Architecting Microservices

Mark Heath
CLOUD ARCHITECT

@mark_heath www.markheath.net

Overview

Evolving towards microservices

Microservices are autonomous

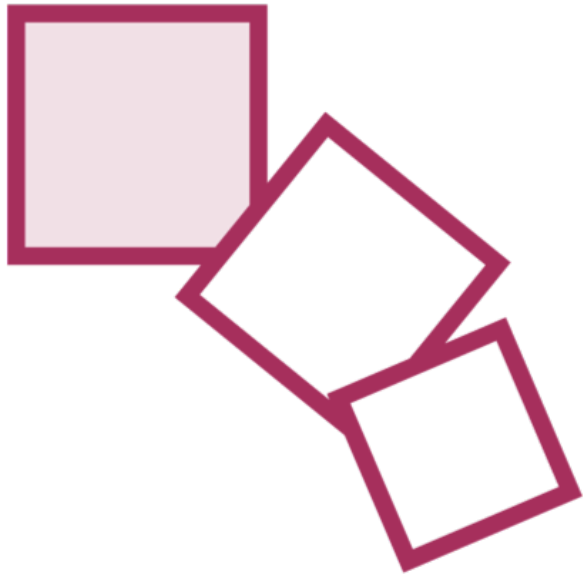
Microservices own their data

Microservices are independently deployable

Identifying microservice boundaries

What if I already have an existing (monolithic) application?

Evolving towards Microservices



Augment a monolith

- Add new microservices

Decompose a monolith

- Extract microservices

You don’t need to start with microservices

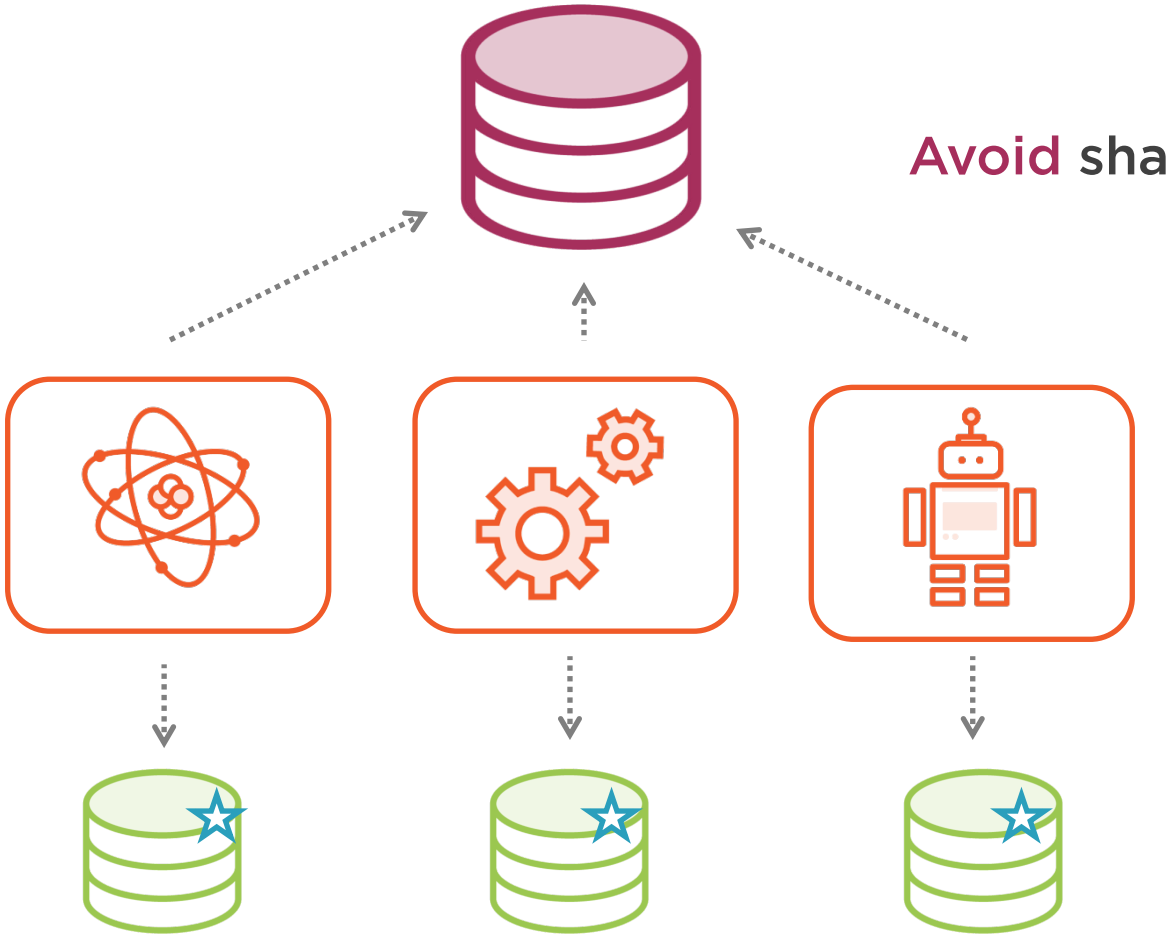
- It’s hard to get service boundaries right

Defining microservice responsibilities

- Public interface

Microservices own their own data

Microservice Data Ownership



Avoid sharing a data store

Limitations:

