



Modernizing Applications with Containers and Orchestrators

Microsoft Services





Module 4 - Microservices and Containers

Microsoft Services



Objectives

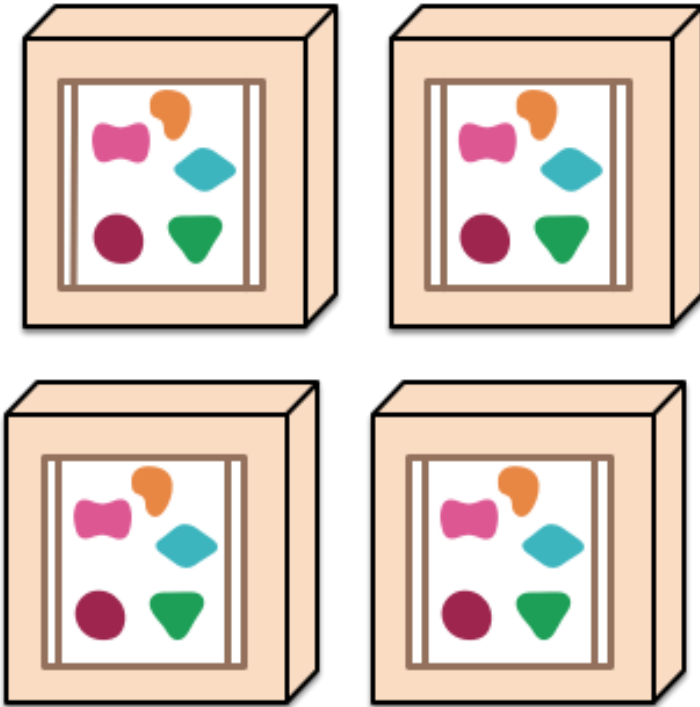
- What are Microservices?
- Microservices Patterns
- Microservices Real World Case Studies
- Microsoft Platform and Microservices
- Containers & Microservices
- Demo

Microservices Architecture

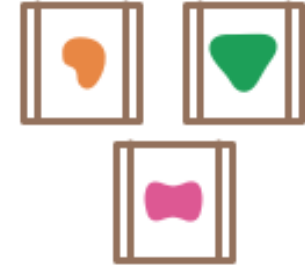
A monolithic application puts all its functionality into a single process...



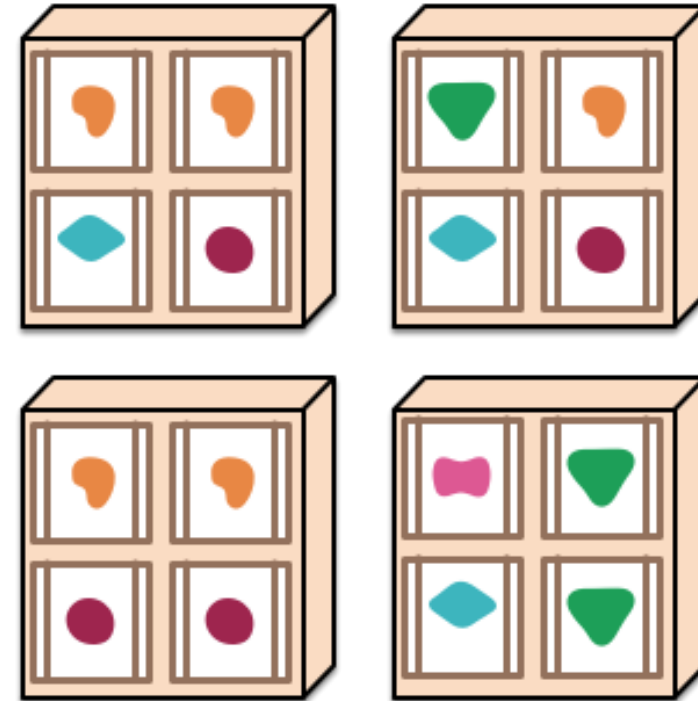
... and scales by replicating the monolith on multiple servers



A microservices architecture puts each element of functionality into a separate service...

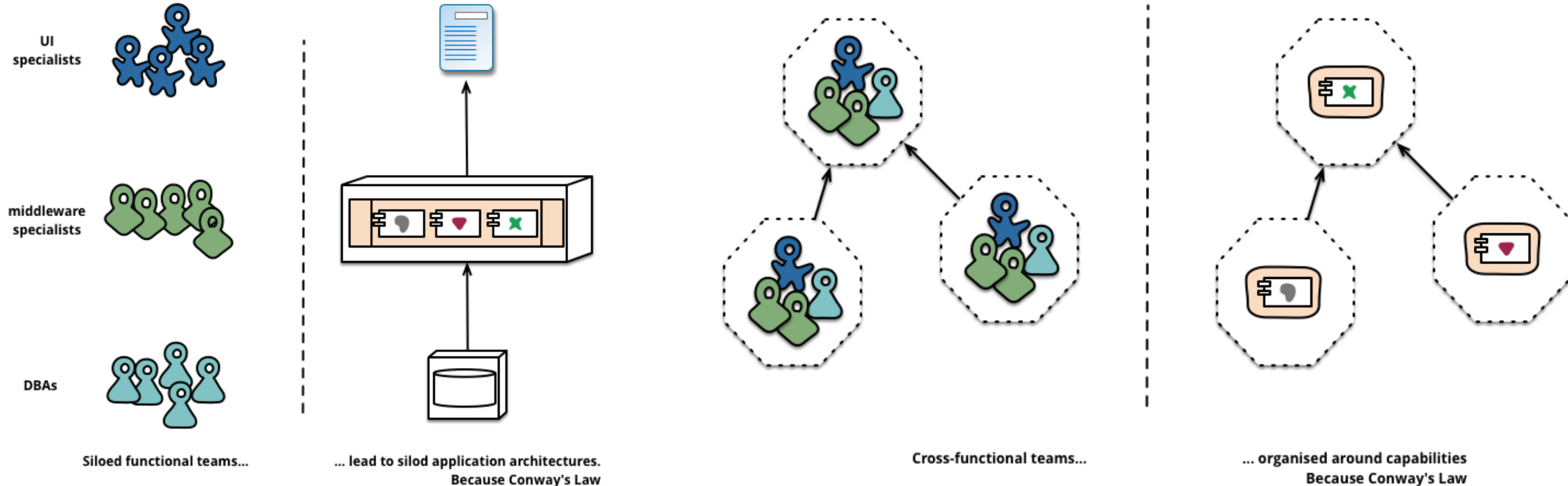


... and scales by distributing these services across servers, replicating as needed.



Microservices Architecture (Cont.)

"Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure." -- Melvyn Conway, 1967



Monolithic

Microservices

SOA and Microservices

SOA

- Services are interfaces of a large monolith
- Orchestration is often required and tend to contain business logic
- Spans across the enterprise

Microservices

- Services are individually developed and deployed
- Does not require integration technology
- Logic resides in microservices
- Can be limited to an individual project

Microservices Design Patterns

API Gateway

Challenges this pattern solves:

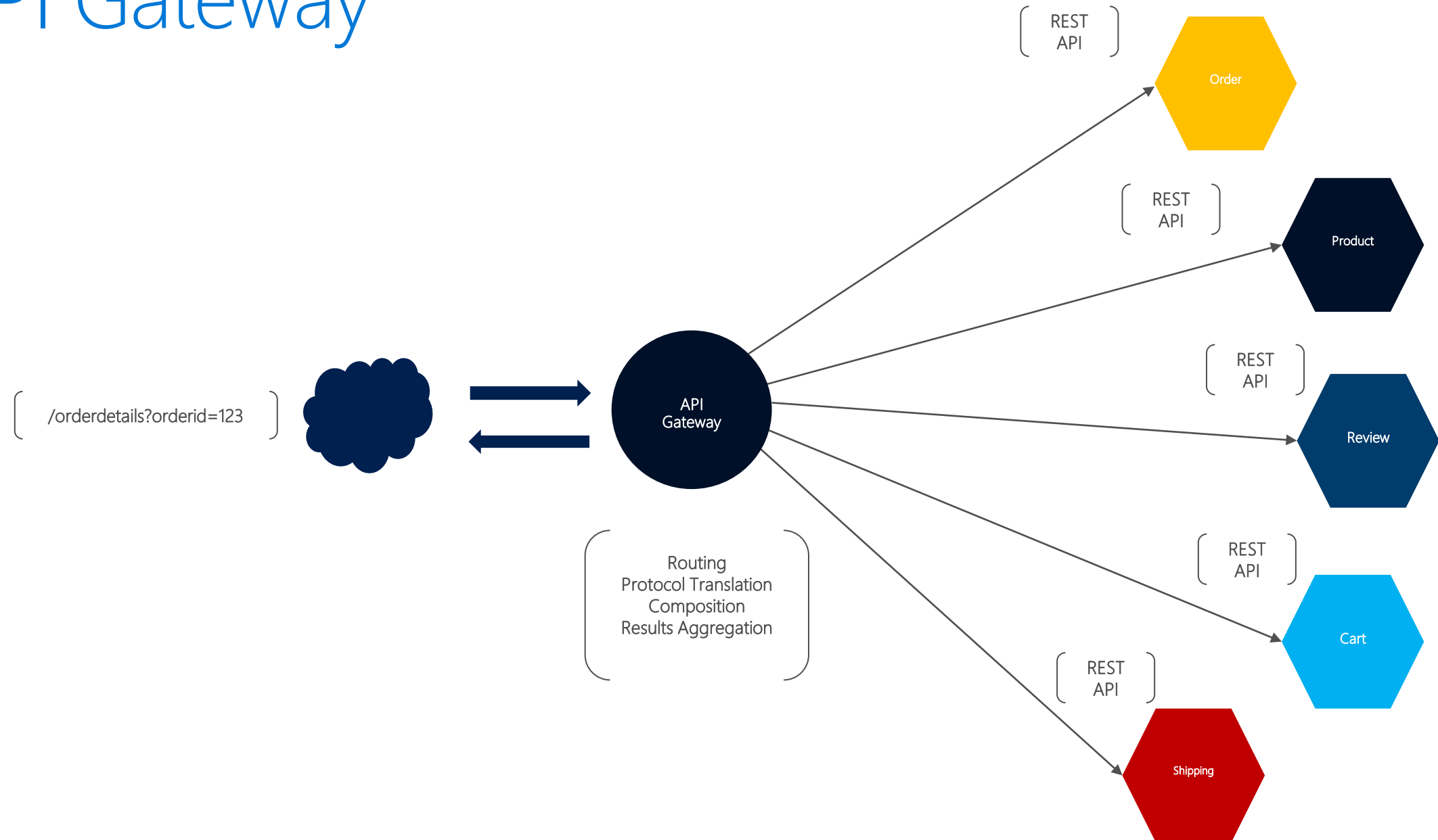
- Granularity of services is often more fine grained than what clients would need
- Different type of clients need different data
- Protocol used by services differ greatly. E.g. AMQP, WebSocket etc.
- Partitioning of services should be hidden from the clients

