Gateway Group

MICROSERVICES

THE GATEWAY CORP.

INTRODUCTION & PURPOSE

- Binkal Patel
- Technical Leader
- The Gateway Corp.

Gateway Group

WHAT IS MICROSERVICES?

- Microservices are an architectural approach to building applications.

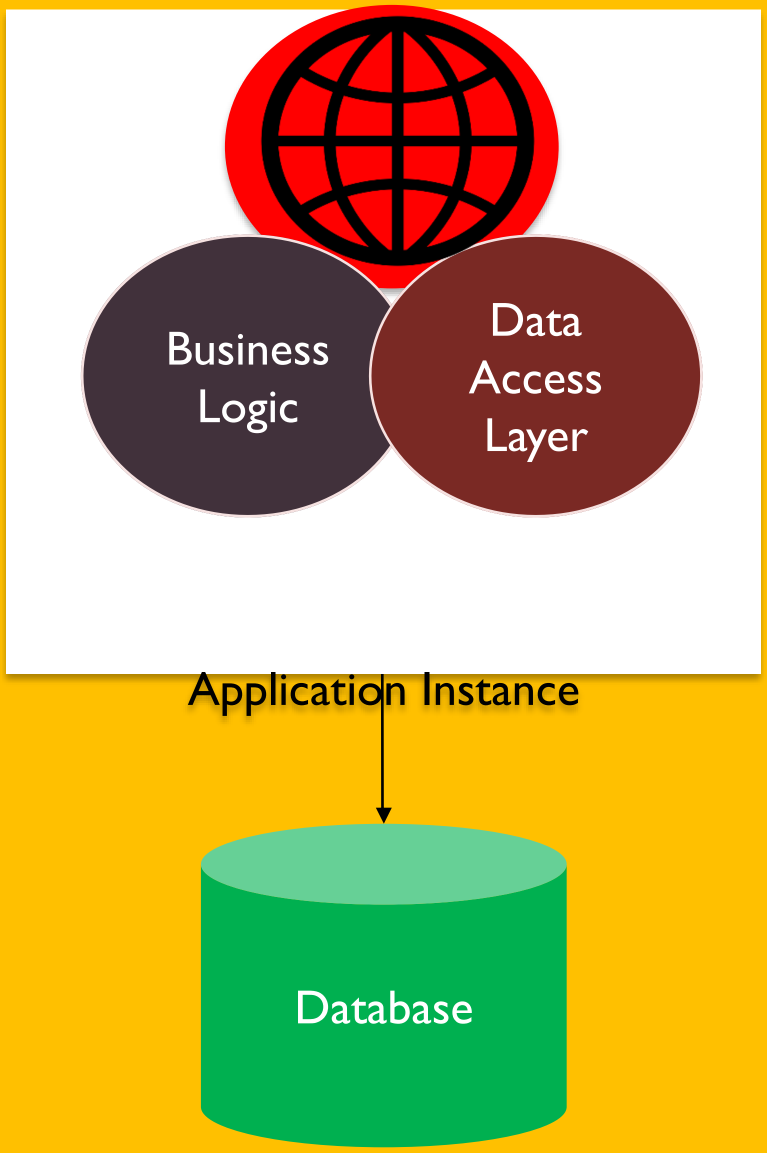
- As an architectural framework, Microservices are distributed and loosely coupled, so one team's changes won't break the entire app.

- The benefit to using Microservices is that development teams are able to rapidly build new components of apps to meet changing business needs.

- You are developing a server-side enterprise application. It must support a variety of different clients including desktop browsers, mobile browsers and native mobile applications. The application might also expose an API for 3rd parties to consume. It might also integrate with other applications via either web services or a message broker.

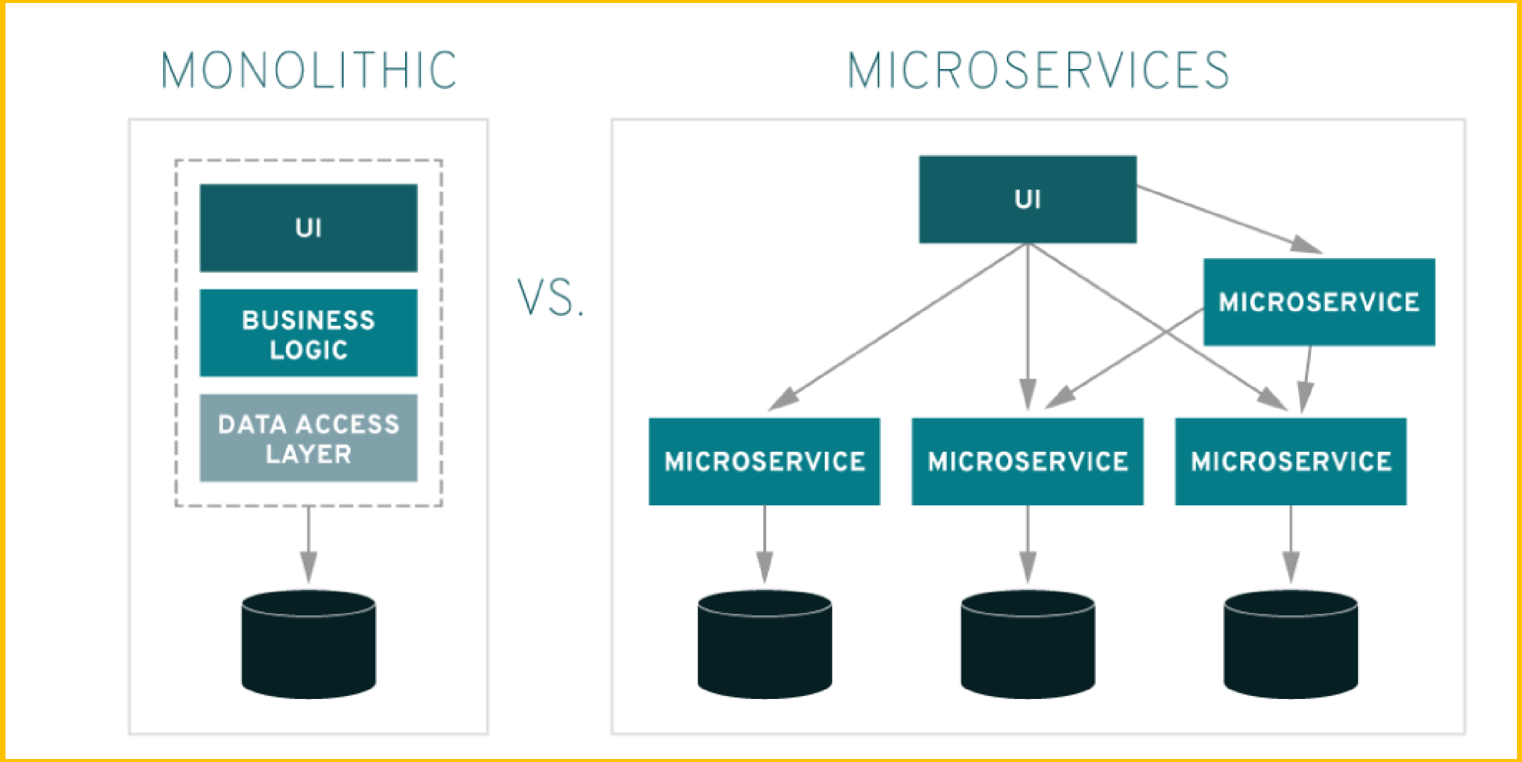
MONOLITHIC ARCHITECTURE

- A monolithic architecture is an architecture where all components for an application are collocated within a single unit.
- Monolithic application often consist of User interface, Business logic and Data Access layer.
- All this layers are combined on a single runtime instance of an application.
- Often suitable for small application



Gateway Group

ARCHITECTURE

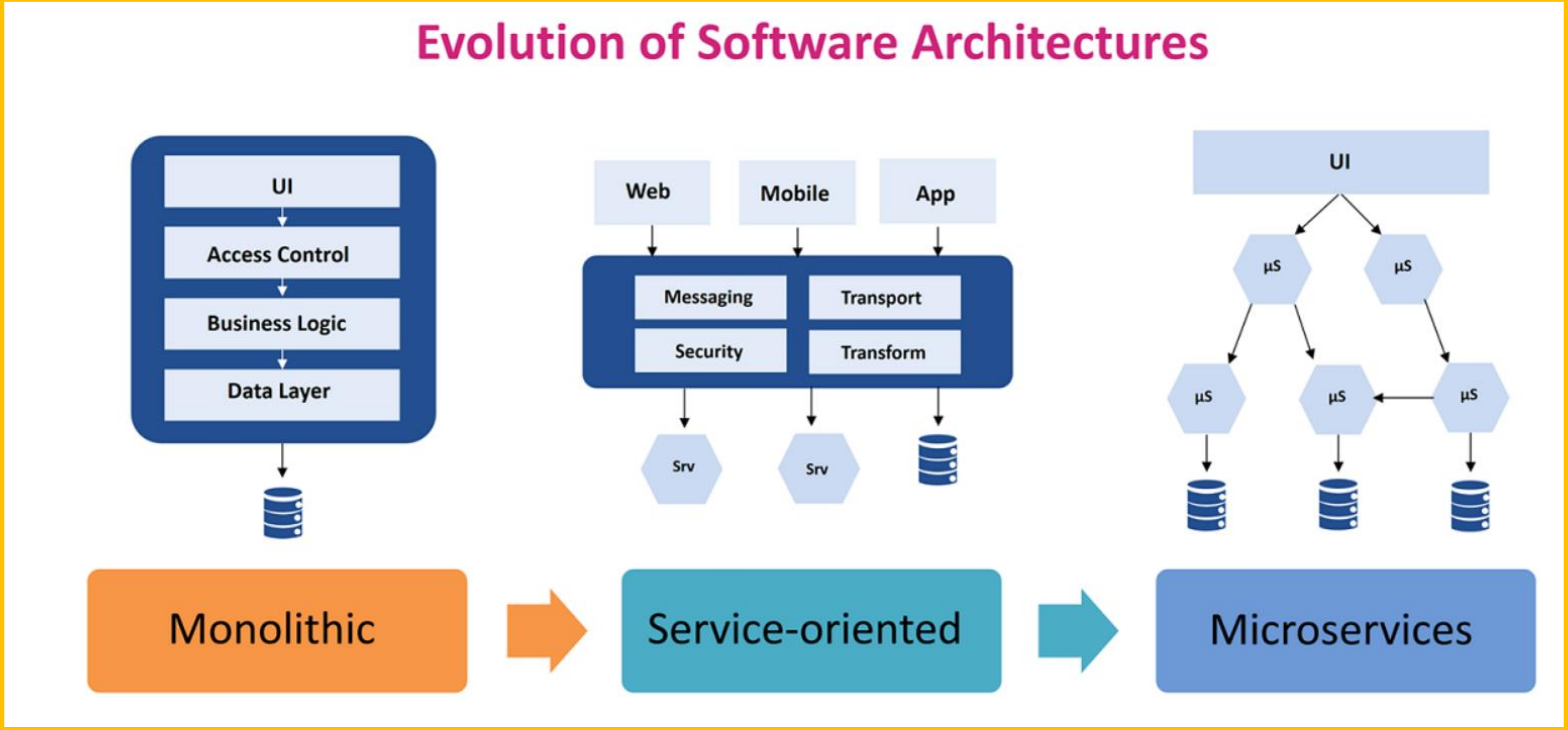


Gateway Group

LIMITATIONS OF MONOLITHIC

- Difficult to maintain
- Difficult to scale
- Difficult to manage deployments
- Tied to single technology stack, which limits innovation in new platforms and SDKs
- Difficult to reuse – Part of the application inside another application
- Fault in application instance brings down the entire application
- Difficult to update database schema