- Database joins
- Transactions
- Eventual consistency

Data store per microservice

Mitigating Data Ownership Limitations

Define service boundaries well

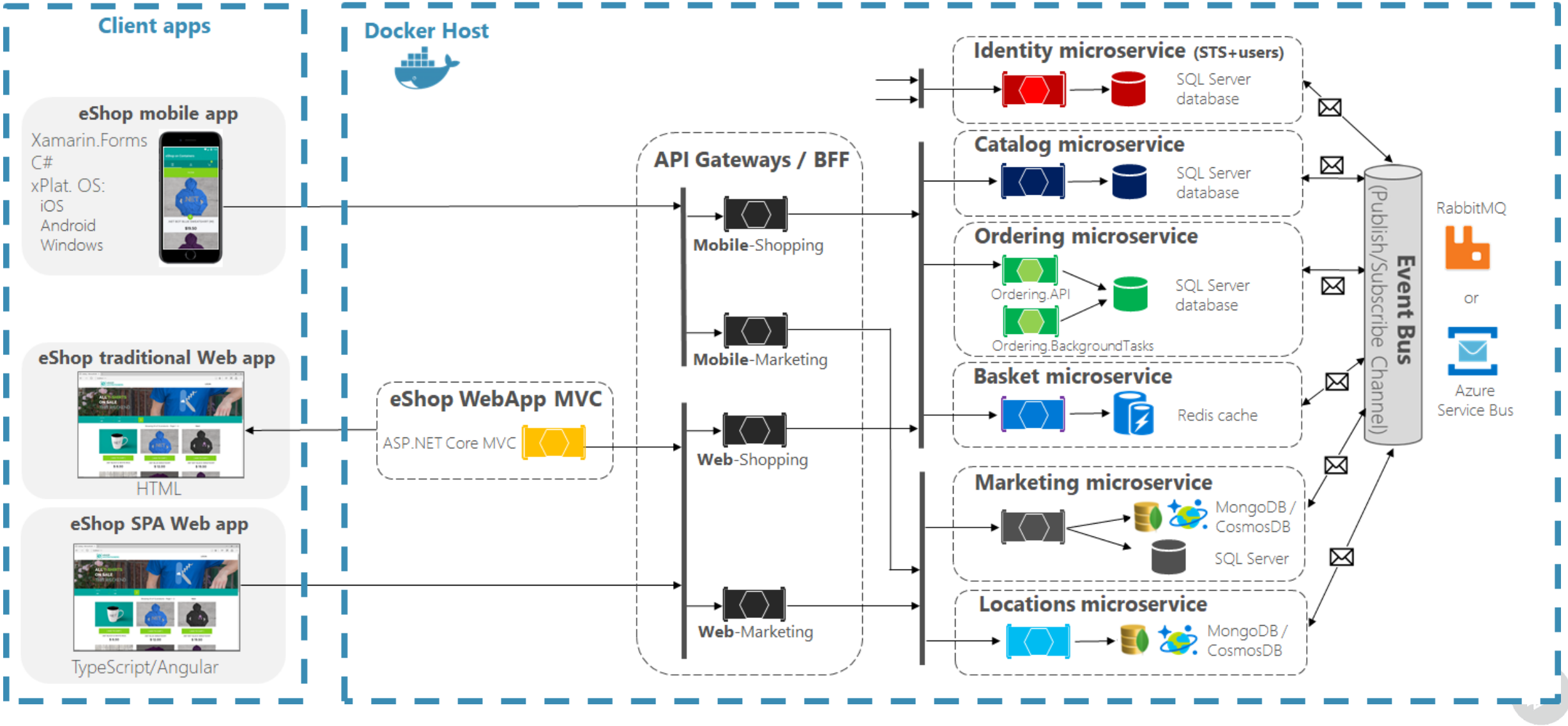
- Minimize the need to aggregate data

Caching

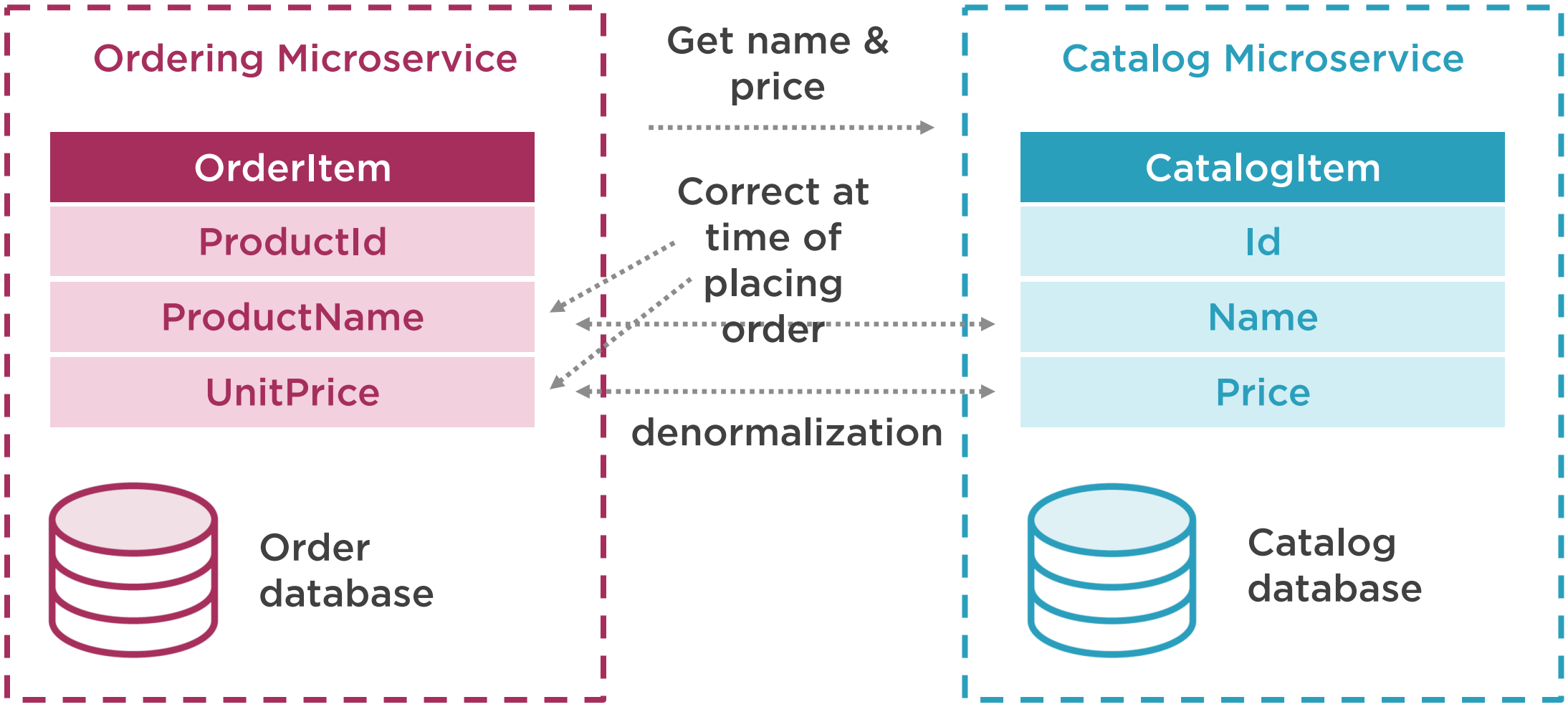
- Improved performance
- Improved availability