API Gateway

Solution:

- API Gateway acts as an entry point for all access to Microservices by encapsulating the internal system design and provides:
 - API that is tailored for each client
 - Security features such as authentication, token cache etc.
 - Protocol transition
 - Load balancing

API Gateway



Service Discovery

Challenges this pattern solves:

- Services address change dynamically due to auto scaling
- Discovering services is inherently more challenging as more services are added

Solution

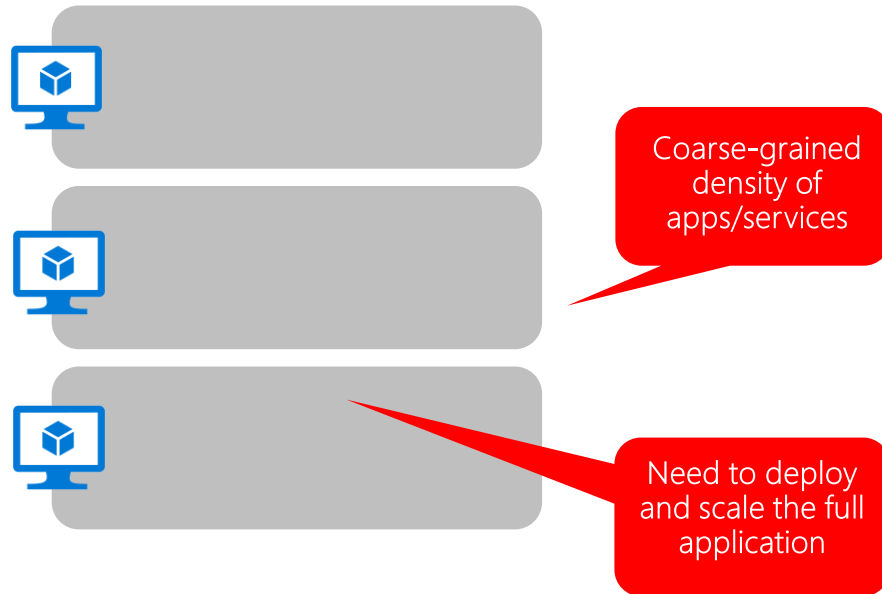
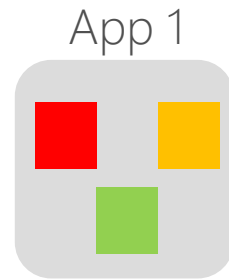
Discover services dynamically using service registry (database of available services) that will locate the instance of service to call

- Client Side Discovery
- Server Side Discovery

Microservices Design Patterns in Practice

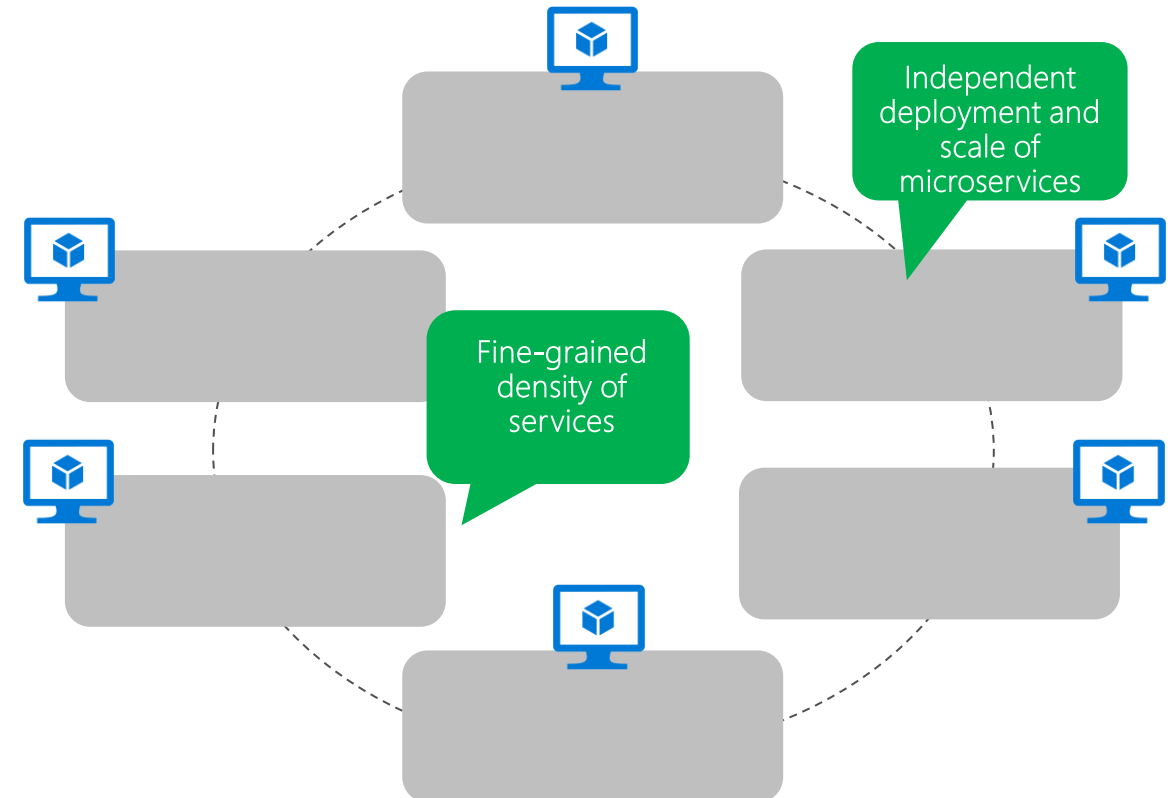
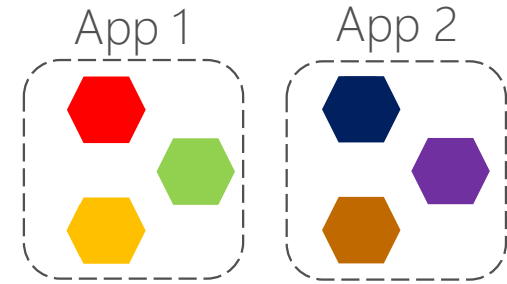
Traditional application approach

- A traditional application has most of its functionality within a few processes that are componentized with layers and libraries.
- Scales by cloning the app on multiple servers/VMs