EVOLUTION OF MICROSERVICE



Gateway Group

MICROSERVICES ARCHITECTURE

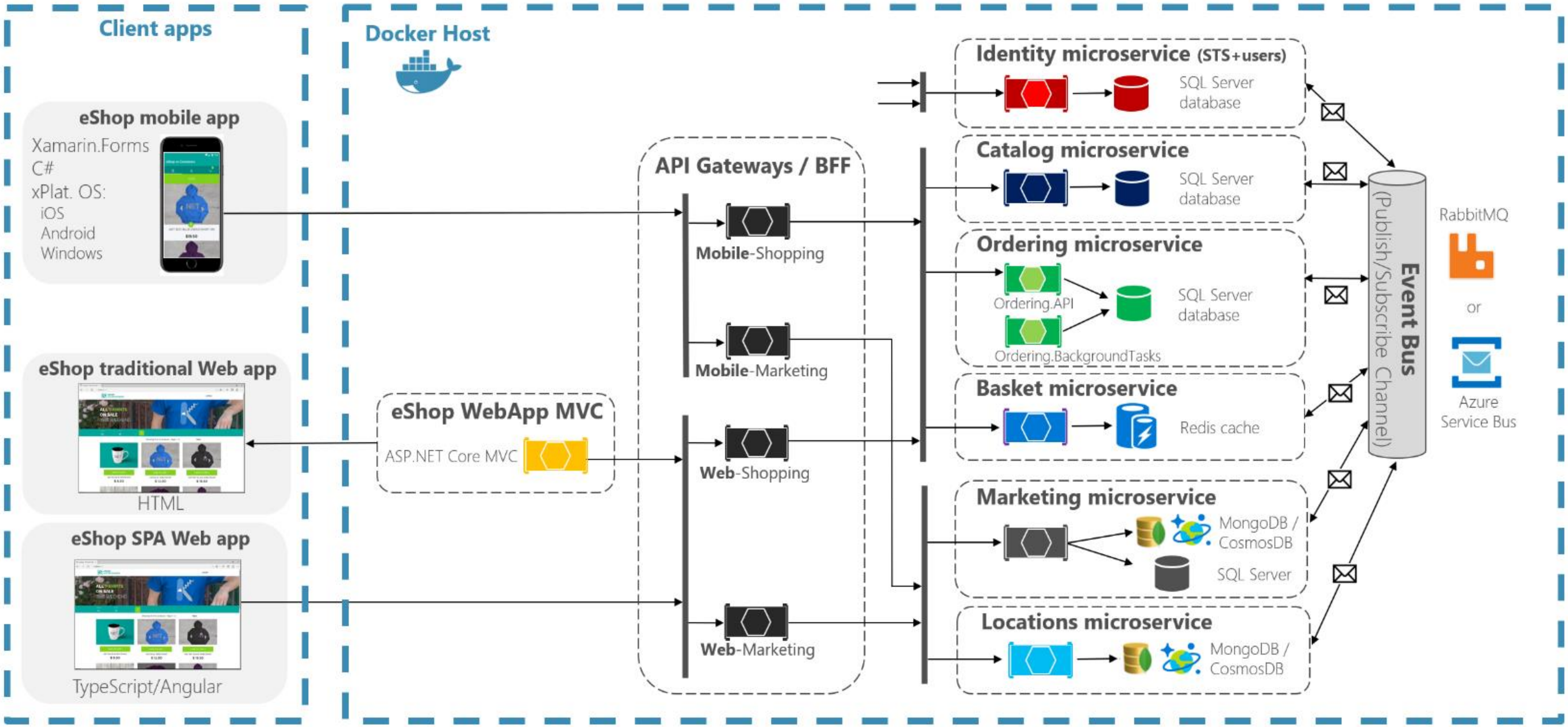
- Microservices are a software architecture style. It is a software development technique.
- Structure an application as collection of services.
- Each service implement a single business capability
- These services communicate with each other to achieve business goal
- These services are small, independent, and loosely coupled
- Services can be deployed independently. Facilitates continues delivery and deployment
- Services are responsible for persisting their own data or external state.
- Services don't need to share the same technology stack, libraries, or frameworks.

WHY USE MICROSERVICE

- Easier to scale service
- Better fault isolation
- Use the best approach
- Deliver value faster
- Easier to maintain and deploy
- Enables us to choose latest technologies
- Supports continuous integration and continuous delivery
- Easier to understand
- Facilitates code reuse
- Easier to integrate with other systems
- Suitable for large applications

eShopOnContainers reference application

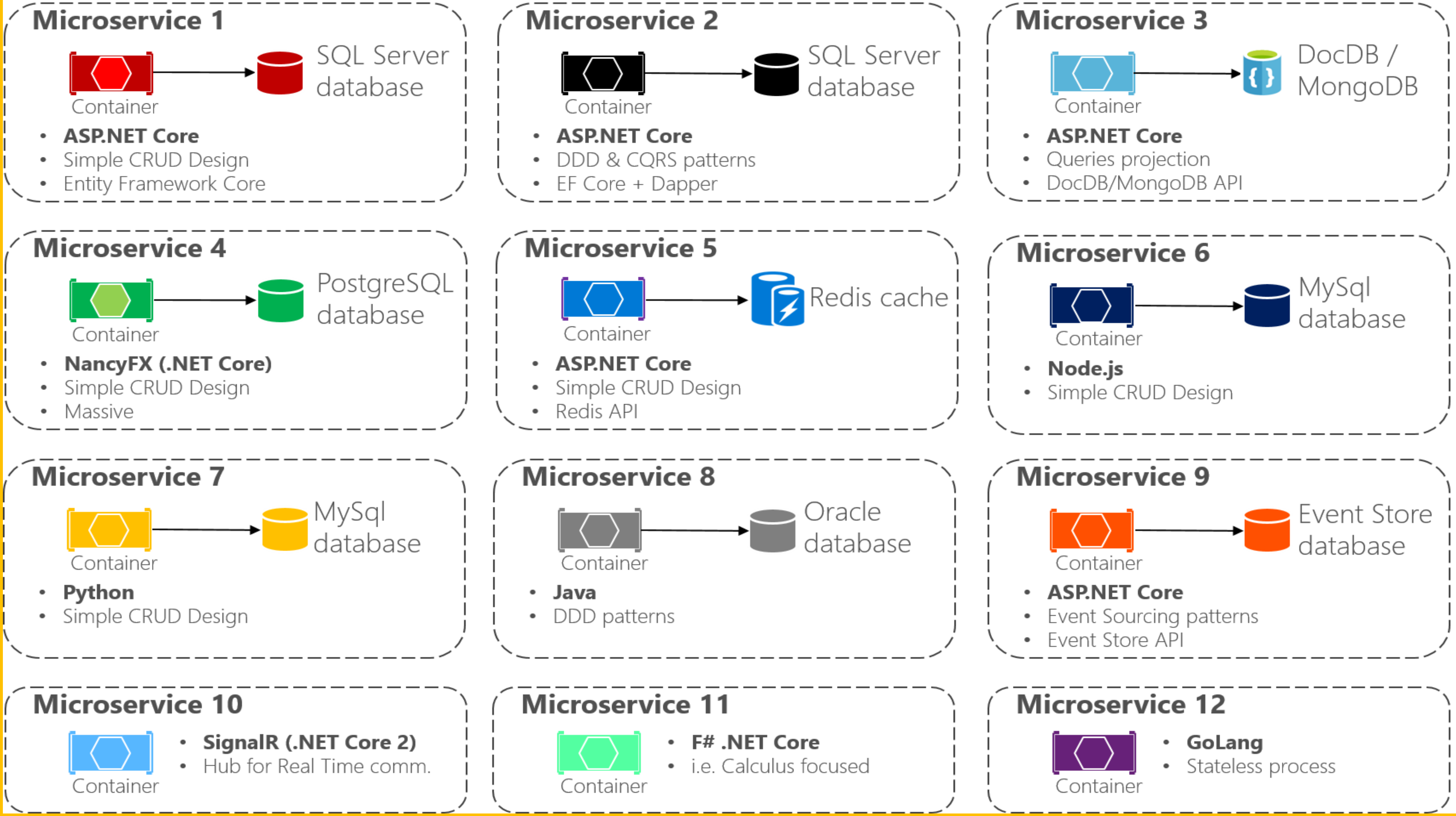
(Development environment architecture)



Gateway Group

Binkal Patel

The Multi-Architectural-Patterns and polyglot microservices world



Microservice 1

Container → SQL Server database

- **ASP.NET Core**
- Simple CRUD Design
- Entity Framework Core

Microservice 2

Container → SQL Server database

- **ASP.NET Core**
- DDD & CQRS patterns
- EF Core + Dapper

Microservice 3

Container → DocDB / MongoDB

- **ASP.NET Core**
- Queries projection
- DocDB/MongoDB API

Microservice 4

Container → PostgreSQL database

- **NancyFX (.NET Core)**
- Simple CRUD Design
- Massive

Microservice 5

Container → Redis cache

- **ASP.NET Core**
- Simple CRUD Design
- Redis API

Microservice 6

Container → MySql database

- **Node.js**
- Simple CRUD Design

Microservice 7

Container → MySql database

- **Python**
- Simple CRUD Design

Microservice 8

Container → Oracle database

- **Java**
- DDD patterns

Microservice 9

Container → Event Store database

- **ASP.NET Core**
- Event Sourcing patterns
- Event Store API

Microservice 10

Container

- **SignalR (.NET Core 2)**
- Hub for Real Time comm.

Microservice 11

Container

- **F# .NET Core**
- i.e. Calculus focused

Microservice 12

Container

- **GoLang**
- Stateless process

Gateway Group

Binkal Patel

WHAT ARE THE PROBLEMS CAUSING PEOPLE MOVE TOWARD THE MICROSERVICE WORLD

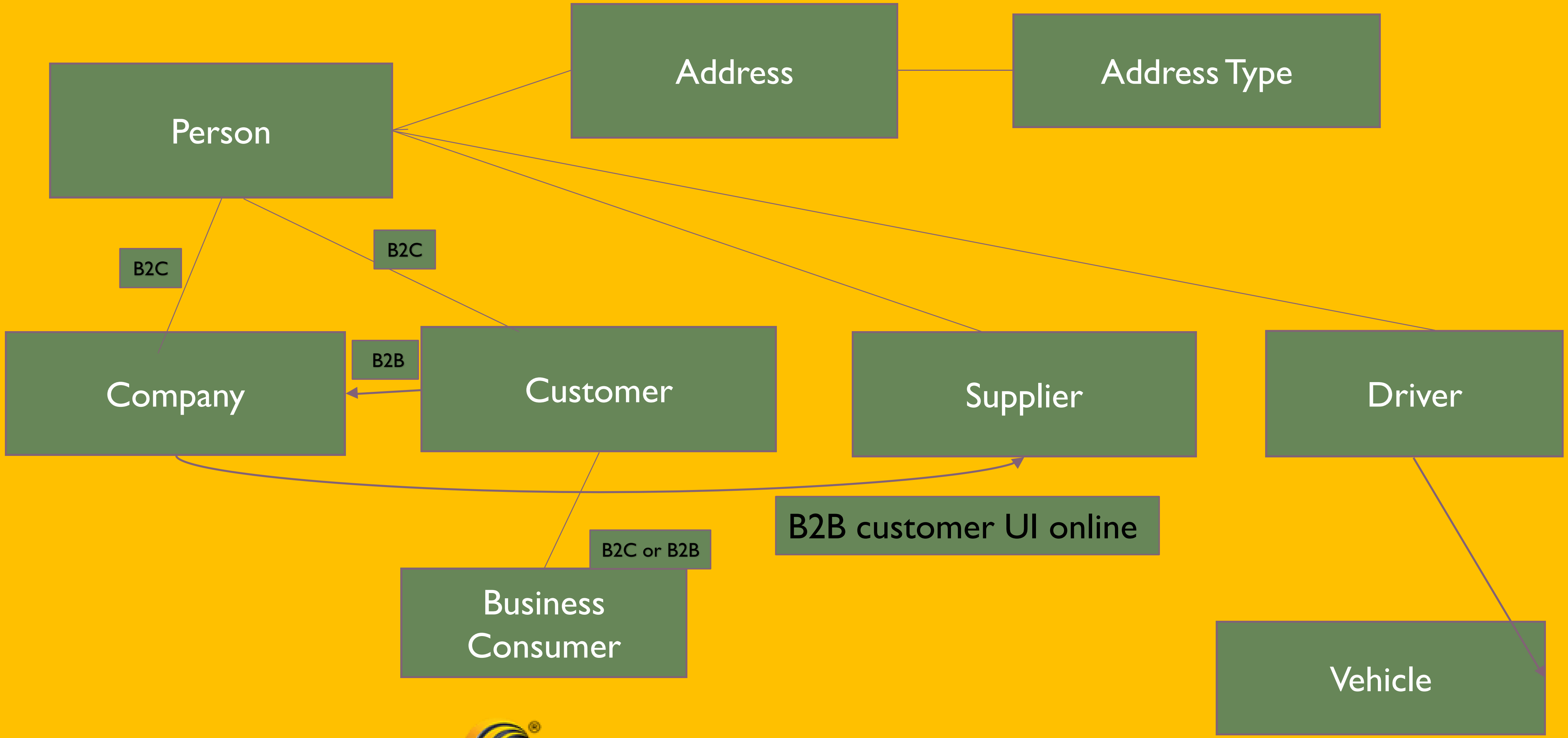
- How much of the workload should be moved to Microservices?
- Should you allow code to be migrated to different services?
- How do you decide what the boundaries of each microservice will be while the operation is running?
- How do you monitor the performance of Microservices?

