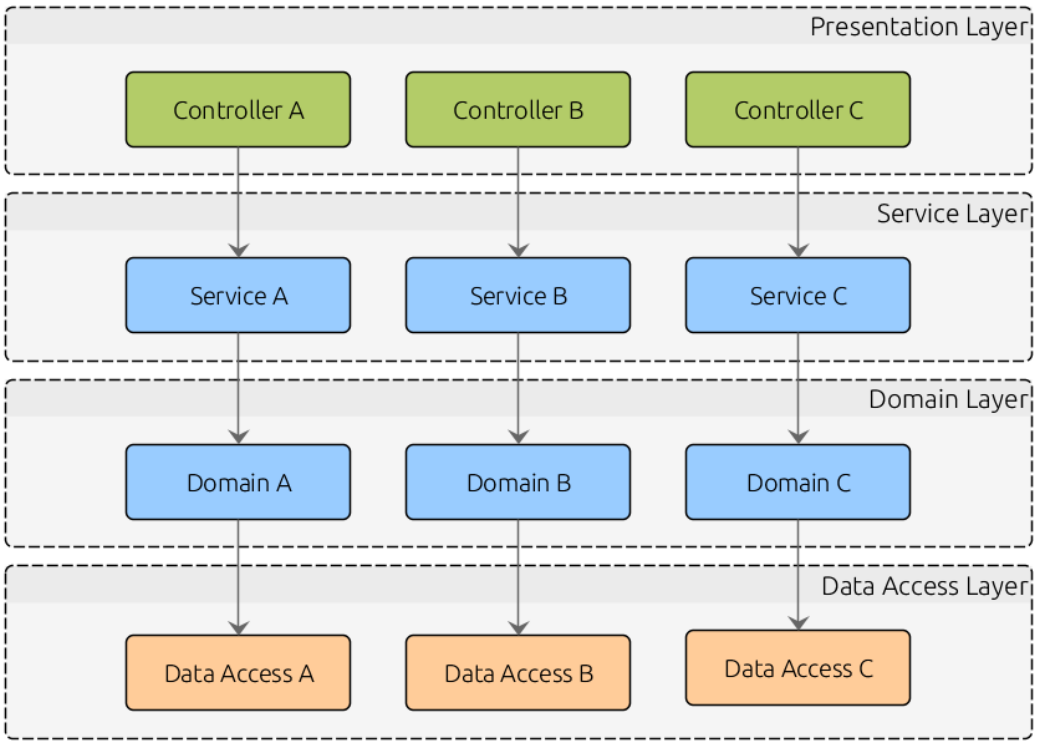
- Monolith: new code added to existing codebase
- Microservices: new code added as new service