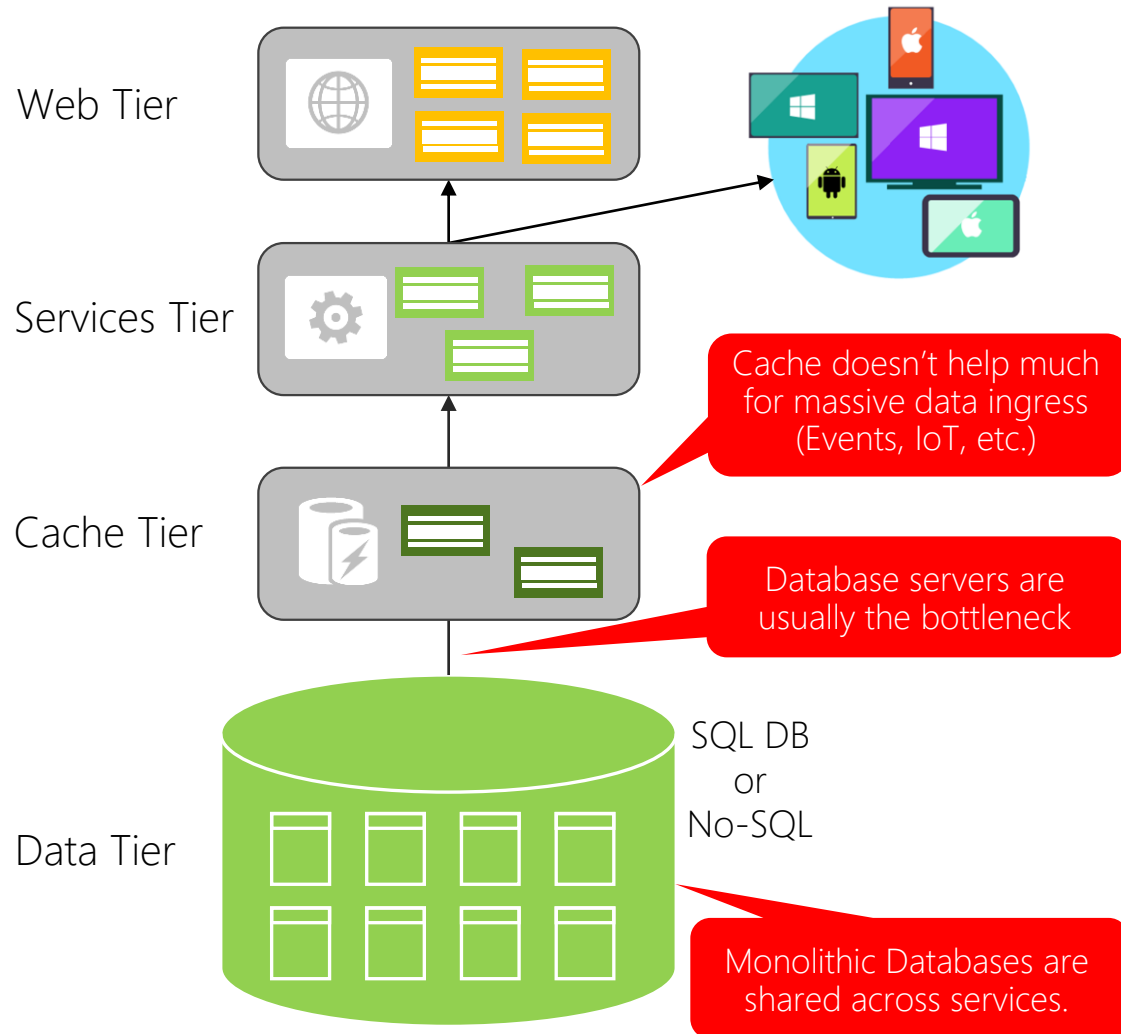
Microservices application approach

- A microservice application segregates functionality into separate smaller services.
- Scales out by **deploying each service independently** with multiple instances across servers/VMs



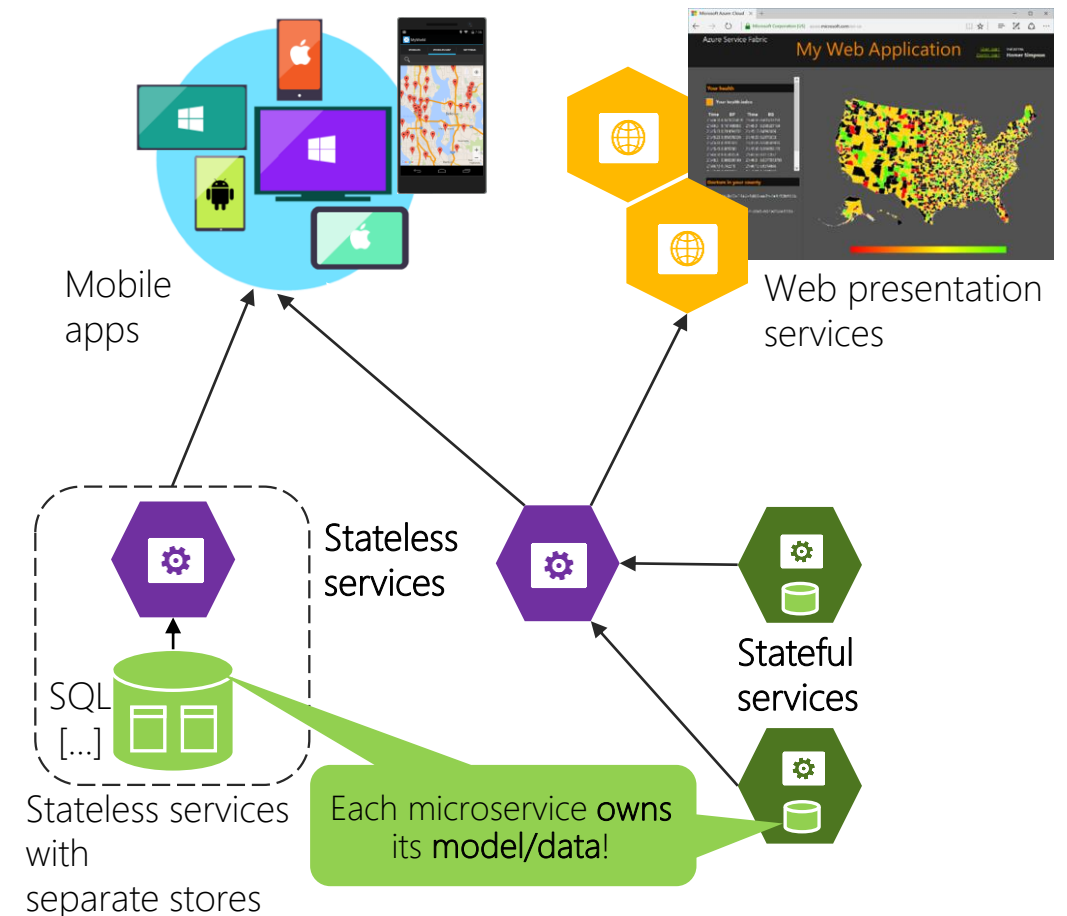
Data in Traditional approach

- Single monolithic database
- Tiers of specific technologies

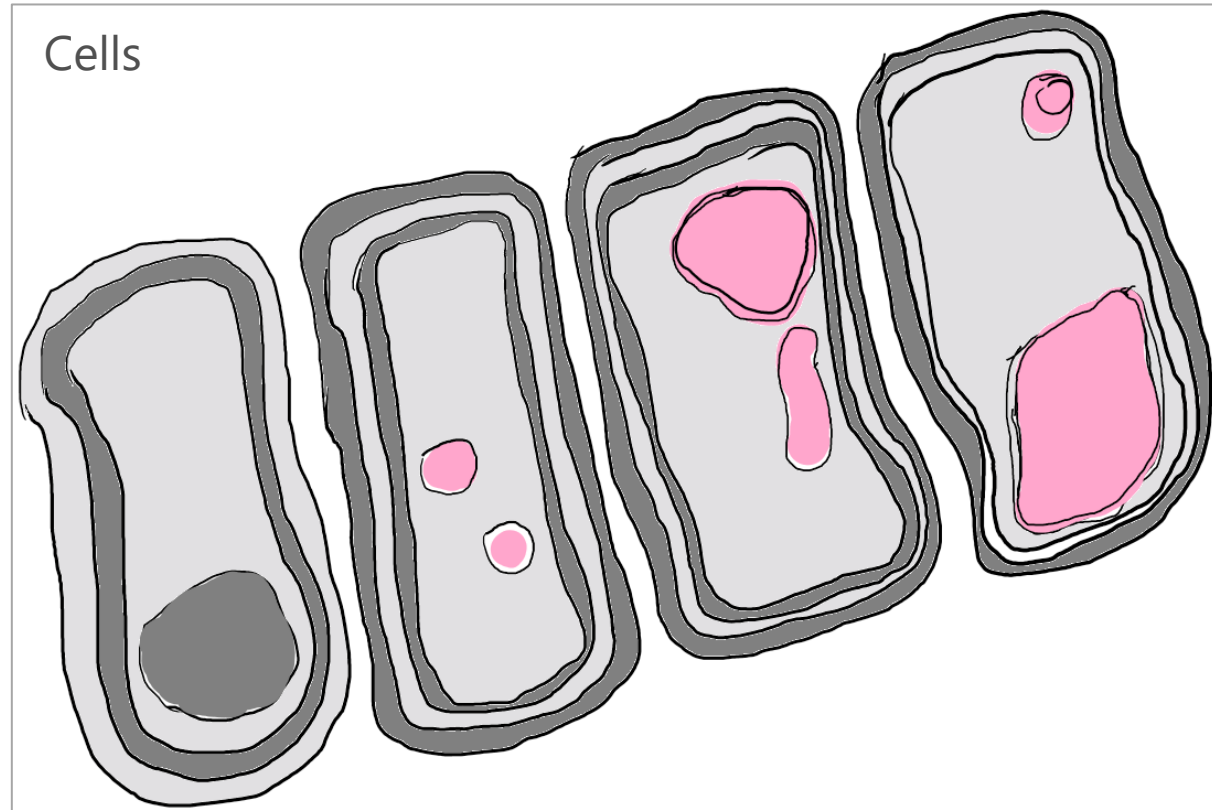


Data in Microservices approach

- Graph of interconnected microservices
- State typically scoped to the microservice
- Remote Storage for cold data



The Bounded Context Pattern



Independent
Autonomous
Loosely coupled composition

*"Cells can exist because their membranes define
what is in and out and determine what can pass"*
[Eric Evans]

