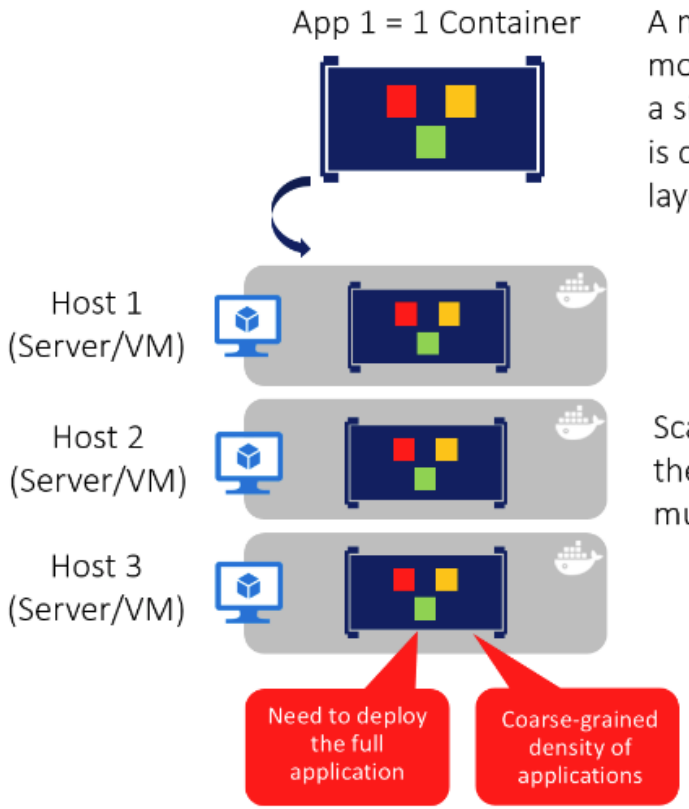
Microservices

KAL ACADEMY

Monolithic application



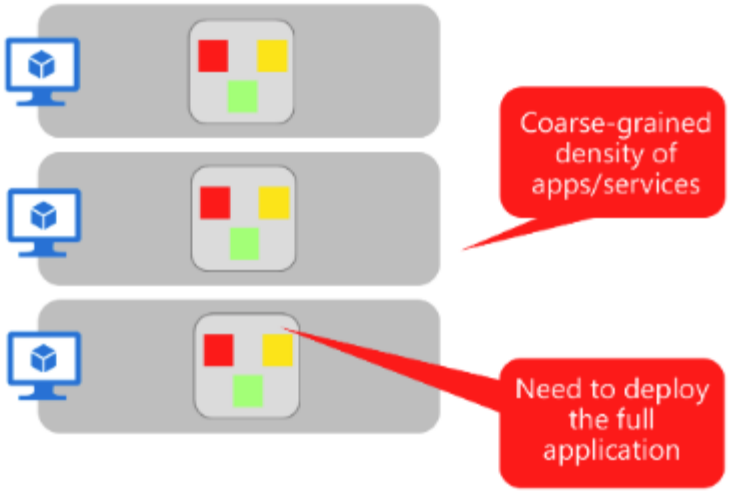
A monolithic application has most of its functionality within a single process/container that is componentized with internal layers or libraries.