# Productivity

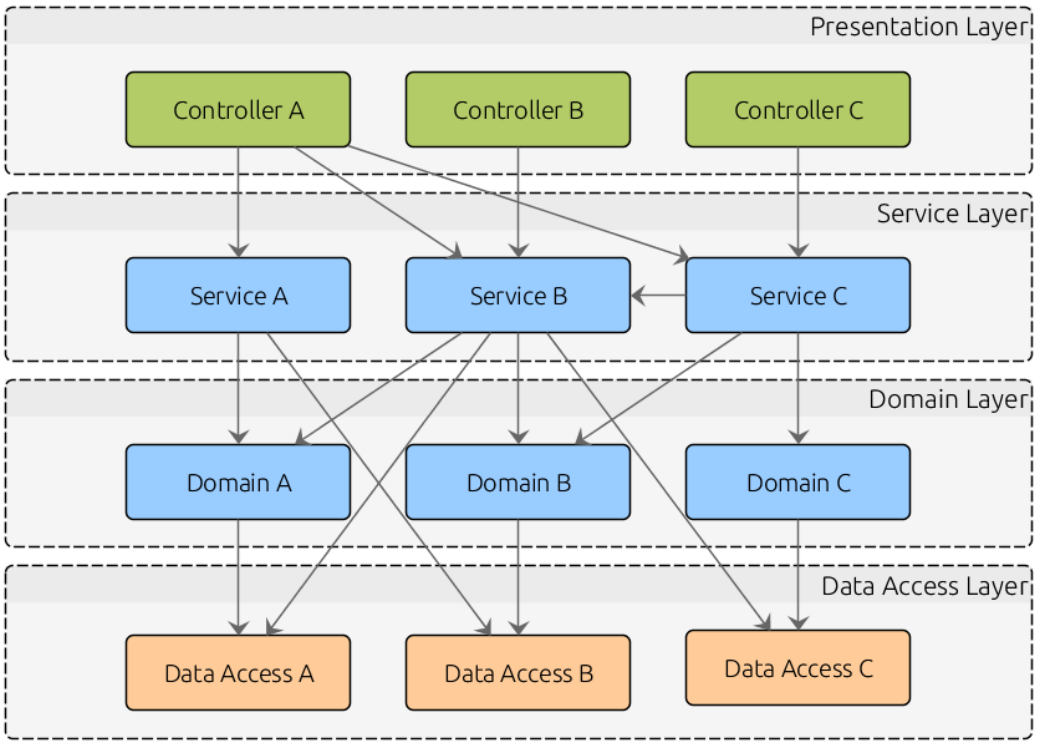


# Bookstore

# Monolith - hope

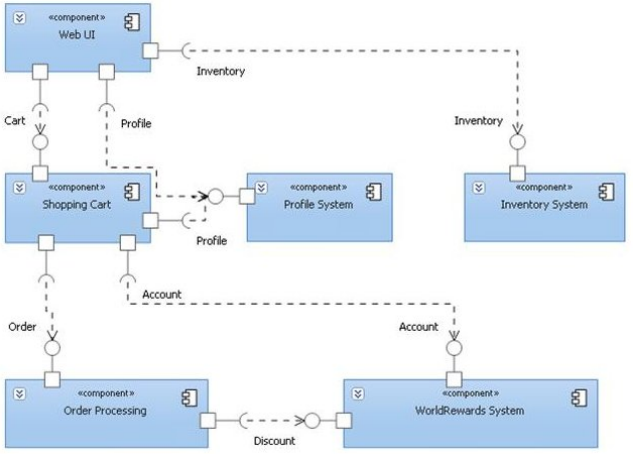


# Monolith - reality



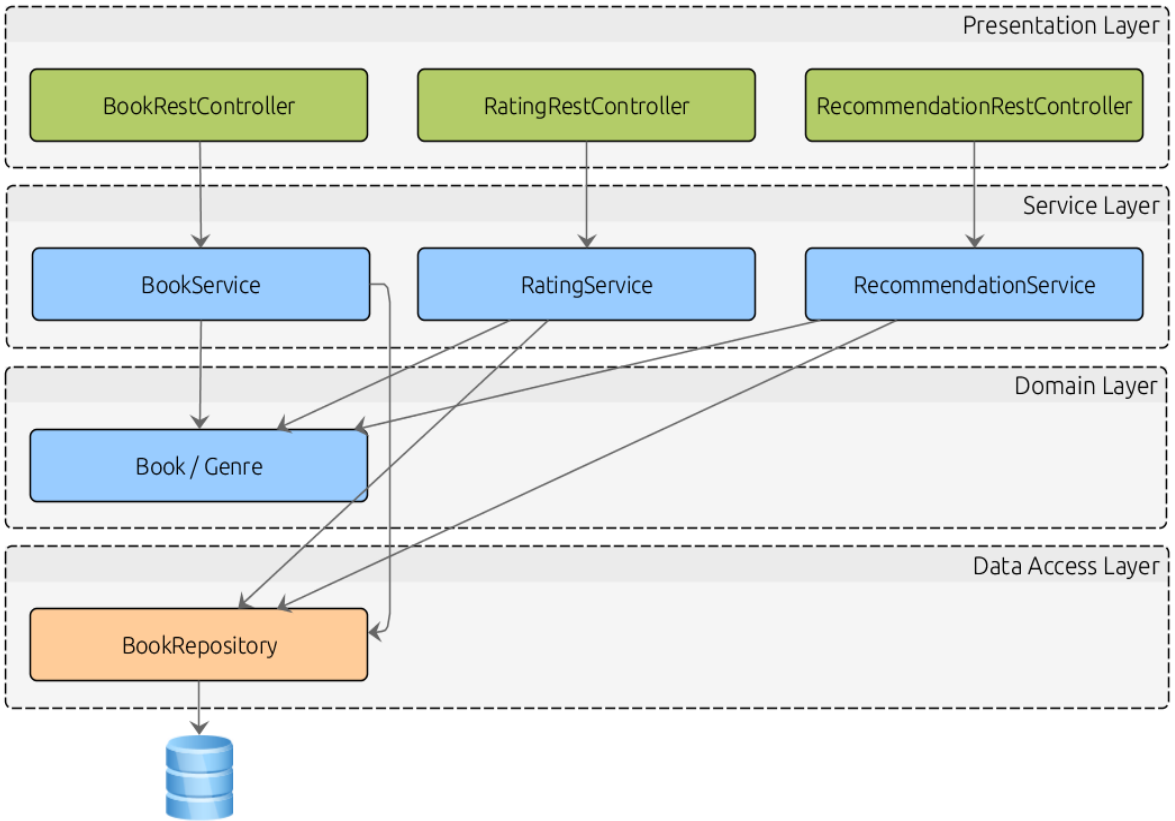
# The model-code gap

The architecture models include