Identify “seams” in the database

eShopOnContainers Architecture

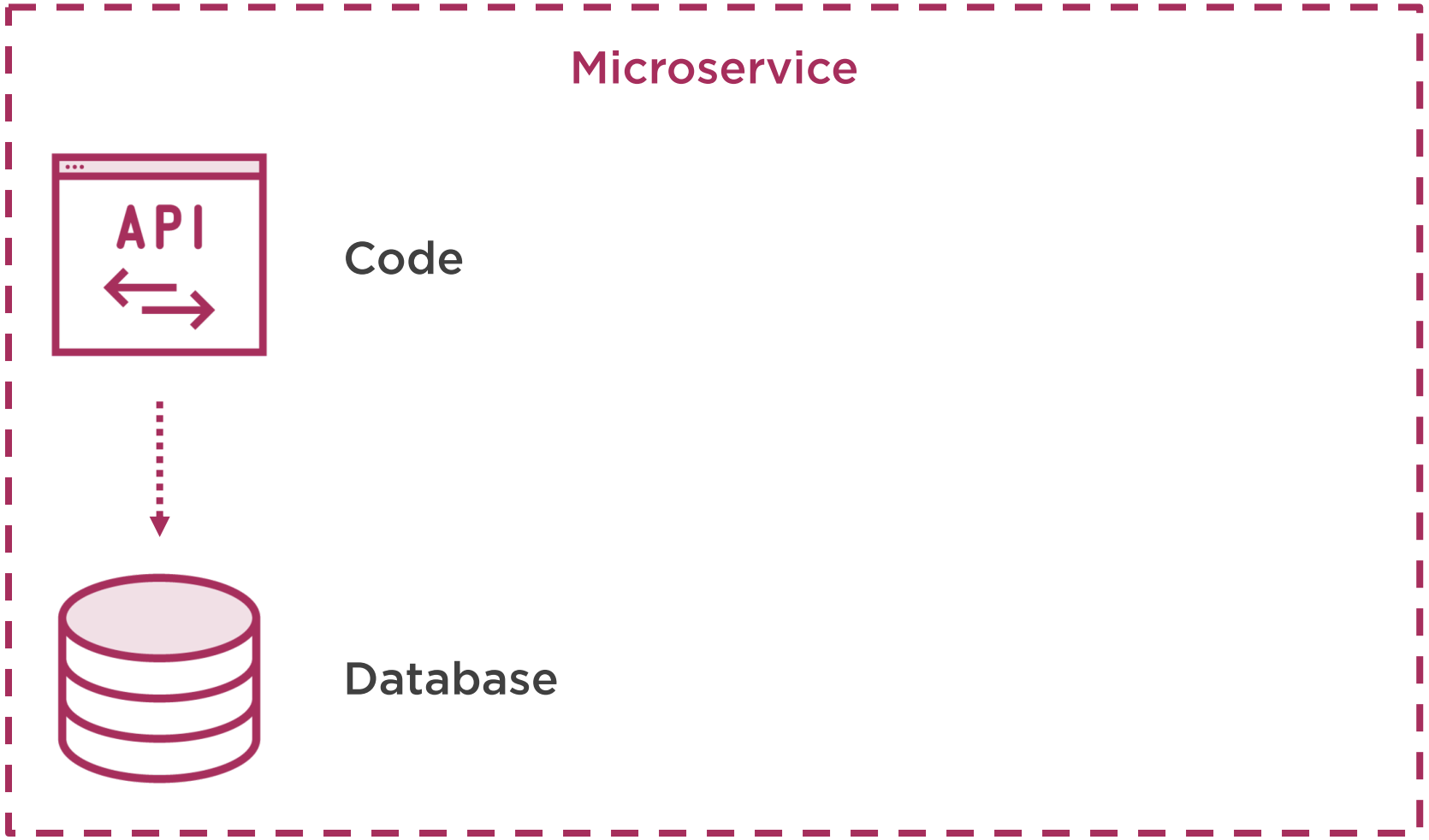


Duplicating Data

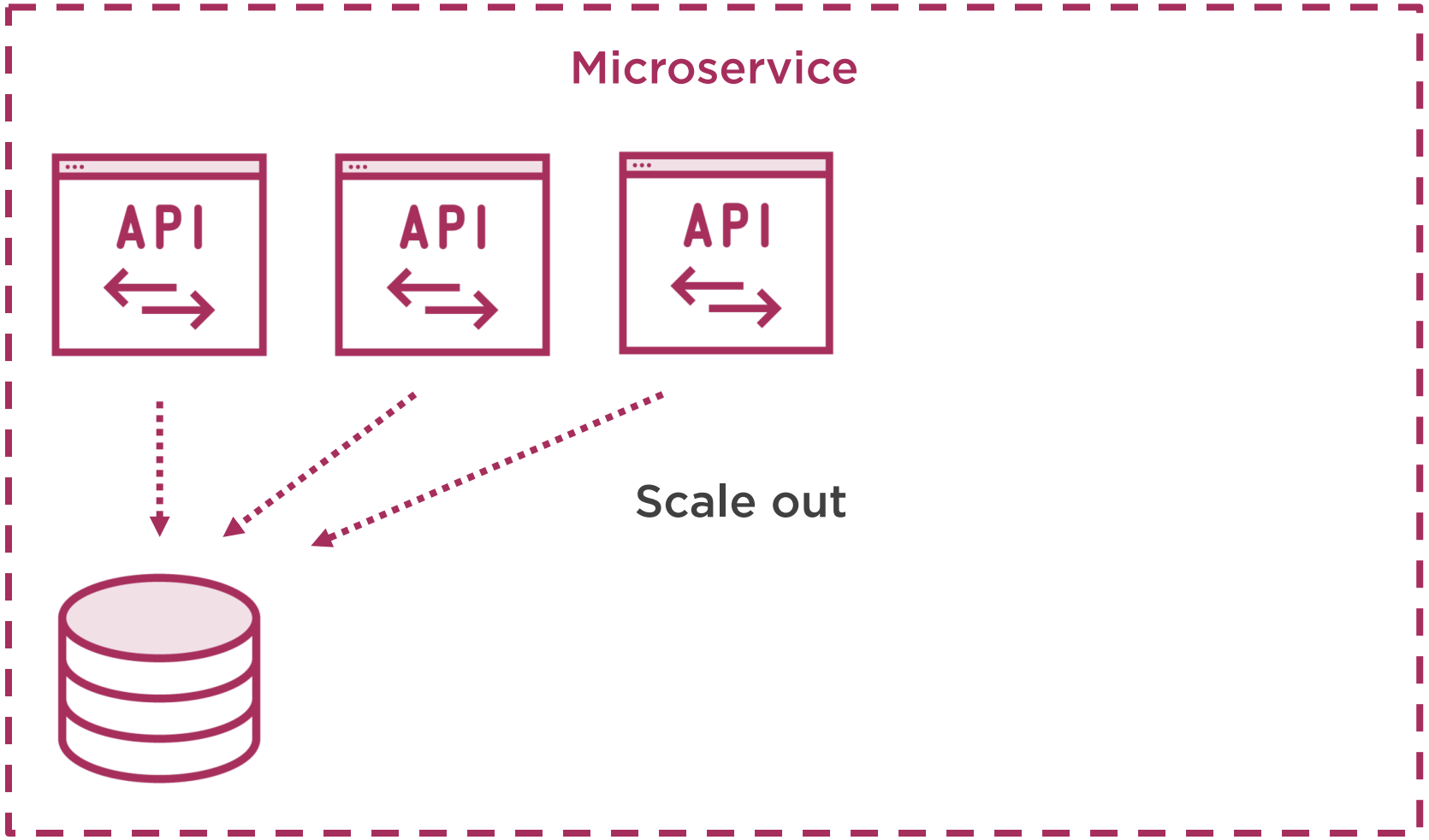


Microservices can consist of more than one process

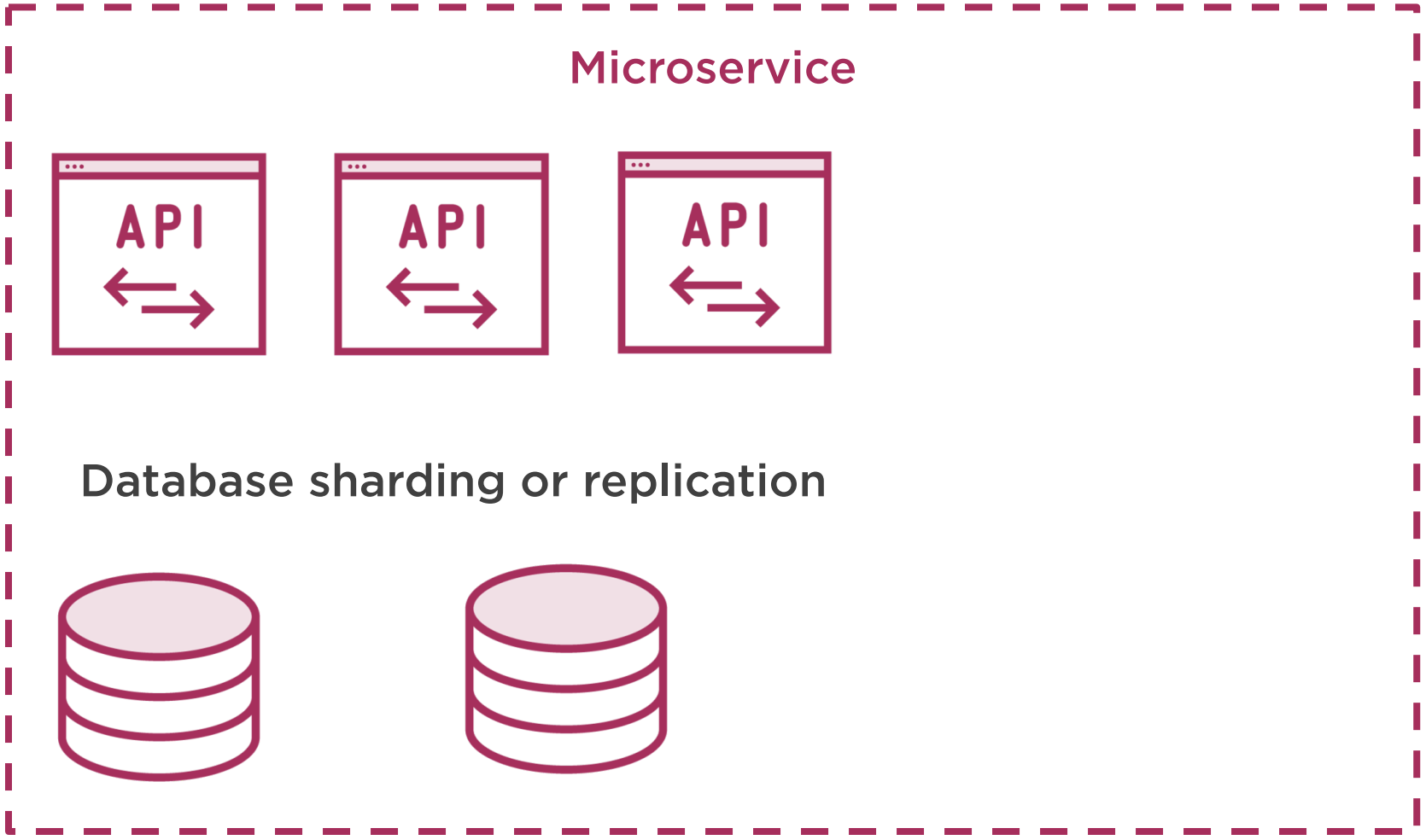
Microservice Components



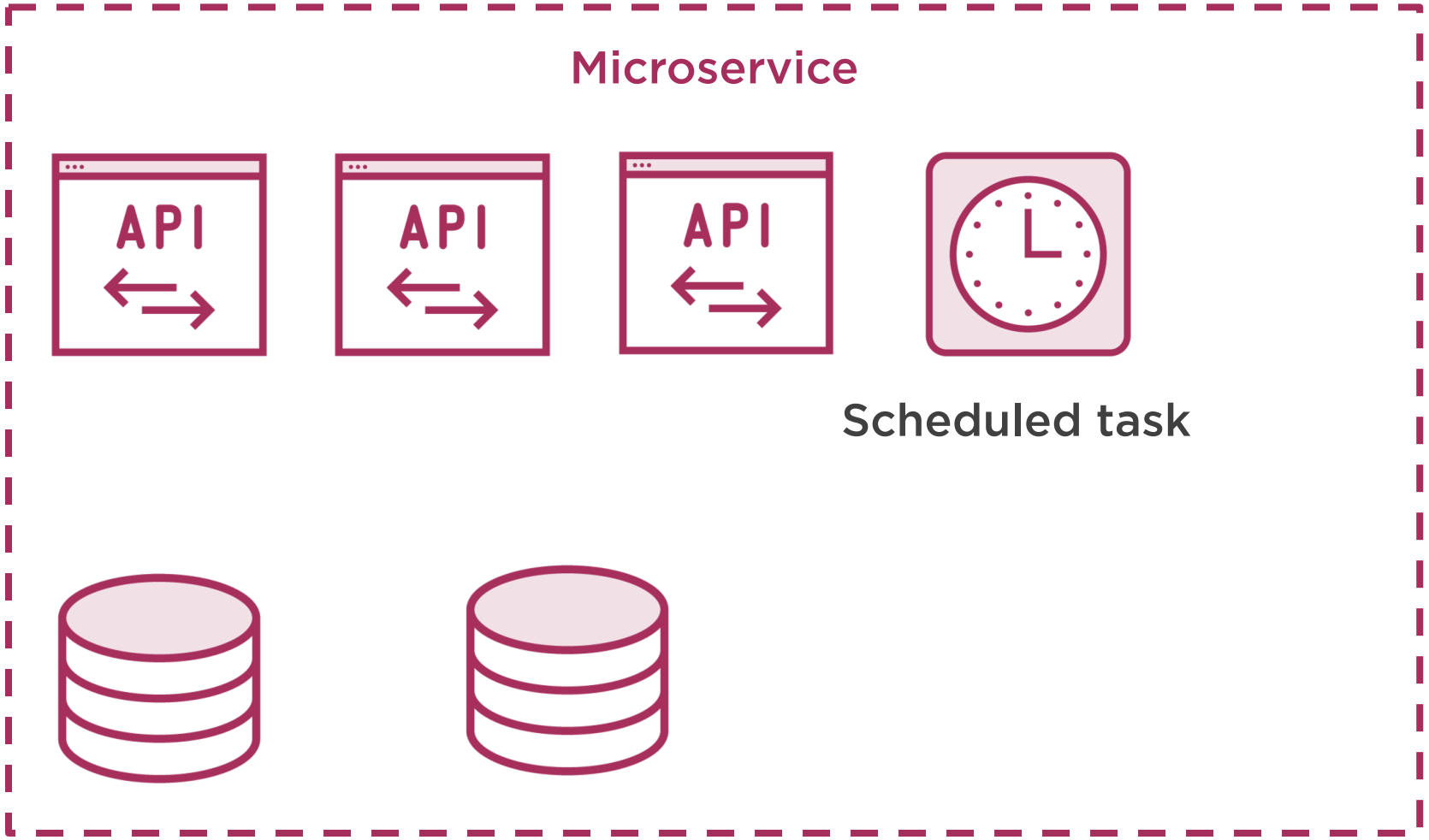