Bounded Contexts and Microservices

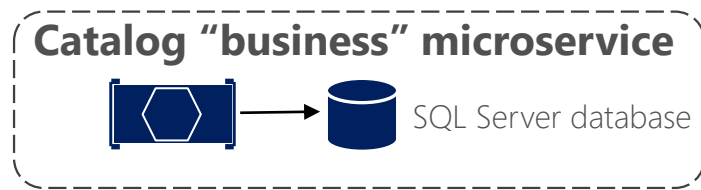
Bounded Context == “Business Microservice” boundary

Each Bounded Context has:

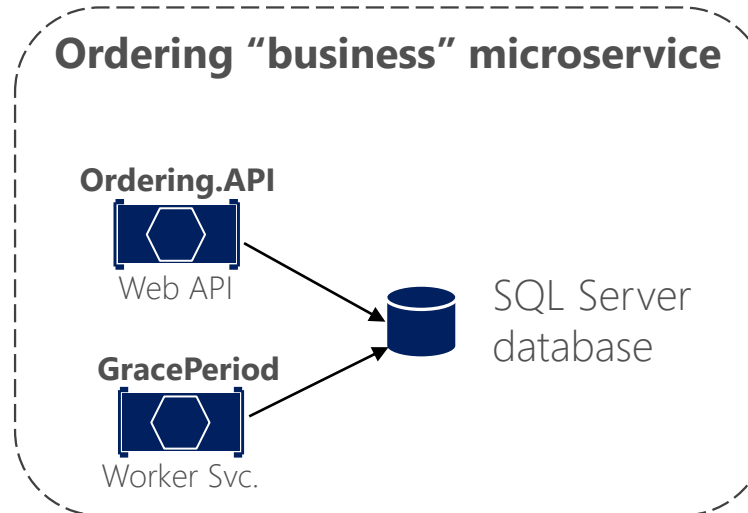
- Its own Domain Model → i.e. Database
- Its own context, invariants, rules, code!
- IT IS AUTONOMOUS!

Business/Logical Microservices (Bounded Contexts)

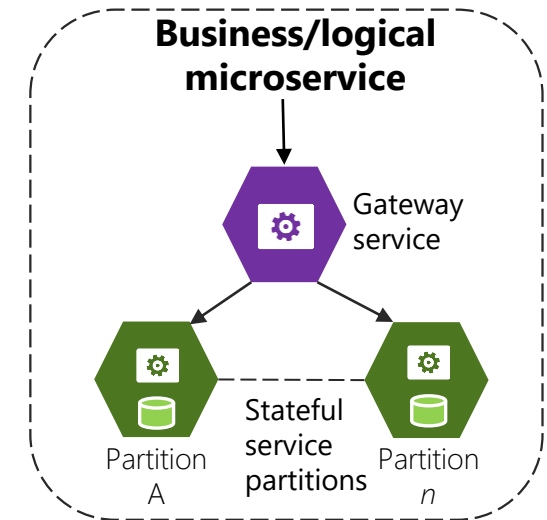
Example 1



Example 2



Example 3

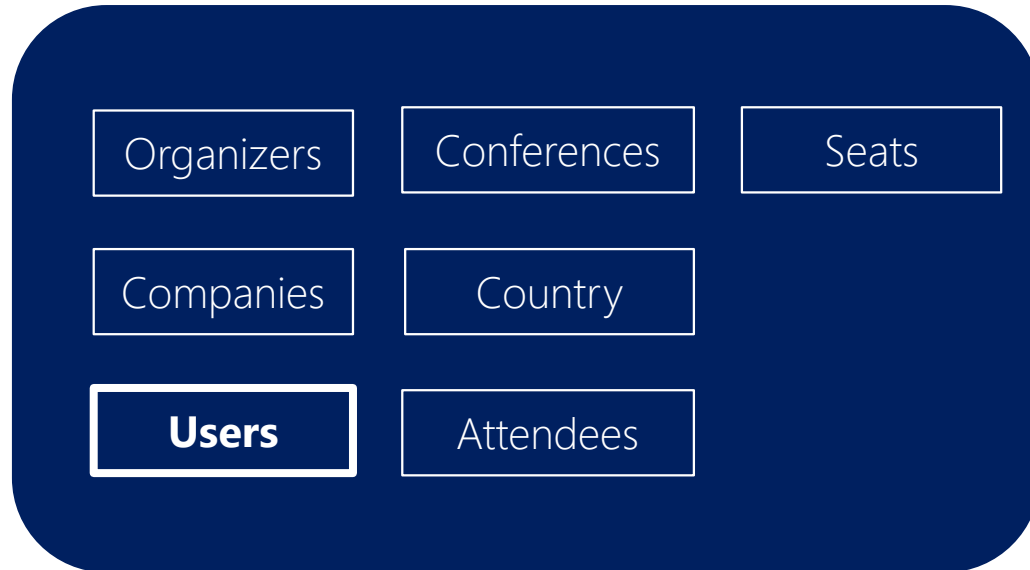


(Using Azure Service Fabric Stateful Reliable Services)

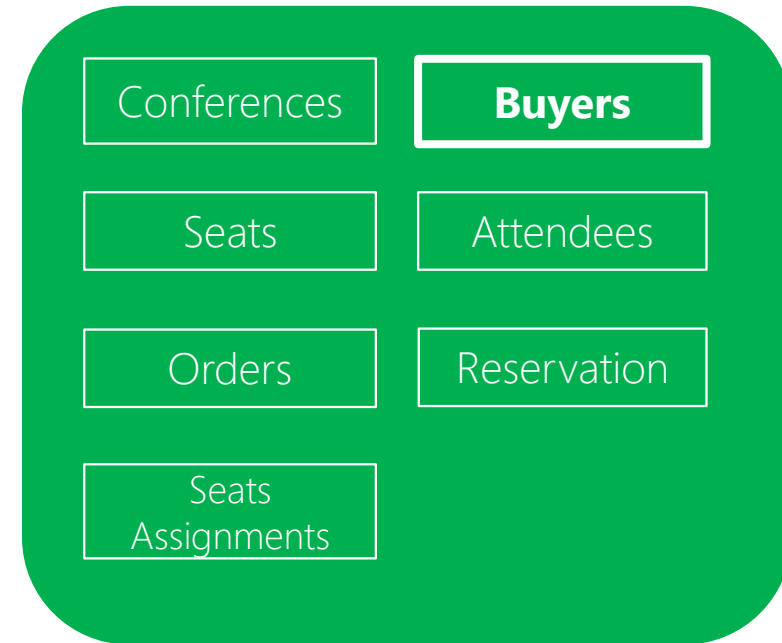
- The Logical Architecture can be different from the Physical/Deployment Architecture
- A Bounded Context can be implemented by 1 or more services (i.e. ASP.NET Web API)

Identifying a Domain Model per Microservice/BoundedContext

Conferences Management



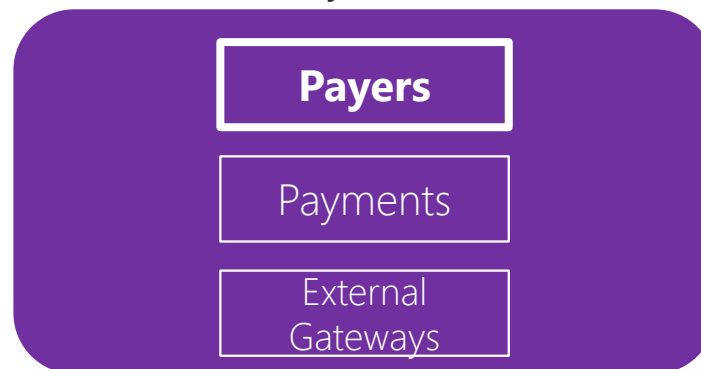
Orders and Registration



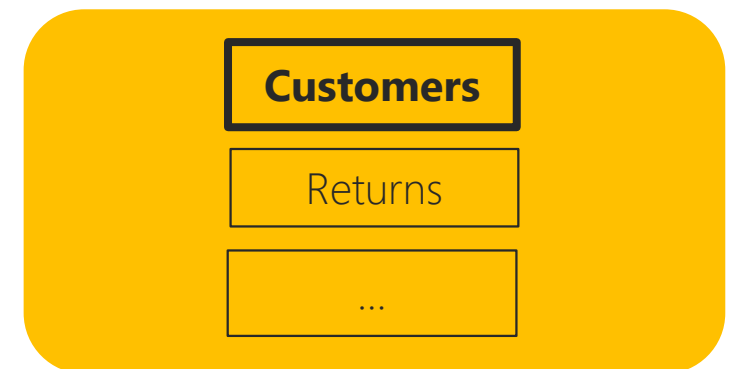
Pricing and Marketing



Payment



Customer Service



.Net Microservice And Container Guidance

Trending C# repositories

GitHub, Inc. [US] github.com/trending/c%23?since=monthly

Features Business Explore Pricing

Search GitHub Your dashboard

Explore GitHub Showcases Integrations Trending

Trending in open source

See what the GitHub community is most excited about this month.

Unwatch 496 Unstar 2,580 Fork 731

dotnet-architecture / eShopOnContainers

Easy to get started sample reference microservice and container based application (Currently in ALPHA state, ongoing progress, accepting feedback and pull-requests). Cross-platform on Linux and Windows Containers, powered by .NET Core and Docker engine. Supports .CSPROJ with Visual Studio 2017 and also CLI based environments with Docker CLI, dot...