Scales out by cloning the app/container on multiple servers/VMs

Microservices Architecture

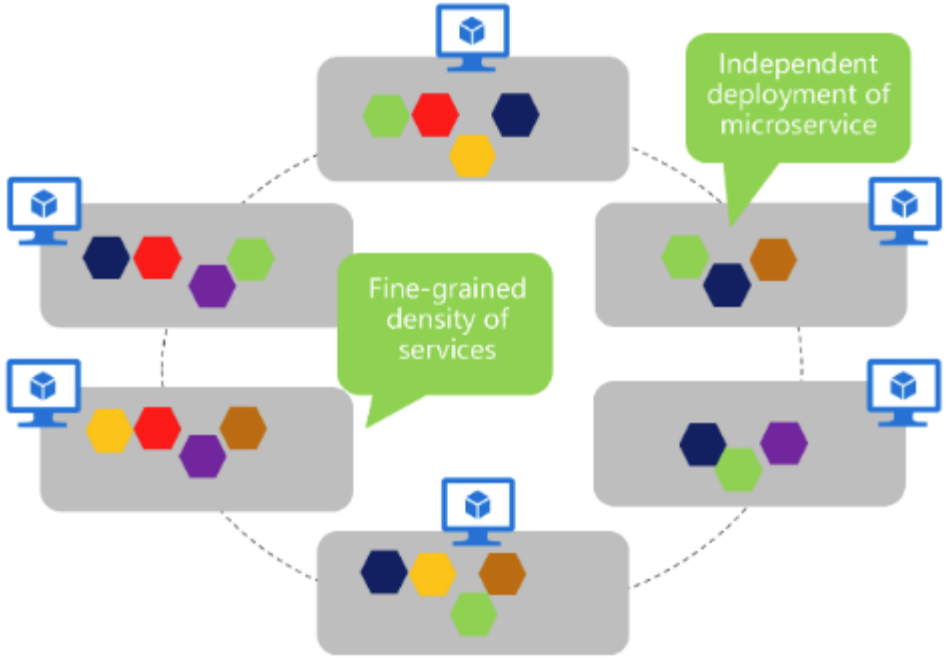
Monolithic deployment approach

- 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



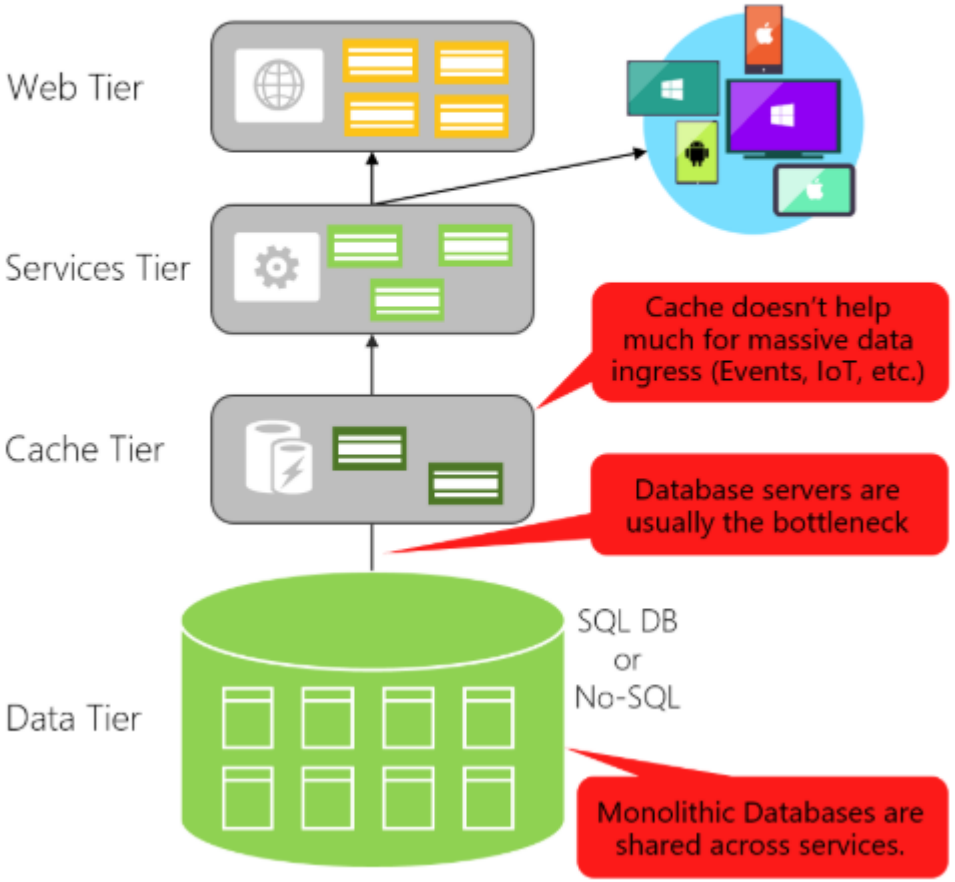
Microservices application approach

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



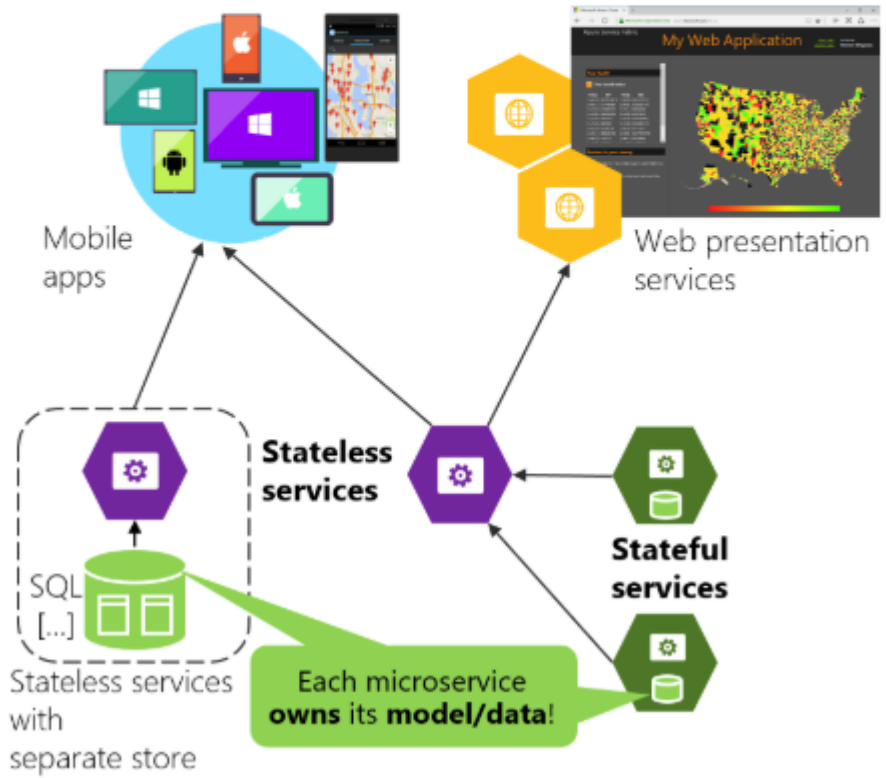
Data in Traditional approach

- Single monolithic database
- Tiers of specific technologies

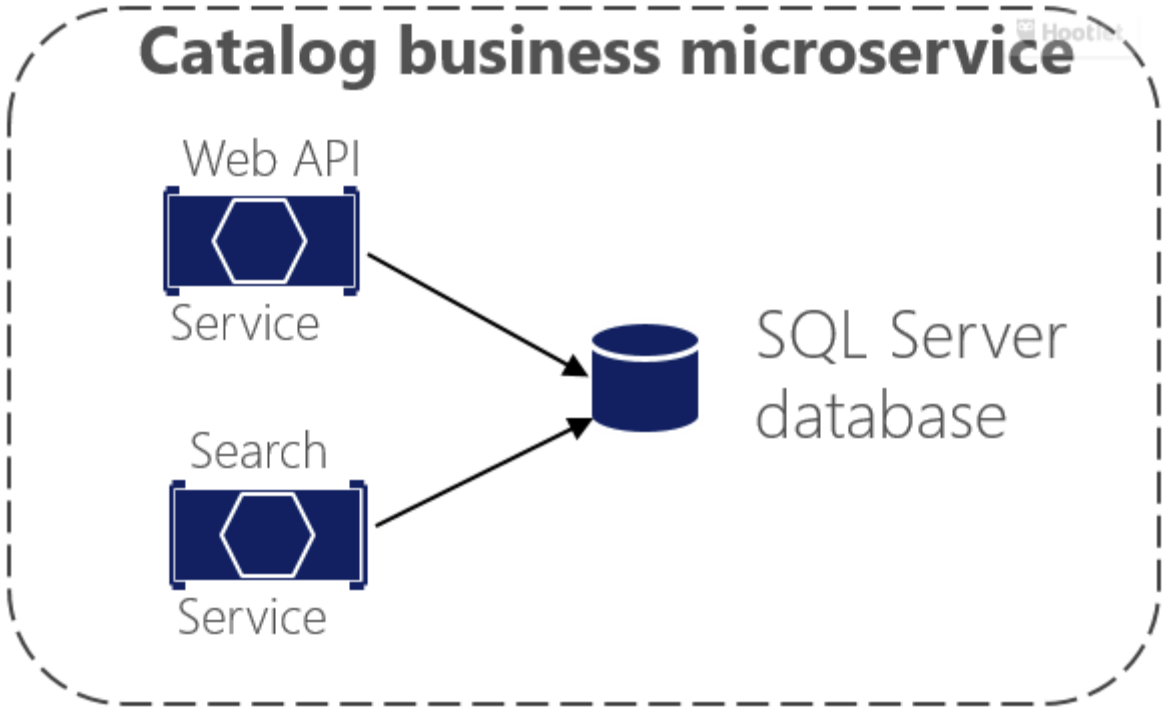


Data in Microservices approach

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



Logical Vs Physical Architecture



Challenges

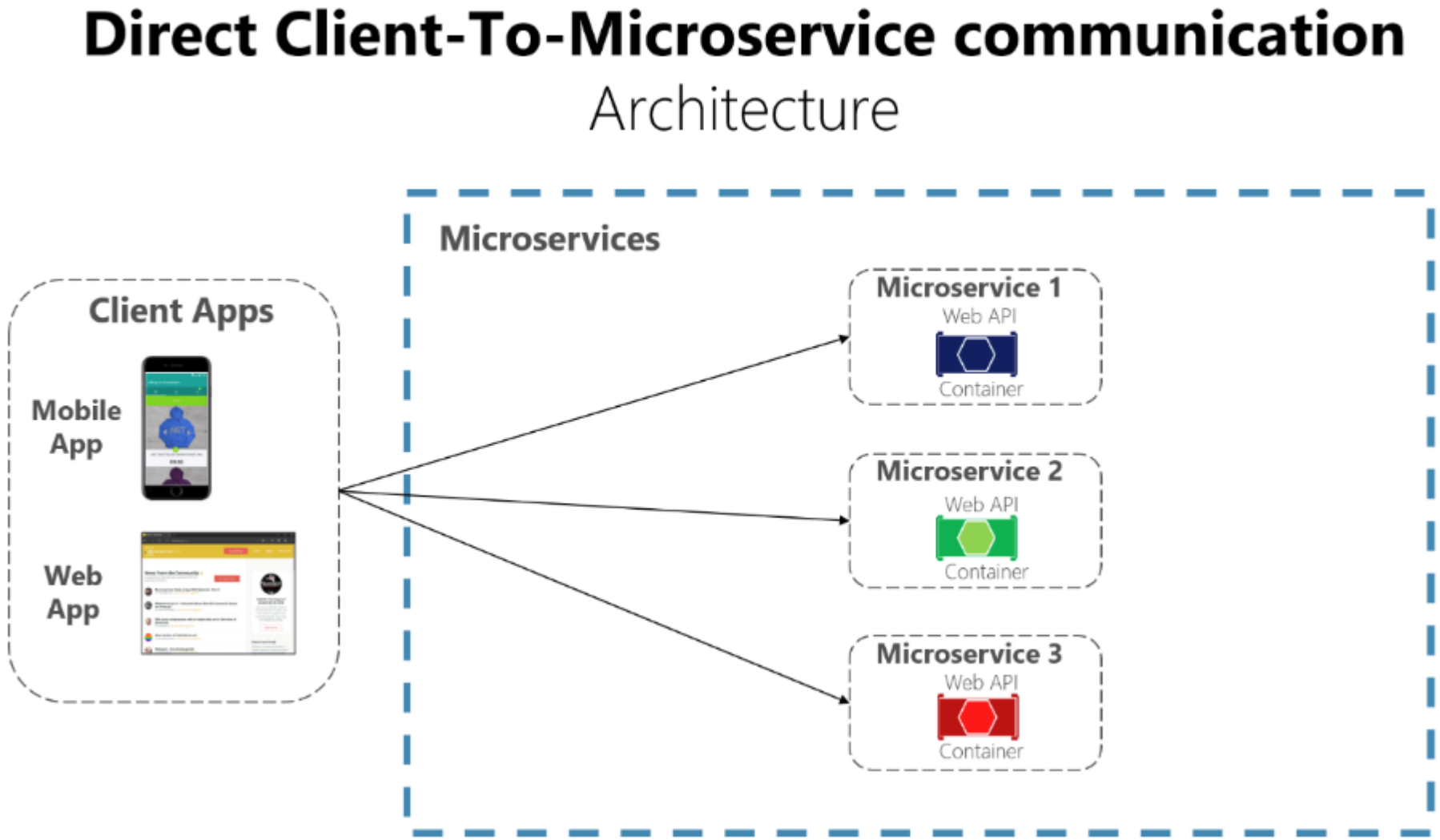
Challenge #1: How to define the boundaries of each microservice