Gateway Group

CHALLENGES OF MICROSERVICES

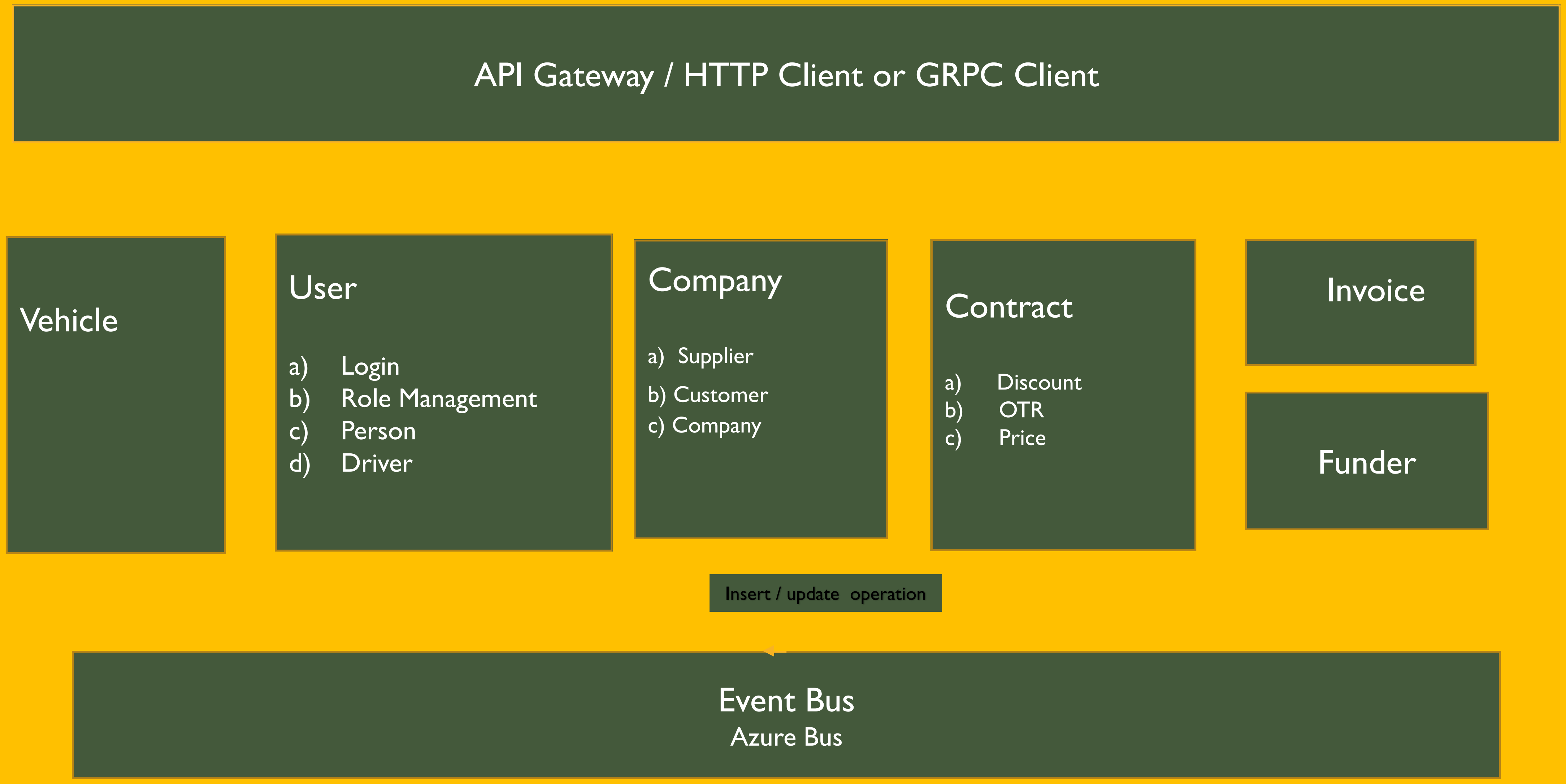
- Due to distributed deployment, testing can become complicated
- Increasing number of services can result in information barriers
- The architecture brings additional complexity as the developers have to mitigate fault tolerance, network latency, and deal with a variety of message formats as well as load balancing
- Being a distributed system, it can result in duplication of effort
- Handling use cases that span more than one service without using distributed transactions is not only tough but also requires communication and cooperation between different teams
- Partitioning the application into Microservices requires experienced and skilled architects

HOW TO CONVERT MONOLITH ARCH TO MICROSERVICES EVOLUTION



Gateway Group

Binkal Patel



API Gateway / HTTP Client or GRPC Client

Vehicle

User

a) Login
b) Role Management
c) Person
d) Driver

Company

a) Supplier
b) Customer
c) Company

Contract

a) Discount
b) OTR
c) Price

Invoice

Funder

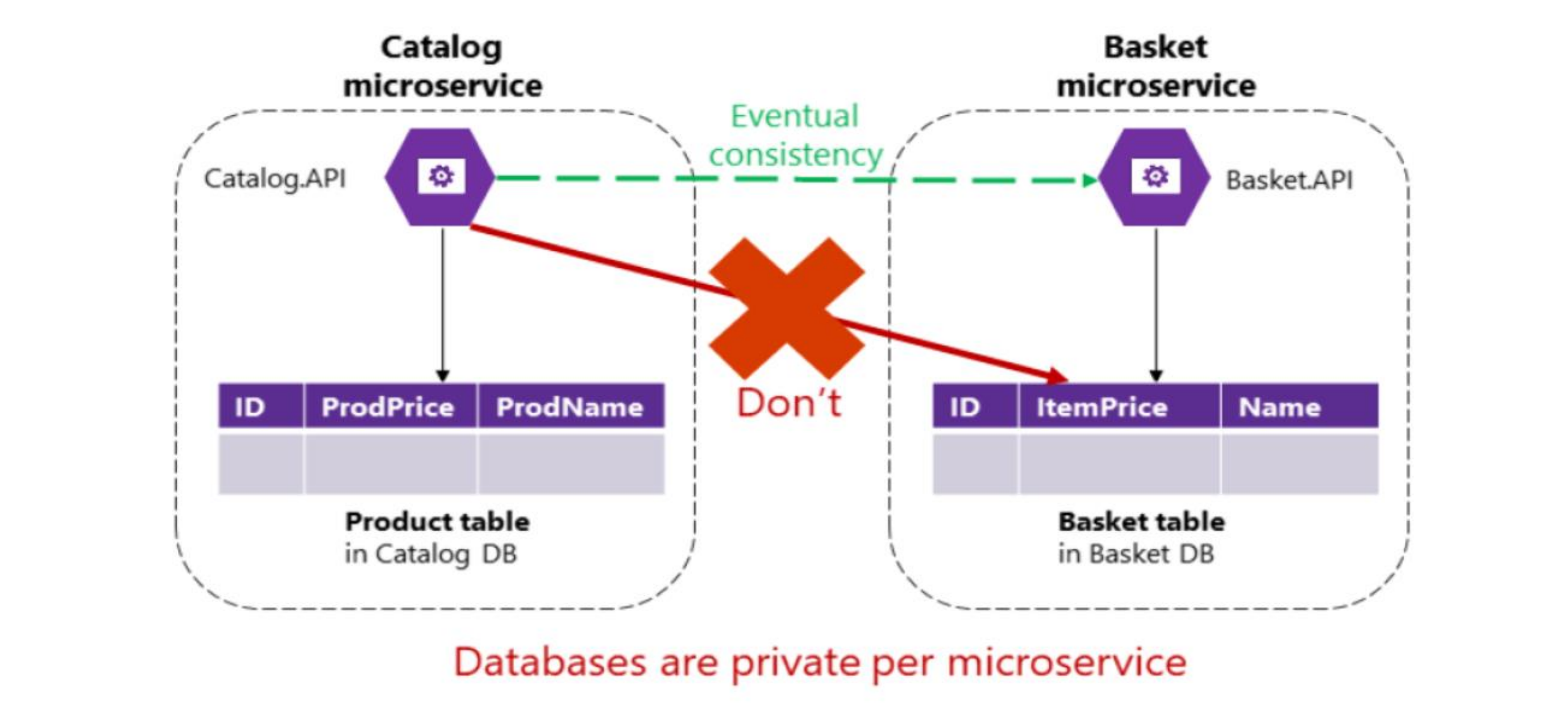
Insert / update operation

Event Bus
Azure Bus

Gateway Group

Binkal Patel

MICROSERVICE BOUNDARIES

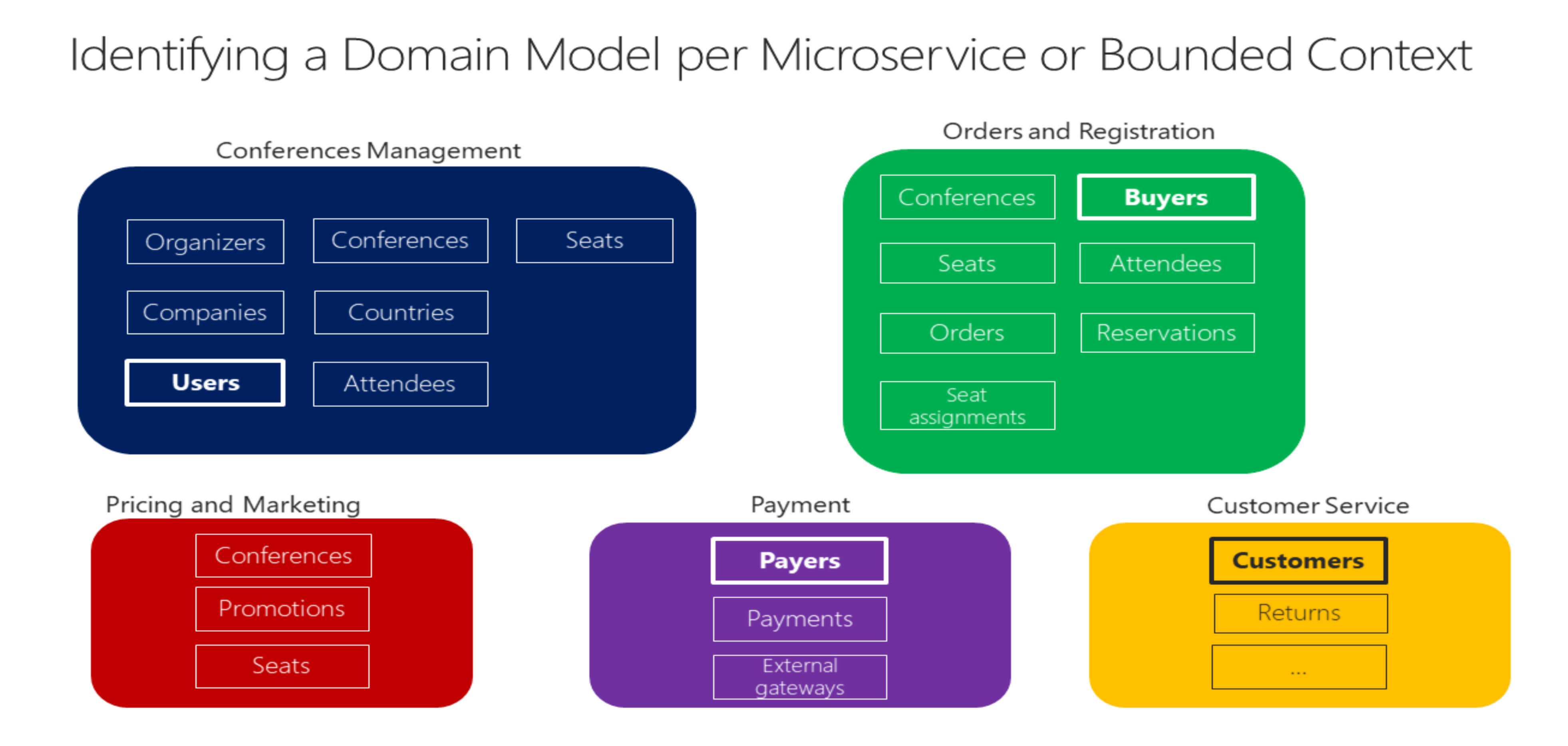


ID	ProdPrice	ProdName

ID	ItemPrice	Name

Databases are private per microservice

IDENTIFY DOMAIN-MODEL BOUNDARIES FOR EACH MICROSERVICE



Identifying a Domain Model per Microservice or Bounded Context

Conferences Management
- Organizers
- Conferences
- Seats
- Companies
- Countries
- **Users**
- Attendees