C# 1,571 377 Built by

★ 491 stars this month

thedillonb / CodeHub

CodeHub is an iOS application written using Xamarin

C# 8,748 349 Built by

★ 534 stars this month

Microsoft / react-native-windows

A framework for building native UWP and WPF apps with React.

C# 2,190 168 Built by

★ 261 stars this month

PowerShell / PowerShell

PowerShell for every system!

C# 6,851 876 Built by

★ 243 stars this month

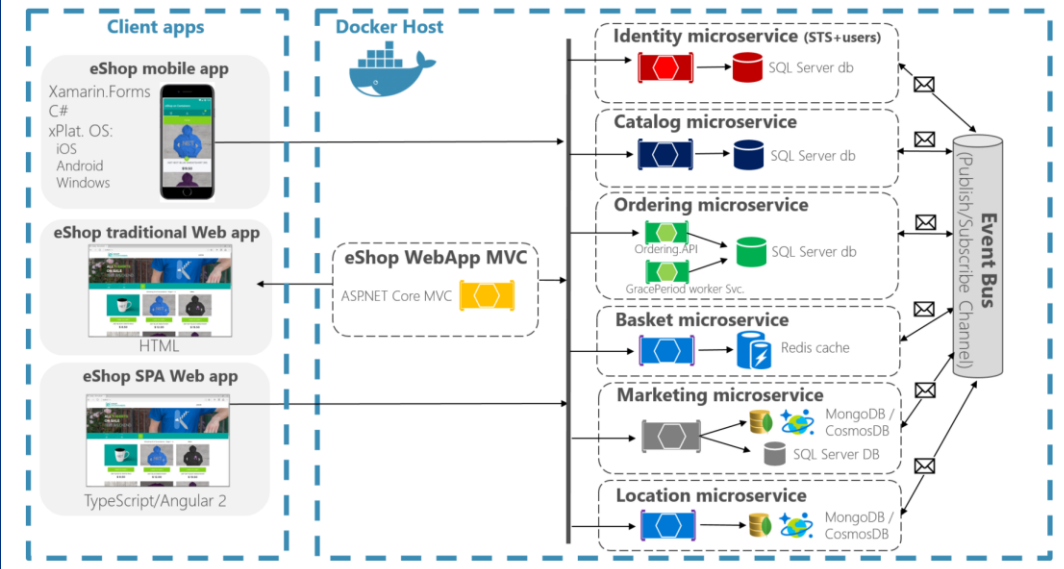
← #2 – May/June

Reference application

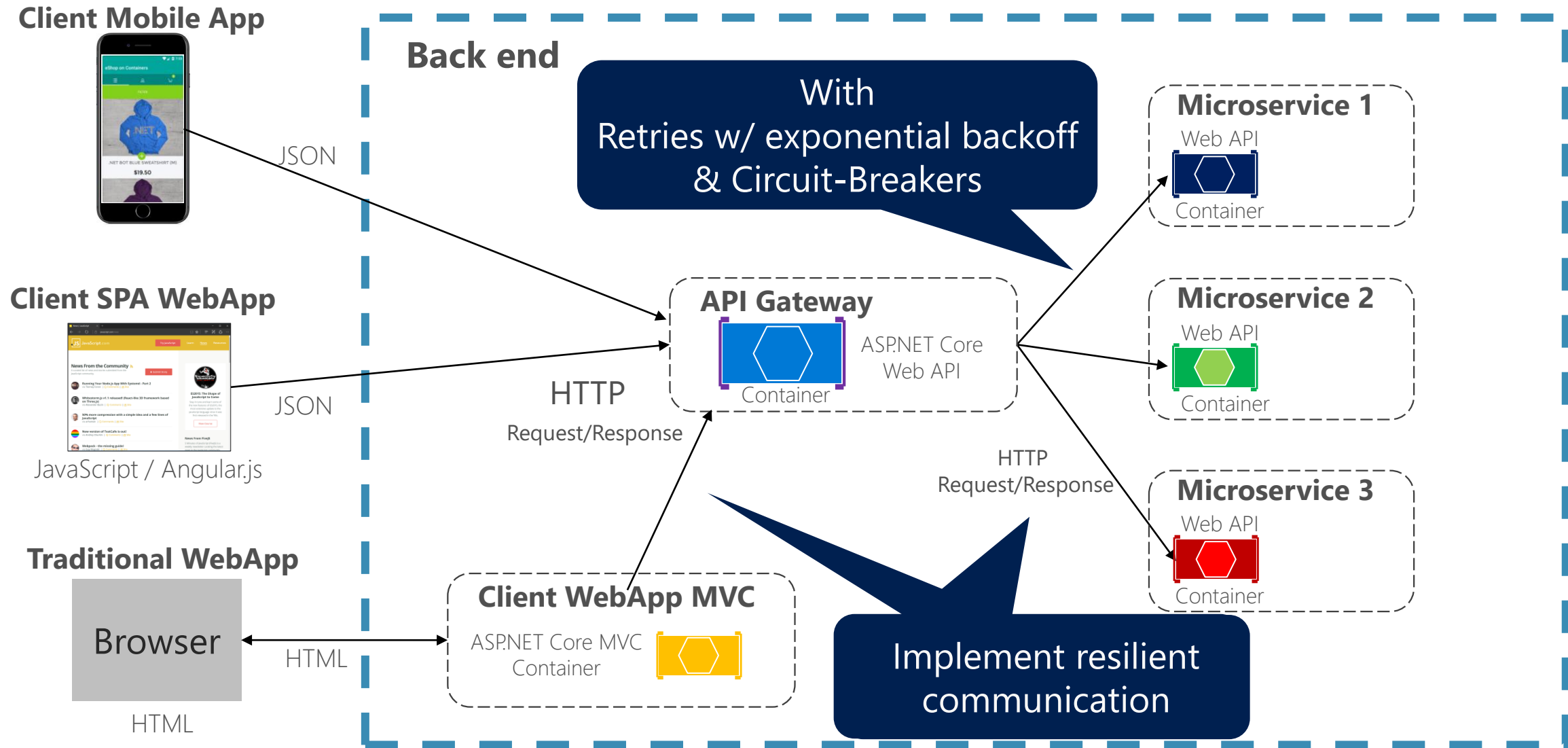
aka.ms/MicroservicesArchitecture

eSHOP onCONTAINERS

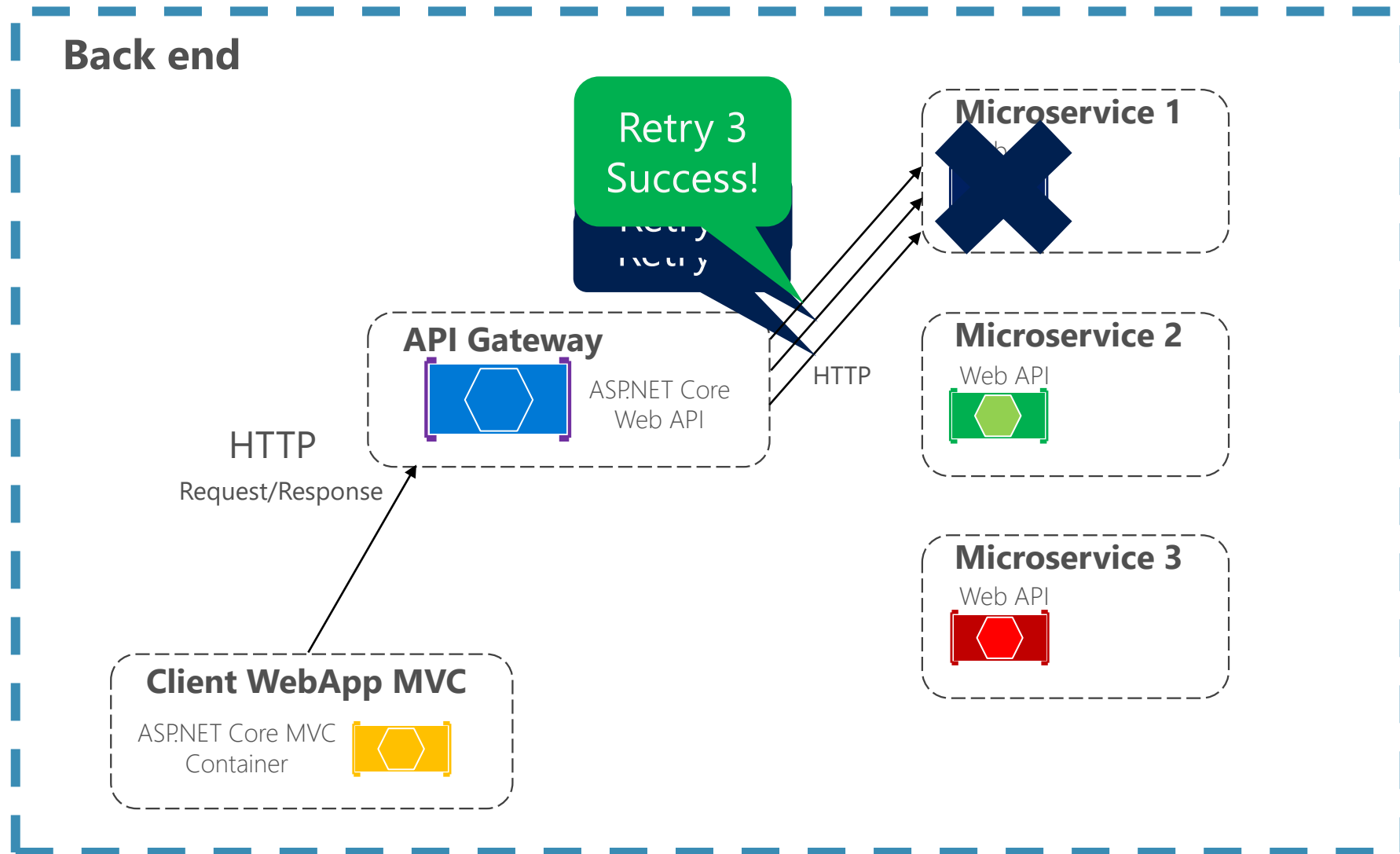
eShopOnContainers Reference Application - Architecture