Challenge #2: How to create queries that retrieve data from several microservices

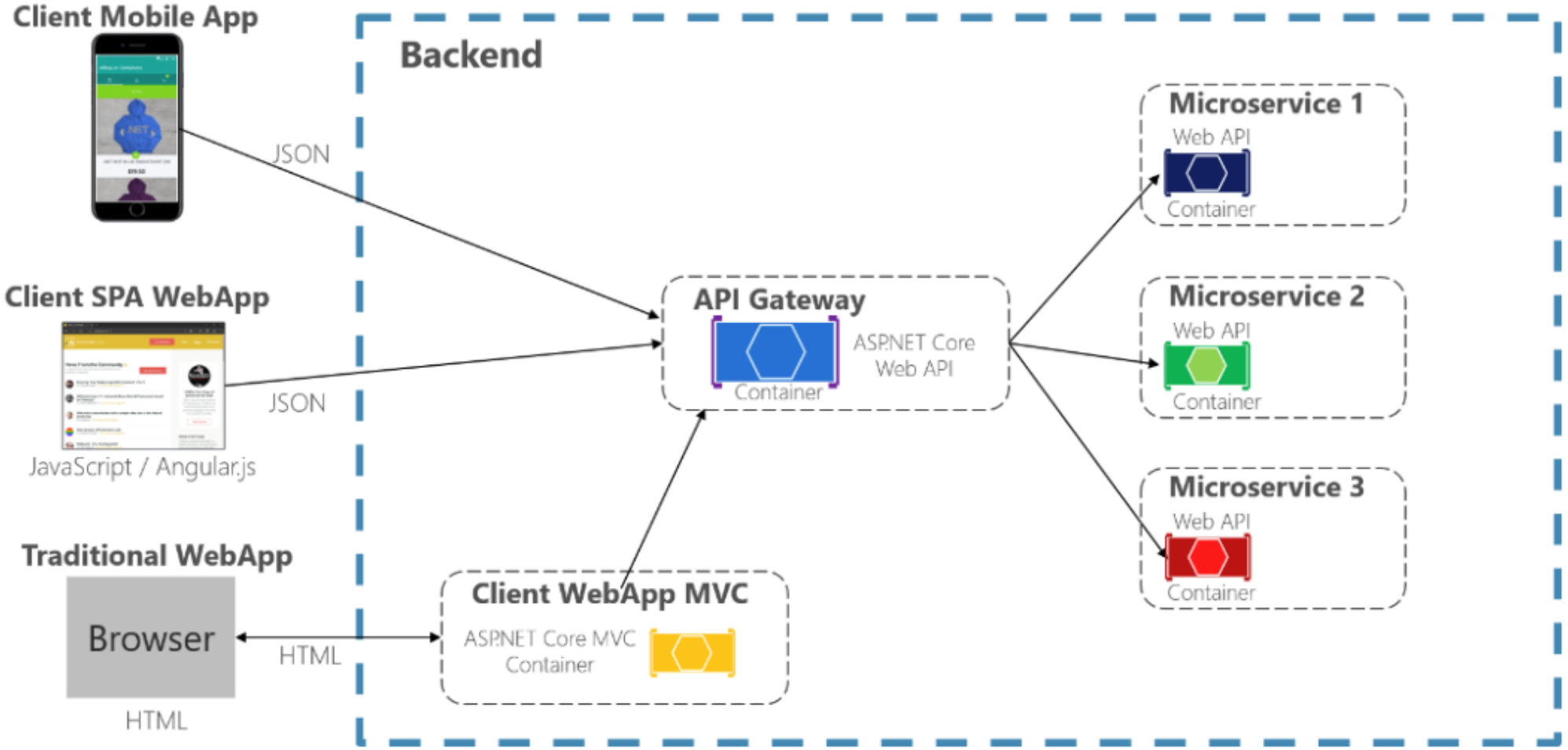
Challenge #3: How to achieve consistency across multiple microservices