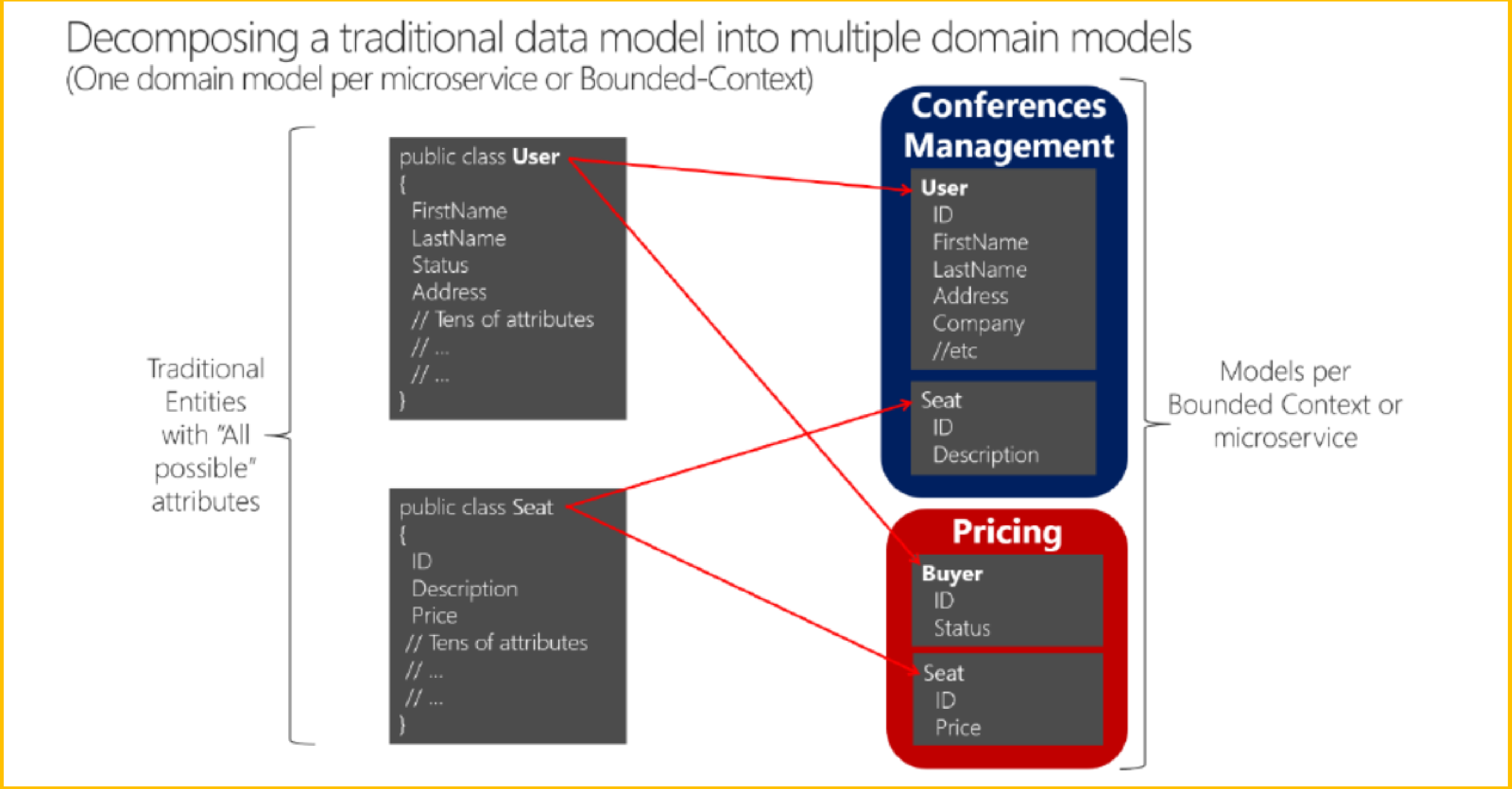
Orders and Registration
- Conferences
- **Buyers**
- Seats
- Attendees
- Orders
- Reservations
- Seat assignments

Pricing and Marketing
- Conferences
- Promotions
- Seats

Payment
- **Payers**
- Payments
- External gateways

Customer Service
- **Customers**
- Returns
- ...

DATABASE



Decomposing a traditional data model into multiple domain models

(One domain model per microservice or Bounded-Context)

Traditional Entities with "All possible" attributes

```
public class User
{
  FirstName
  LastName
  Status
  Address
  // Tens of attributes
  // ...
  // ...
}
```

```
public class Seat
{
  ID
  Description
  Price
 // Tens of attributes
 // ...
 // ...
}
```

Conferences Management

```
User
  ID
  FirstName
  LastName
  Address
  Company
  //etc
```

```
Seat
  ID
  Description
```

Pricing

```
Buyer
  ID
  Status
```

```
Seat
  ID
  Price
```

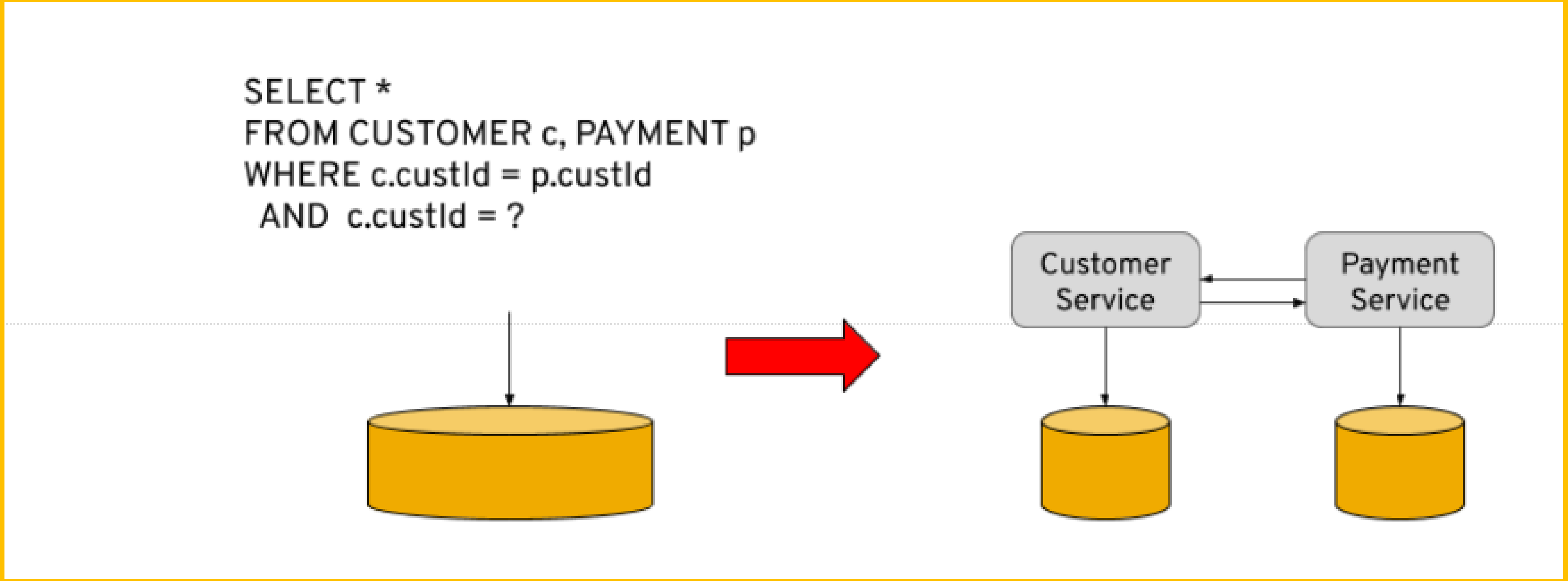
Models per Bounded Context or microservice

Gateway Group

MICROSERVICE FUNDAMENTALS

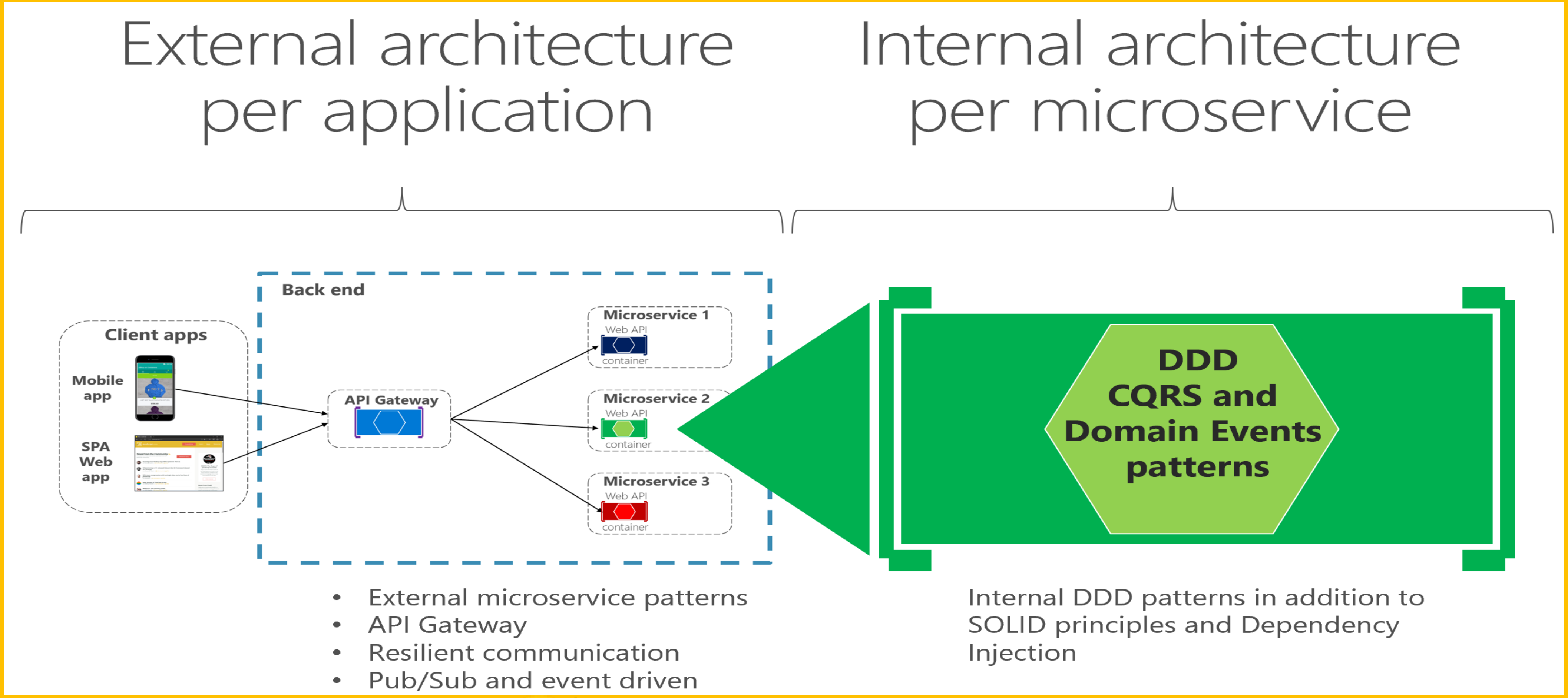
- The Scope Of Functionality
- High Cohesion Combined With Loose Coupling
- Unique Source Of Identification
- API Integration
- Data Storage Segregation
- Traffic Management
- Automating The Process
- Minimal Database Tables
- Constant Monitoring

MICROSERVICE DATA



```
SELECT *
FROM CUSTOMER c, PAYMENT p
WHERE c.custId = p.custId
 AND  c.custId = ?
```

DIFFERENT TYPES OF MICROSERVISE , MODULER, DDD



External architecture per application

Internal architecture per microservice

- External microservice patterns
- API Gateway
- Resilient communication
- Pub/Sub and event driven

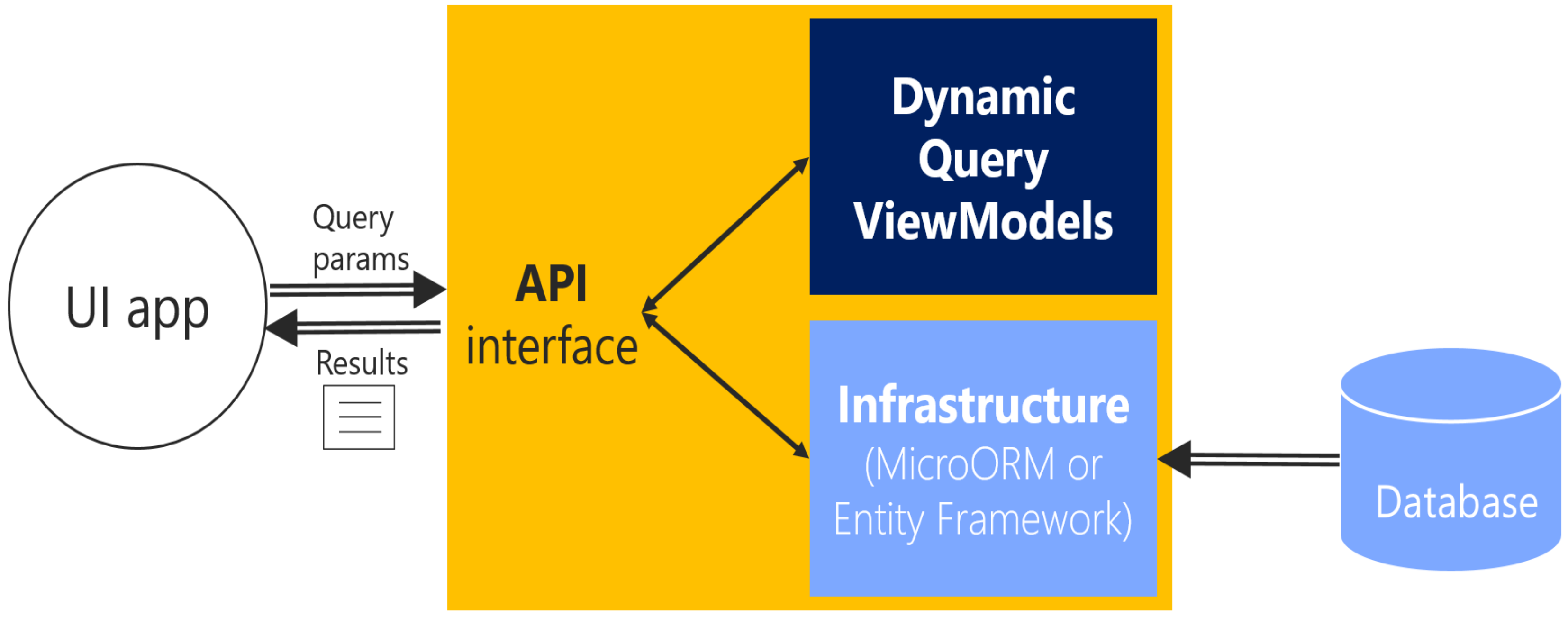
Internal DDD patterns in addition to SOLID principles and Dependency Injection