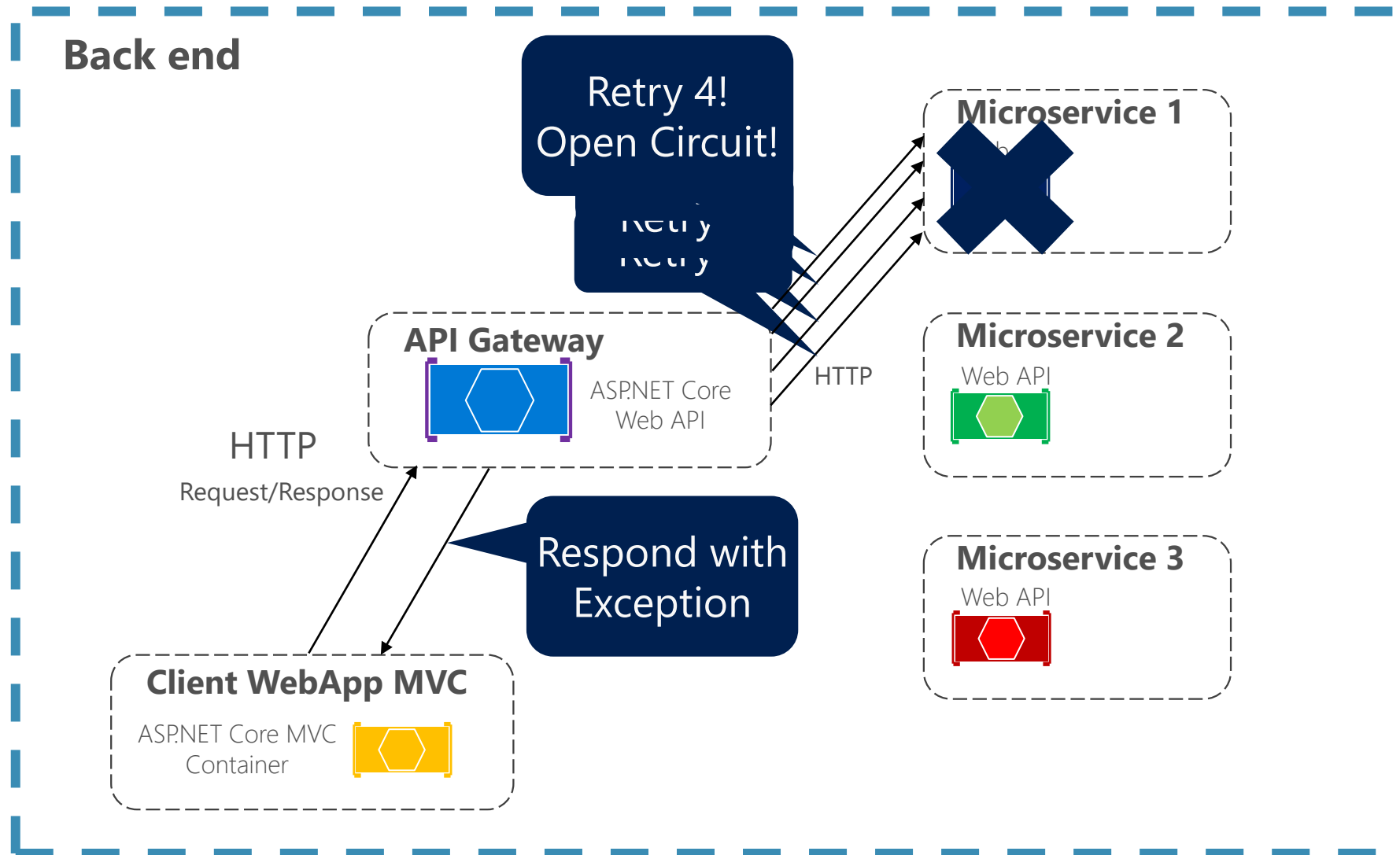
Building Resilient Cloud Applications



Retries with Exponential Backoff

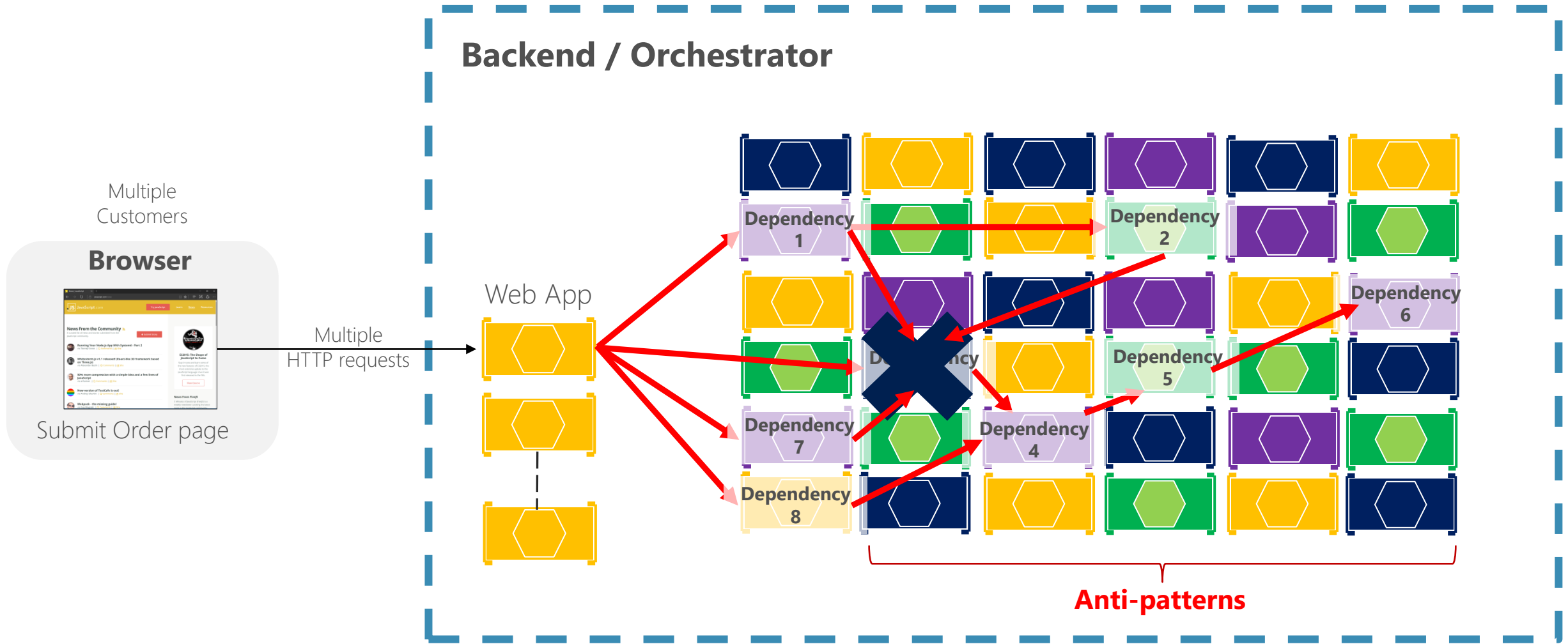


Retries with Exponential Backoff + Circuit Breaker



Risk of Partial Failure Amplified by Microservices

HTTP request/response communication

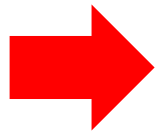


How To Minimize Exponential Failures In Microservices

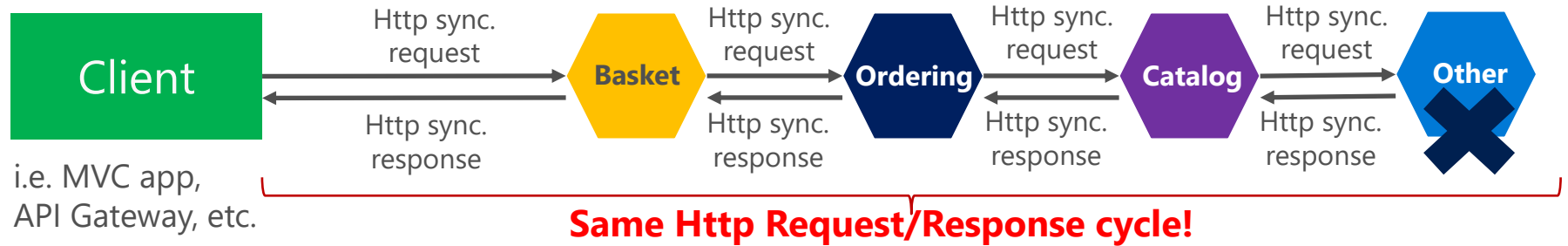
- Circuit-Breakers
- Avoid long Http call chains within the same request/response cycle