Challenge #4: How to design communication across microservice boundaries

Communication Architecture

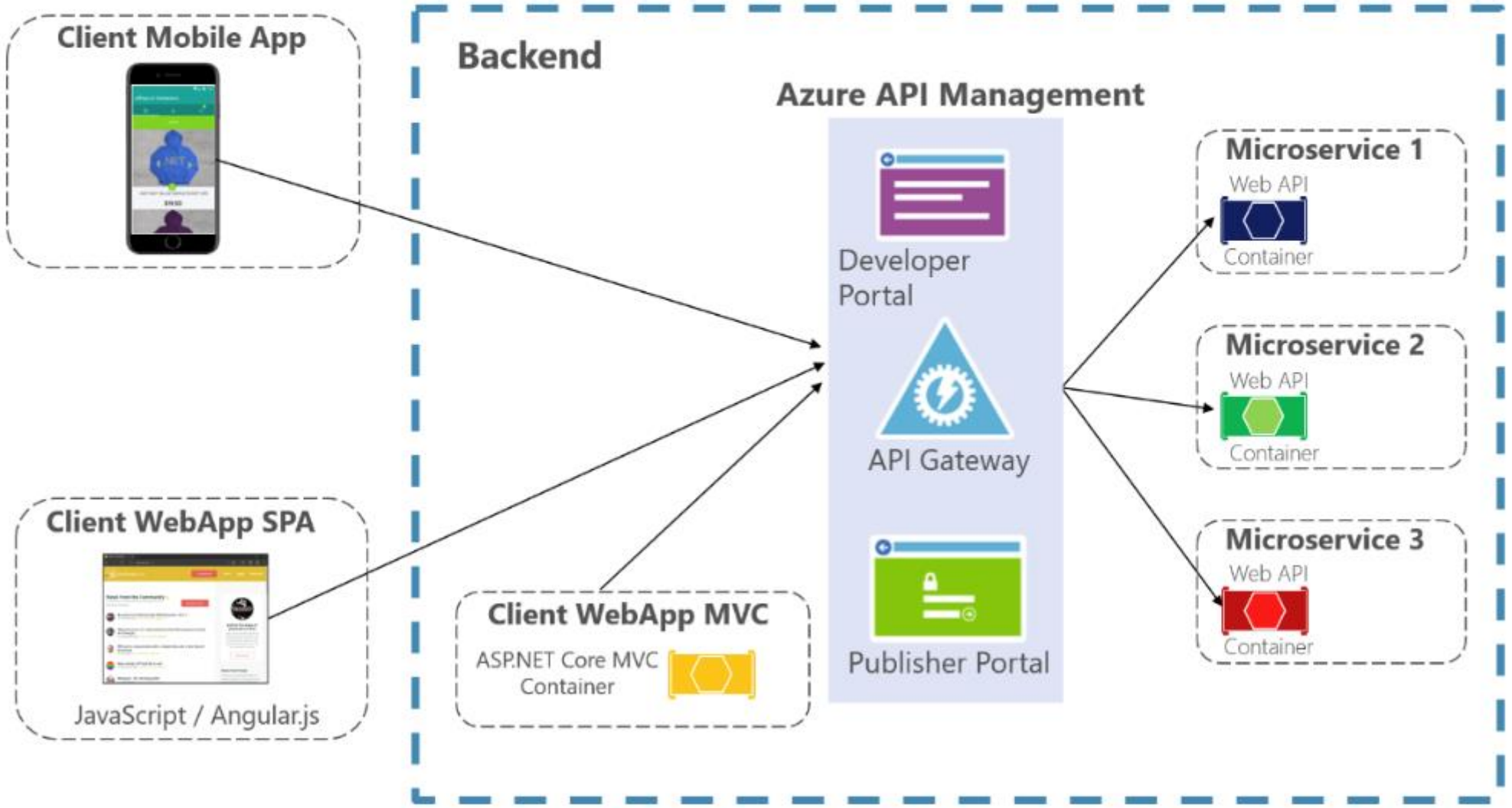


Using the **API Gateway Service**



API Gateway with Azure API Management

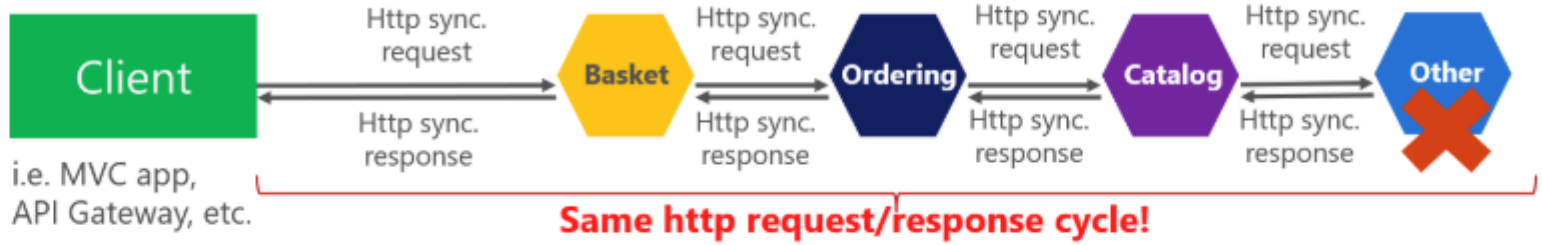
Architecture



Synchronous vs. async communication across microservices

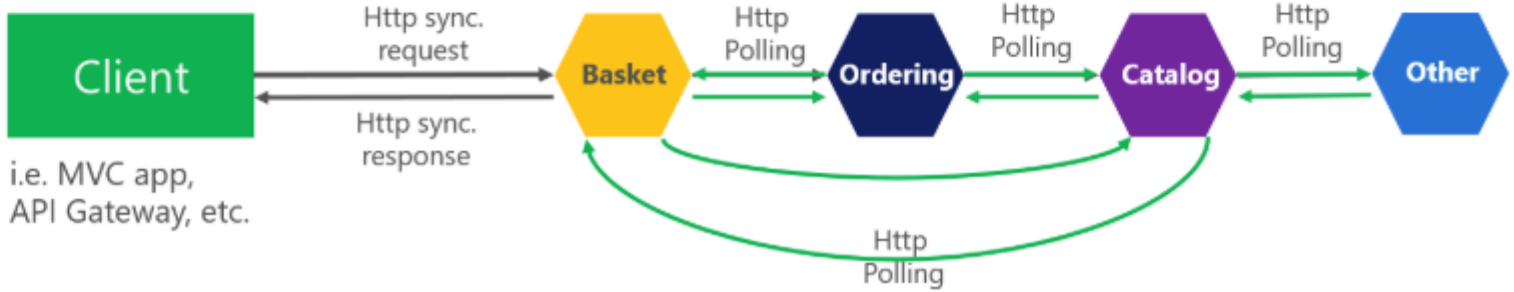
Anti-pattern

Synchronous
all req./resp. cycle



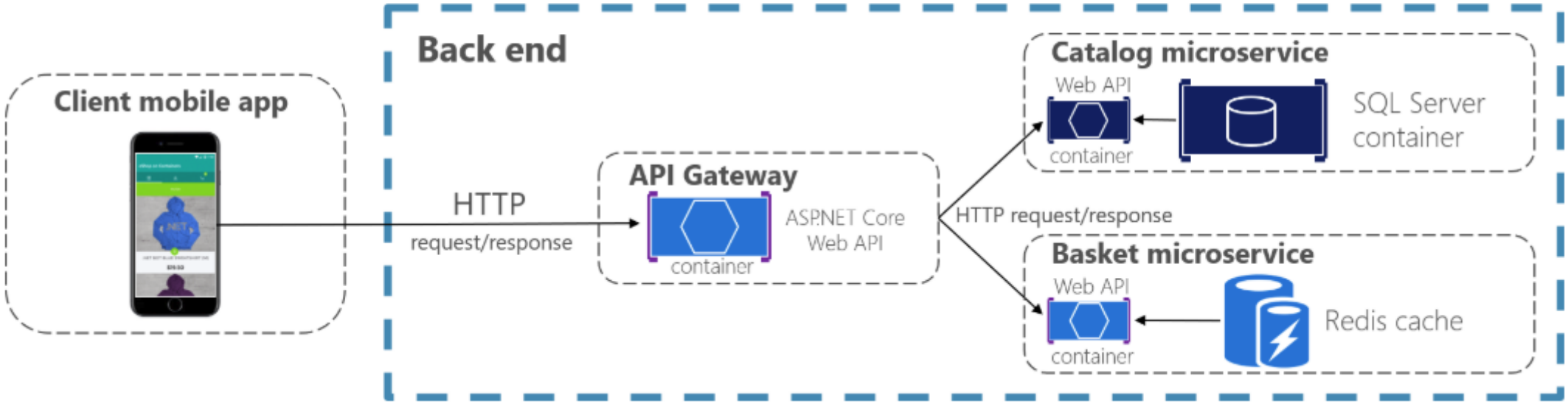
Asynchronous
Comm. across internal microservices (EventBus: i.e. **AMPQ**)