concepts such as component, services, modules, etc. but the code doesn't reflect this - the implementation often happens to be a bunch of classes sitting inside a traditional layered architecture.

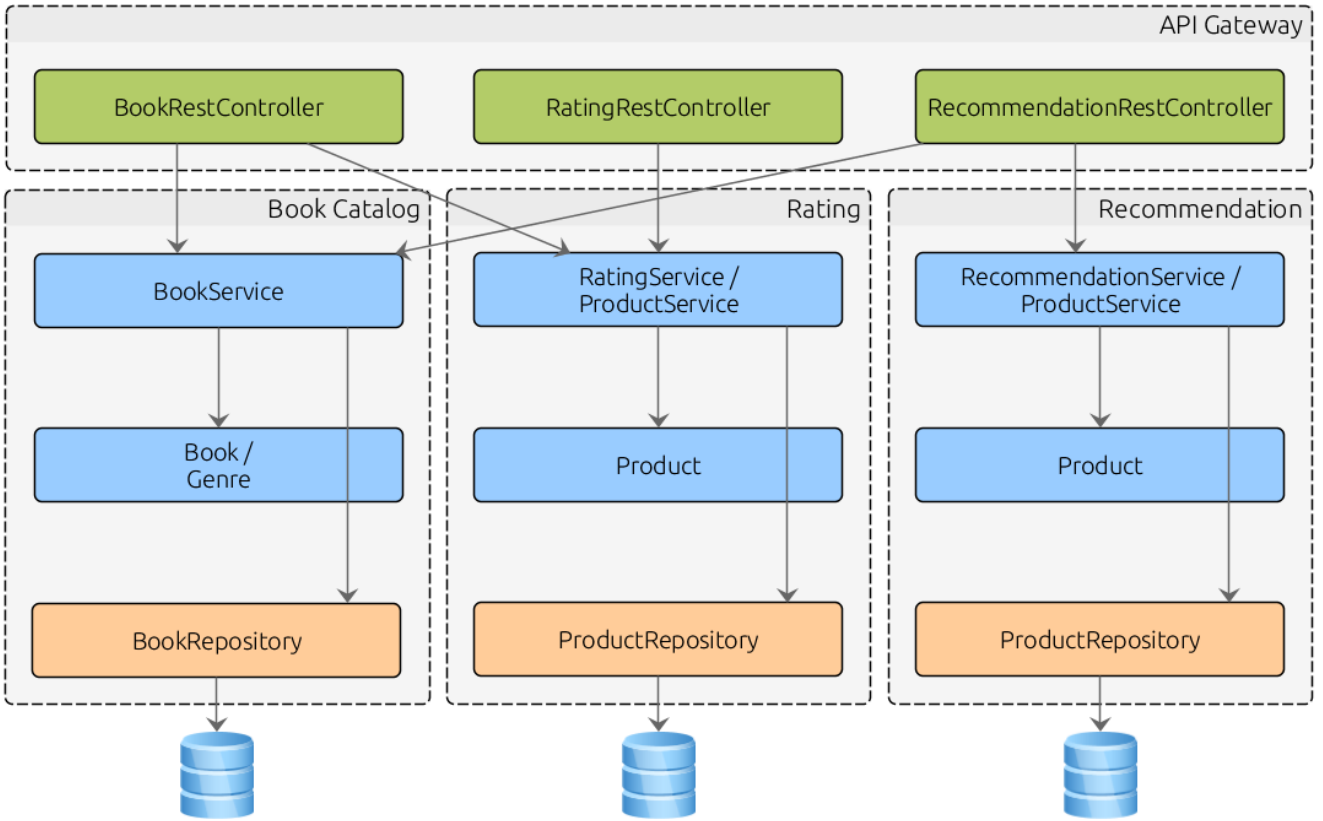



vs.

