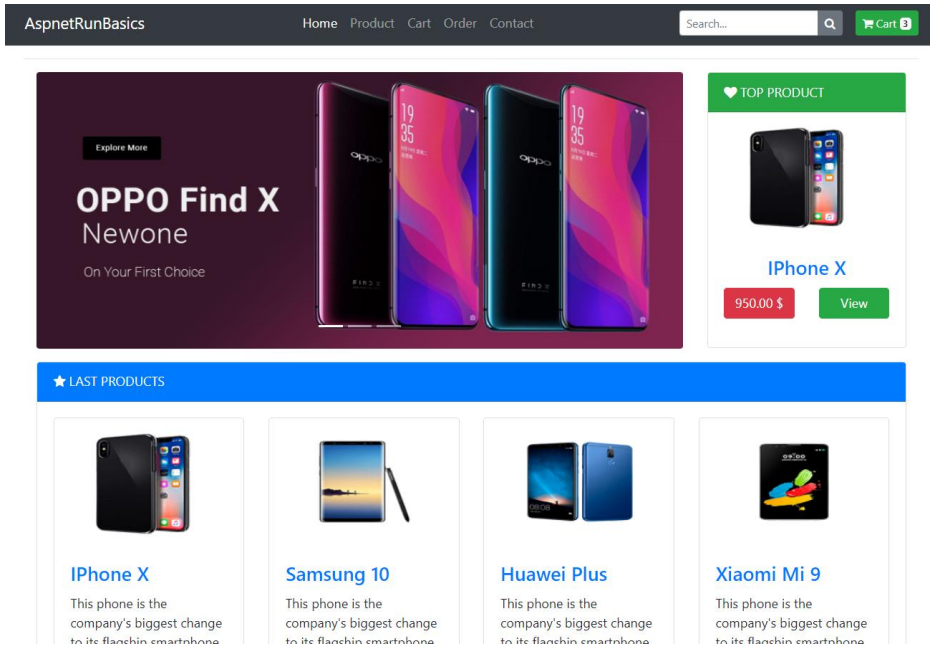
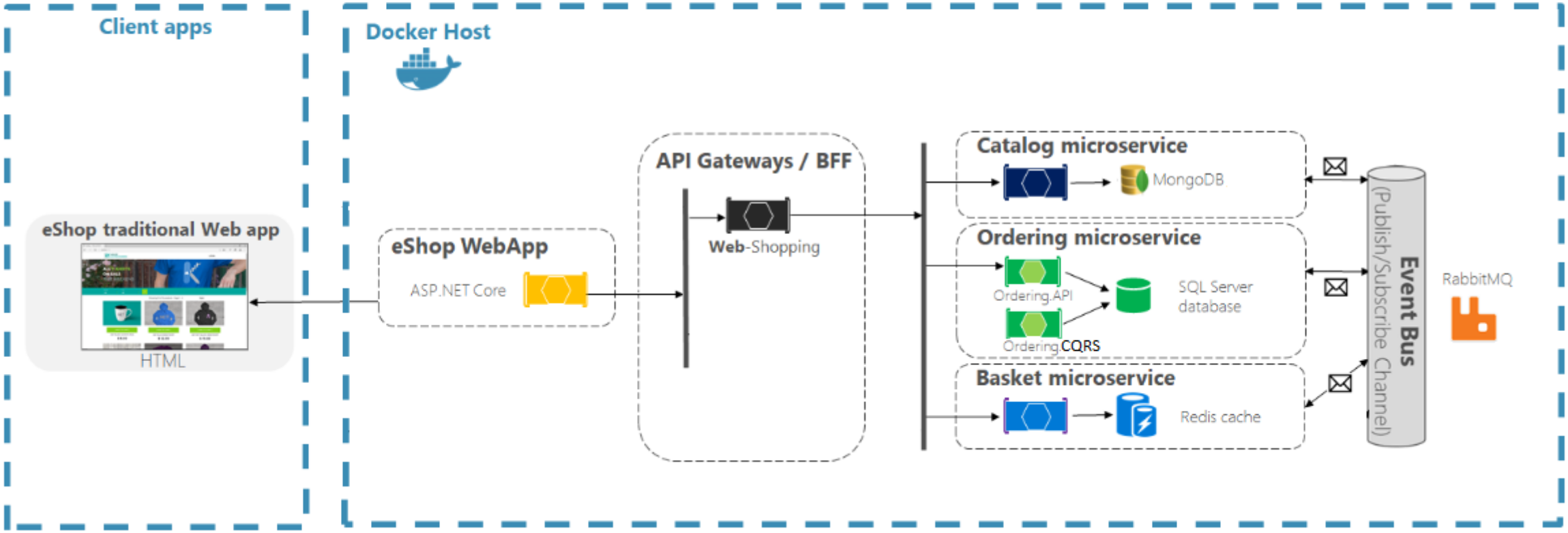
Microservices Architecture and Implementation on .NET



Mehmet OZKAYA

aspnetrun

aspnetrun-microservices Environment Architecture



Prerequest

Basics of C# and Asp.Net Core

Basics of Docker

Source Code

aspnetrun / run-aspnetcore-microservices

Sponsor | Used by 5 | Unwatch 6 | Unstar 92 | Fork 11

Code | Issues 0 | Pull requests 0 | Actions | Projects 0 | Wiki | Security 0 | Insights | Settings

Building Microservices on .Net platforms which used Asp.Net Web API, Docker, RabbitMQ, Ocelot API Gateway, MongoDB, Redis, SqlServer, Entity Framework Core, CQRS and Clean Architecture implementation. Download Microservices Architecture and Step by Step Implementation on .NET Book -> https://aspnetrun.azurewebsites.net/M...

Edit

aspnet-core | docker | ocelot-gateway | micorservices | aspnetcore-microservices | rabbitmq | mongodb | redis | sql-server | cqrs-pattern | clean-architecture | event-sourcing | eventbus | event-driven | microservices-architecture | aspnet-web-api | swagger | rest-api | api-gateway | mediator-pattern

Manage topics

87 commits | 1 branch | 0 packages | 0 releases | 1 contributor | MIT

https://github.com/aspnetrun/run-aspnetcore-microservices

Development Environment

- .NET Core 3.x or above SDK
- Visual Studio 2019 v16.x or above
- Docker Desktop
- Postman

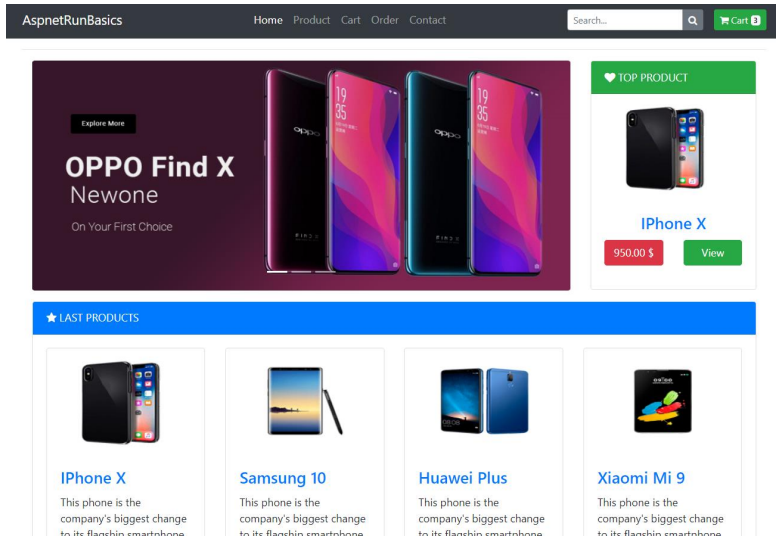
Run Final Application

- Clone the repository
- Run command on top of **docker-compose.yml** files;

docker-compose -f docker-compose.yml

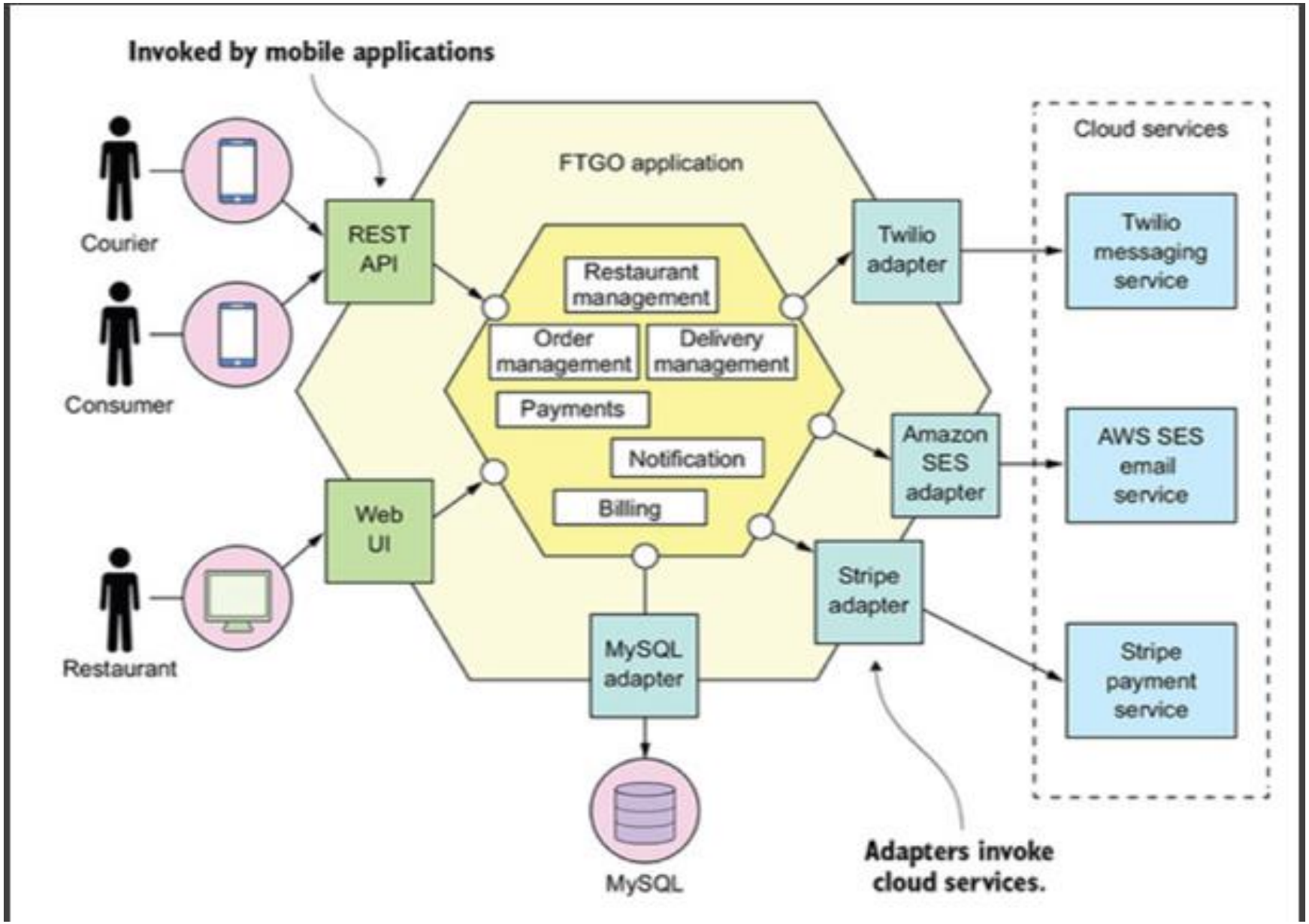
-f docker-compose.override.yml up –d

- RabbitMQ -> http://localhost:15672/
- Catalog API -> http://localhost:8000/swagger/index.html
- Basket API -> http://localhost:8001/swagger/index.html
- Order API -> http://localhost:8002/swagger/index.html
- API Gateway -> http://localhost:7000/Order?username=swn
- Shopping Web UI -> http://localhost:8003/



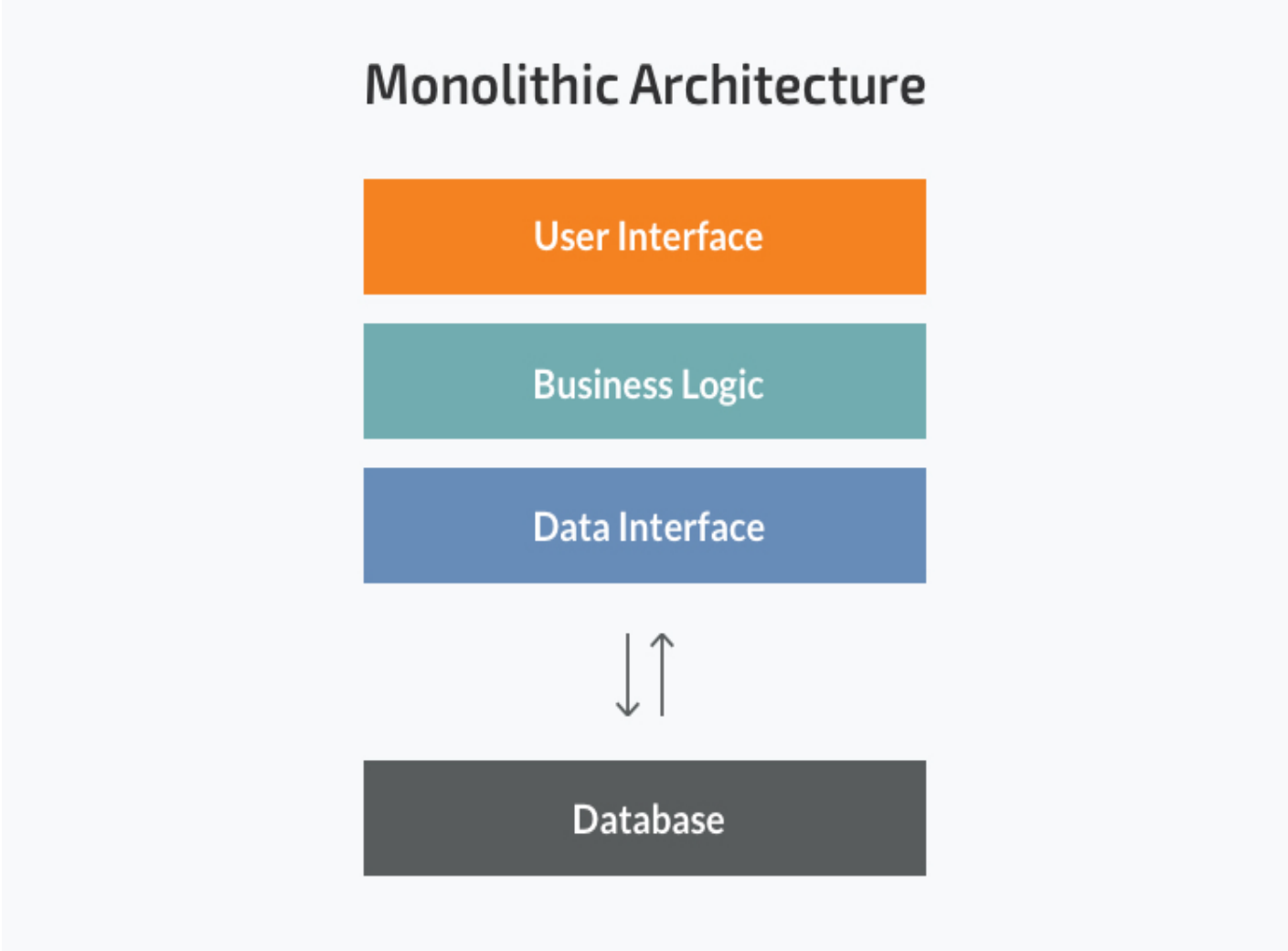
Service Oriented Architecture (SOA)

- Services to communicate in the distribution system
- Same Data Layer
- Consumes Common Applications from integration or Omni-Channel


Invoked by mobile applications
Courier
Consumer
Restaurant
REST API
Web UI
FTGO application
Restaurant management
Order management
Delivery management
Payments
Notification
Billing
MySQL adapter
MySQL
Twilio adapter
Amazon SES adapter
Stripe adapter
Cloud services
Twilio messaging service
AWS SES email service
Stripe payment service
Adapters invoke cloud services.