Microservice Components



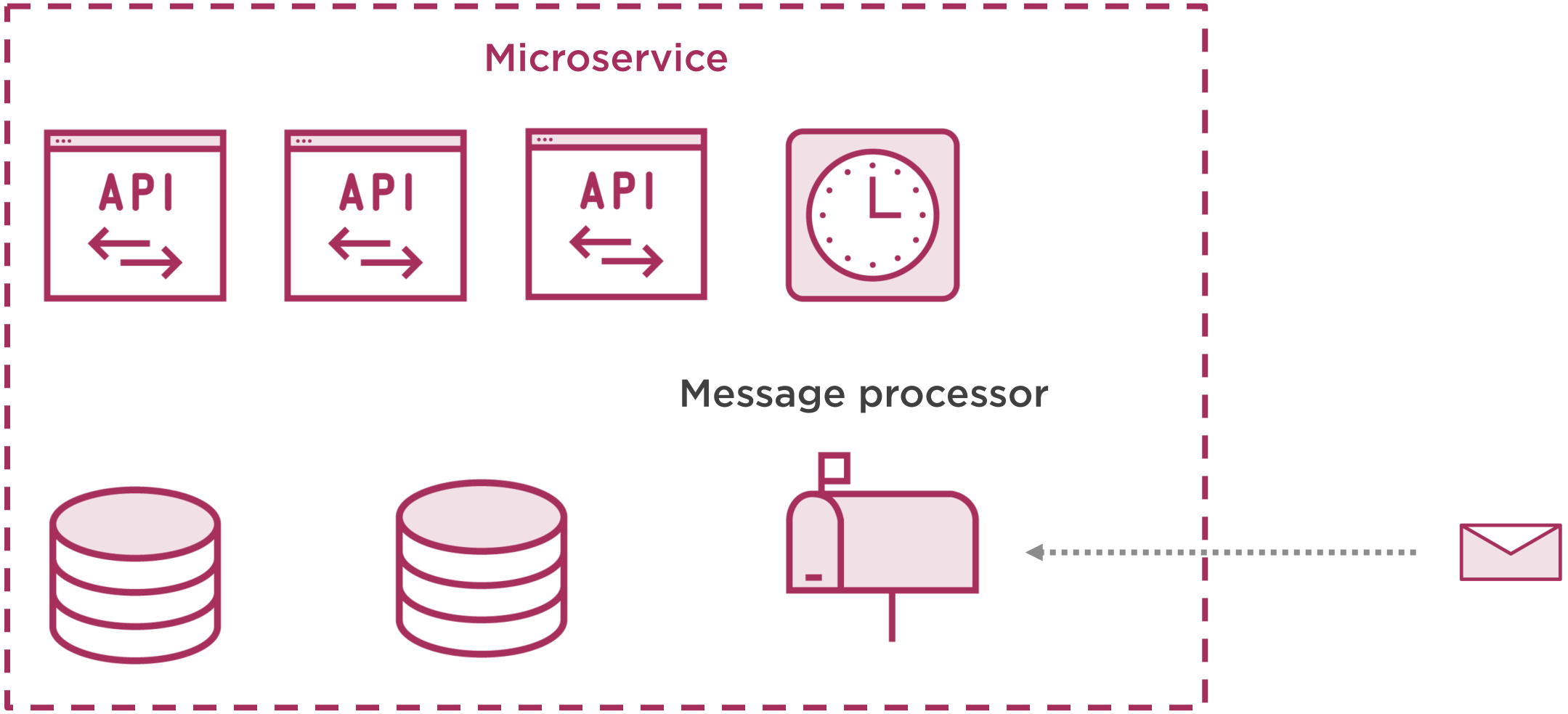
Microservice Components



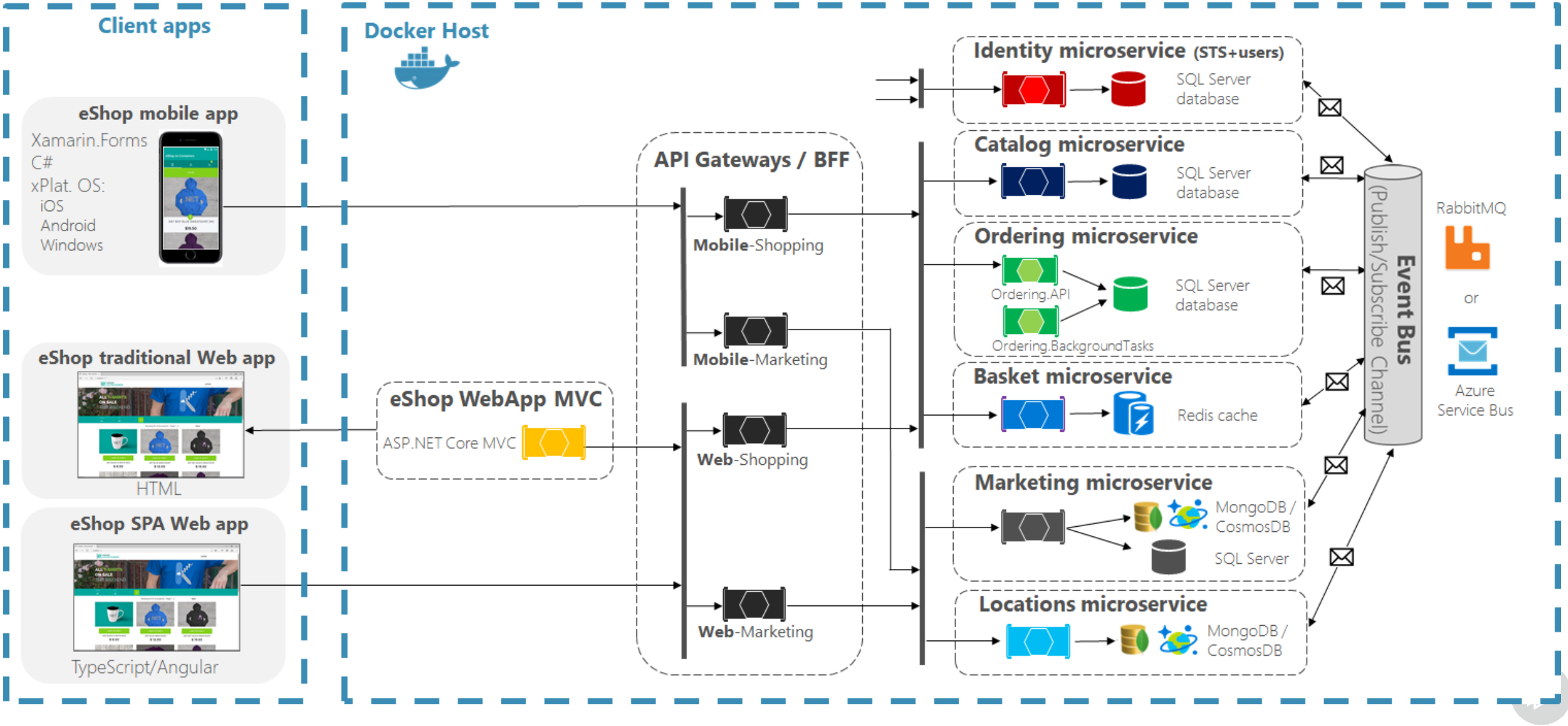
Microservice Components



Microservice Components



eShopOnContainers Architecture



Microservices should have a clearly defined public interface