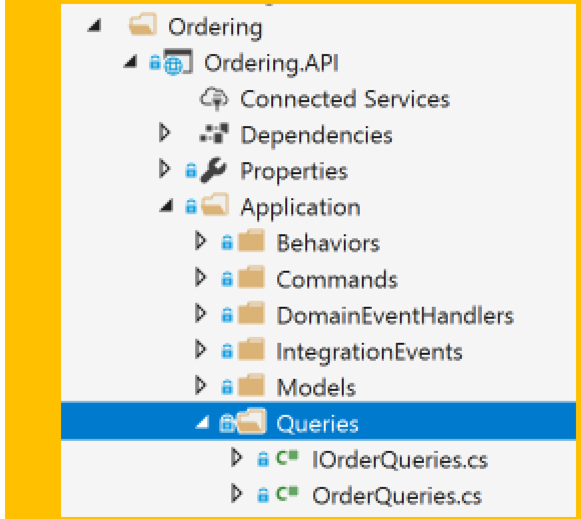
Gateway Group

Binkal Patel

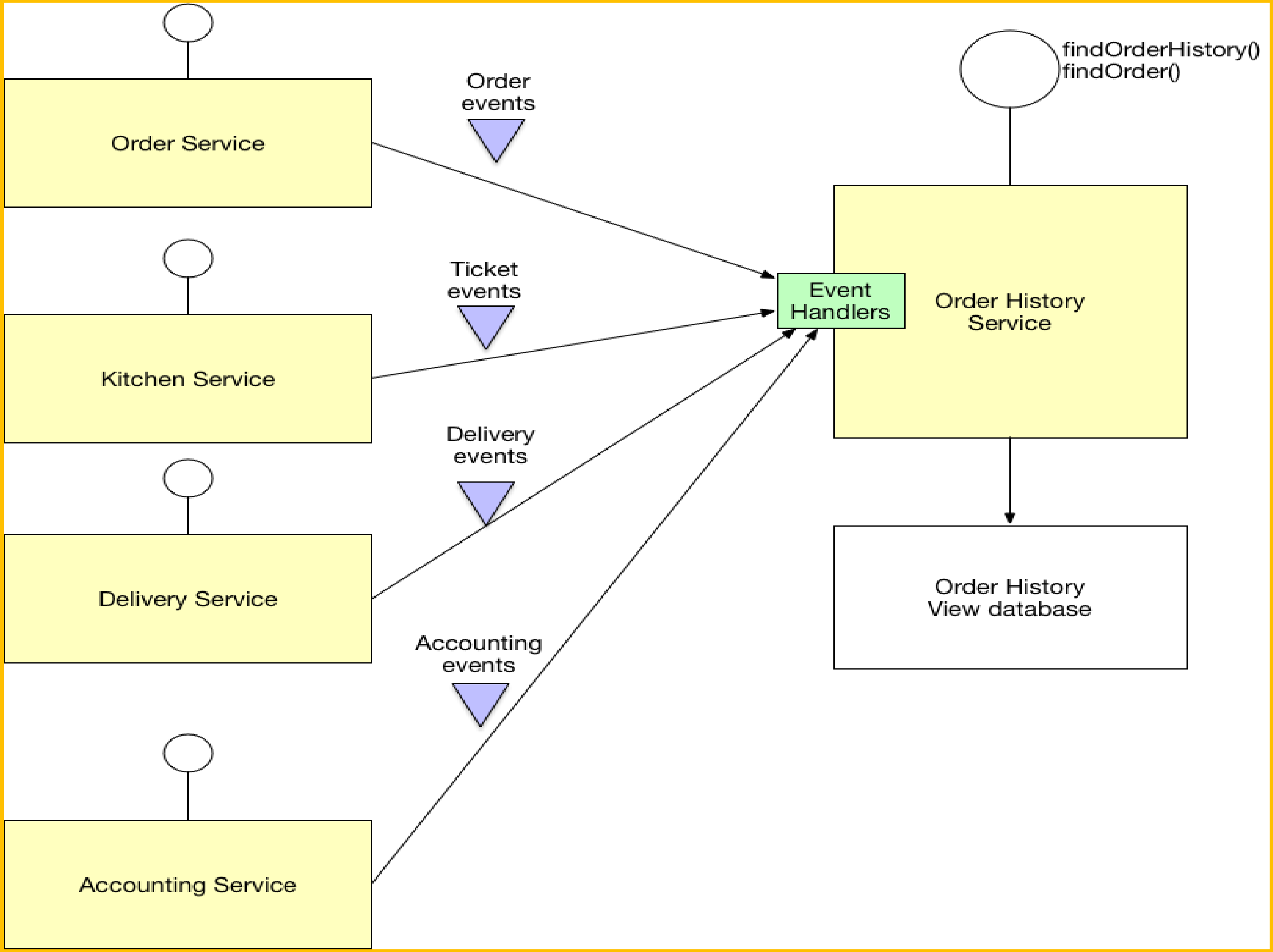
IMPLEMENT READS/QUERIES IN A CQRS MICROSERVICE

High level "Queries-side" in a simplified CQRS





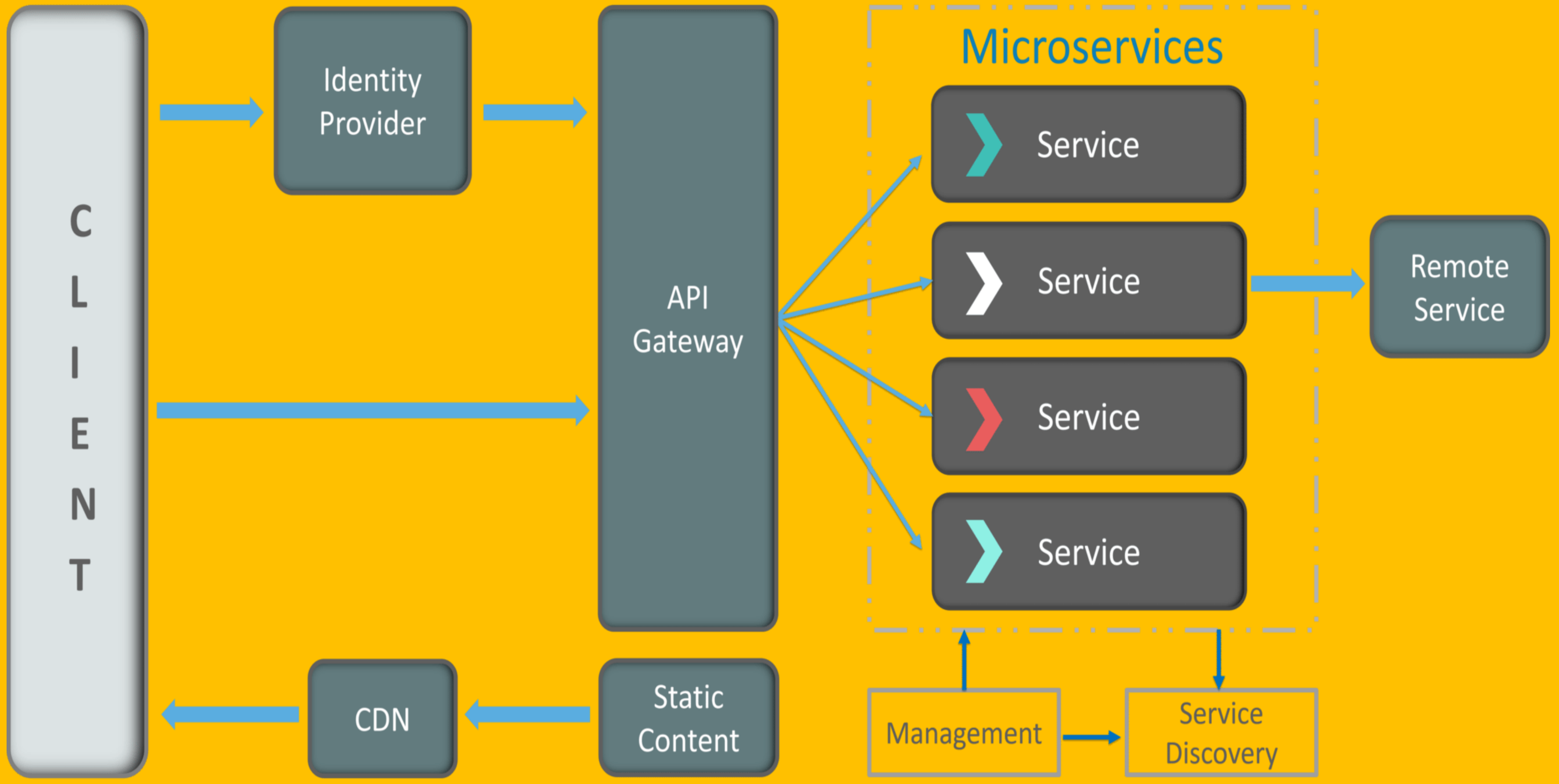
```
Ordering
  Ordering.API
    Connected Services
    Dependencies
    Properties
    Application
      Behaviors
      Commands
      DomainEventHandlers
      IntegrationEvents
      Models
      Queries
        IOrderQueries.cs
        OrderQueries.cs
```



Gateway Group

MICROSERVICE COMPONENTS

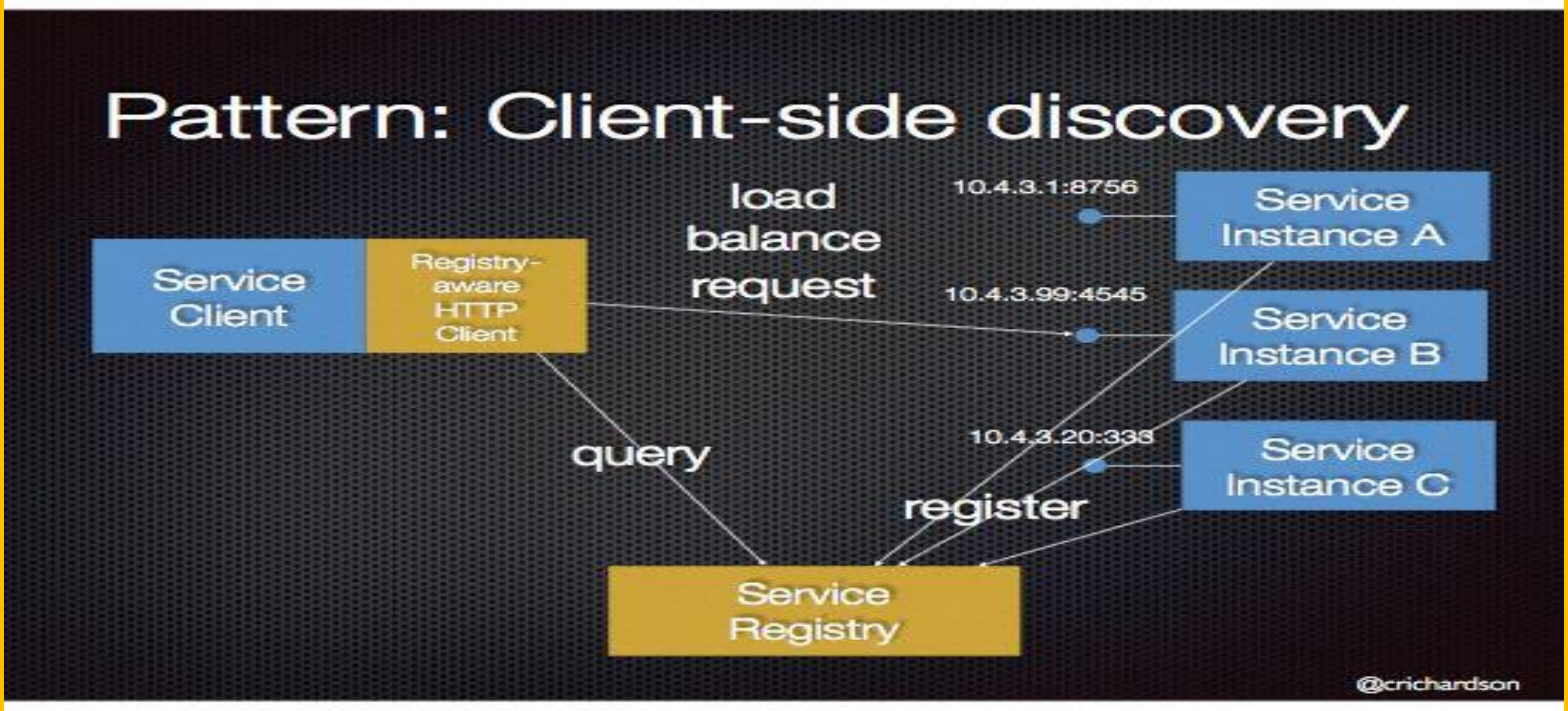
- Clients.
- Identity Providers.
- API Gateway.
- Messaging Formats.
- Databases.
- Static Content.
- Management.
- Service Discovery.