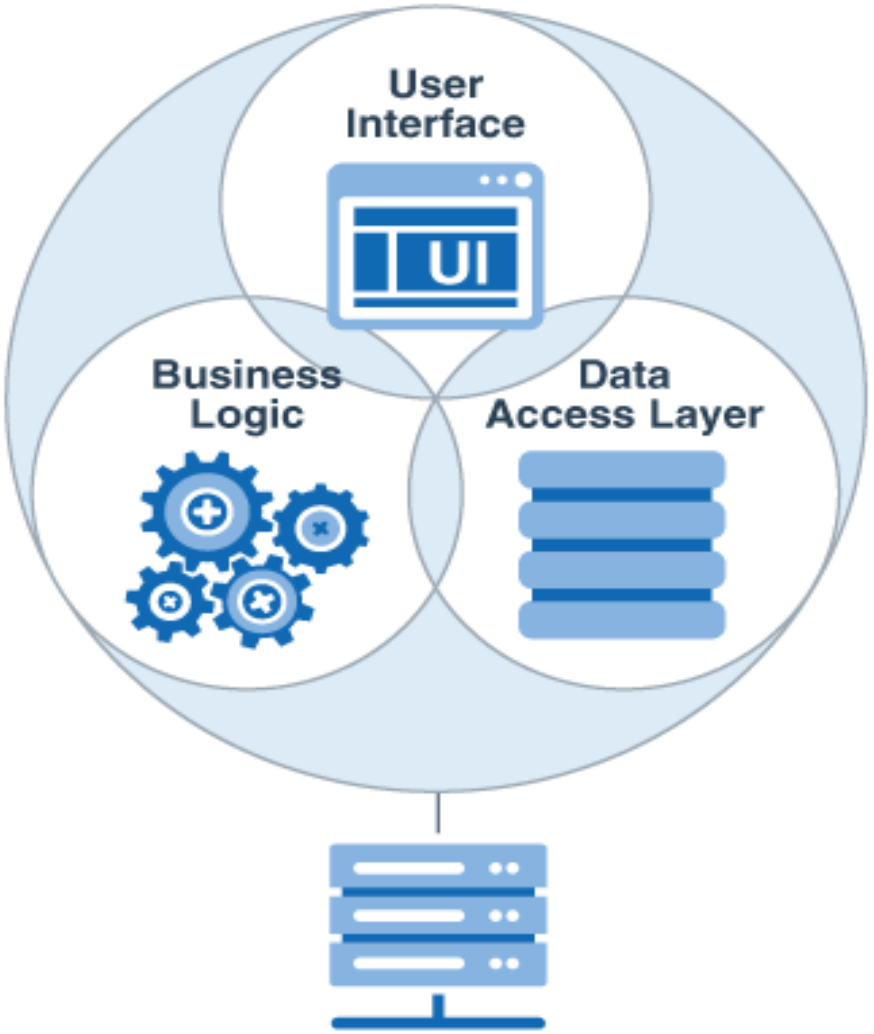
Monolithic Architecture

- All Layers in Single Project
- Self-contained applications
- Interdependent rather than loosely coupled



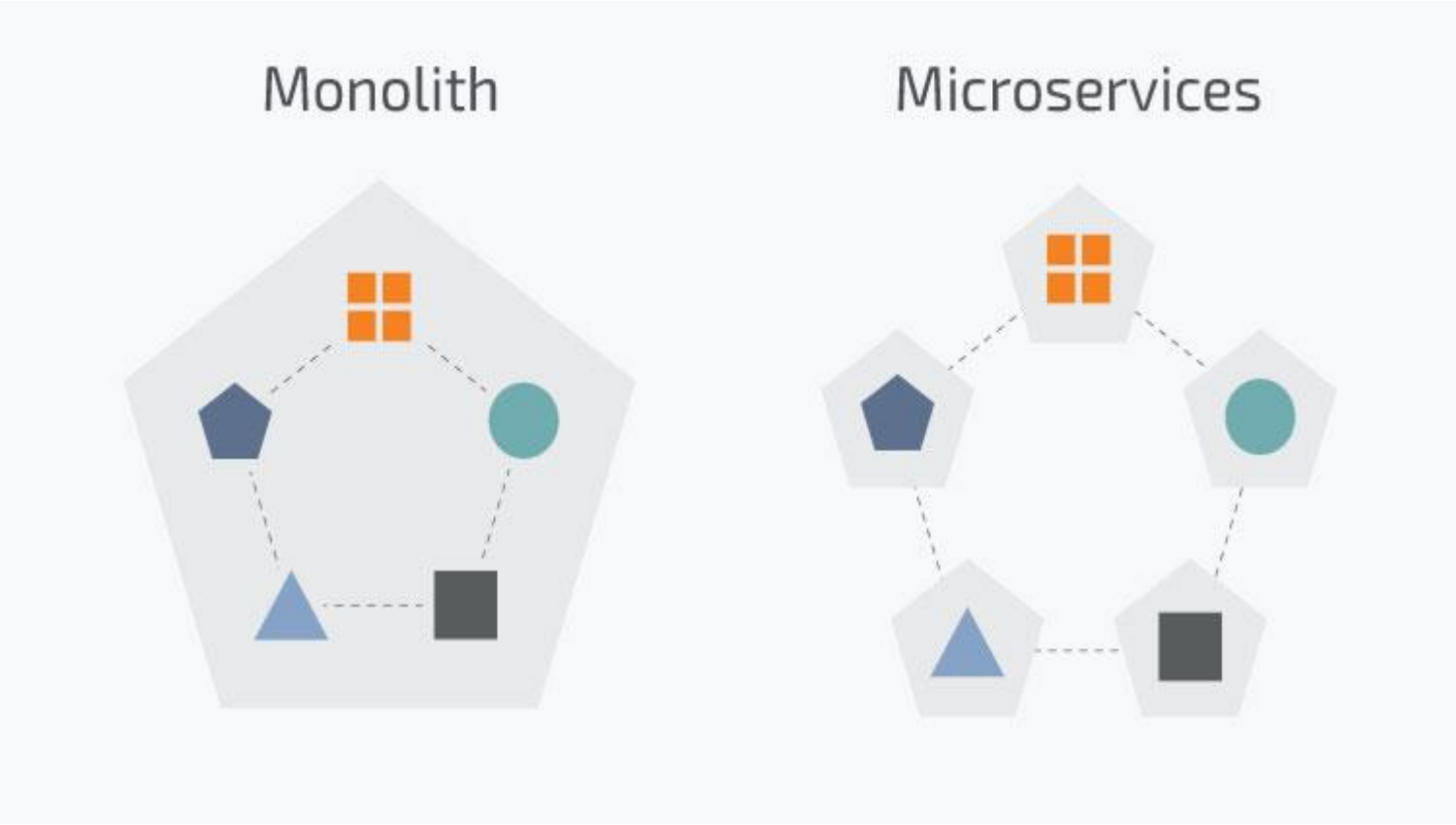
Weaknesses of Monolithic Architecture

- Long Training Requirements
- Crushed Under Traffic Needs
- Deploying the Entire App
- Failure to adapt to innovations



Strengths of Monolithic Architecture

- Easy to Debug
- Easy Deployment
- Find bug fast
- Single peace

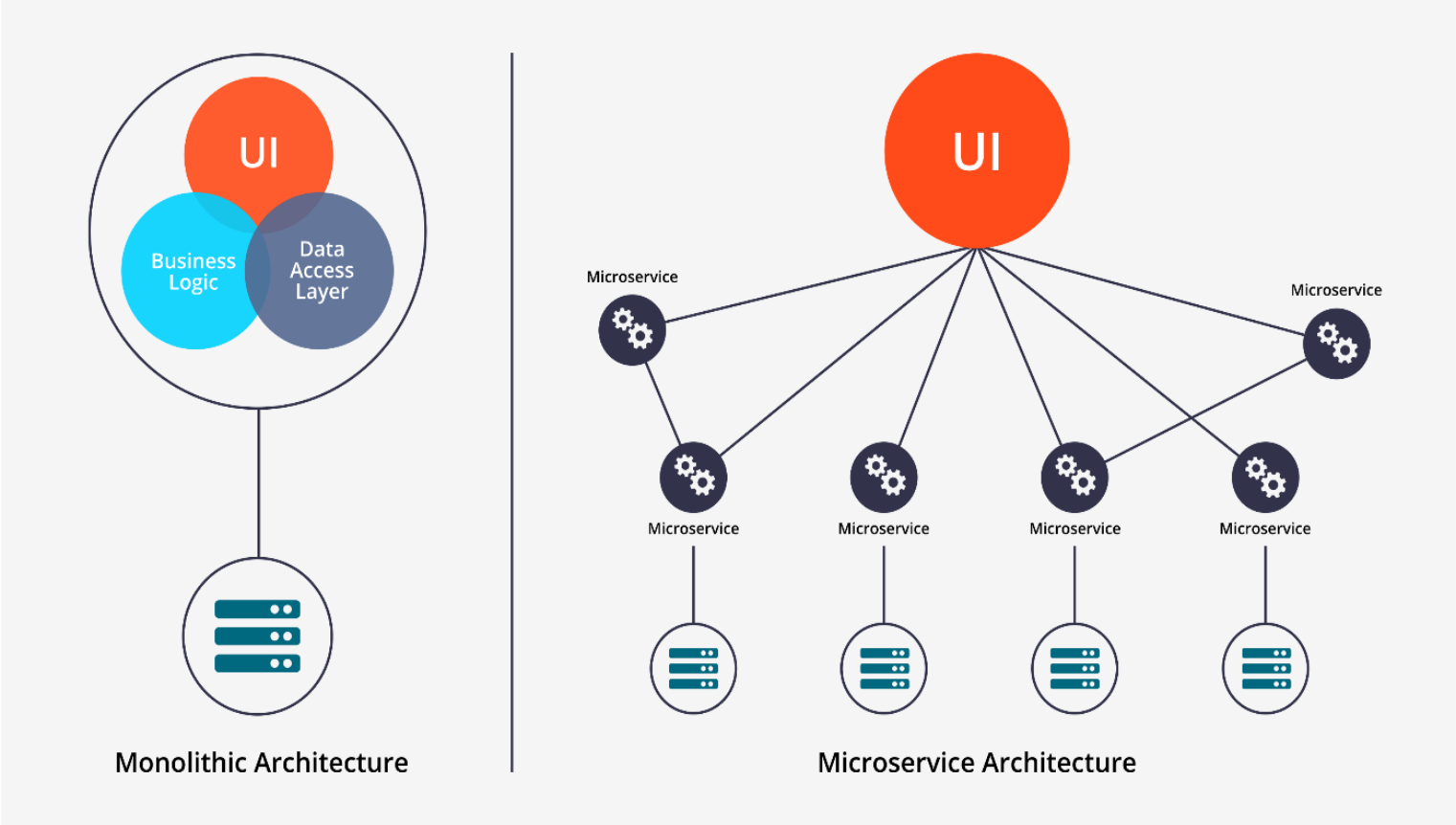


Monolith Architecture

- Single deployable unit
- Good choice for simple applications
- Even a small change requires rebuilt and redeployment
- Scaling requires scaling of the entire application

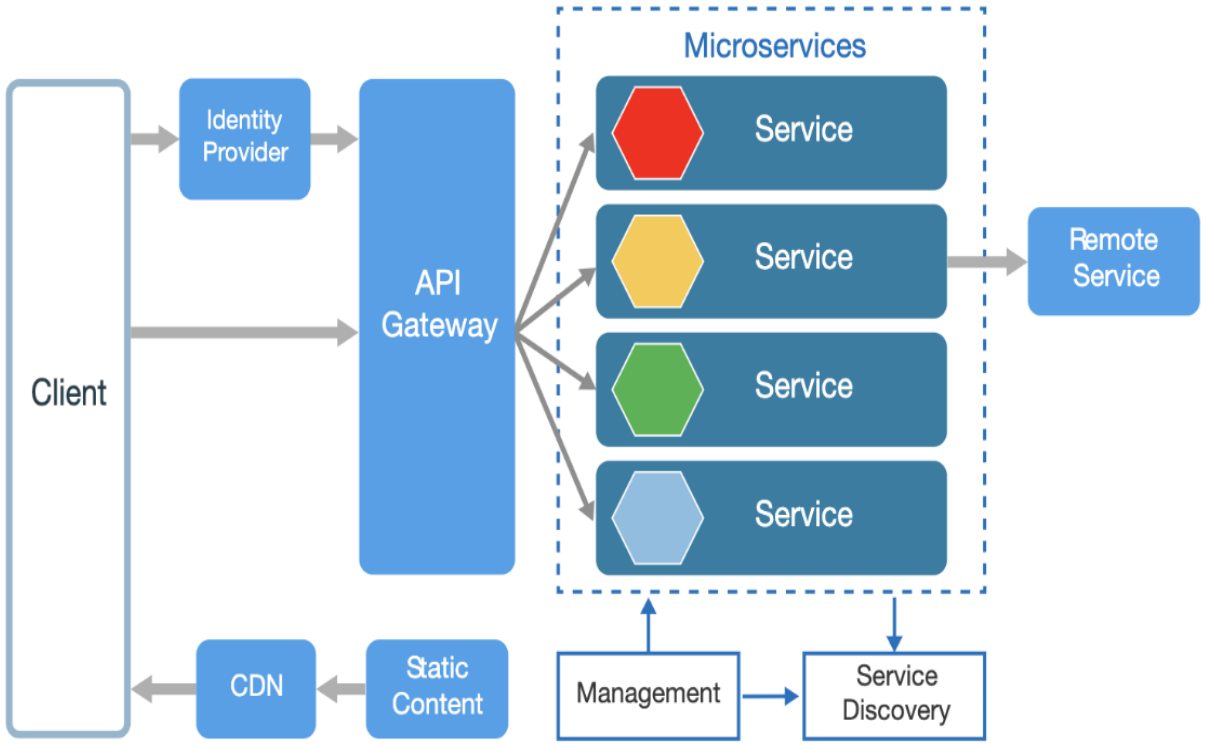
Microservices Architecture

- Divided into smaller parts
- Every peace do one single job
- Developed independently
- Continuous Delivery
- Decentralized Governance



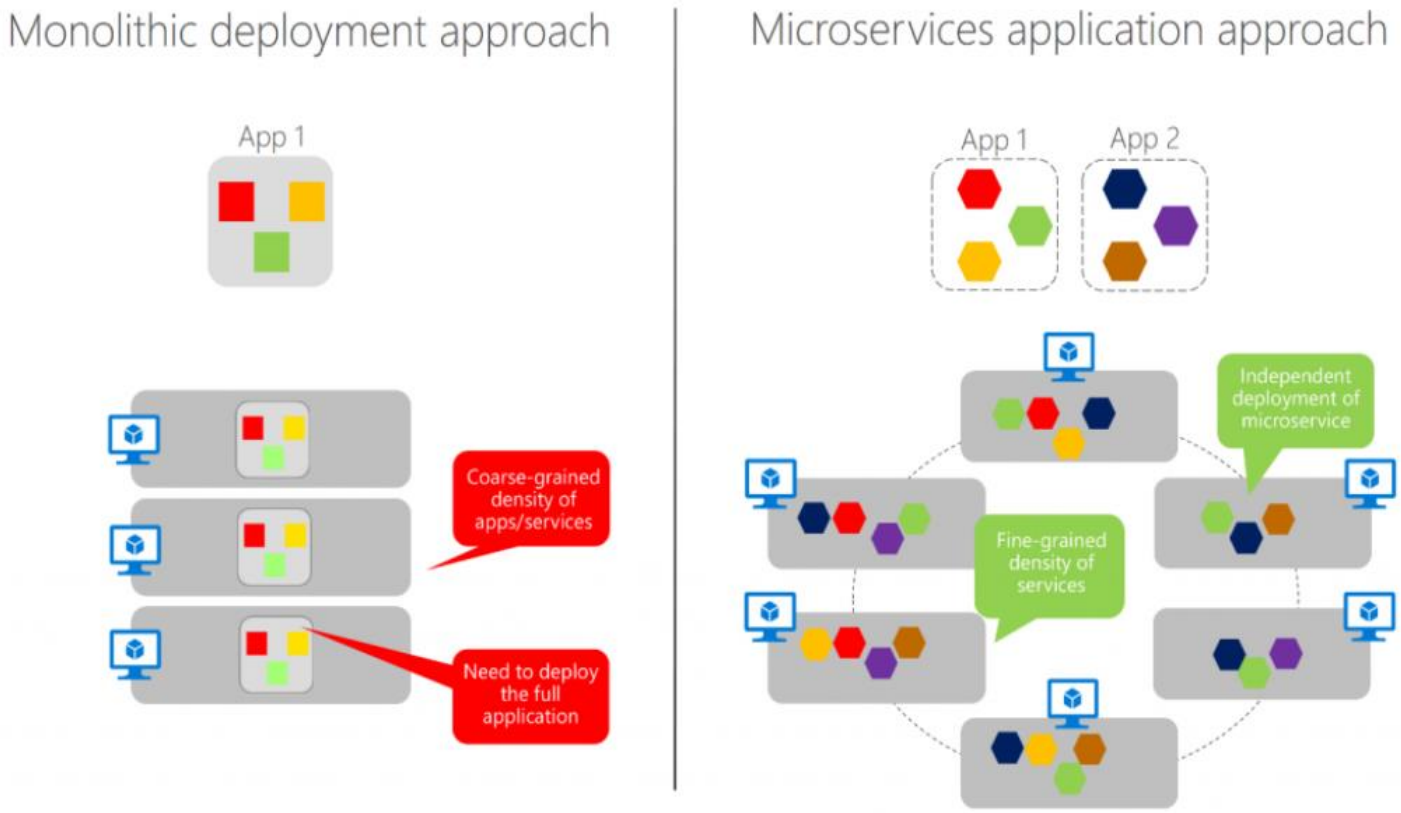
Weaknesses of Microservices Architecture