Microservices can be upgraded without their clients needing to upgrade

Microservice Contracts

Make additive changes

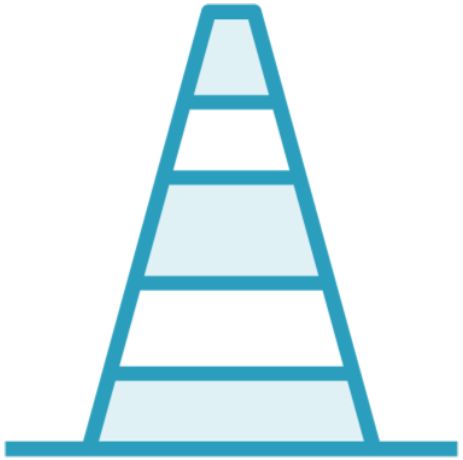
- New endpoints
- New properties on DTOs

Introduce version 2 API

- Version 1 clients must still be supported

Easily forgotten in development

Avoiding Upgrade Issues



Team ownership of microservices

- First, add a new capability
- Then, deploy updated microservice
- Later, update clients

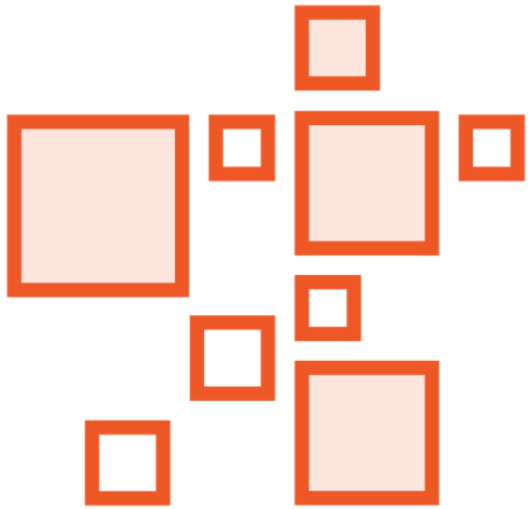
Create automated tests

- Ensure that older clients are supported
- Run as part of a CI build process

Beware of shared code

- Can result in tight coupling

Identifying Microservice Boundaries



Getting it wrong can be costly

- Poor performance
- Hard to change

Start from an existing system

- Identify loosely coupled components
- Identify database seams

Organize microservices around business capabilities

Domain Driven Design

Identify “bounded contexts”