Gateway Group

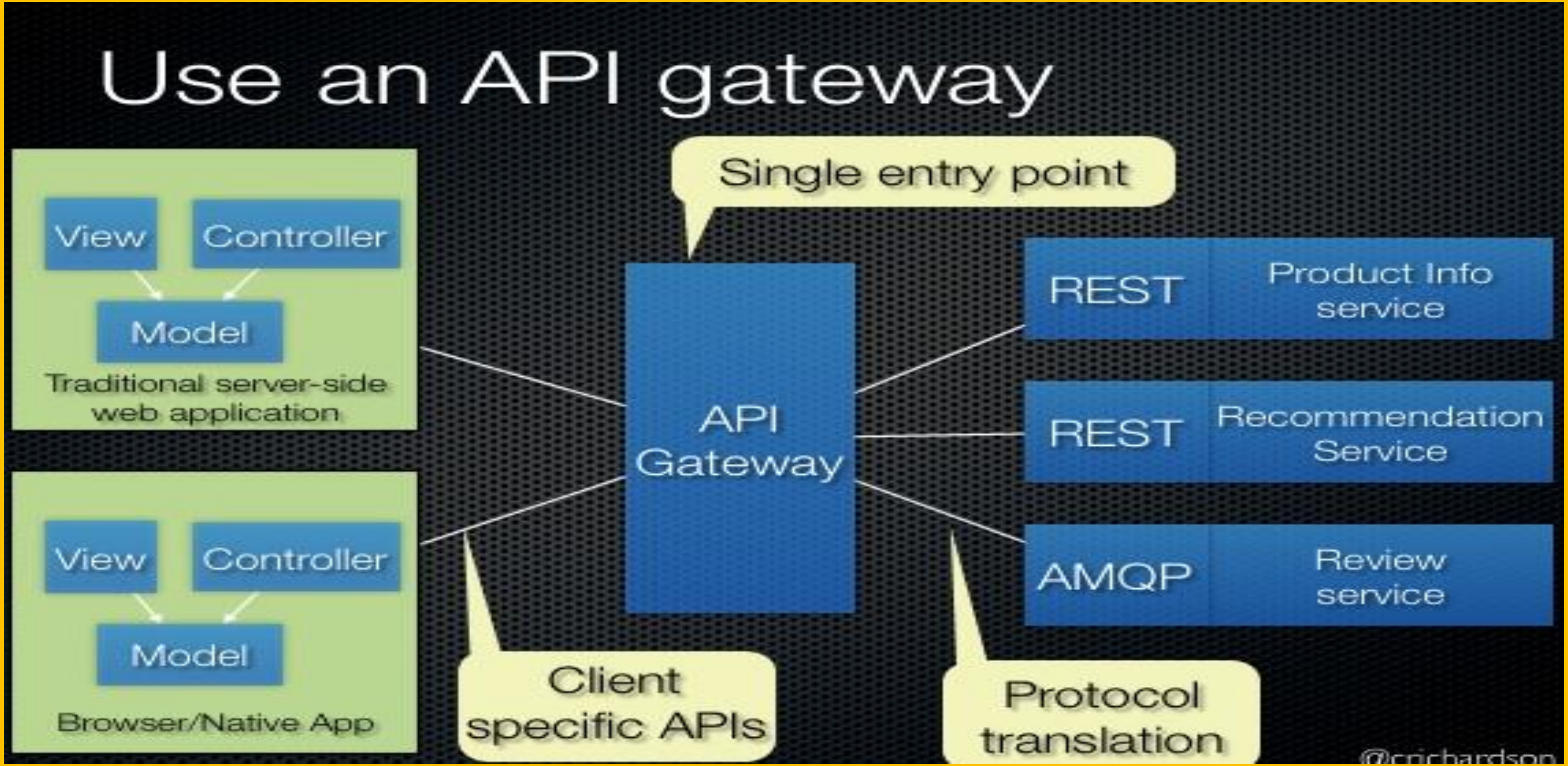
Binkal Patel

SERVICE DISCOVERY.



Gateway Group

API GATEWAY



Use an API gateway

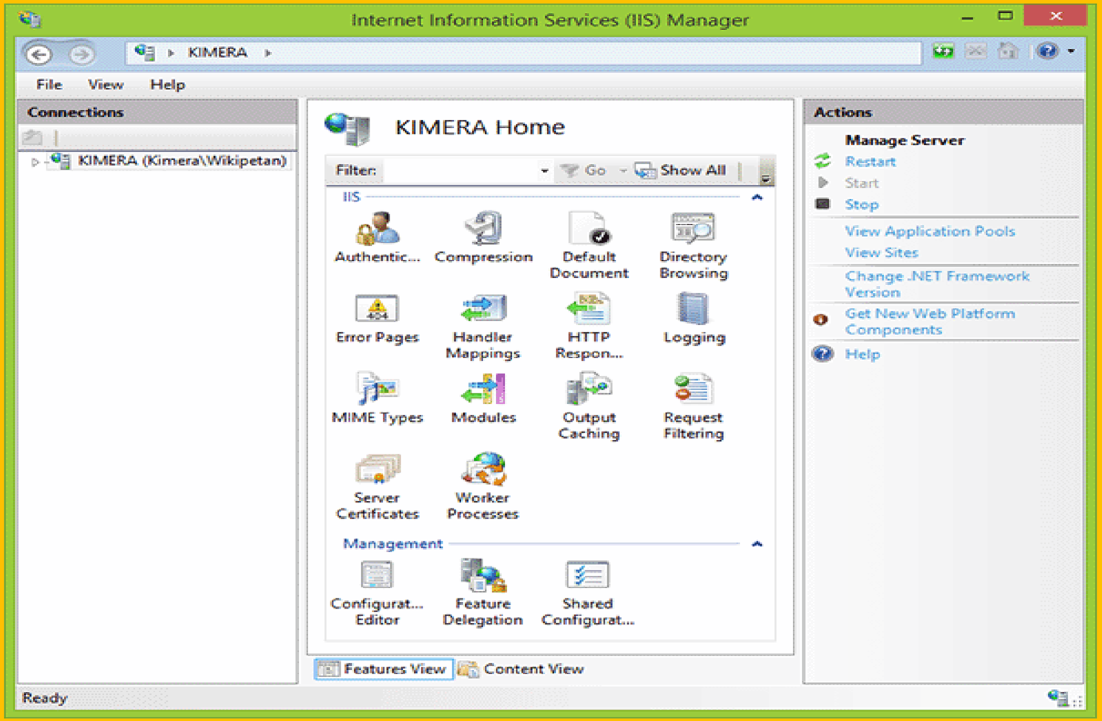
Single entry point

Client specific APIs

Protocol translation

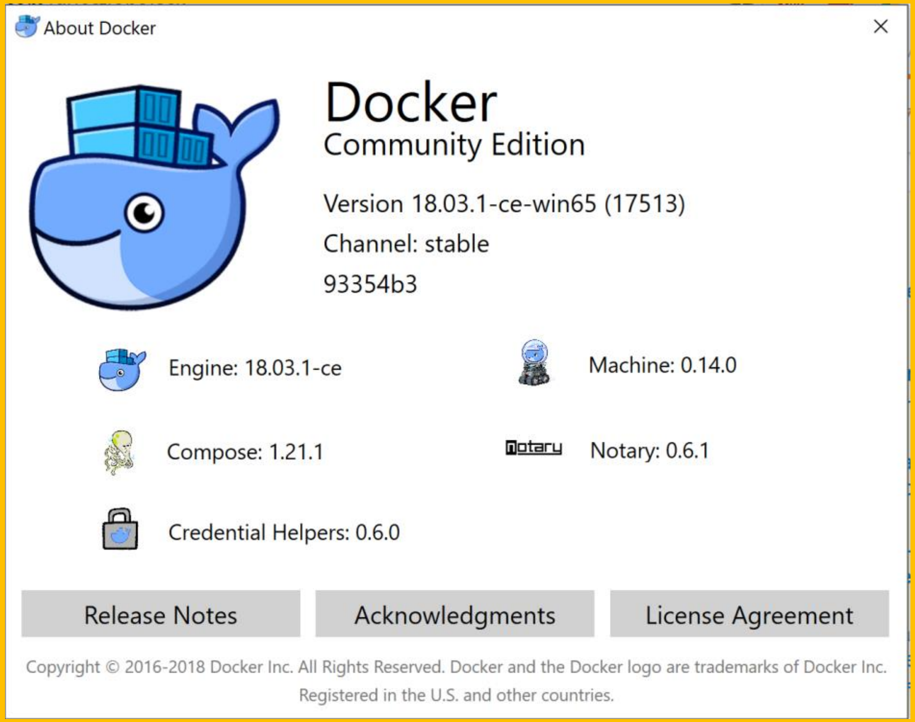
DEPLOYMENT MICROSERVICE

IIS



OR

Docker Host



Docker
Community Edition

Version 18.03.1-ce-win65 (17513)
Channel: stable
93354b3

Engine: 18.03.1-ce

Machine: 0.14.0

Compose: 1.21.1

Notary: 0.6.1

Credential Helpers: 0.6.0

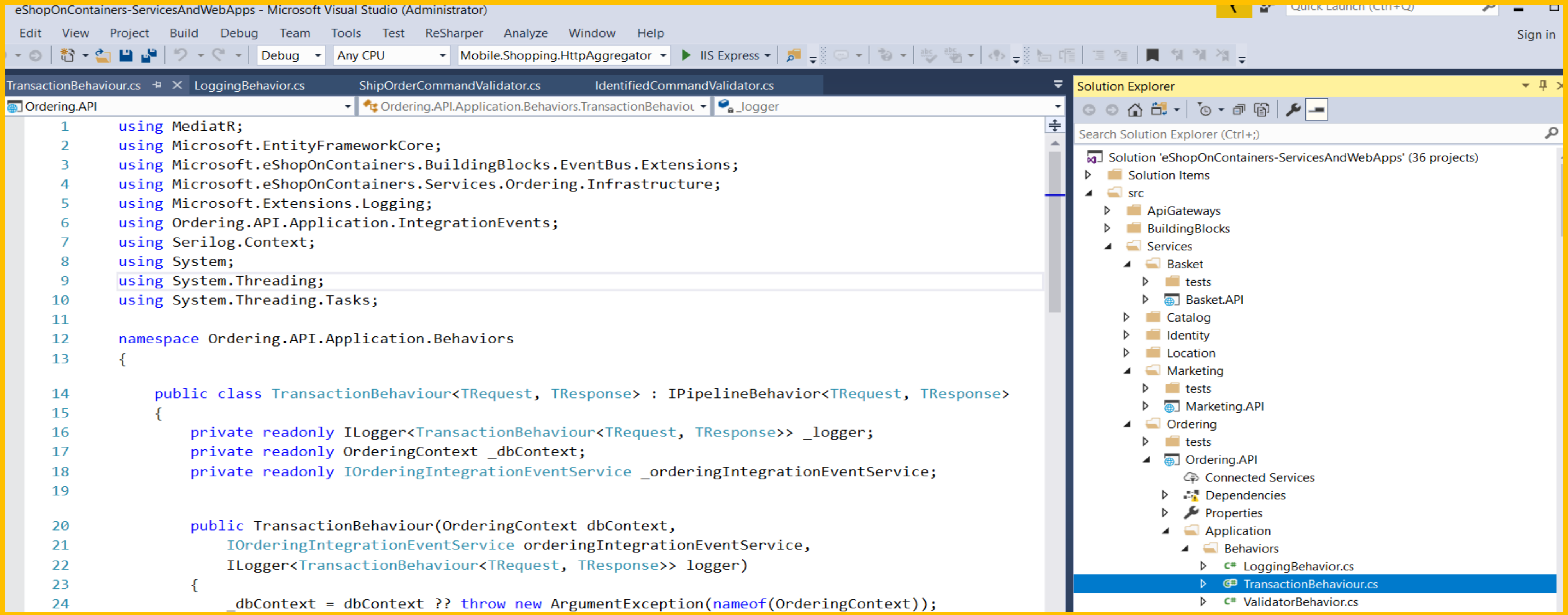
Copyright © 2016-2018 Docker Inc. All Rights Reserved. Docker and the Docker logo are trademarks of Docker Inc. Registered in the U.S. and other countries.