- Complex System
- Hard Communication
- Separate Deployments
- Finding Root Causes



Strengths of Microservices Architecture

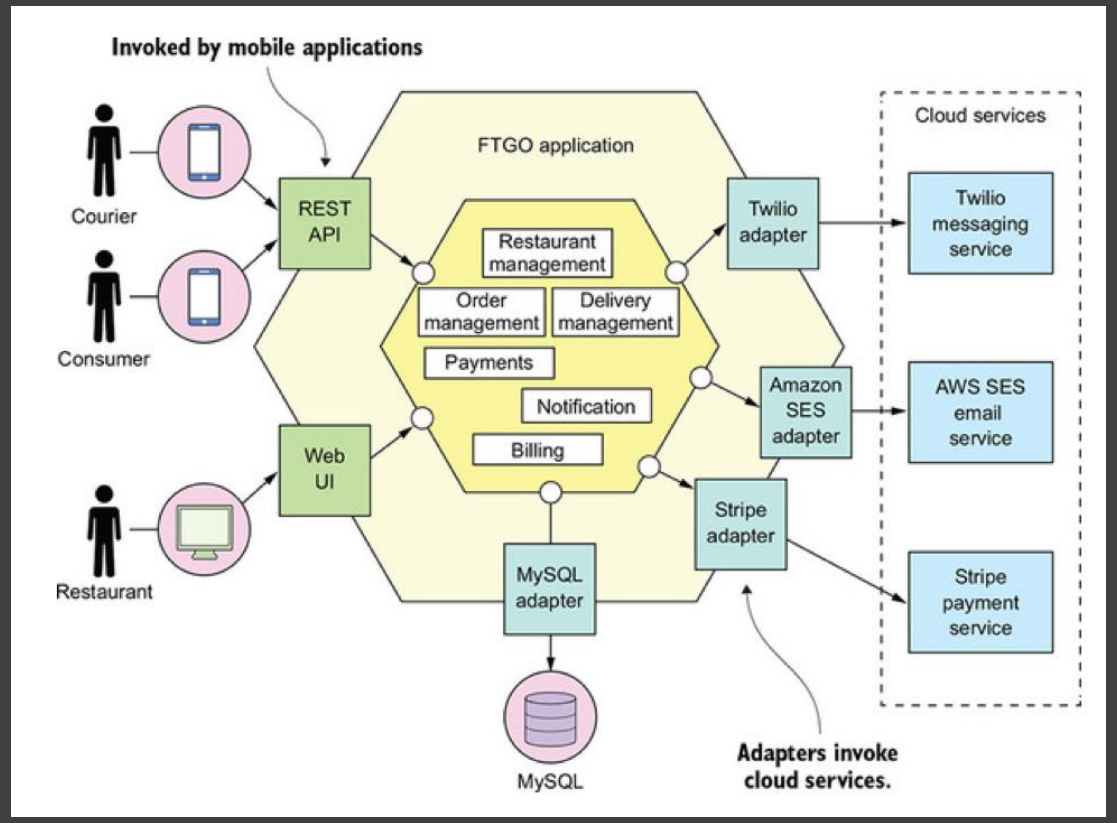
- Divided Small to understand
- Scaled Independently
- Select the Technologies
- Speed the Deployment

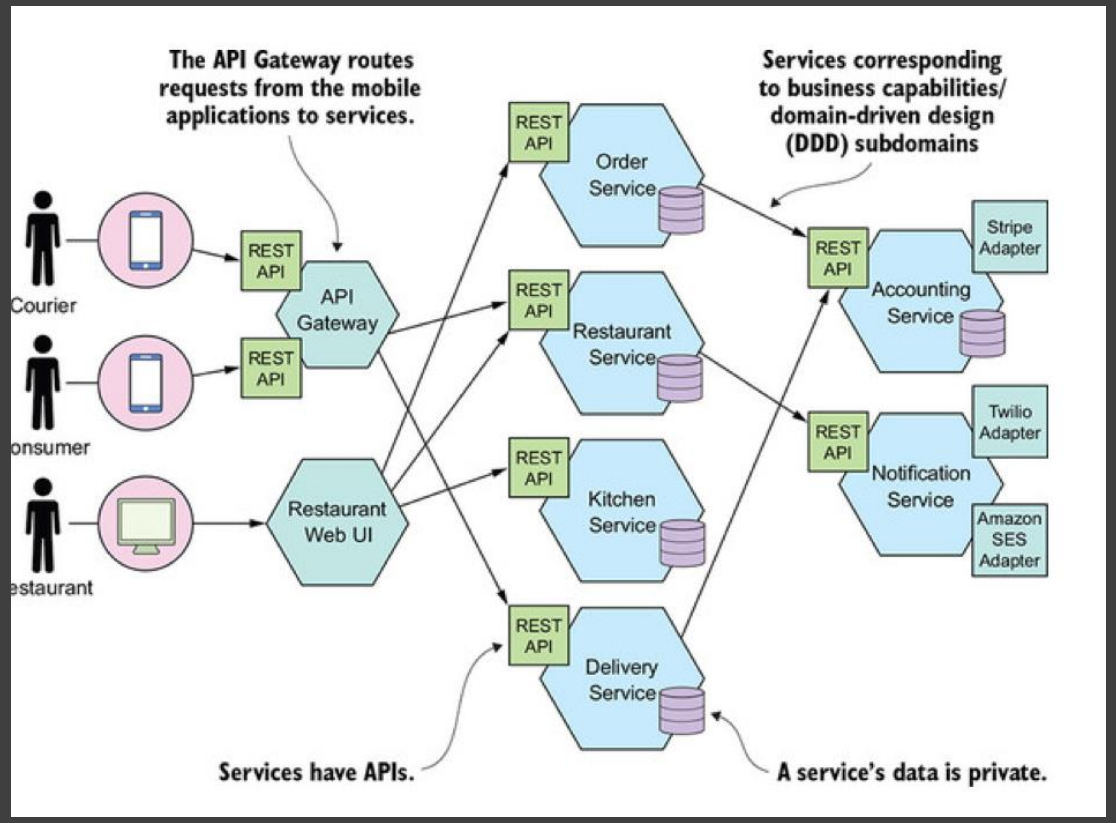


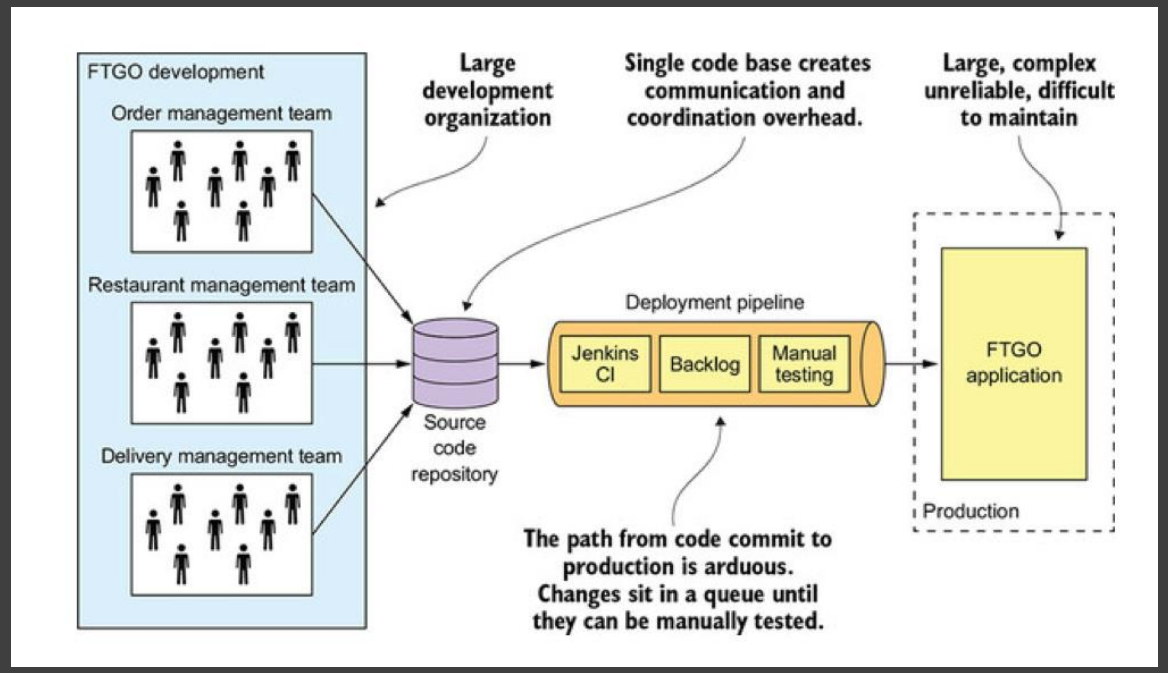
Microservice Architecture

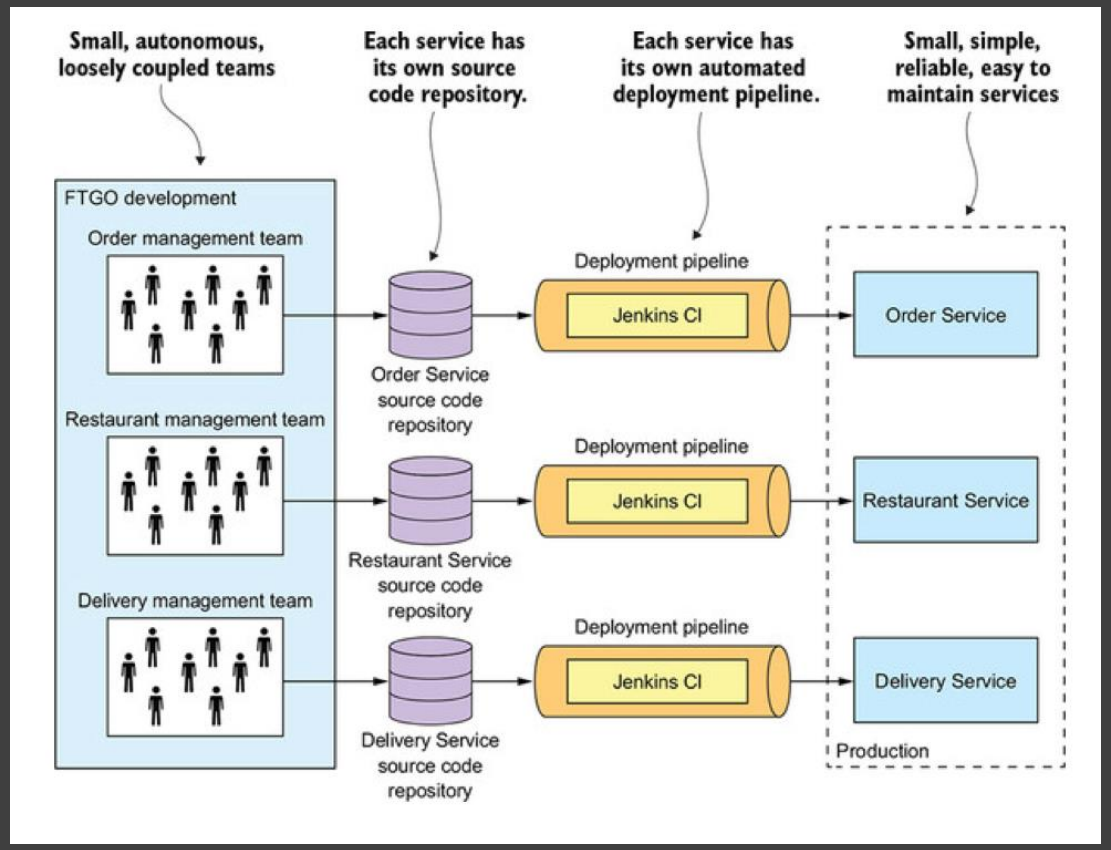
- decomposes a system into a set of independently deployable and scalable services
- Each microservice has its own database
- better choice for large, complex applications.
- accelerates the velocity of software development by enabling small, autonomous teams to work in parallel.
- requires DevOps and small, autonomous teams.