- Break up large domains
- “Ubiquitous language”
- Microservices do not share models
- e.g. OrderItem and CatalogItem

Sketch your ideas on a whiteboard

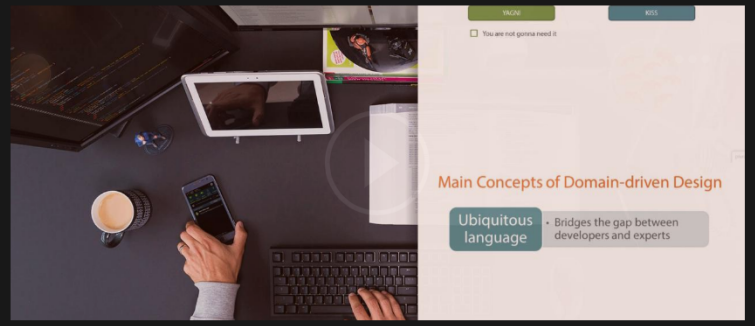
- Run them through real-world use cases
- Identify potential problems

Product Representations

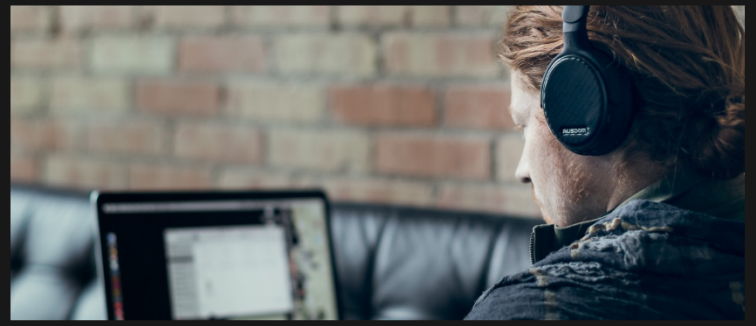
```
class OrderItem
{
    string ProductId;
    Guid? OrderId;
    decimal UnitPrice;
    string ProductName;
    string PictureUrl;
    int Quantity;
    decimal Discount;
}
```

```
class CatalogItem
{
    int Id;
    decimal Price;
    string Name;
    string PictureUri;
}
```

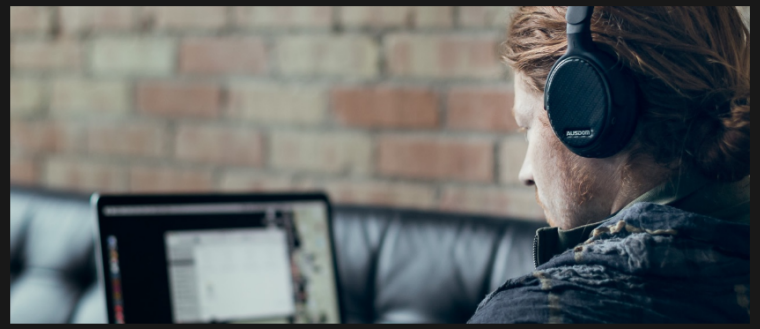
Learn More About Domain Driven Design



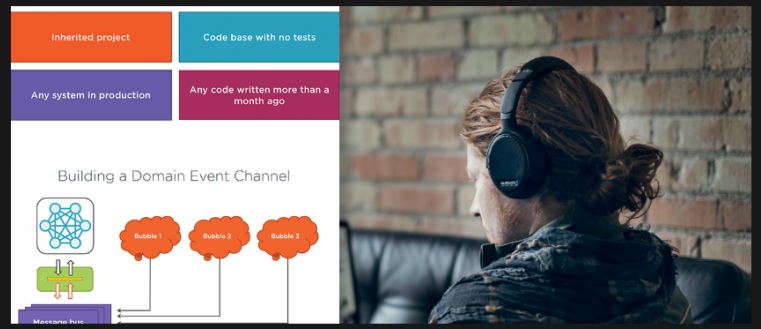
Domain-Driven Design in Practice
Vladimir Khorikov
Intermediate · 4h 19m · Sept 16, 2019



Refactoring from Anemic Domain Model Towards a Rich One
Vladimir Khorikov
Intermediate · 3h 36m · Nov 13, 2017



Specification Pattern in C#
Vladimir Khorikov
Intermediate · 1h 27m · June 27, 2017



Domain-Driven Design: Working with Legacy Projects
Vladimir Khorikov
Intermediate · 3h 51m · Mar 27, 2018

Microservice Boundary Pitfalls

Don’t turn every noun into a microservice

- Anemic CRUD microservices
- Thin wrappers around databases
- Logic distributed elsewhere

Avoid circular dependencies

Avoid chatty communications

eShopOnContainers Service Boundaries

Catalog

Browsing for products

Large dataset

Support flexible queries

Non-sensitive data

Basket

Preparing for purchase

Short-lived data

Redis cache

Ordering

Processing orders

Mostly writes new data

Reliability is vital

Highly sensitive data

Temporarily Unavailable

Identity

Authentication

Marketing

Email campaigns

Location

Customer locations

How would you break your application into microservices?

Summary

- Evolving towards microservices
- Microservices own their own data
- Microservices may consist of multiple processes
- Microservices should be independently deployable
- Avoid breaking changes
- Identify “bounded contexts” for microservice boundaries
- Getting boundaries right is important

Up next...

Building microservices