Synchronous vs. Async communication across Microservices

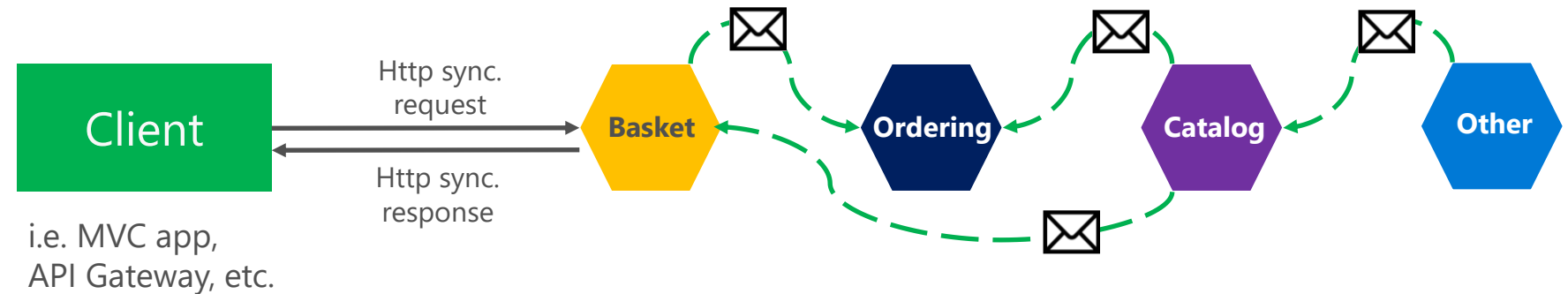
Anti-pattern



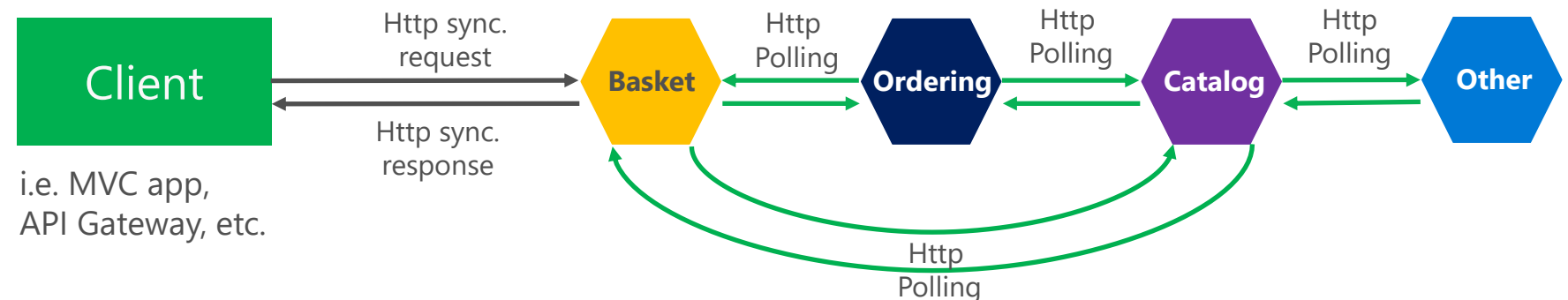
Synchronous
all req./resp. cycle



Asynchronous
Comm. across
internal microservices
(EventBus: i.e. **AMPQ**)

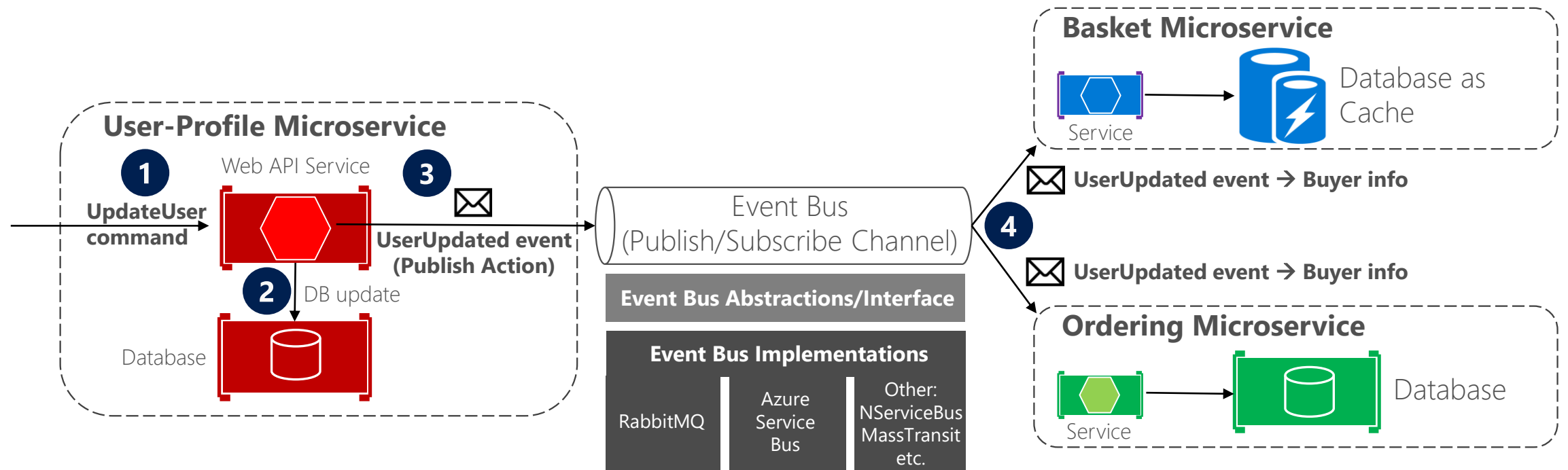


"Asynchronous"
Comm. across
internal microservices
(Polling: **Http**)



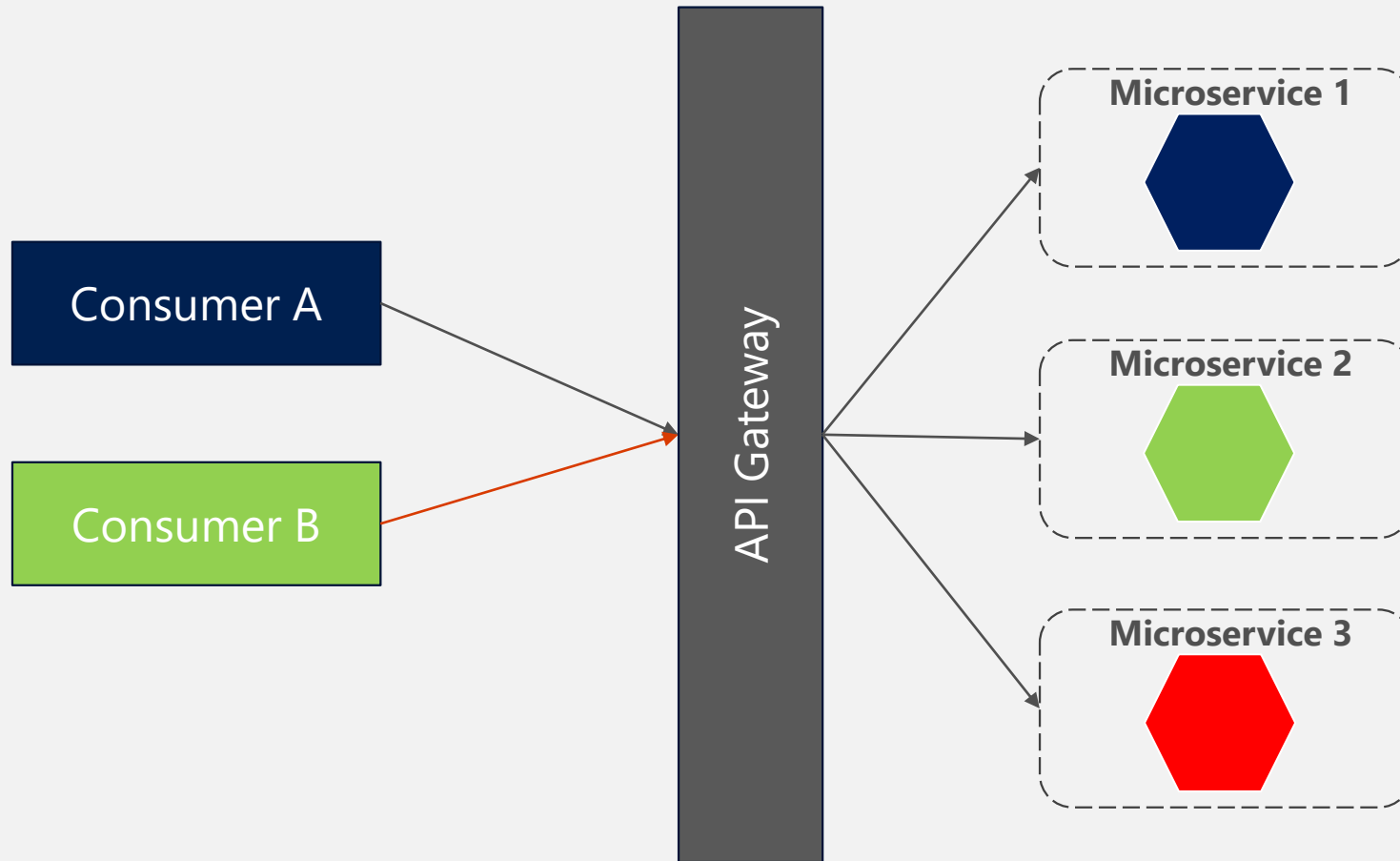
Asynchronous Event-Driven communication with an Event Bus