Architecture Comparison





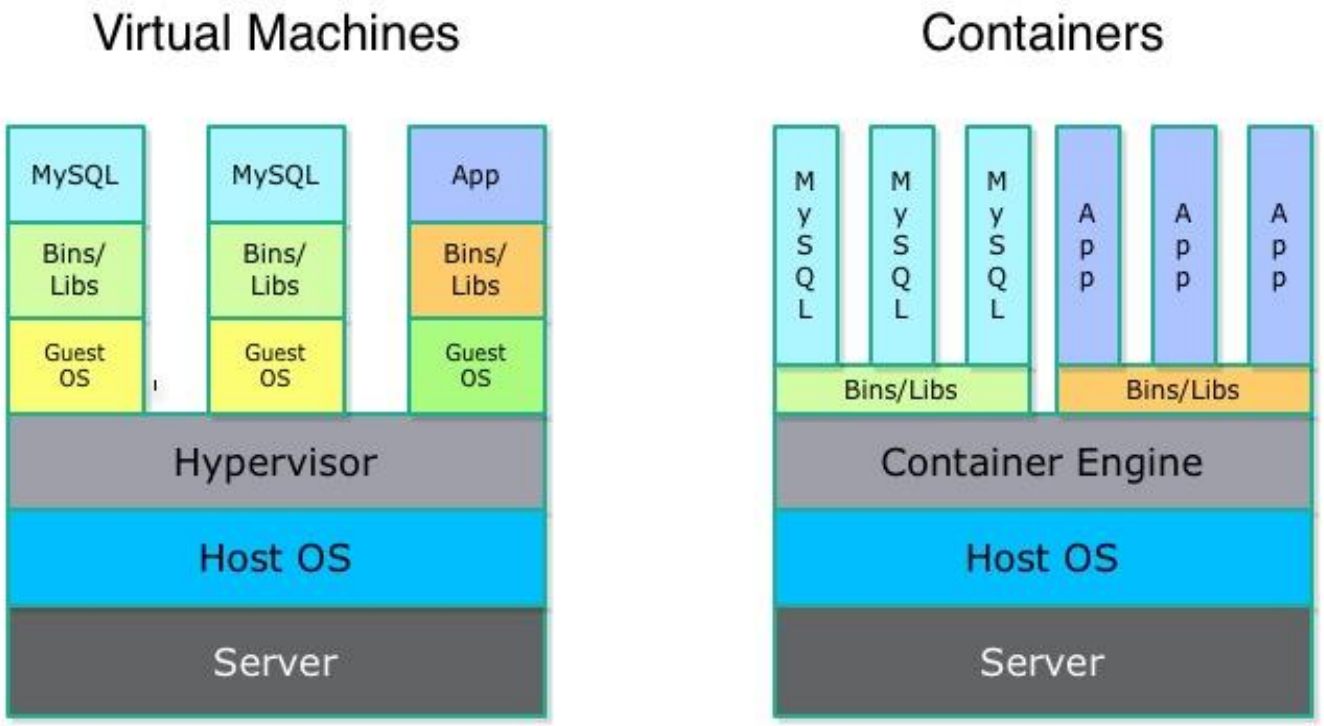




Deployment Pipeline Comparison

DOCKER

- Container Technology
- Share OS
- Shipping and Running any system
- Loosely Isolated Environments



https://docs.docker.com/get-started/overview/

DOCKER Commands

docker –version

docker pull <image name>

docker run -it -d <image name>
This command is used to create a container from an image

docker images
This command lists all the locally stored docker images

docker rm <container id>
This command is used to delete a stopped container

docker rmi <image-id>
This command is used to delete an image from local storage

docker ps
This command is used to list the running containers

docker ps -a
This command is used to show all the running and exited containers

docker exec -it <container id> bash
This command is used to access the running container

docker start <container id>
docker stop <container id>
docker restart <container id>

docker info

docker logs <container id>

DOCKER Commands (Continue)

docker volume create
docker volume ls

docker build <path to docker file>
This command is used to build an image from a specified docker file

docker compose up
This command run multiple container

Bonus Example docker hub pull:
docker run -d --hostname swn-rabbit --name swn-rabbit -p 5672:5672 -p 15672:15672 rabbitmq:3-management

Single Container
For aspnetcore app after adding docker file -- for single container add docker file build and run = create new container
$ docker build -t aspnetapp .
$ docker run -d -p 8080:80 --name myapp aspnetapp

Multi Container - docker-compose.yml
docker-compose up
$ docker-compose -f docker-compose.yaml -f docker-compose-infrastructure.yaml up --build

docker-compose -f docker-compose.yml -f docker-compose.override.yml up –build

None Image Delete
docker image prune

BUILDING CATALOG MICROSERVİCES

- ASP.NET Core **Web API** application
- **REST** API principles, CRUD operations
- **Mongo DB** NoSQL database connection on docker
- **N-Layer** implementation
- **Repository** Design Pattern
- **Swagger** Open API implementation
- **Dockerfile** implementation

Catalog Microservices

- MongoDB
- Docker Container
- Swagger

Catalog API v1 OAS3

/swagger/v1/swagger.json

Catalog

GET	/api/v1/Catalog
POST	/api/v1/Catalog
PUT	/api/v1/Catalog
GET	/api/v1/Catalog/{id}
DELETE	/api/v1/Catalog/{id}
GET	/api/v1/Catalog/GetProductByCategory/{category}

Asp.Net Core Web API

- Program.cs
- Built-in Dependency Injection (DI)
- Application Startup
- Middleware
- Configuration

ASP.NET Core

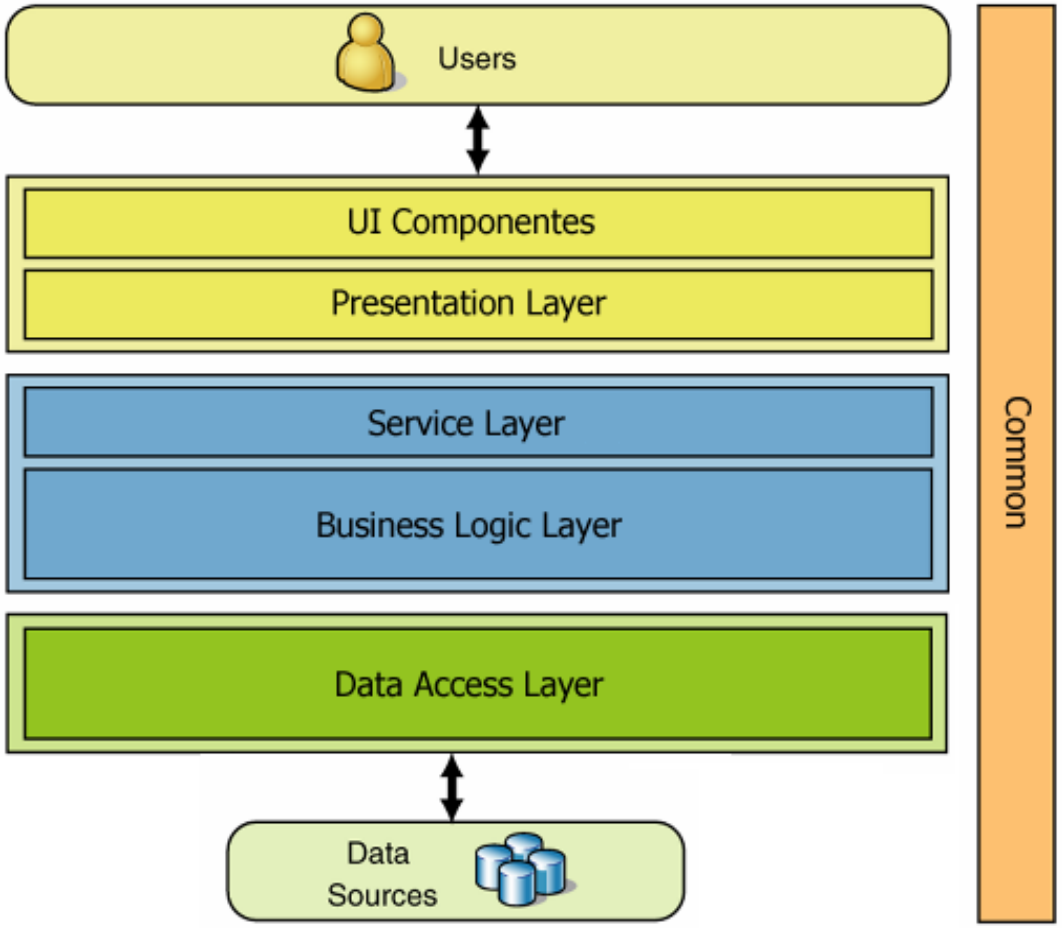
Analysis & Design of Catalog Microservices

- List Product and Categories
- Get Product by Id
- Get Products by Category
- Create new Product
- Update Product
- Delete Product

Method	Request URI	Use Case
GET	api/v1/Catalog	Listing Products and Categories
GET	api/v1/Catalog/{id}	Get Product with product Id
GET	api/v1/Catalog/ GetProductByCategory /{category}	Get Products with category
POST	api/v1/Catalog	Create new Product
PUT	api/v1/Catalog	Update Product
DELETE	api/v1/Catalog/{id}	Delete Product

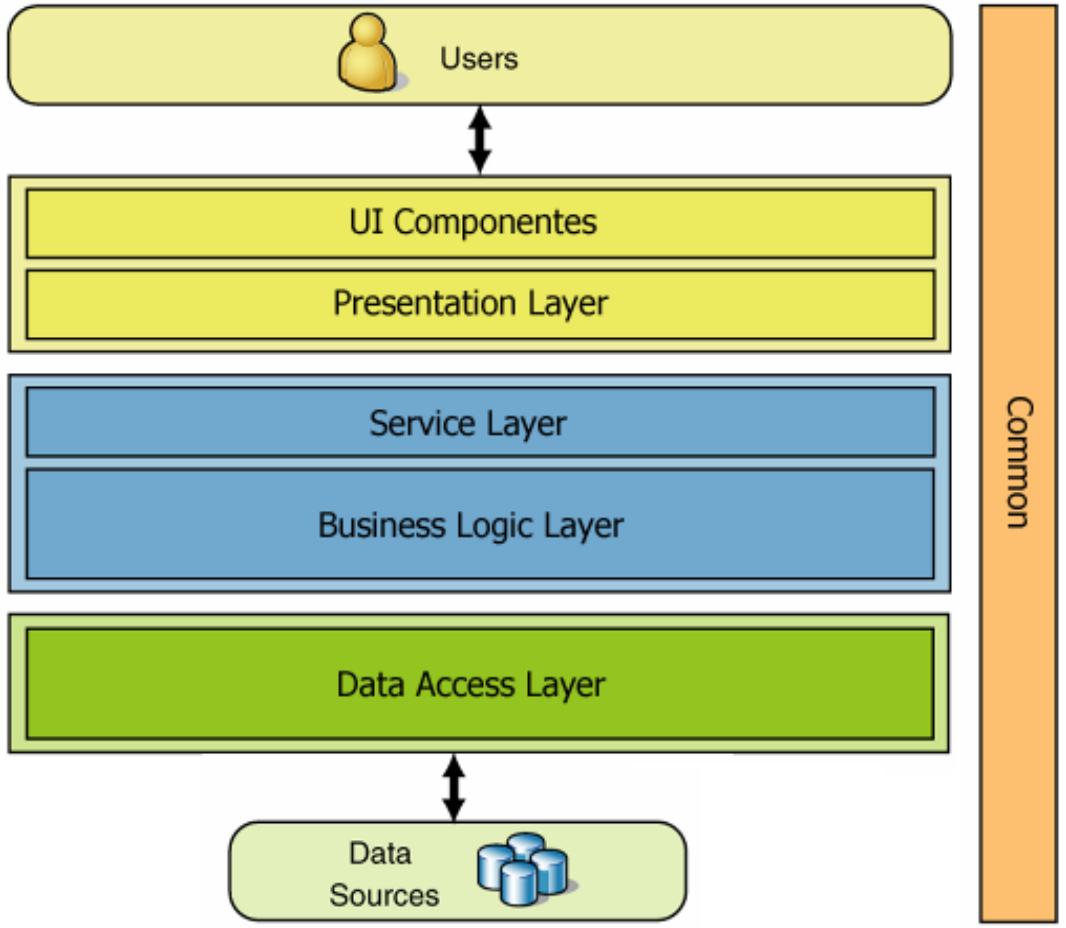
Architecture of Catalog Microservices

- Data Source
- Data Access Layer
- Business Logic Layer
- Presentation Layer
- Common Layer



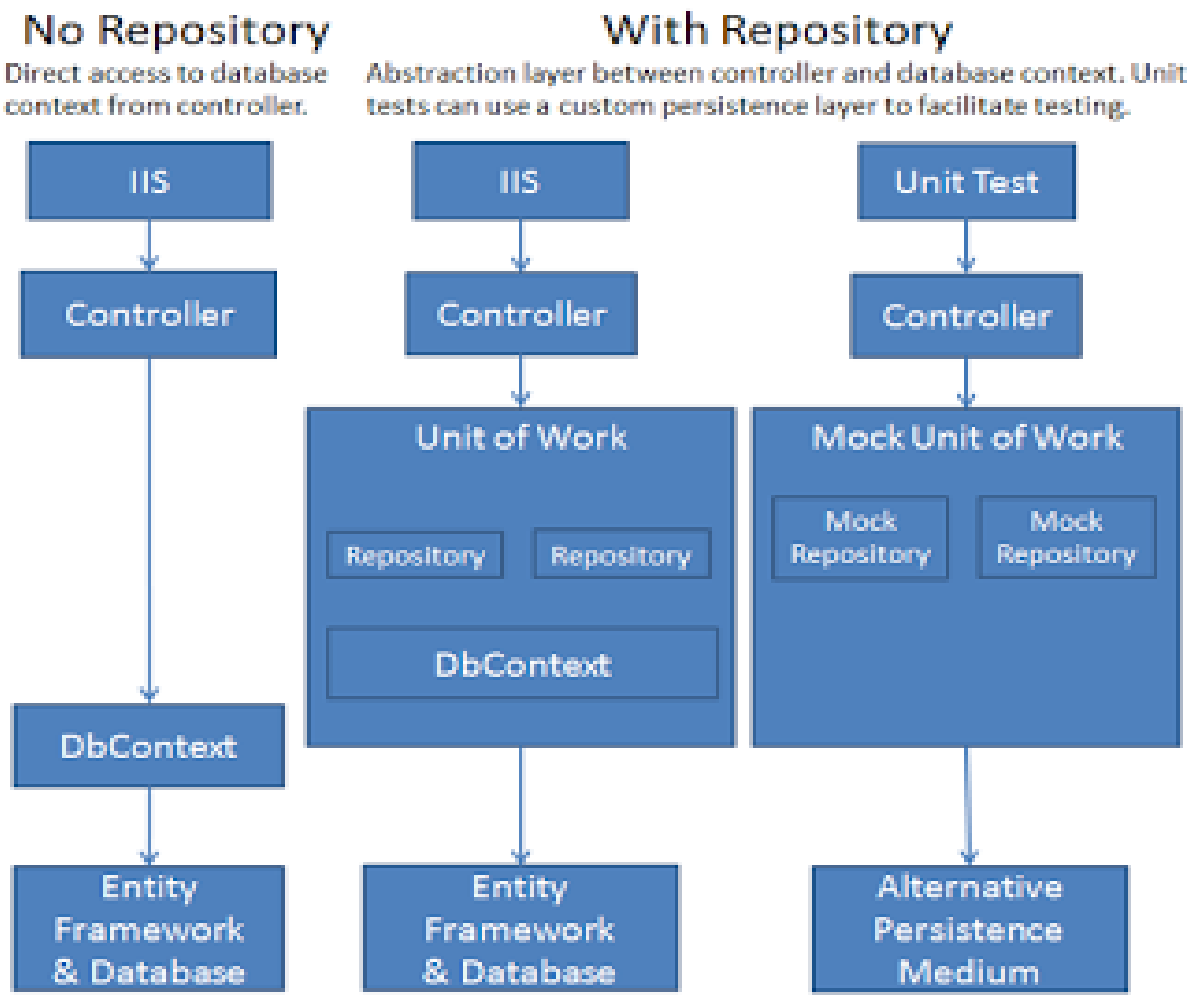
Code Structure of Catalog Microservices

- Entities Folder
- Data Folder
- Repositories Folder
- Controller Folder
- Settings Folder



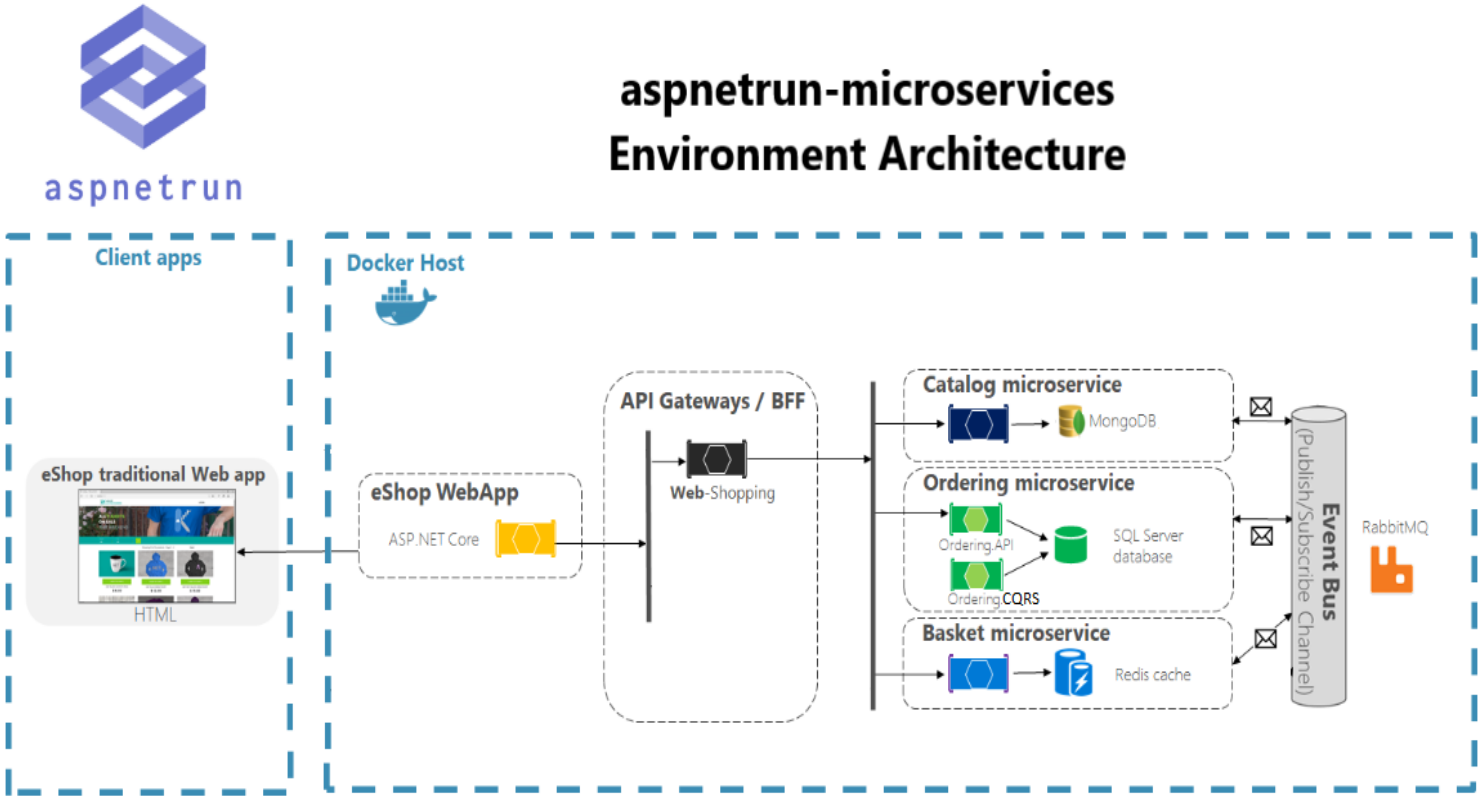
Repository Pattern

- Encapsulate DB Operations
- Prevent Database Works
- Repositories Folder
- Bridge for Data Layer and Business Layer