Gateway Group

Binkal Patel

SAMPLE APPLICATION ON MICROSERVICE BASED ARCHITECTURE

- https://github.com/dotnet-architecture/eShopOnContainers



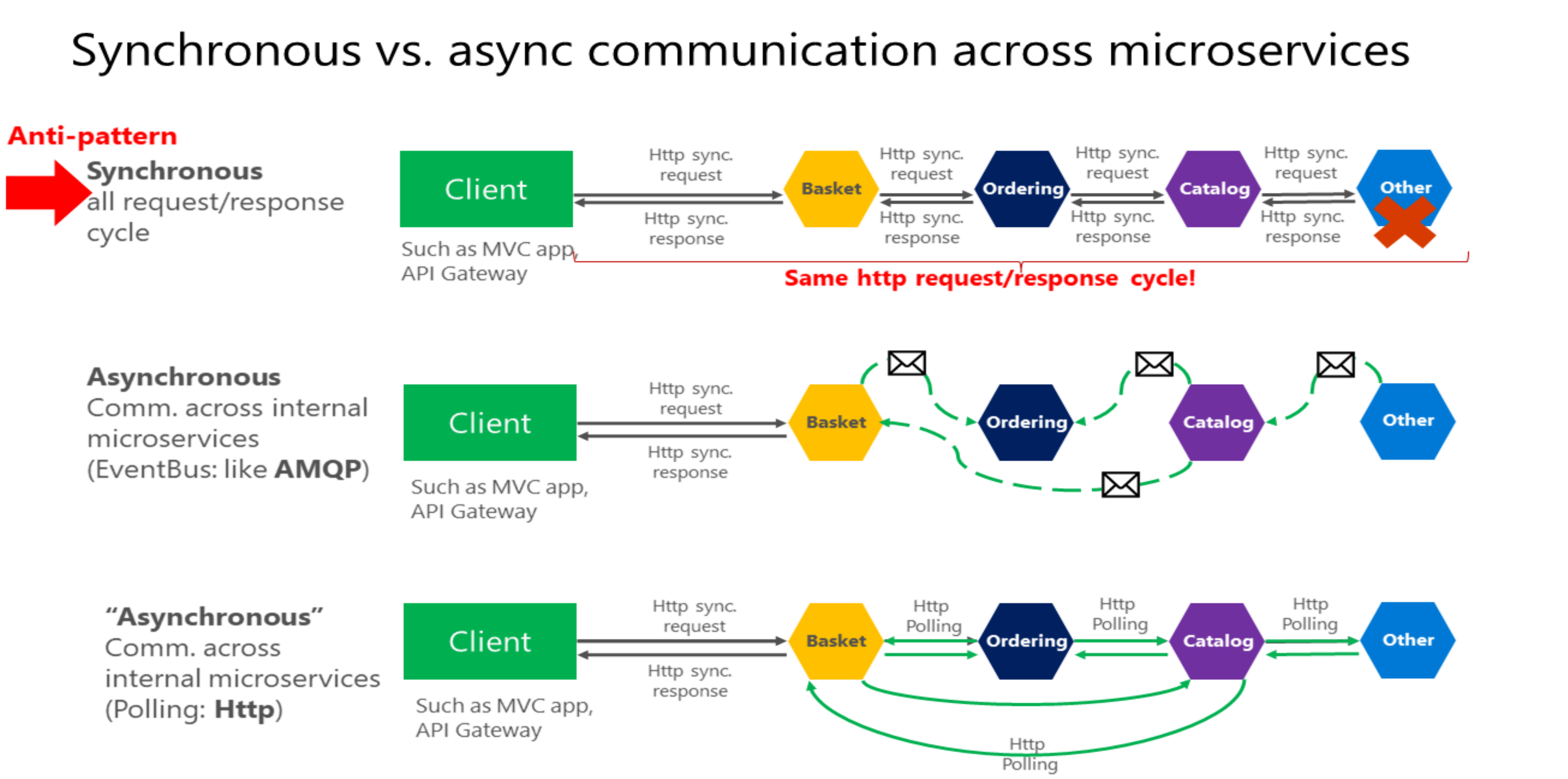
```
using MediatR;
using Microsoft.EntityFrameworkCore;
using Microsoft.eShopOnContainers.BuildingBlocks.EventBus.Extensions;
using Microsoft.eShopOnContainers.Services.Ordering.Infrastructure;
using Microsoft.Extensions.Logging;
using Ordering.API.Application.IntegrationEvents;
using Serilog.Context;
using System;
using System.Threading;
using System.Threading.Tasks;

namespace Ordering.API.Application.Behaviors
{
    public class TransactionBehaviour<TRequest, TResponse> : IPipelineBehavior<TRequest, TResponse>
    {
        private readonly ILogger<TransactionBehaviour<TRequest, TResponse>> _logger;
        private readonly OrderingContext _dbContext;
        private readonly IOrderingIntegrationEventService _orderingIntegrationEventService;

        public TransactionBehaviour(OrderingContext dbContext,
            IOrderingIntegrationEventService orderingIntegrationEventService,
            ILogger<TransactionBehaviour<TRequest, TResponse>> logger)
        {
            _dbContext = dbContext ?? throw new ArgumentException(nameof(OrderingContext));
```

Gateway Group

DIFFERENT COMMUNICAITON PATTERNS BETWEEN MICROSERVICES



Synchronous vs. async communication across microservices

Anti-pattern

Synchronous
all request/response cycle

Asynchronous
Comm. across internal microservices
(EventBus: like **AMQP**)

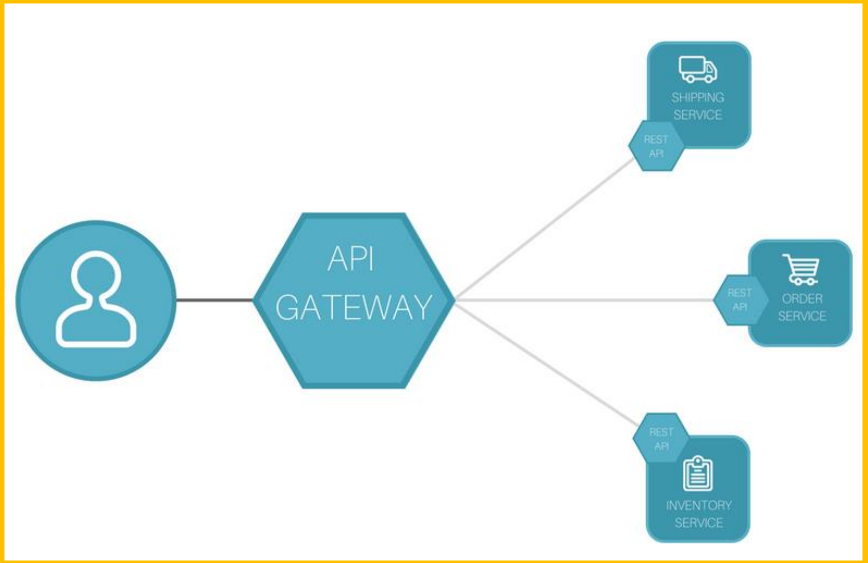
"Asynchronous"
Comm. across internal microservices
(Polling: **Http**)

Same http request/response cycle!

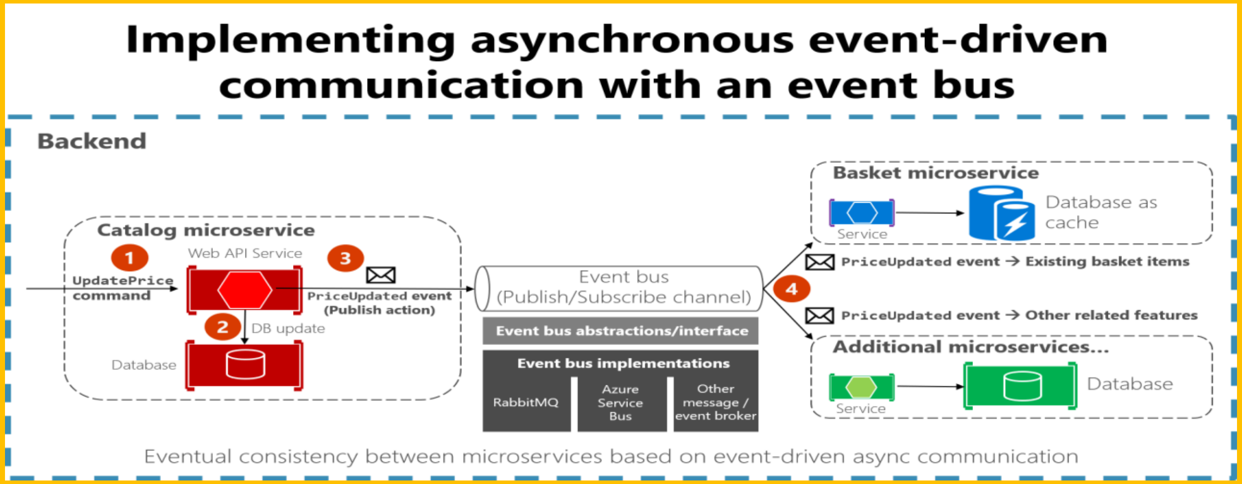
Gateway Group

MICROSERVICE COMMUNICATION

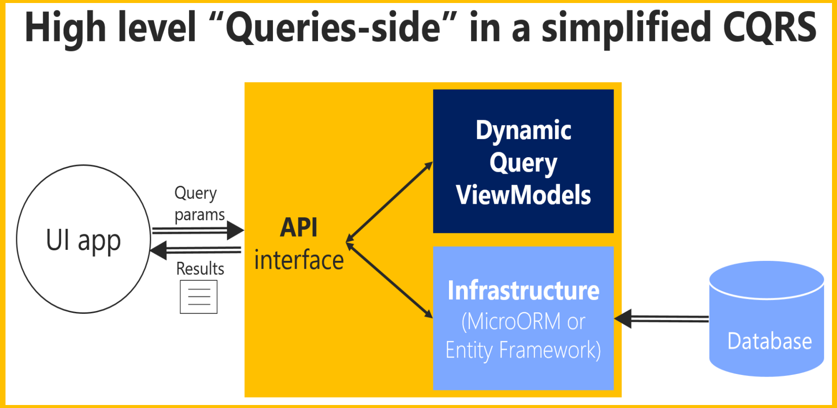
API Gateway



Event Bus /Azure Bus


Implementing asynchronous event-driven communication with an event bus
Backend
Catalog microservice
Web API Service
1
UpdatePrice command
2
DB update
3
PriceUpdated event (Publish action)
Database
Event bus (Publish/Subscribe channel)
Event bus abstractions/interface
Event bus implementations
RabbitMQ
Azure Service Bus
Other message / event broker
4
Basket microservice
Service
Database as cache
PriceUpdated event → Existing basket items
PriceUpdated event → Other related features
Additional microservices...
Service
Database
Eventual consistency between microservices based on event-driven async communication

CQRS


High level "Queries-side" in a simplified CQRS
UI app
Query params
Results
API interface
Dynamic Query ViewModels
Infrastructure (MicroORM or Entity Framework)
Database