Backend



Eventual consistency across microservices' data based on event-driven async communication

API Gateway



Calls aggregation

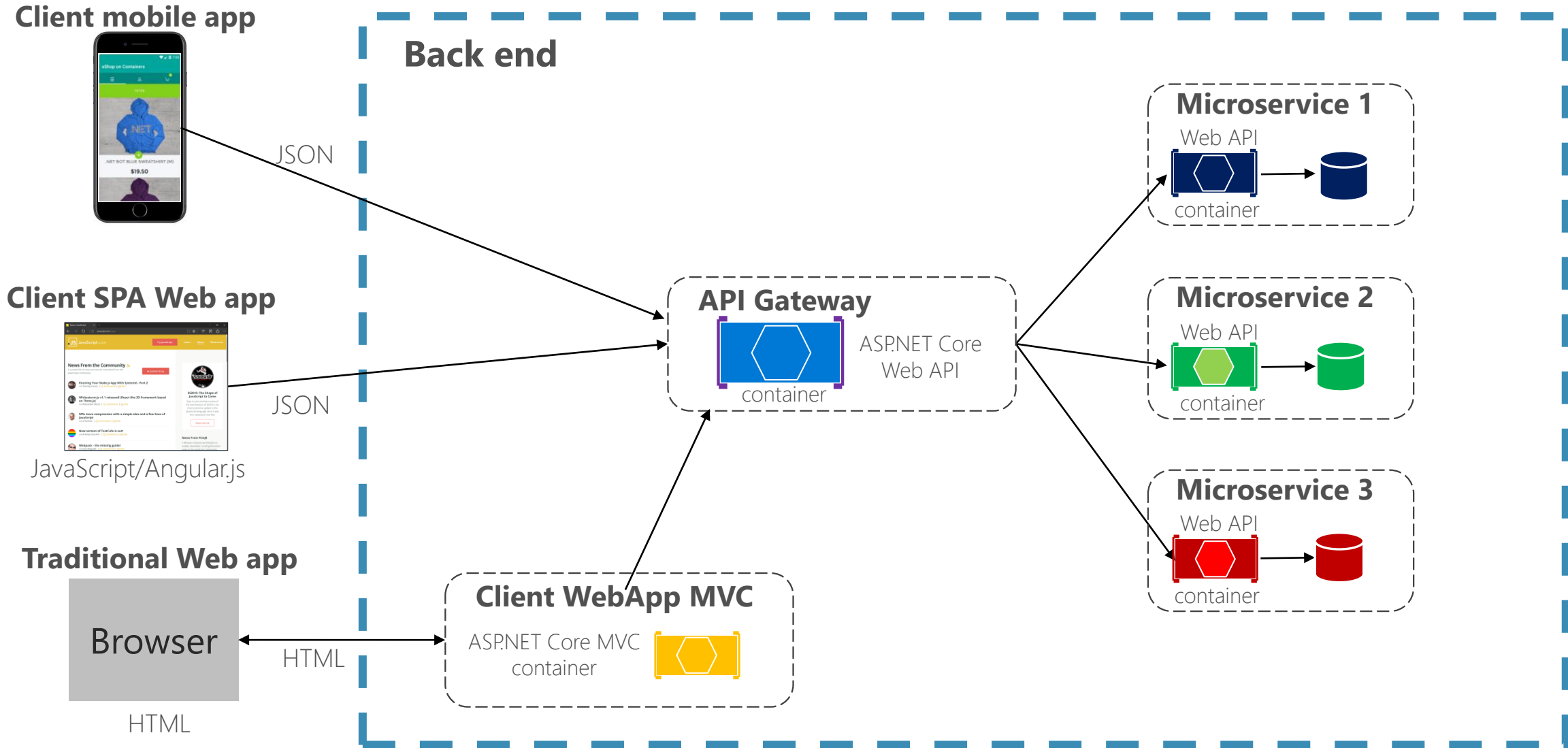
Who is consuming our services?

Who was consuming what?

What rate?

What time?

Using a **custom** API Gateway Service



API Gateway "as a service/product"



AZURE API MANAGEMENT

Third parties



KONG

apigee

Dev environment



Forks/Flavors

Production environment

eShopOnServiceFabric, eShopOnKubernetes
eShopOnSwarm, eShopOnDCOS, etc.

Foundational Development technologies



.NET Core
.NET Framework



Linux Containers
Windows Containers

Cloud infrastructure and Specific Orchestrators



Azure Container Service
Mesos DC/OS
Kubernetes
Docker Swarm



Azure Service Fabric



Service Bus



SQL Database



BLOB Storage



Cosmos DB



Redis Cache



Key Vault

Orchestrators

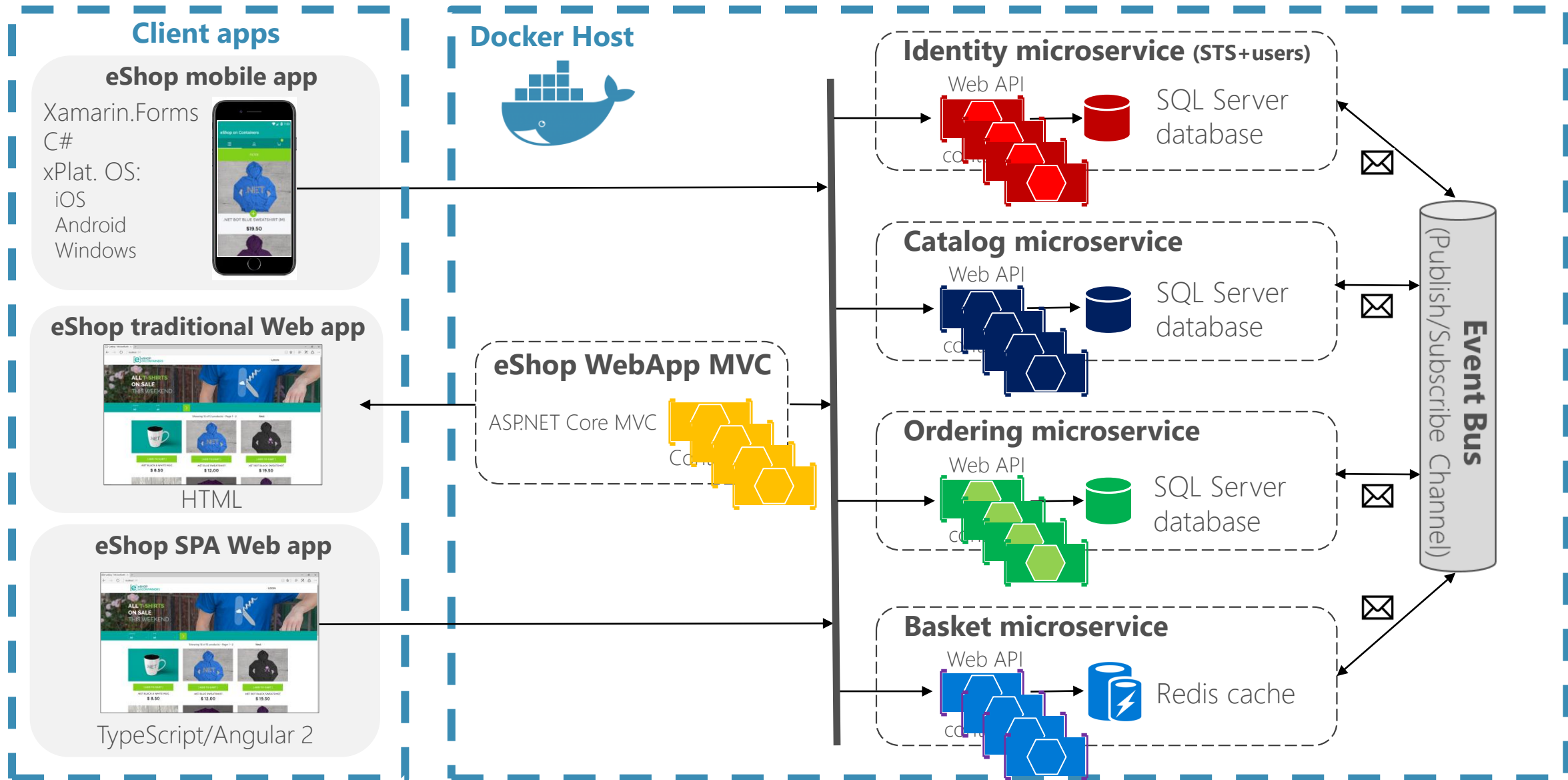
Other Cloud
Infrastructure

Exploring Microservices
Architecture/Design/Development

Infrastructure
Decisions

Production-Ready Microservices

Scaling out eShopOncontainers



Demonstration: *Building Microservices with Containers*

eShopOnContainers