Docker Operations for Catalog Microservices

- Dockerfile creation
- Dockerfile commands
- Docker-compose file creation
- Docker-compose file commands

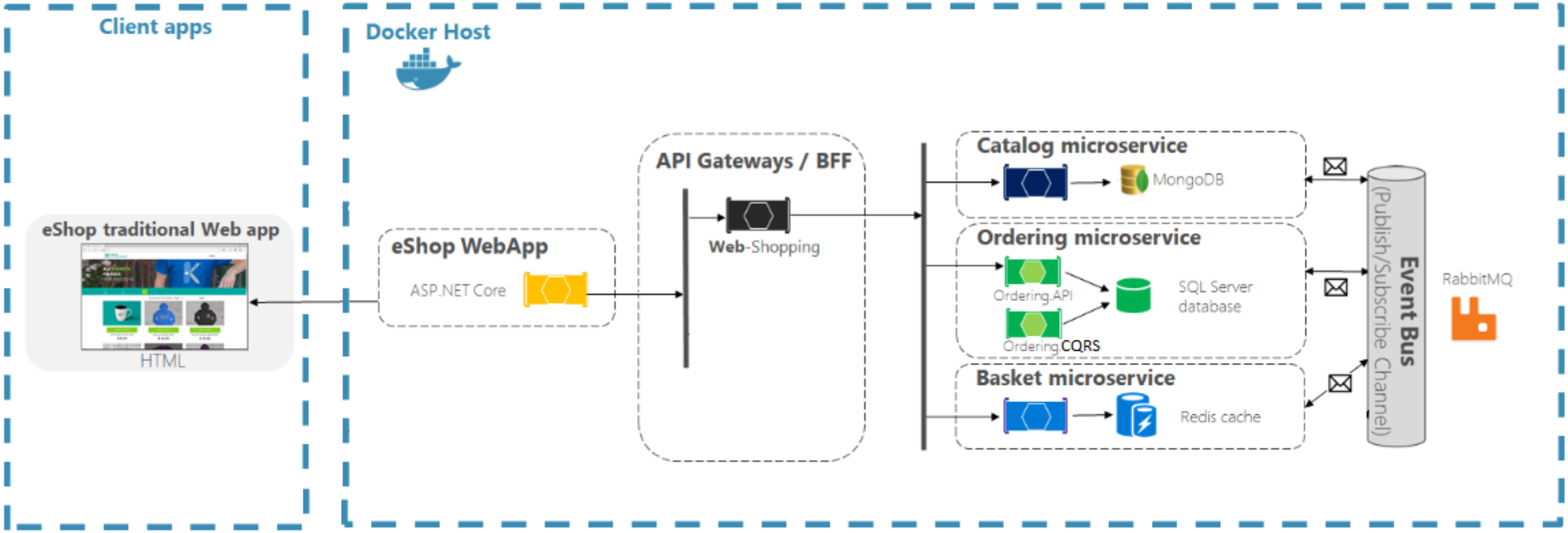


BUILDING BASKET MICROSERVİCES

- ASP.NET Core **Web API** application
- **REST** API principles, CRUD operations
- **Redis DB** NoSQL database connection on docker
- **N-Layer** implementation
- **Repository** Design Pattern
- **Swagger** Open API implementation
- **Dockerfile** implementation

aspnetrun

aspnetrun-microservices Environment Architecture



Basket Microservices

- Redis
- Docker Container
- Swagger

Basket API v1 OAS3

/swagger/v1/swagger.json

Basket

GET	/api/v1/Basket
POST	/api/v1/Basket
DELETE	/api/v1/Basket/{userName}
POST	/api/v1/Basket/Checkout

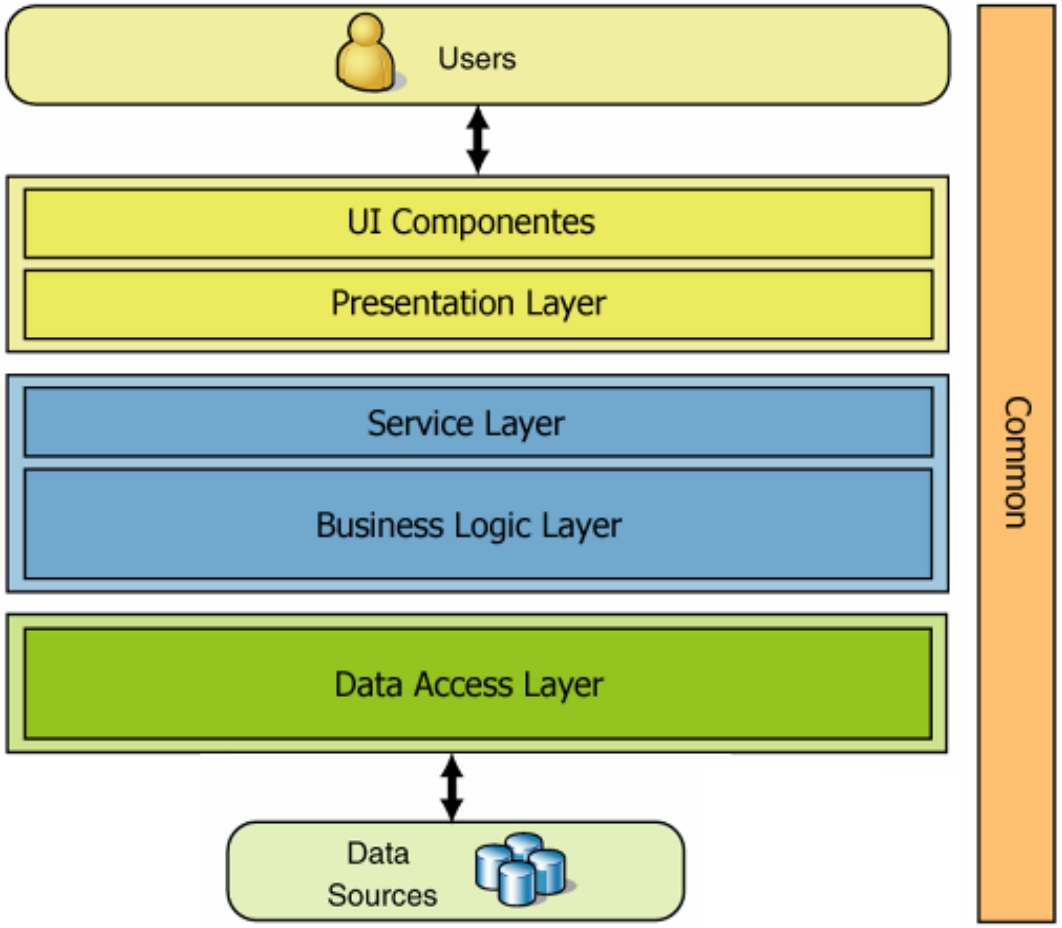
Analysis & Design of Basket Microservices

- Get Basket and Items with username
- Update Basket and Items (add – remove item on basket)
- Delete Basket
- Checkout Basket

Method	Request URI	Use Case
GET	api/v1/Basket	Get Basket and Items with username
POST	api/v1/Basket	Update Basket and Items (add – remove item on basket)
DELETE	api/v1/Basket/{id}	Delete Basket
POST	api/v1/Basket/Checkout	Checkout Basket

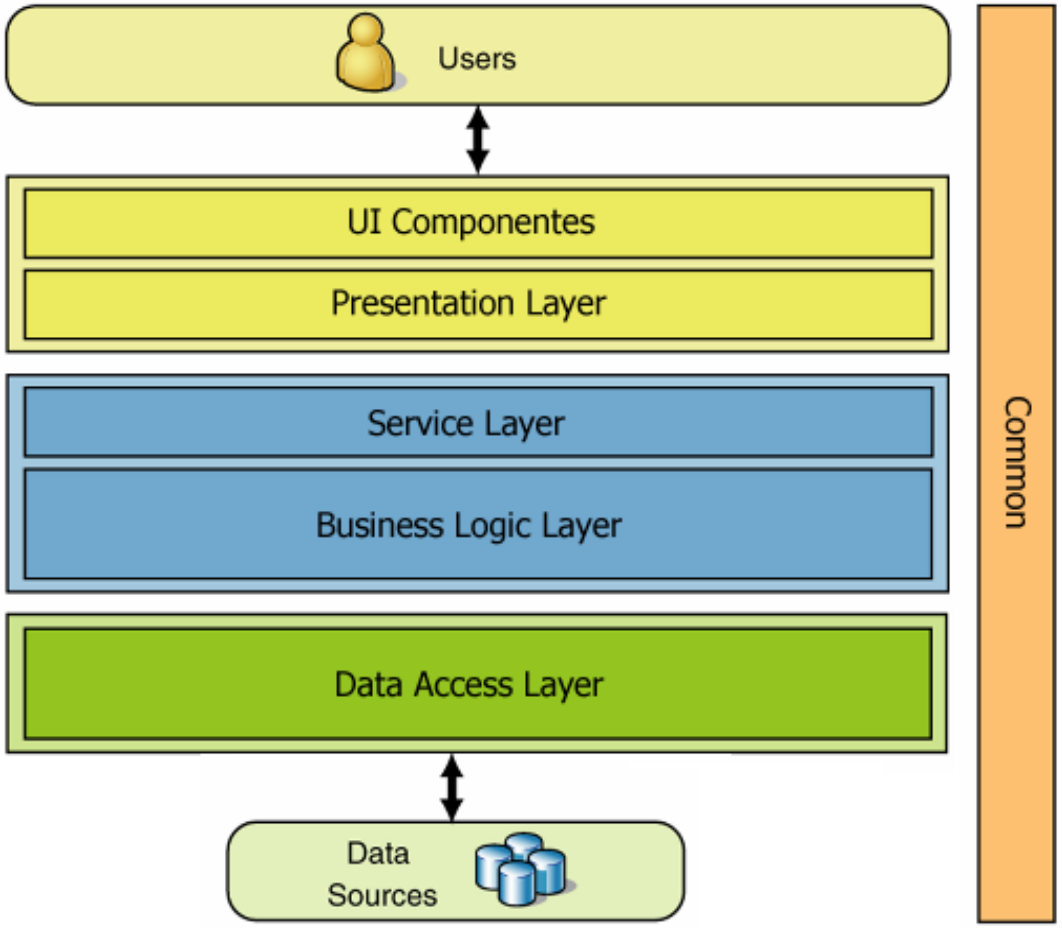
Architecture of Basket Microservices

- Data Source
- Data Access Layer
- Business Logic Layer
- Presentation Layer
- Common Layer



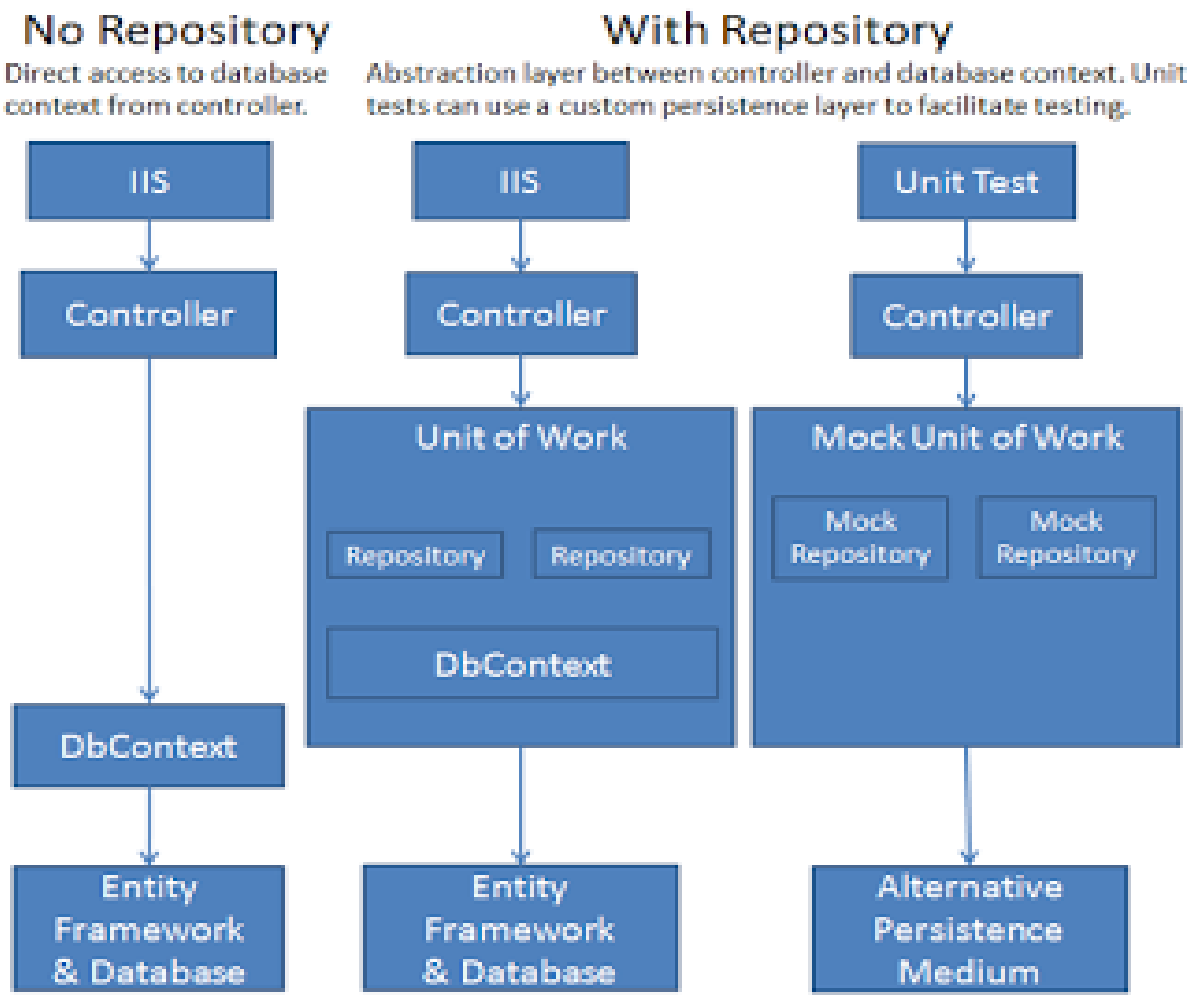
Code Structure of Basket Microservices

- Entities Folder
- Data Folder
- Repositories Folder
- Controller Folder



Repository Pattern

- Encapsulate DB Operations
- Prevent Database Works
- Repositories Folder
- Bridge for Data Layer and Business Layer