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



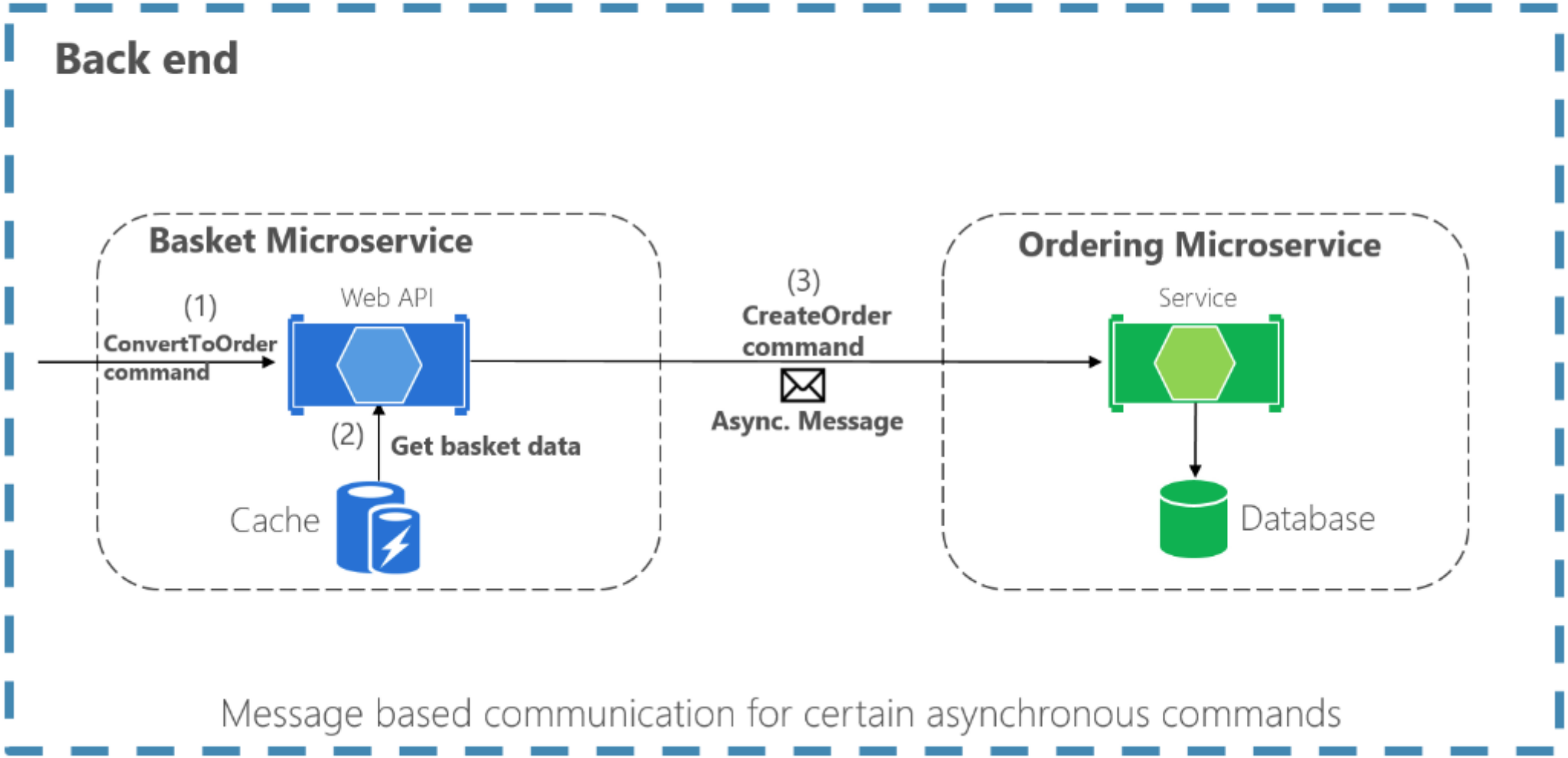
Request/response communication for live queries and updates

HTTP-based Services



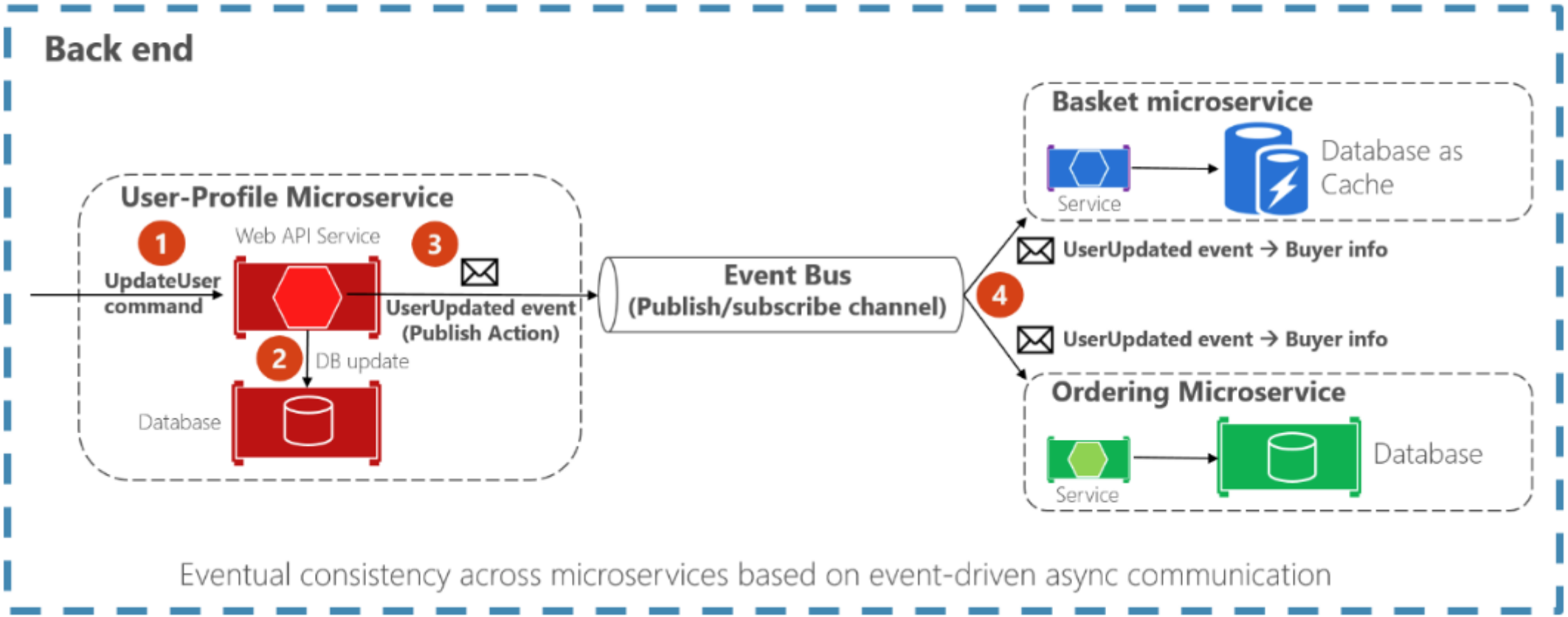
Single receiver message-based communication

(i.e. Message-based Commands)