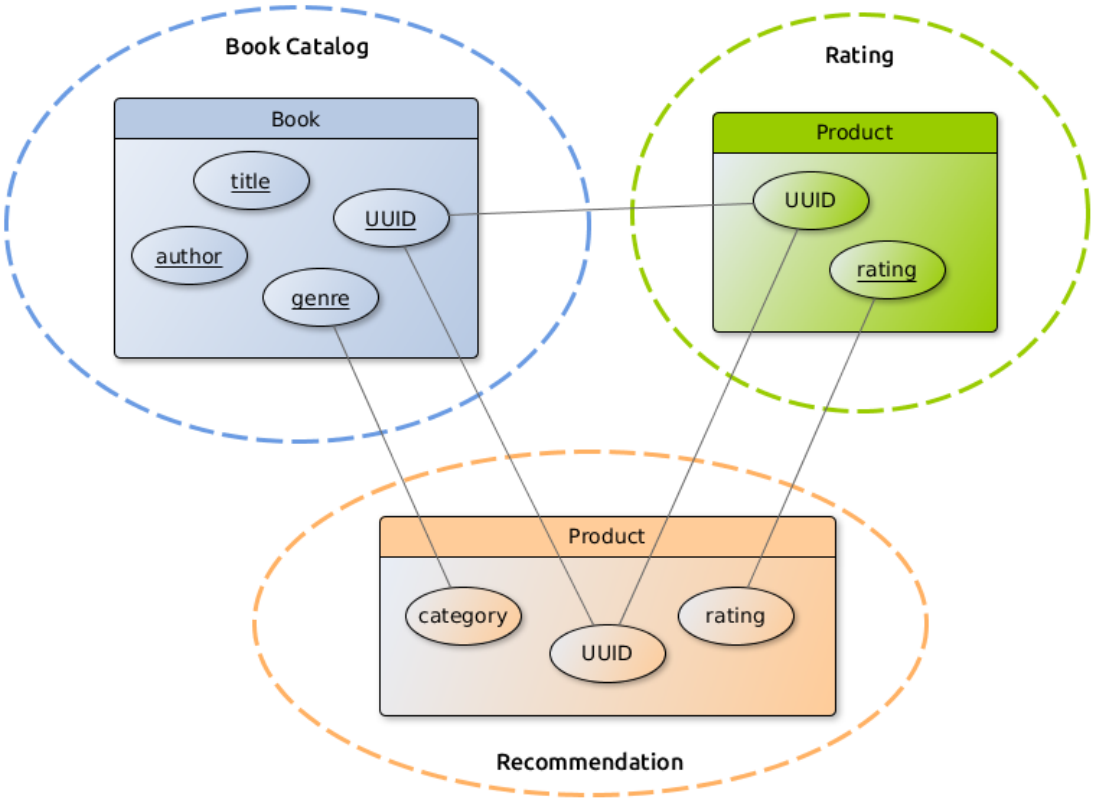
# Bookstore monolith



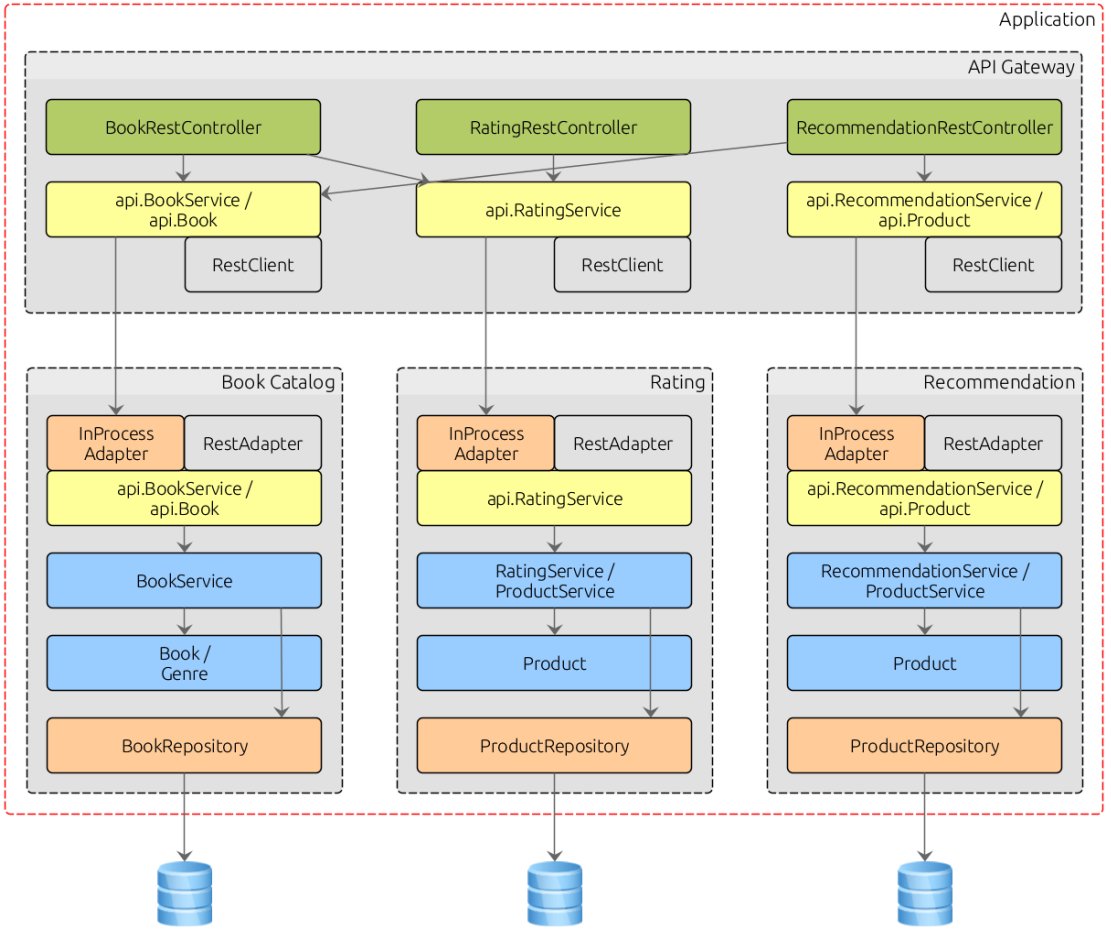
# Bookstore monolith restructured



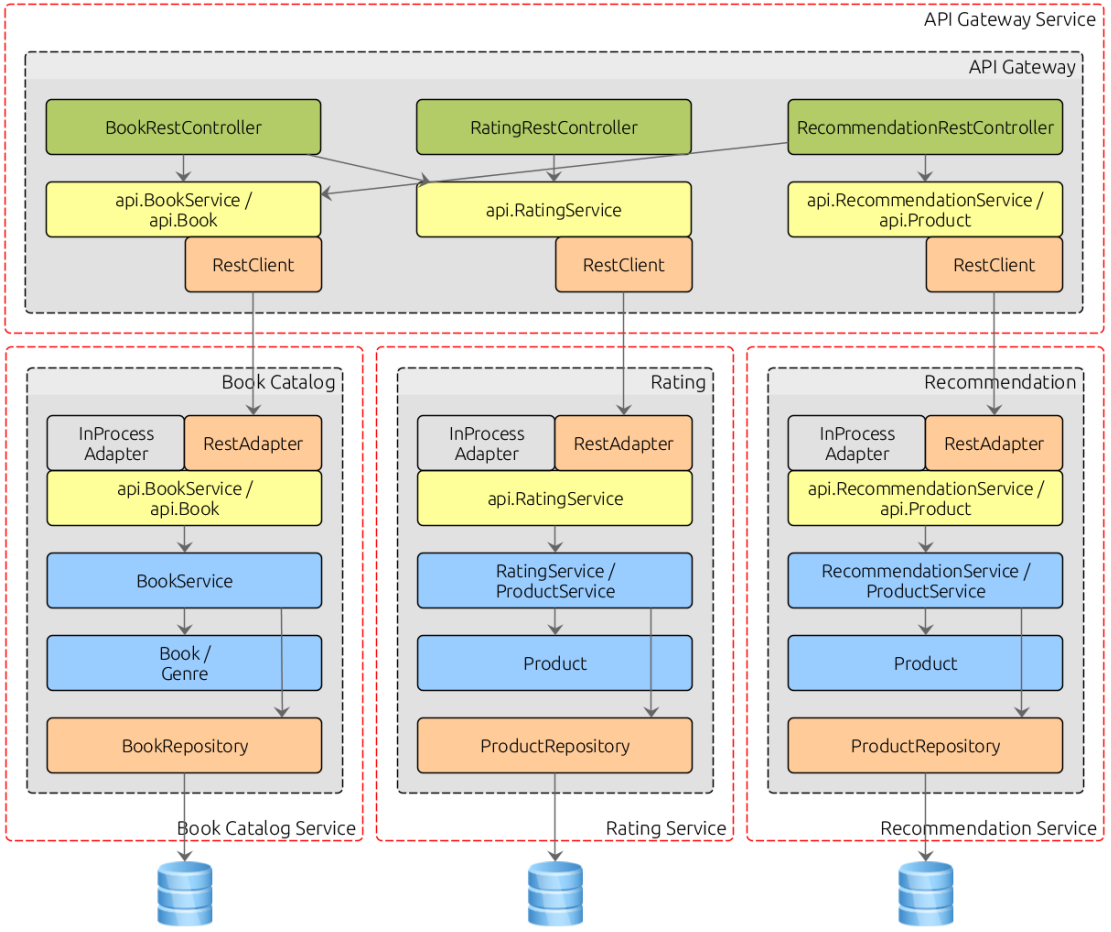
# Context map


Book Catalog
Book
title
UUID
author
genre
Rating
Product
UUID
rating
Product
category
UUID
rating
Recommendation

# Deployment as monolith

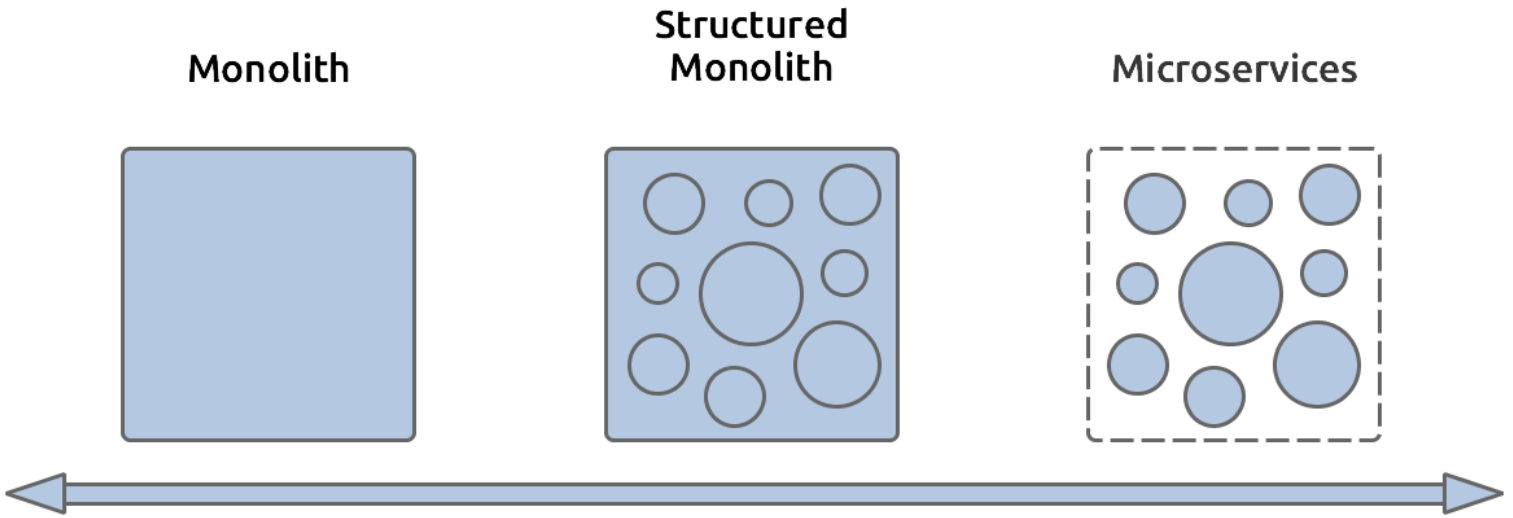


# Deployment as microservices



# Monolith to microservices

# Monolith to microservices



# Microservices architectural prerequisites

- Package structure depicting functional components rather than layers
- Domain model per component (DDD's bounded context, ubiquitous language and context map)
- Data store per component
- Domain events to propagate business-meaningful events in asynchronous manner
- Ports and adapters architecture to isolate components during synchronous calls

# To conclude

**Simon Brown**
@simonbrown

I'll keep saying this ... if people can't build monoliths properly, microservices won't help. #qconlondon #DesignThinking #Modularity

RETWEETS 244 LIKES 100

10:49 AM - 4 Mar 2015

Monolithic

Microservices

# Thank You