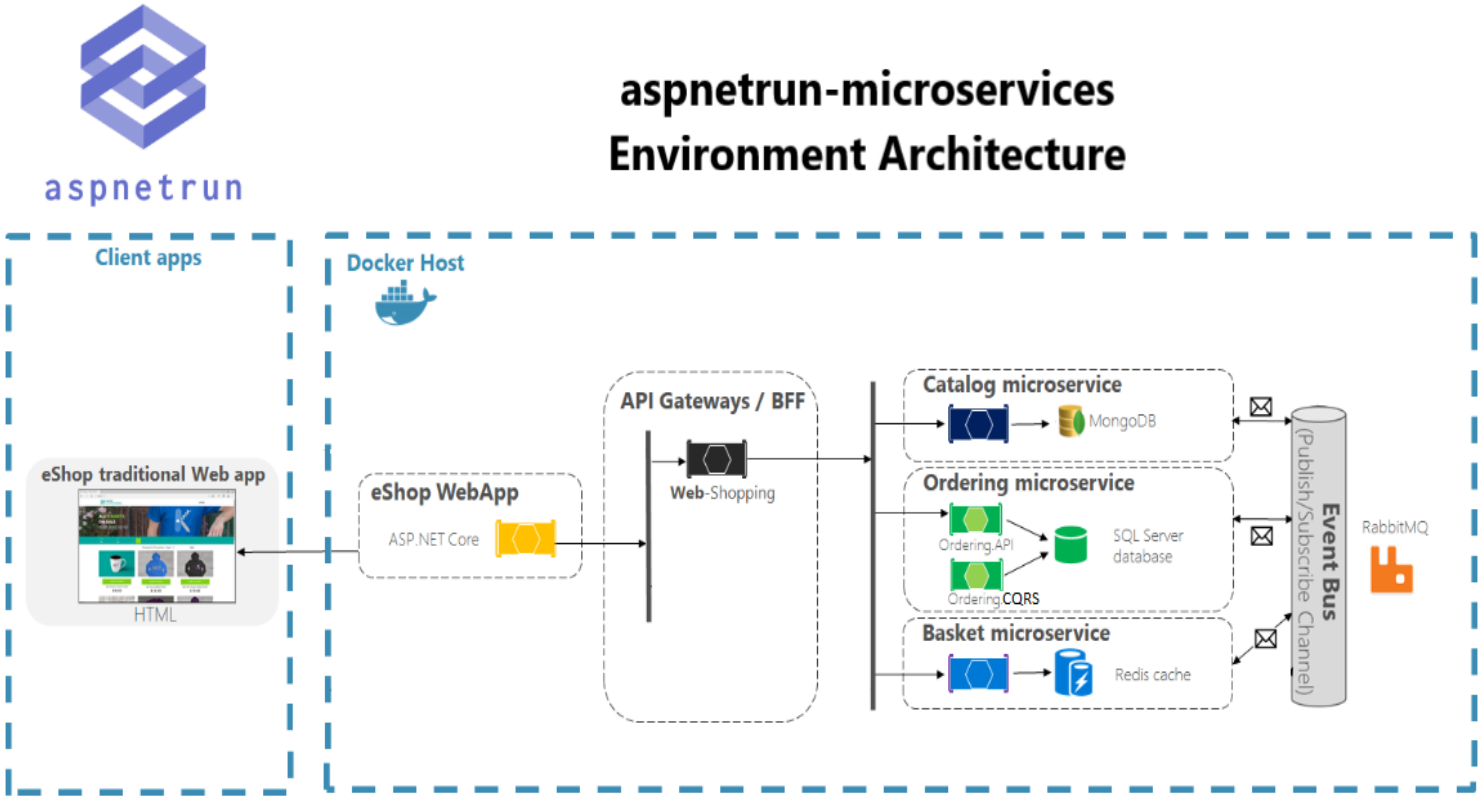


Redis in Use

- Remote Dictionary Server
- Key-Value Dictionary
- High Level Data Structure
- In-memory Database

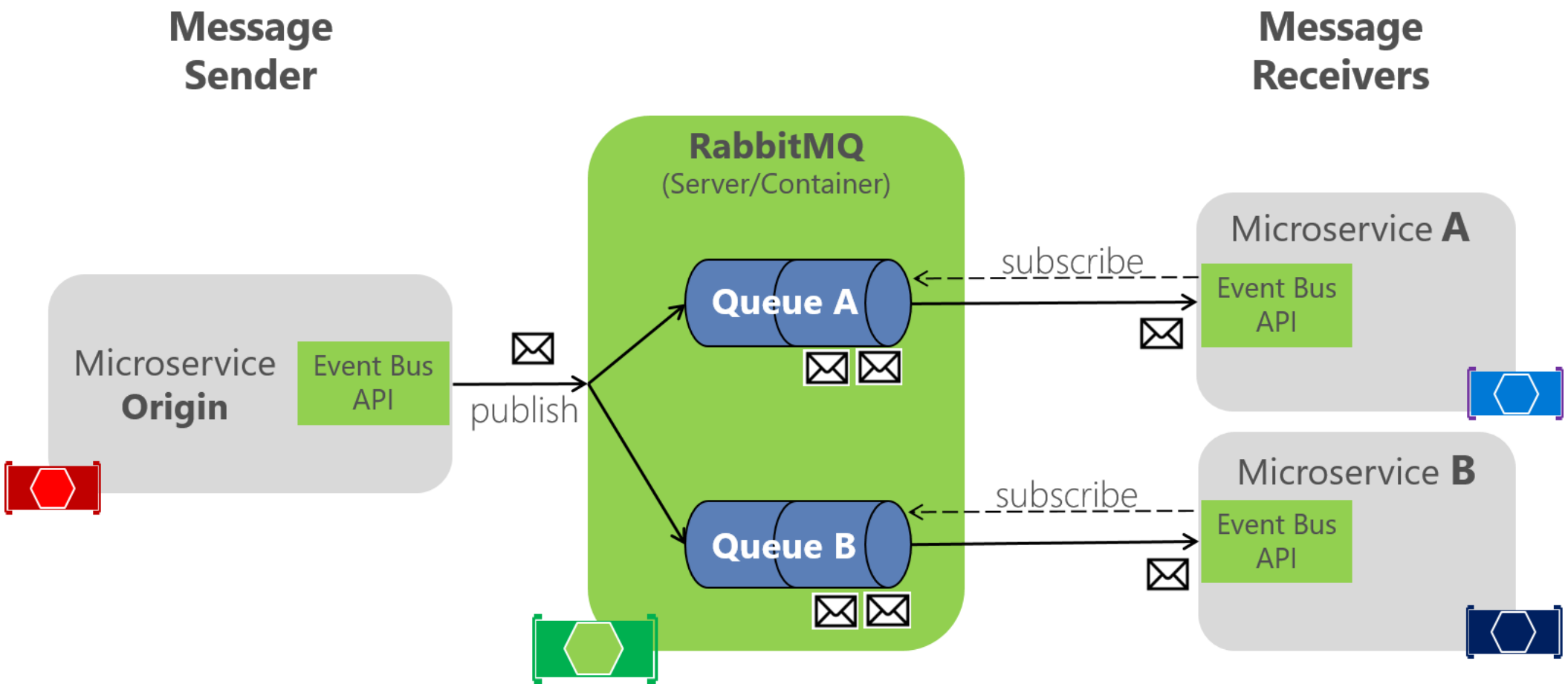
Docker Operations for Basket Microservices

- Dockerfile creation
- Dockerfile commands
- Docker-compose file creation
- Docker-compose file commands



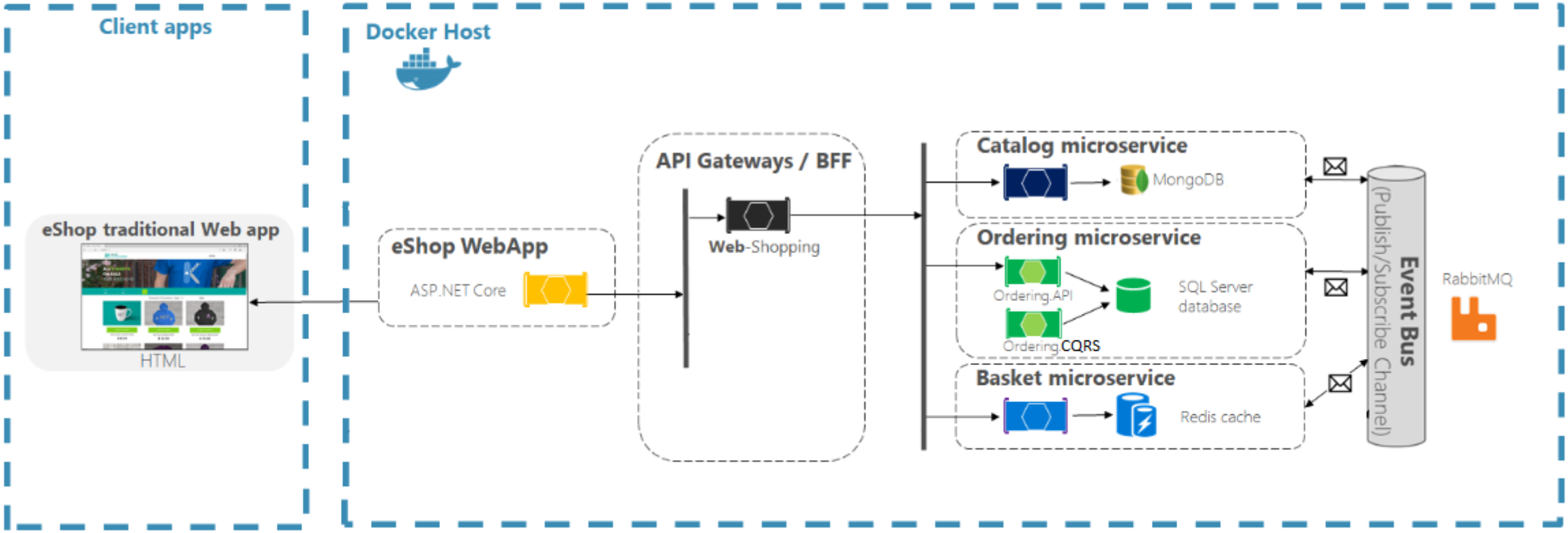
MICROSERVICES COMMUNICATON with BUILDING RABBITMQ LIBRARY

- **Microservice communication** with **RabbitMQ** implementation
- **Class Library** Development for **EventBus** operations
- **RabbitMQ Producer** on Basket Microservice Web API
- **RabbitMQ Consumer** on Ordering Microservice Web API
- **AutoMapper** implementation when mapping Event to Microservices entity
- RabbitMQ **Docker** Implementation


Message Sender
Message Receivers
RabbitMQ
(Server/Container)
Microservice Origin
Event Bus API
publish
Queue A
Queue B
subscribe
subscribe
Microservice A
Event Bus API
Microservice B
Event Bus API

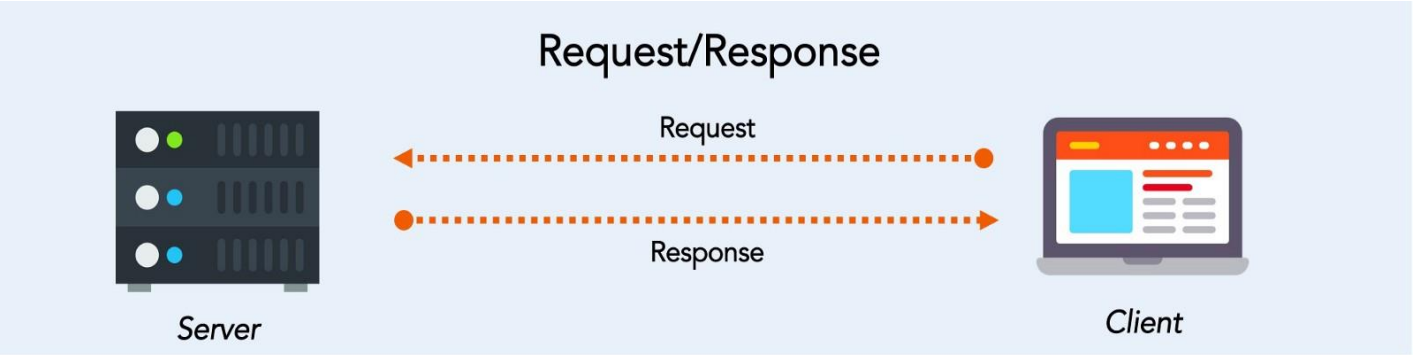
aspnetrun

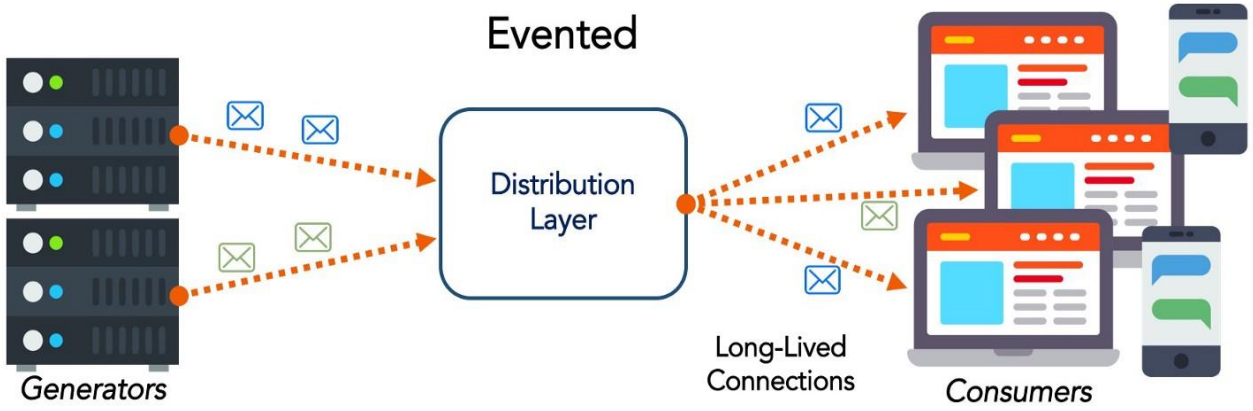
aspnetrun-microservices Environment Architecture



Communications with Microservices

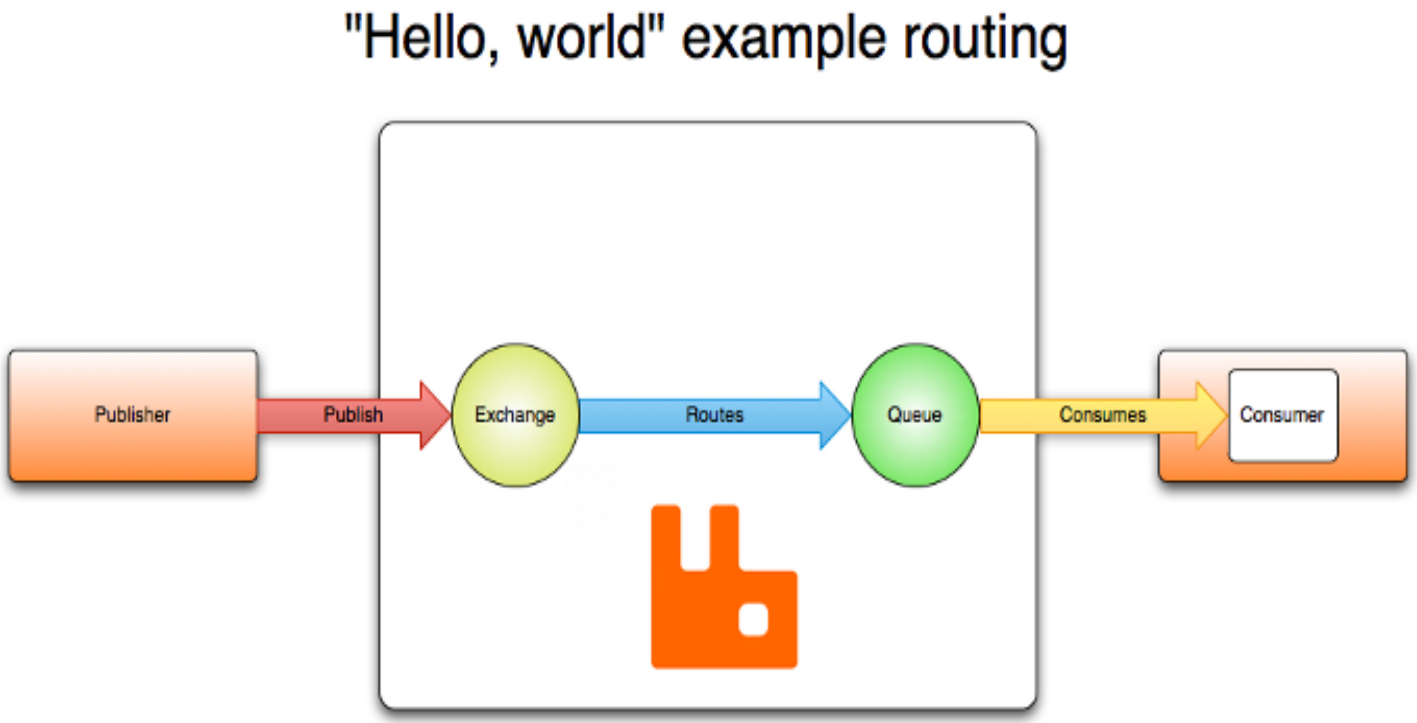
- Request-Driven Architecture
- Event-Driven Architecture
- Hybrid Architecture