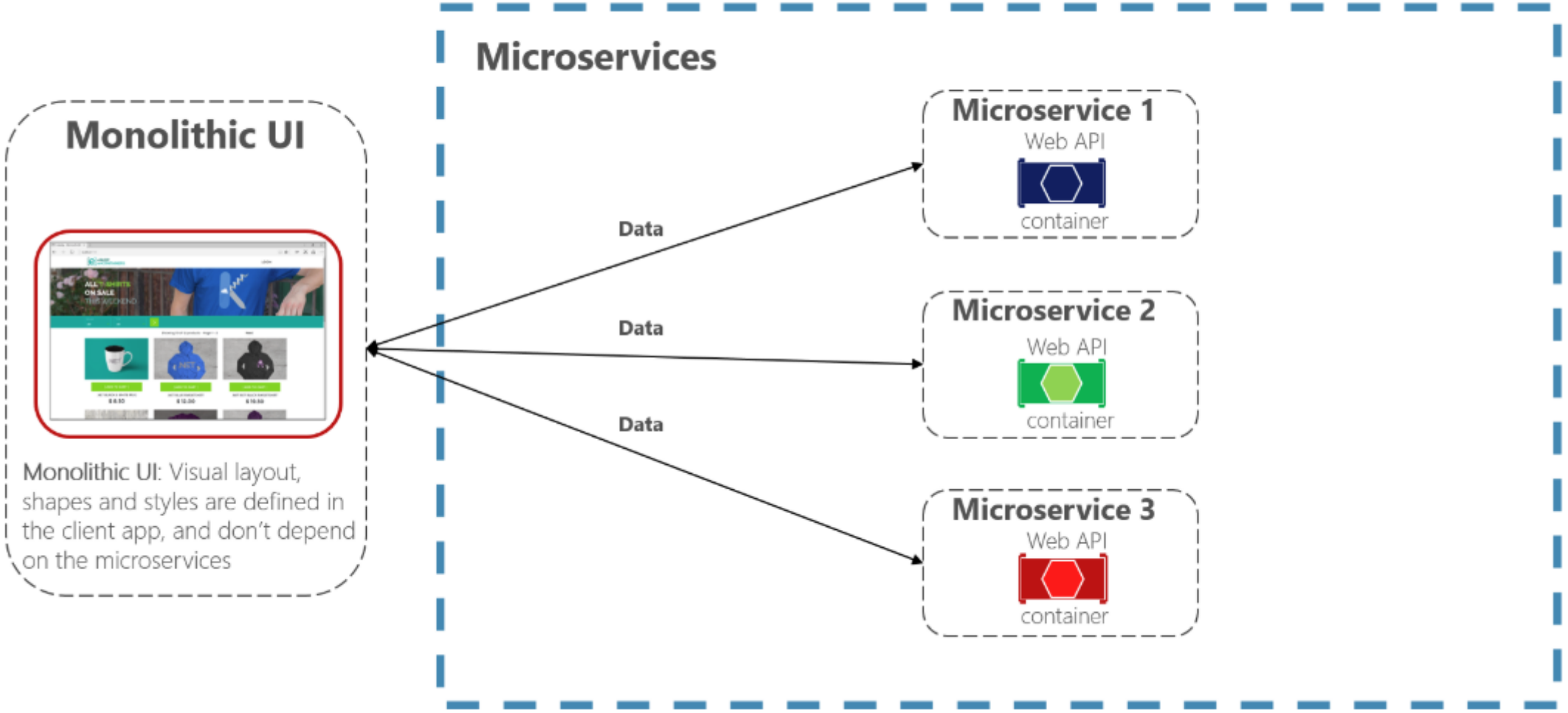


Asynchronous event-driven communication

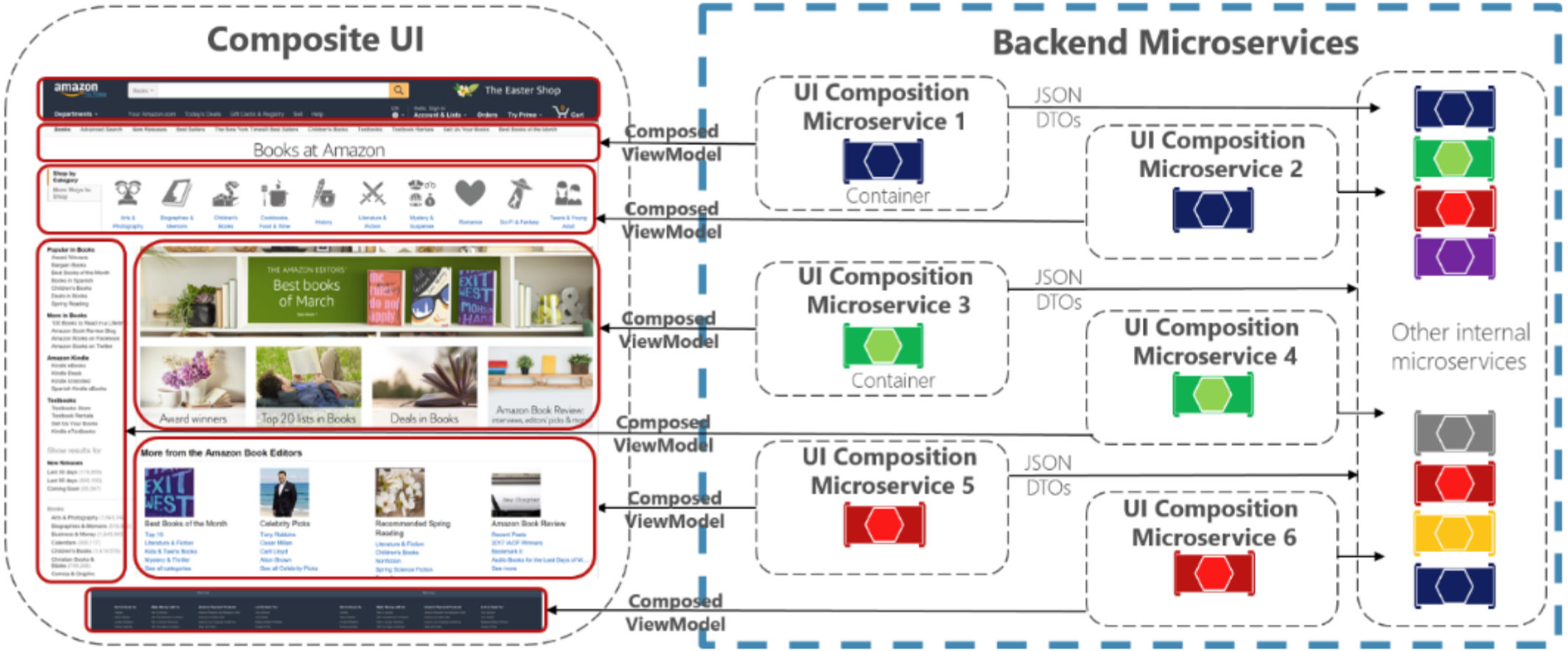
Multiple receivers



Monolithic UI consuming microservices