Gateway Group

TESTING MICROSERVICE

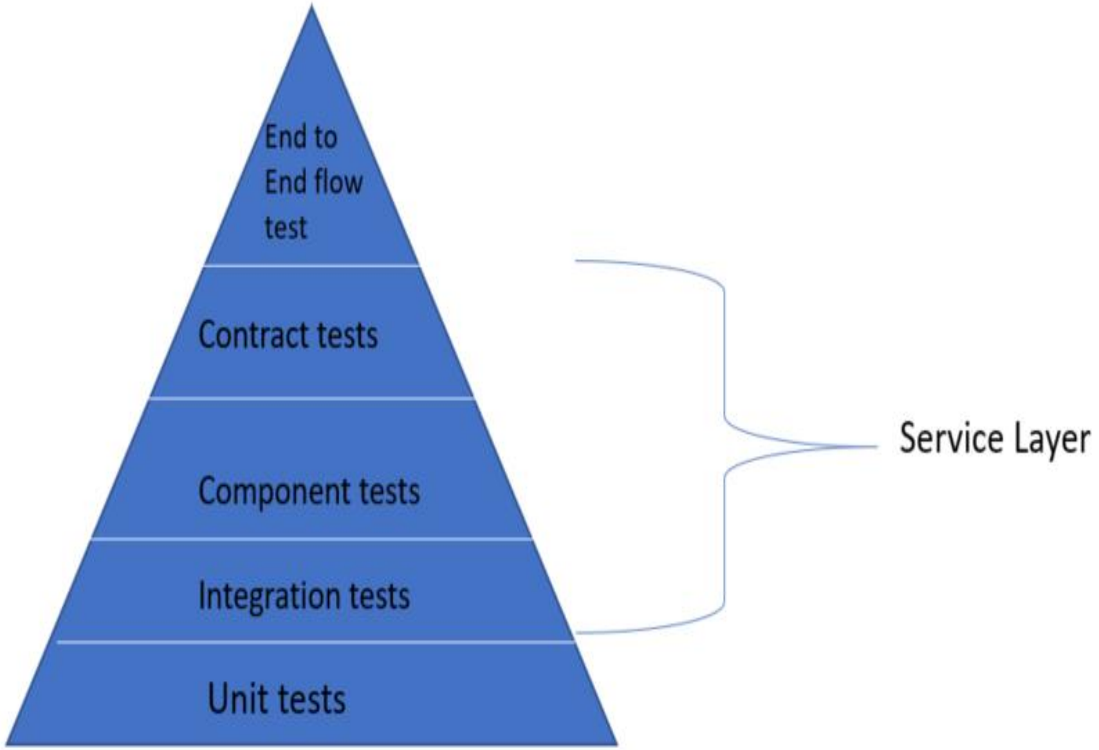
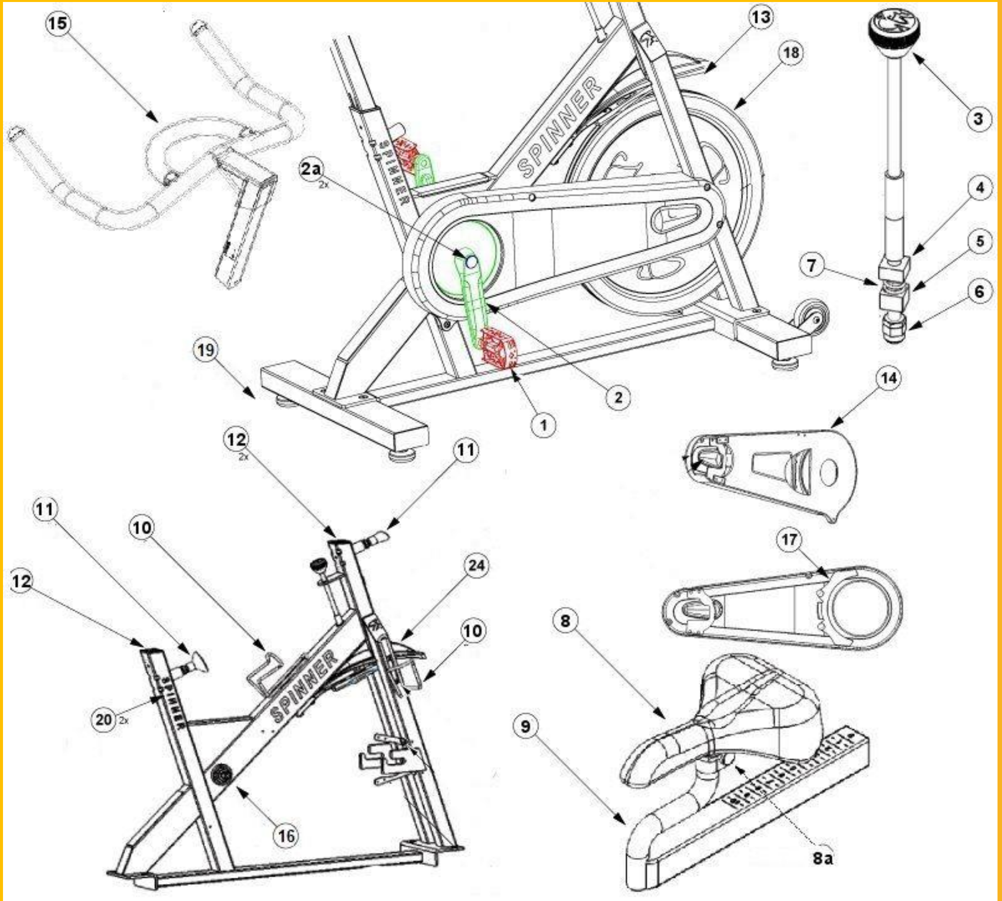


Fig 2. Test Pyramid for microservices



Gateway Group

DEPLOYMENT OF MICROSERVICE ON DOCKER CONTAINERS

- URL :
- https://docs.docker.com/get-docker/
- https://docs.docker.com/develop/

docker docs

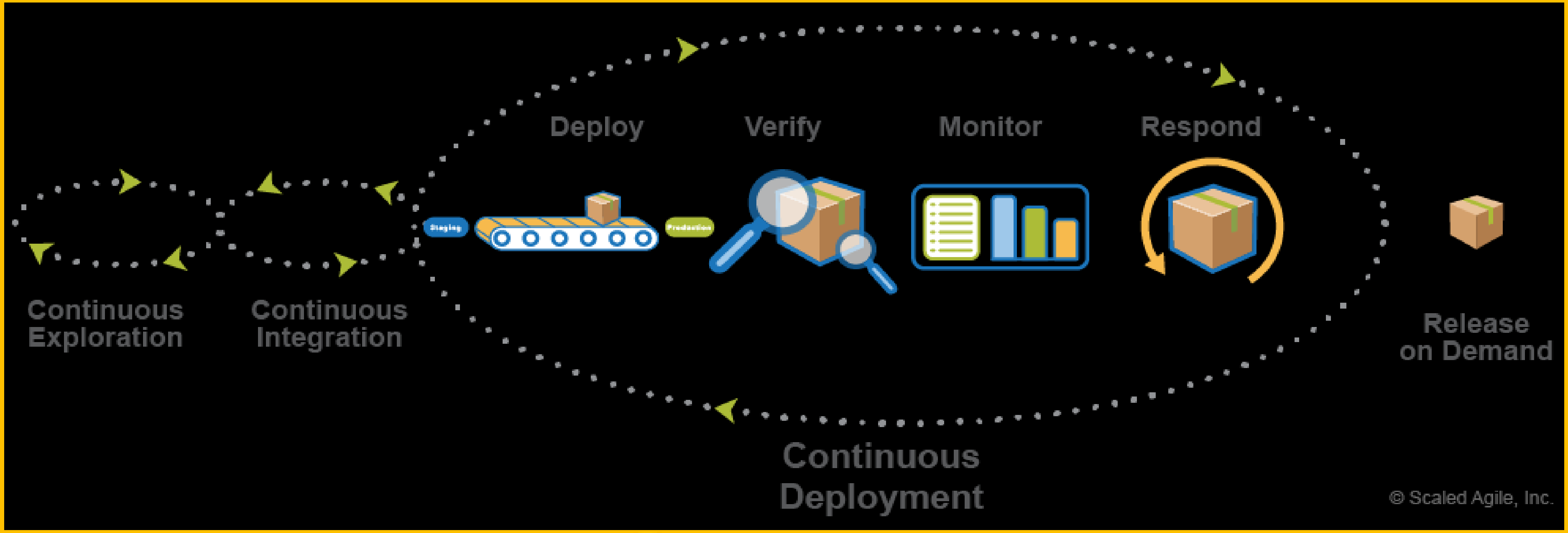
Gateway Group

MICROSERVICE SECURITY DAY

- Data Encryption
- Different types of authentication and authorization practice
- Authentication and Authorization
- Network Security

DEPLOYMENT AND MONITORING DAY

- Manual Deployment
- Automated Deployment
- Deployment Environments
- Centralized Logging Monitoring and health check

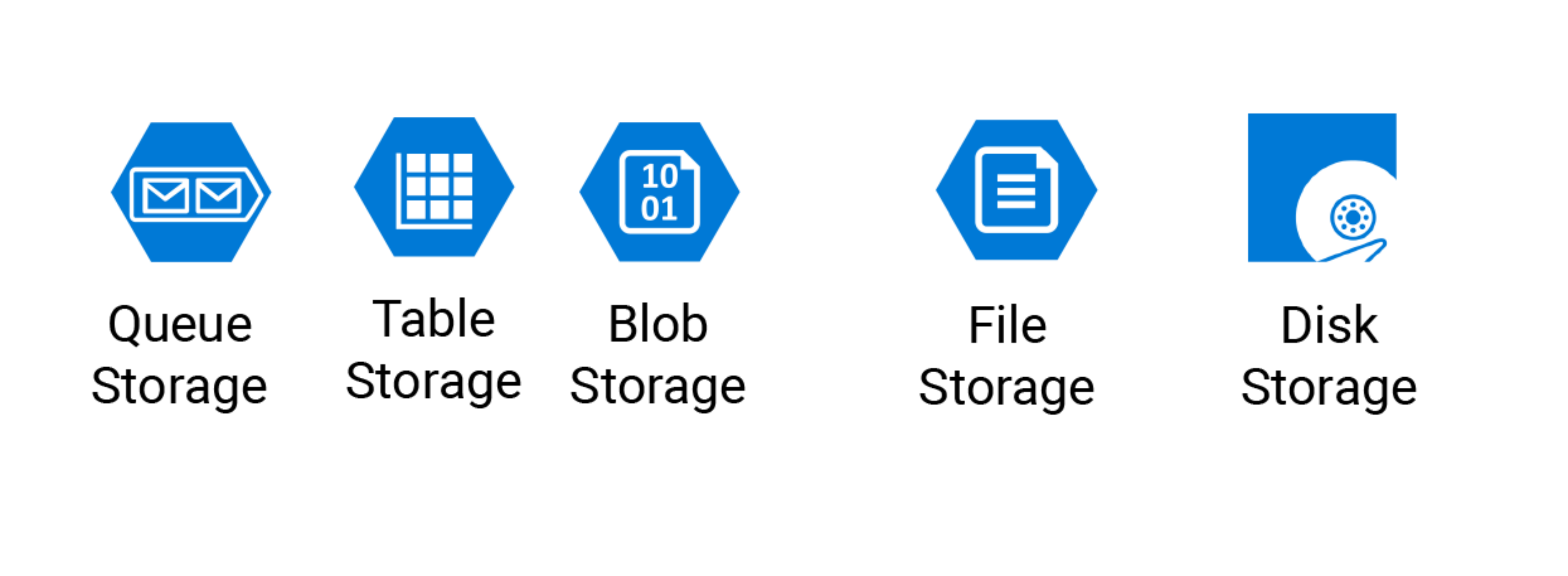


CONTENT DELIVERY NETWORK

- Azure Content Delivery Network (CDN)

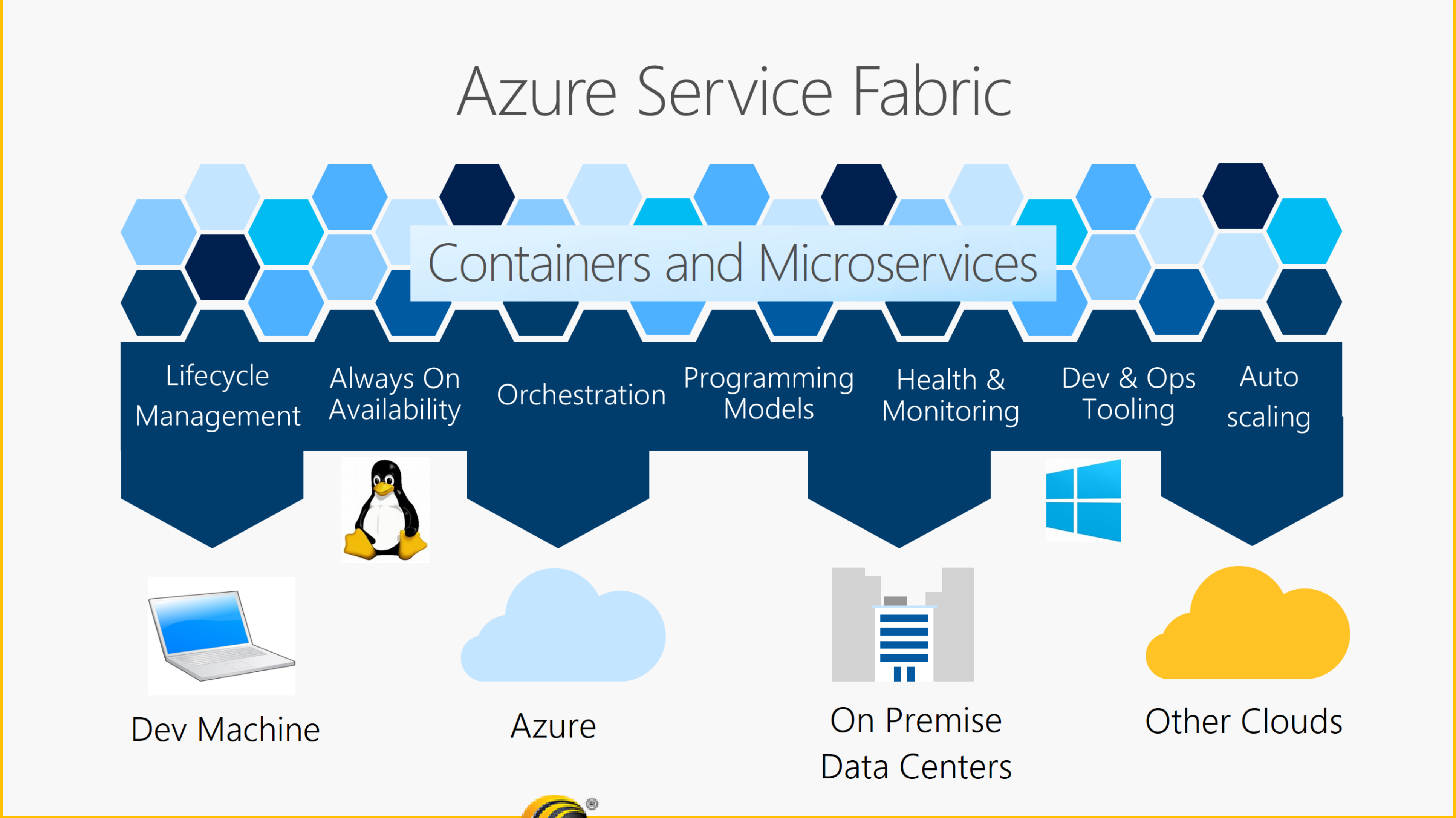
Blob Storage

Images /Document /Audio/Binary data



Gateway Group

AZURE SERVICE FABRIC



Gateway Group

Binkal Patel