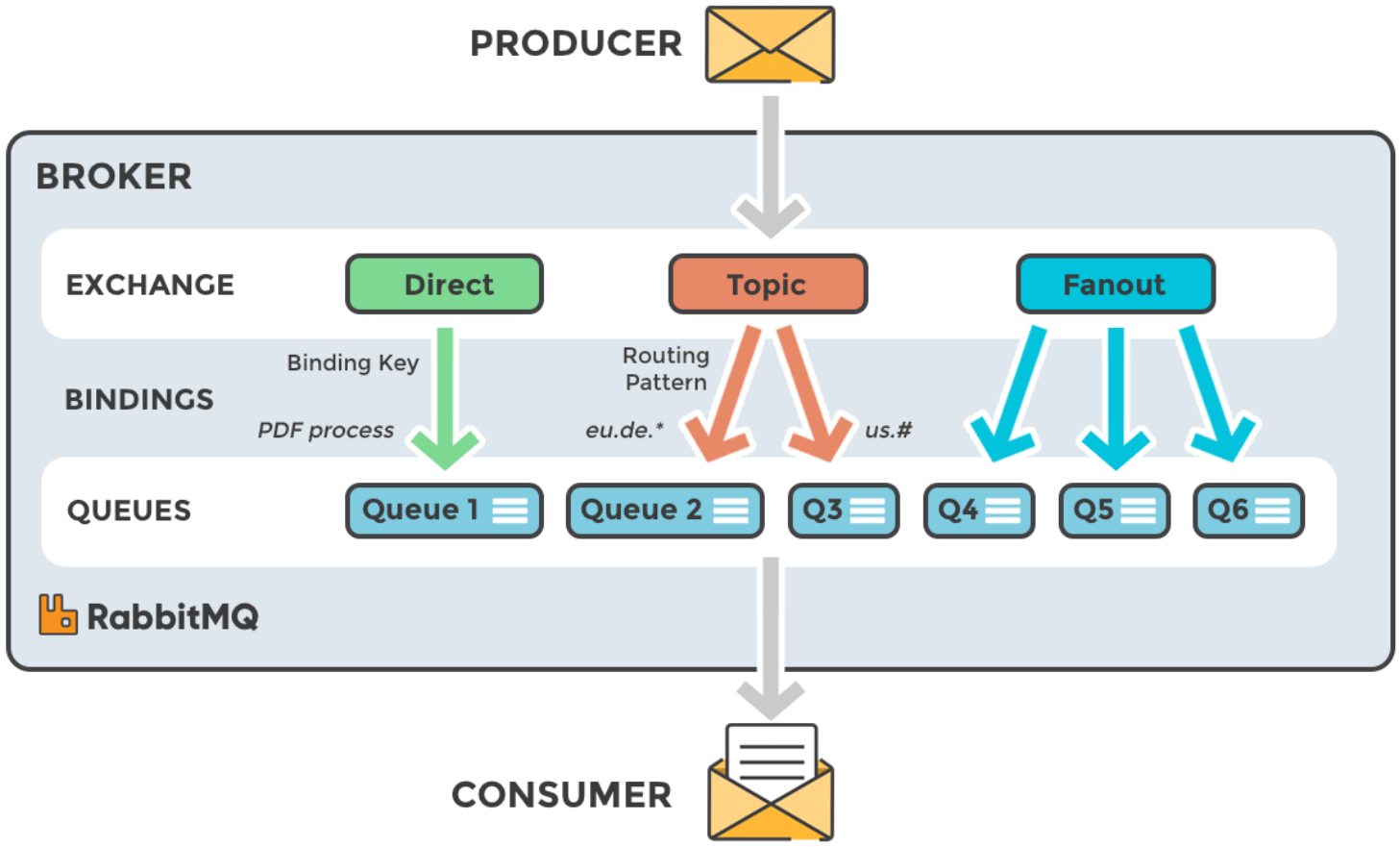
RabbitMQ

- Message Queue System
- Event-Driven Architecture
- Apache Kafka, Msmq, Microsoft Azure Service Bus, Kestrel, ActiveMQ



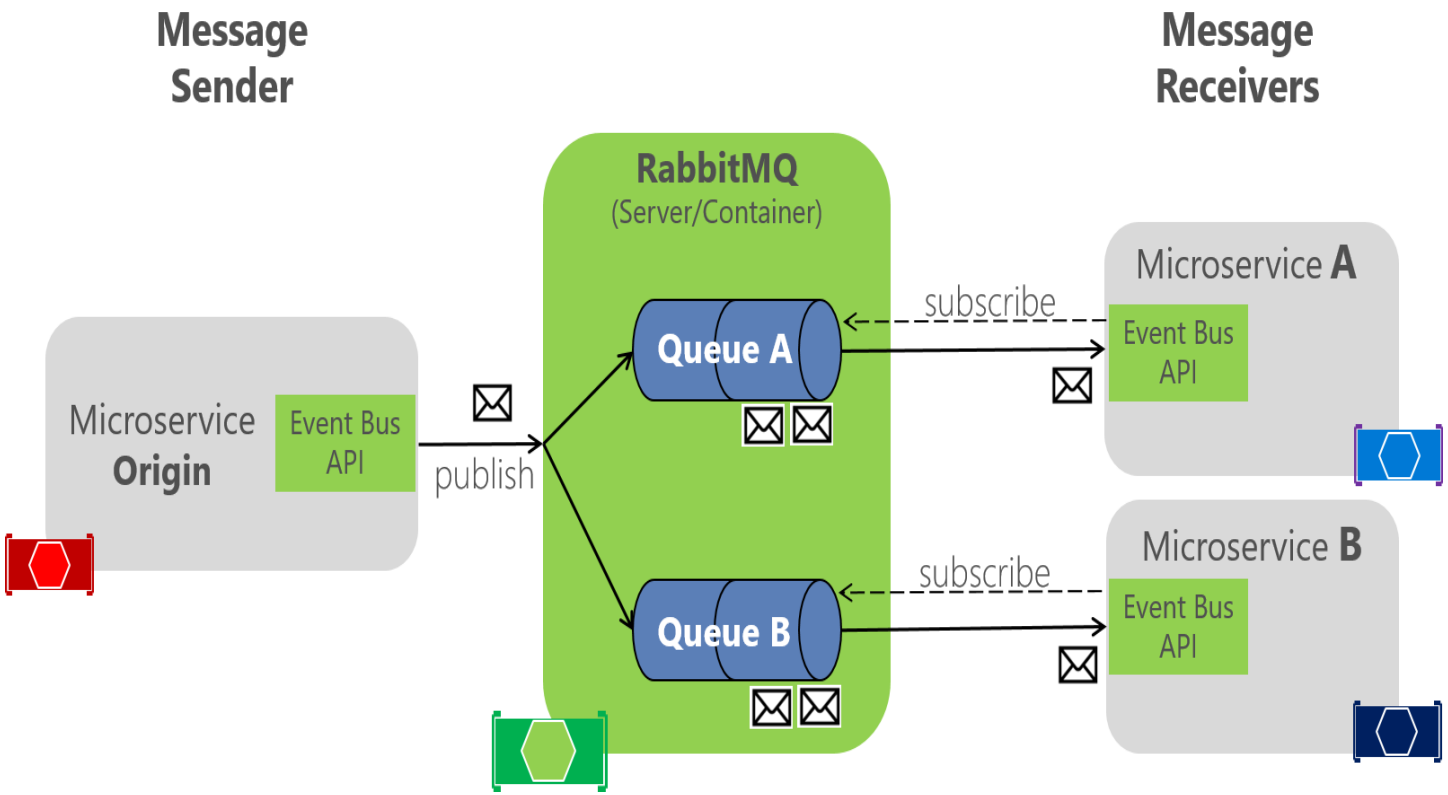
Main Logics of RabbitMQ

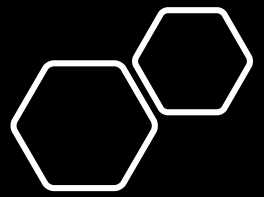
- Producer
- Queue
- Consumer
- Message
- Exchange
- FIFO



Analysis & Design of EventBus Class Library

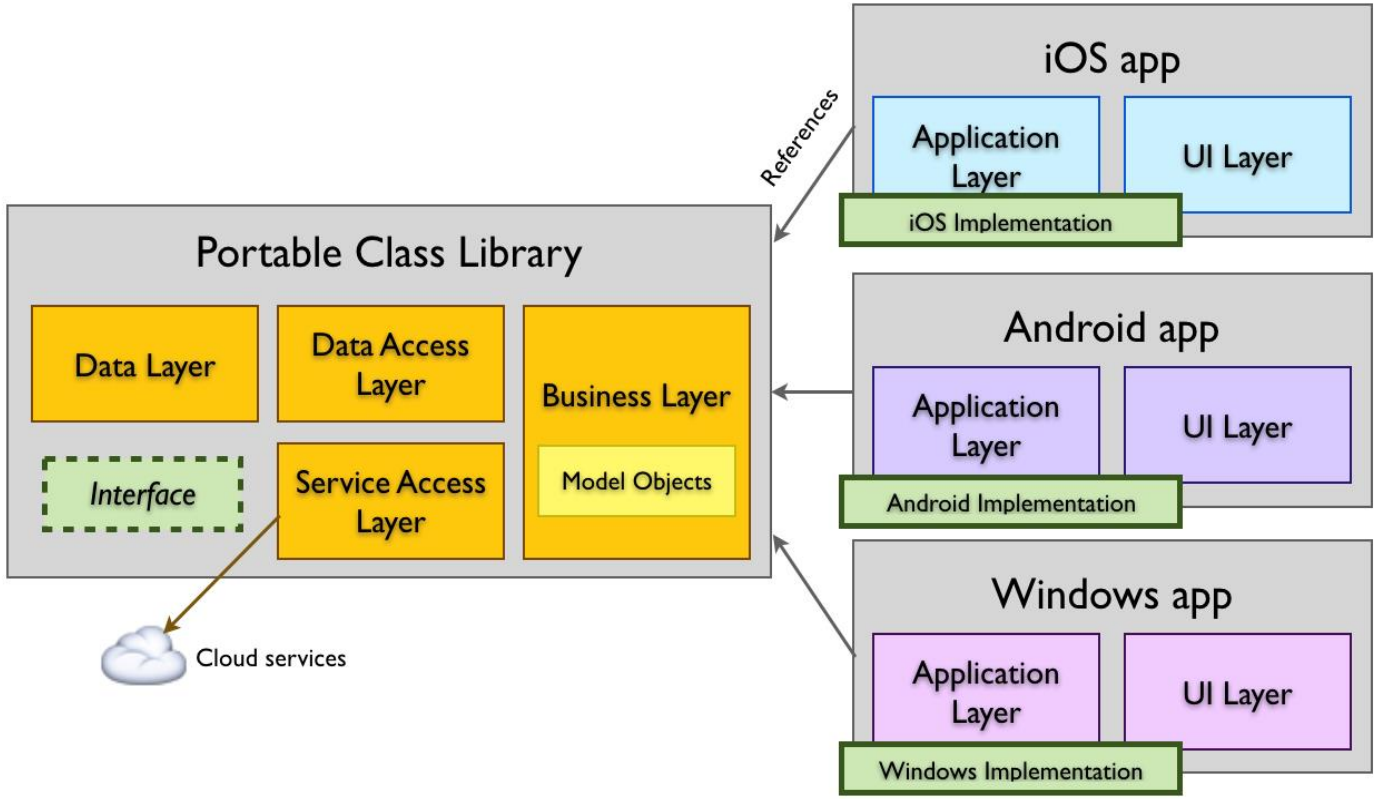
- Create RabbitMQ Connection
- Create BasketCheckout Event
- Develop Basket Microservices as Producer of BasketCheckout Event
- Develop Ordering Microservices as Consumer of BasketCheckout Event