Composite UI *generated* by microservices

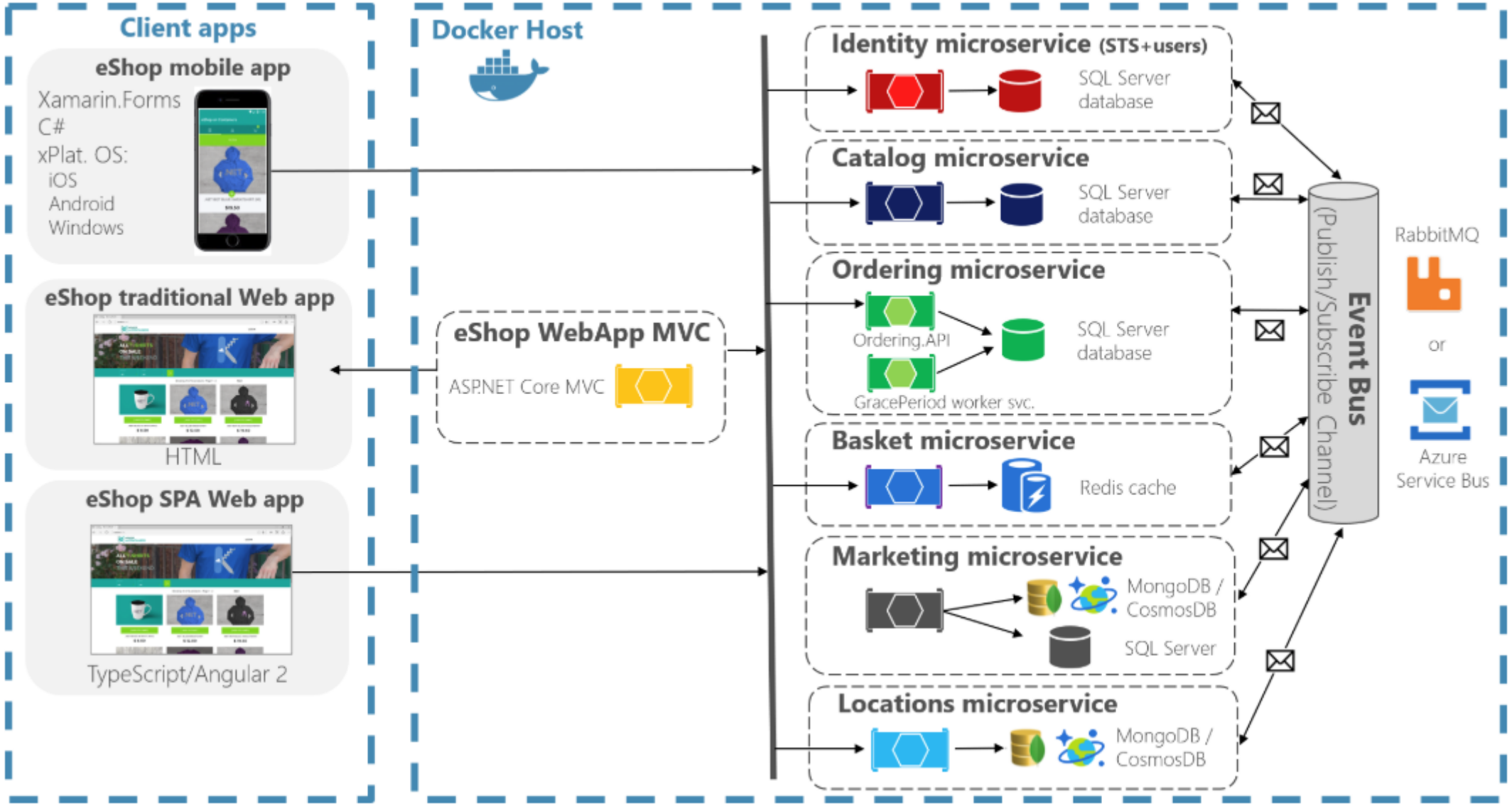


Your microservices

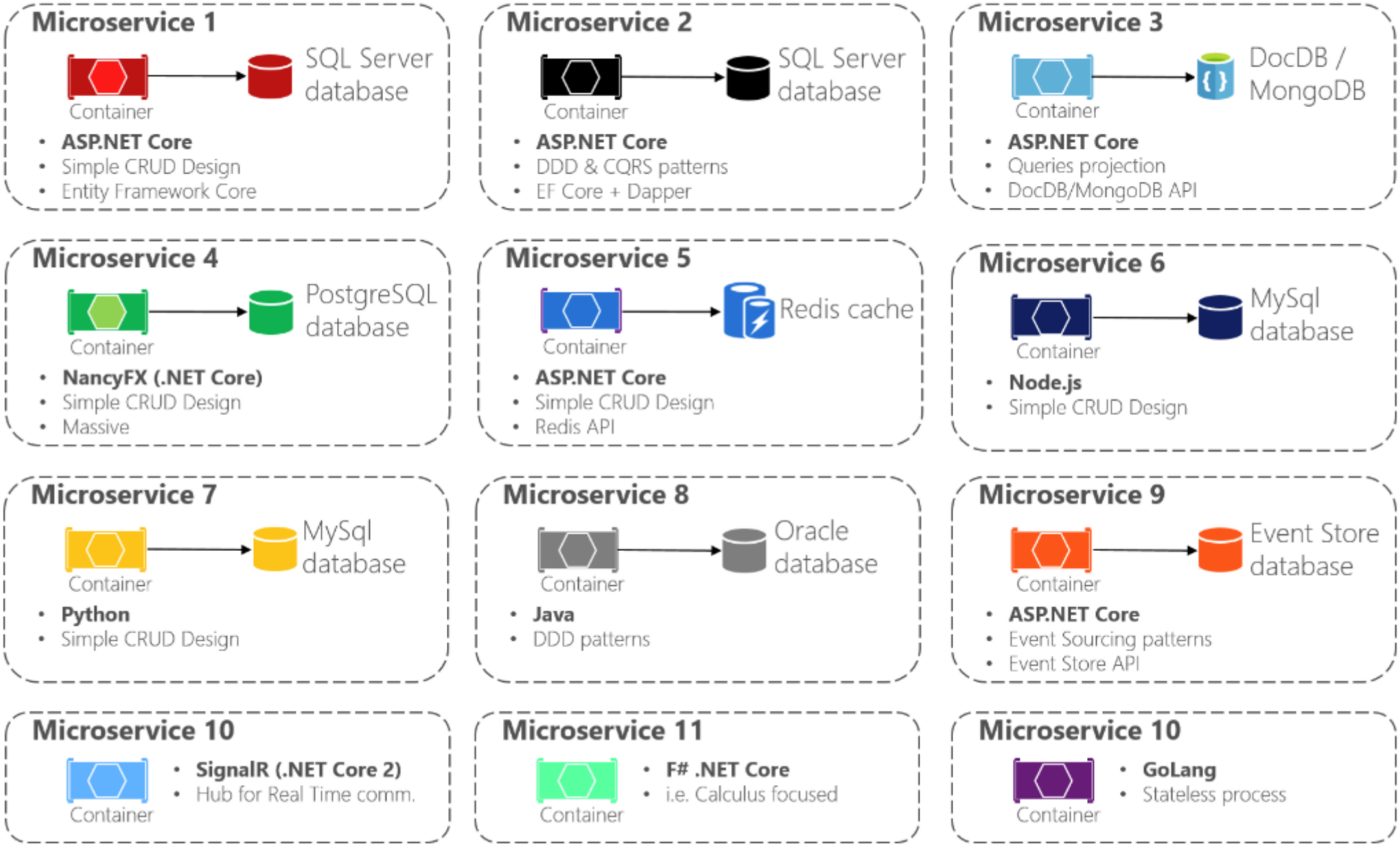
Microservice Platform

(Orchestrators/Clusters)

eCommerce Application

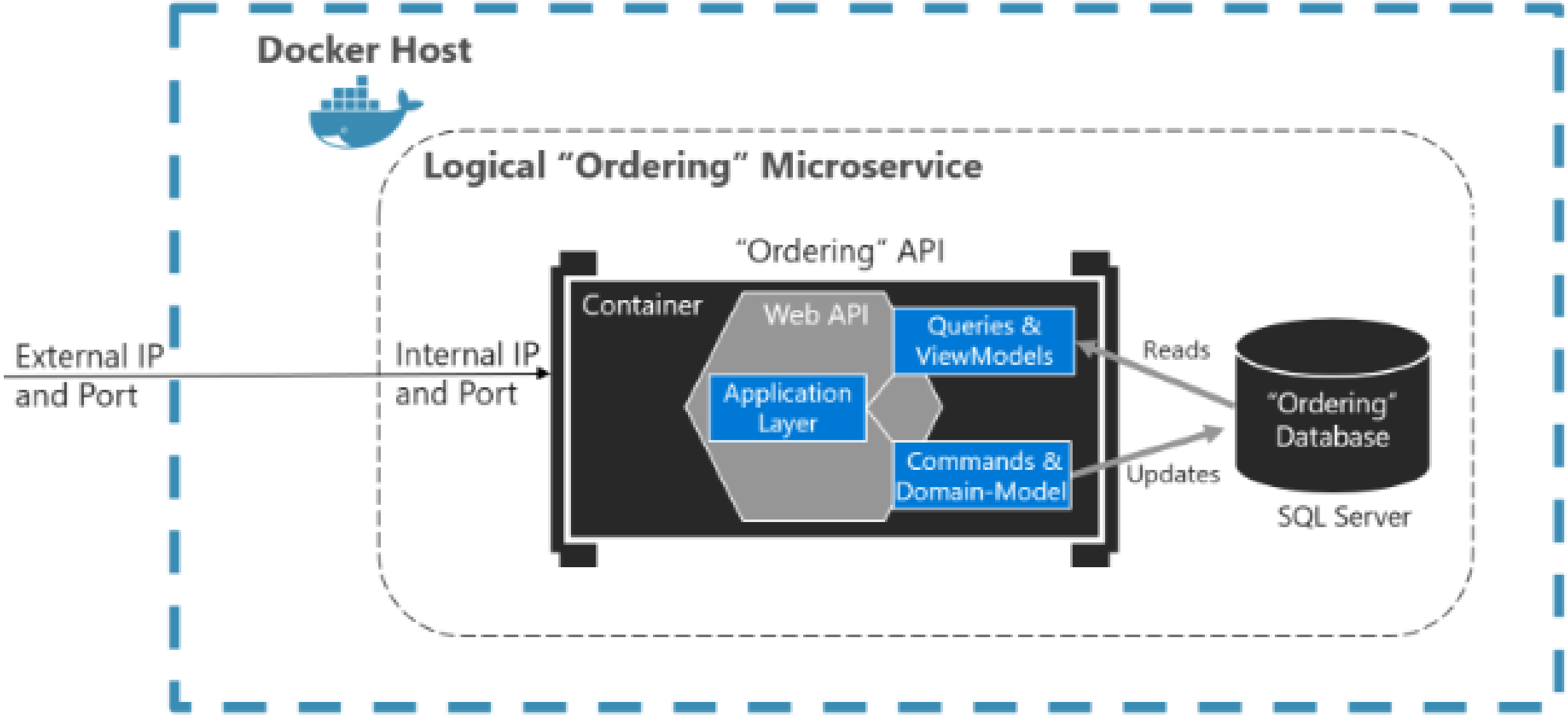


The Multi-Architectural-Patterns and polyglot microservices world

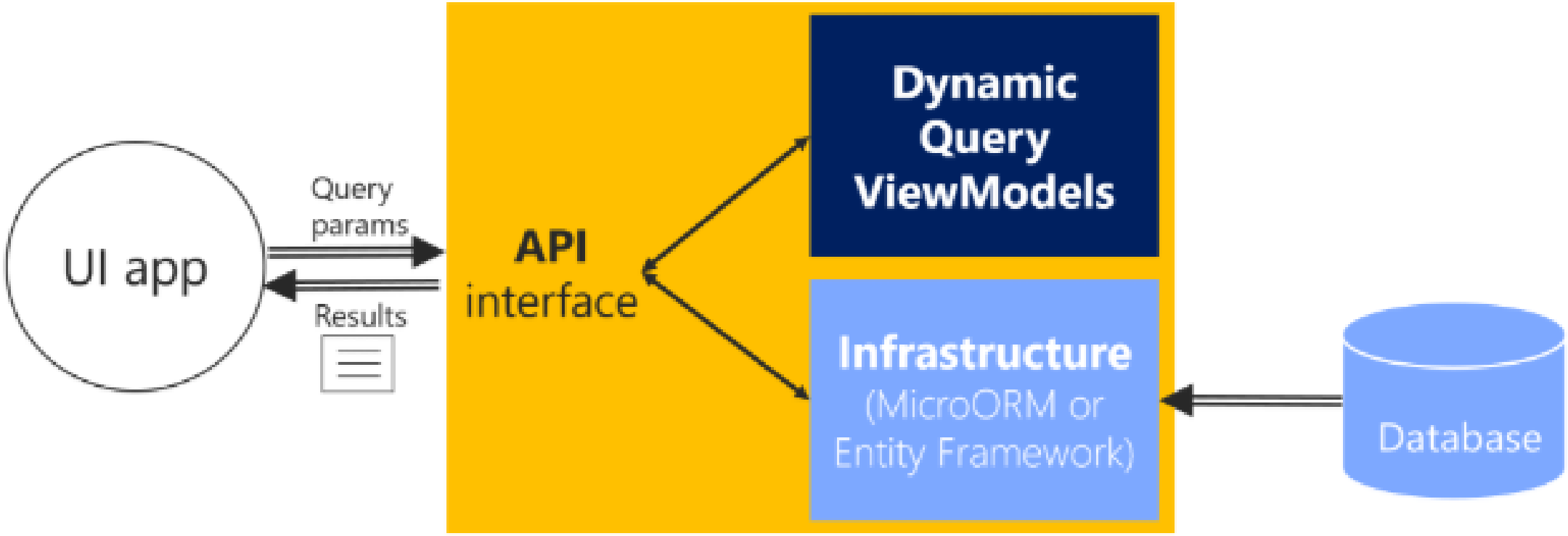


Simplified CQRS and DDD microservice

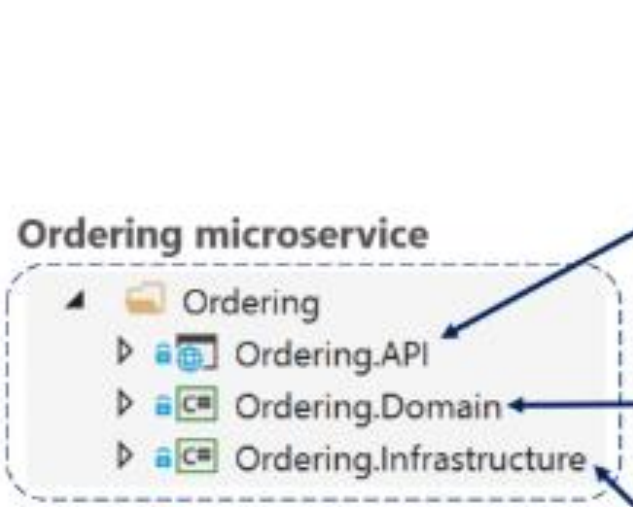
High level design



High level “Queries-side” in a simplified CQRS



Layers in a Domain-Driven Design Microservice



Application layer

- ASP.NET Web API
- Network access to microservice
- API contracts/implementation
- Commands and command handlers
- Queries (when using an CQS approach)
  - Micro ORMs like Dapper

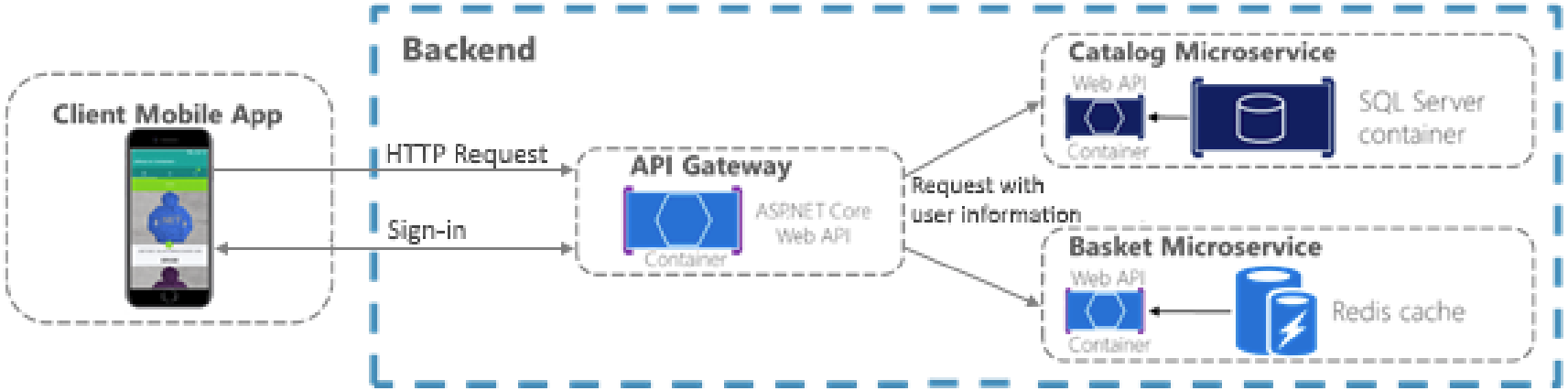
Domain model layer

- Domain entity model
- POCO entity classes (clean C# code)
- Domain entities with data + behavior
- DDD patterns:
  - Domain entity, aggregate
  - Aggregate root, value object
  - Repository contracts/interfaces

Infrastructure layer

- Data persistence infrastructure
  - Repository implementation
- Use of ORMs or data access API:
  - Entity Framework Core or any ORM
  - ADO.NET
  - Any NoSQL database API
- Other infrastructure implementation used from the application layer
  - Logging, cryptography, search engine, etc.

Authentication with API Gateway



Authentication Service