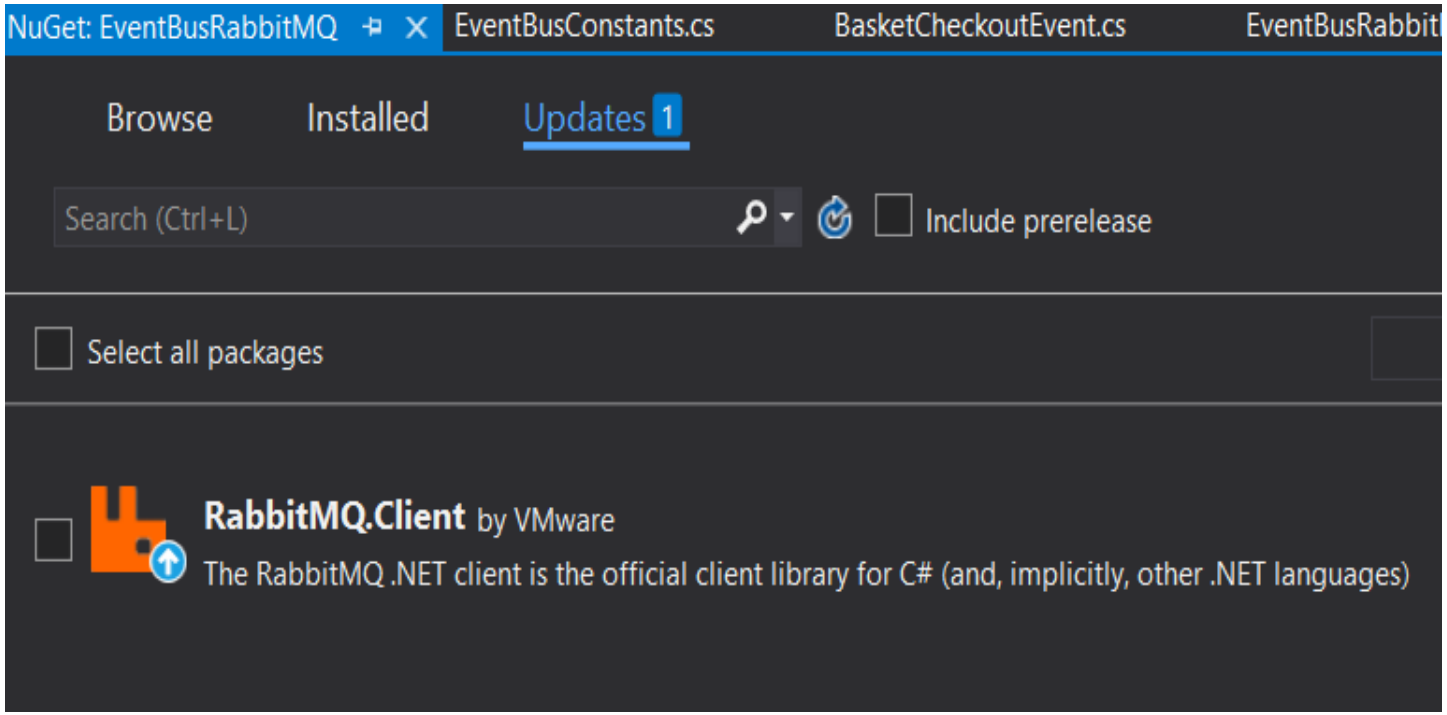
Architecture of Basket Microservices

- EventBus RabbitMQ Class Library



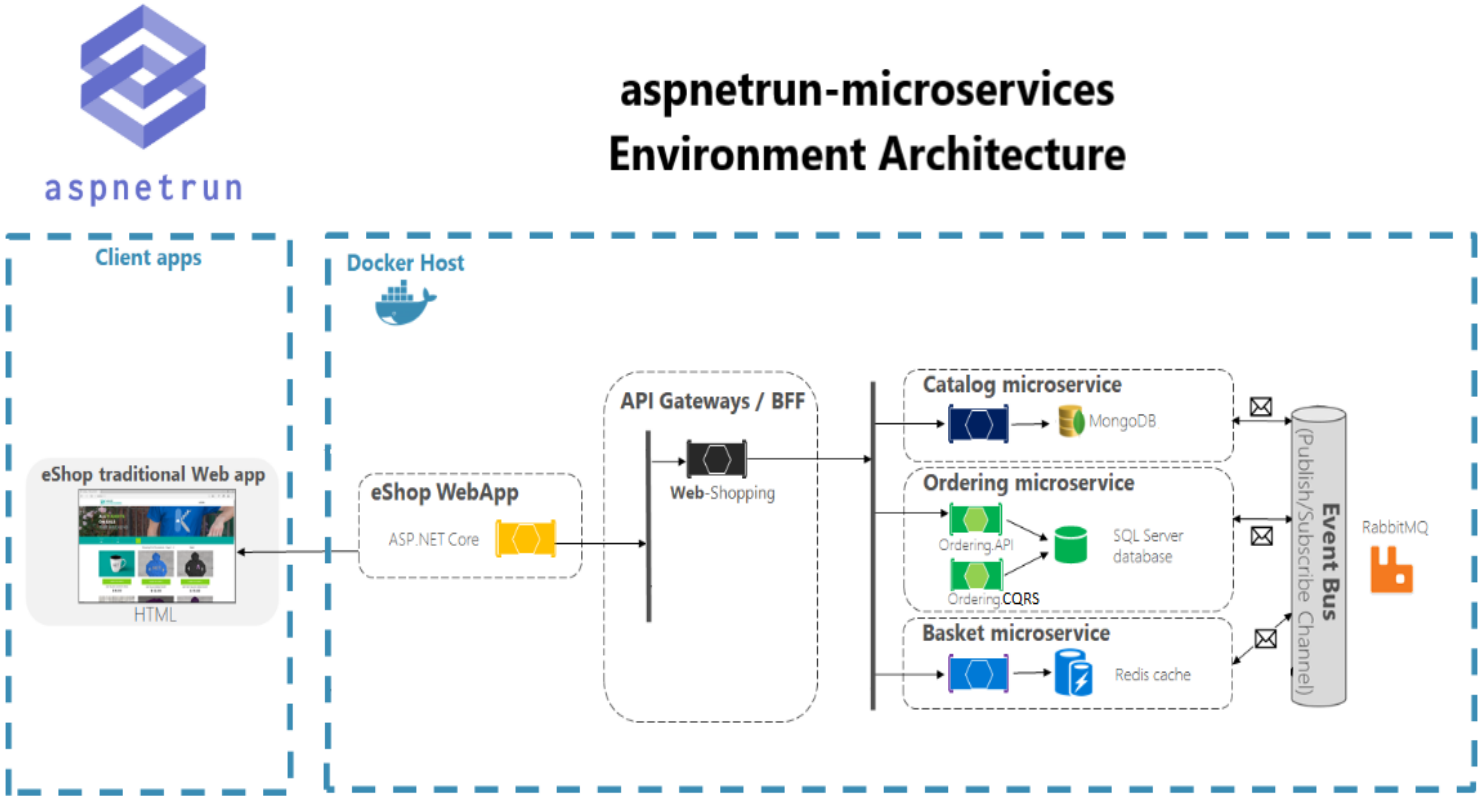
Nuget Package of RabbitMQ Client

- RabbitMQ.Client



Docker Operations for RabbitMQ EventBus

- Docker-compose file mofidication
- Docker-compose file commands

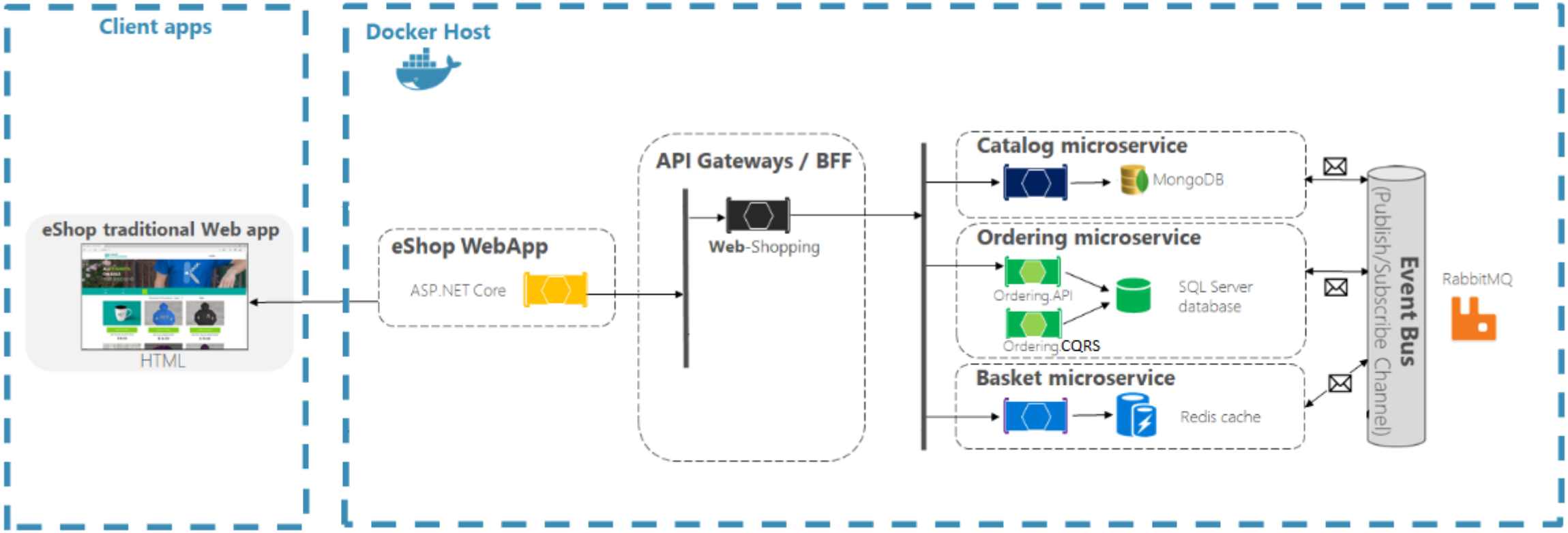


BUILDING ORDERING MICROSERVİCES with Clean Architecture and CQRS

- ASP.NET Core **Web API** application
- **REST** API principles, CRUD operations
- **SQL Server** database connection on docker
- **Entity Framework Core Code-First** Approach
- **Domain Driven Design** (Entities, Repositories, Domain/Application Services, DTO's…)
- **Clean Architecture** implementation with applying **SOLID** principles (Core, Application, Infrastructure and Presentation Layers)
- **CQRS implementation** on commands and queries
- **Swagger** Open API implementation
- **Dockerfile** implementation

aspnetrun

aspnetrun-microservices Environment Architecture



Ordering Microservices

- Sql Server
- Entity Framework Core
- Domain Driven Design
- Clean Architecture
- CQRS
- Swagger
- Docker Container

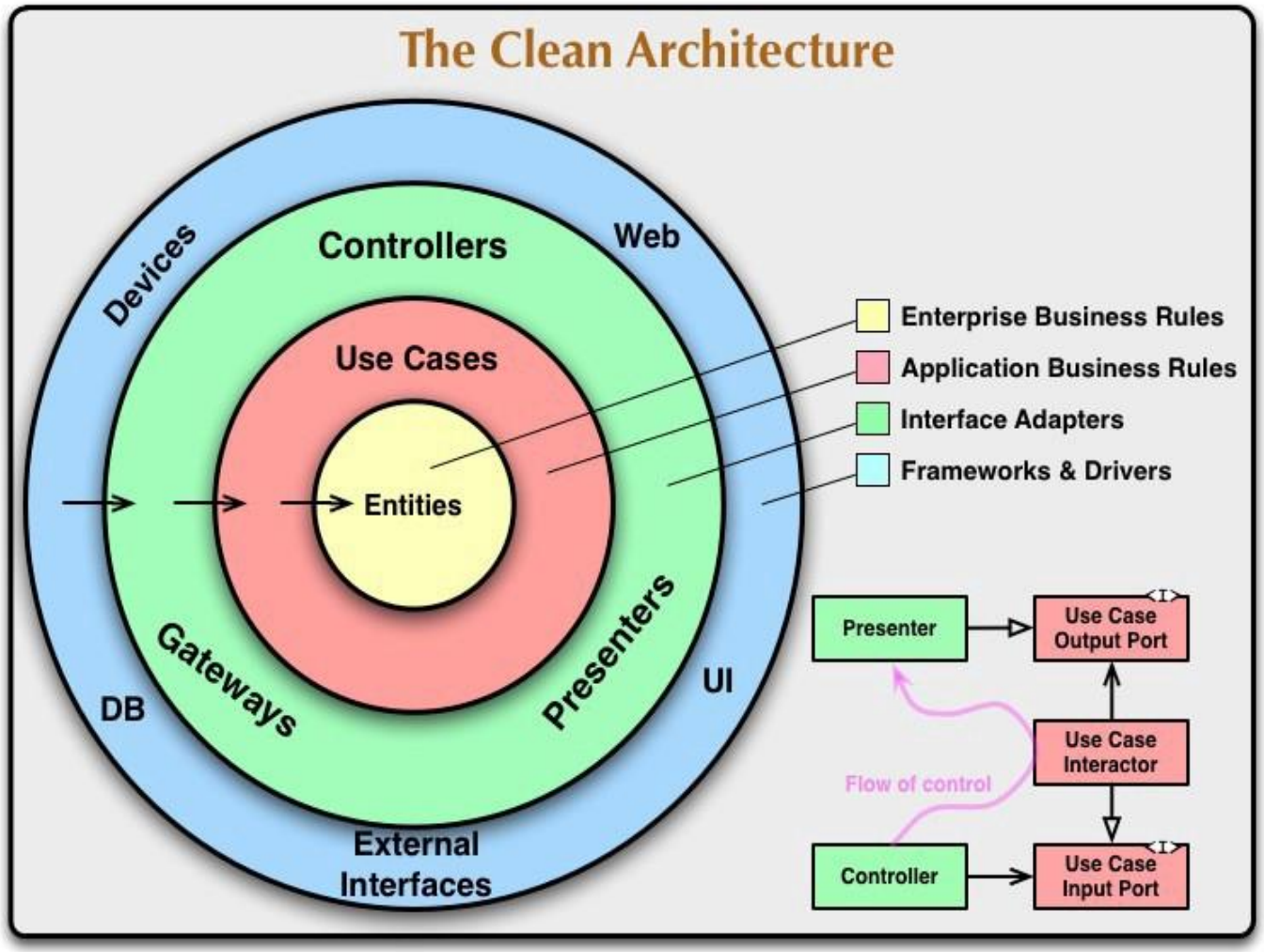
Analysis & Design of Ordering Microservices

- Get Orders with username
- Consume basketCheckout event from RabbitMQ
- CQRS implementation with triggering OrderCommand to insert Order record

Method	Request URI	Use Case
GET	api/v1/Order	Get Orders with username

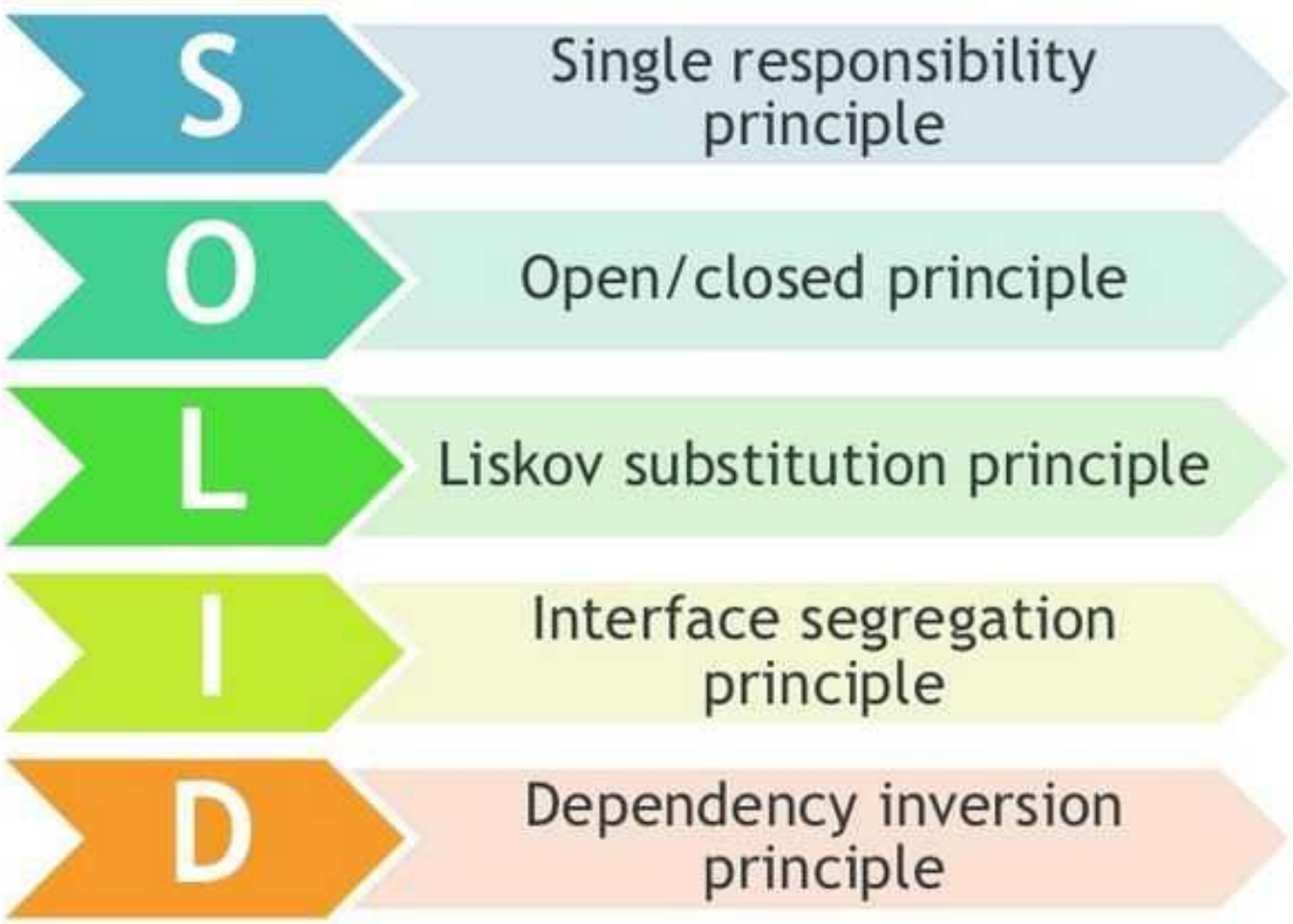
Architecture of Ordering Microservices

- SOLID Principles
- Domain Driven Design
- Clean Architecture
- Mediator Design Pattern
- CQRS Design Pattern
- CQRS & Event Sourcing



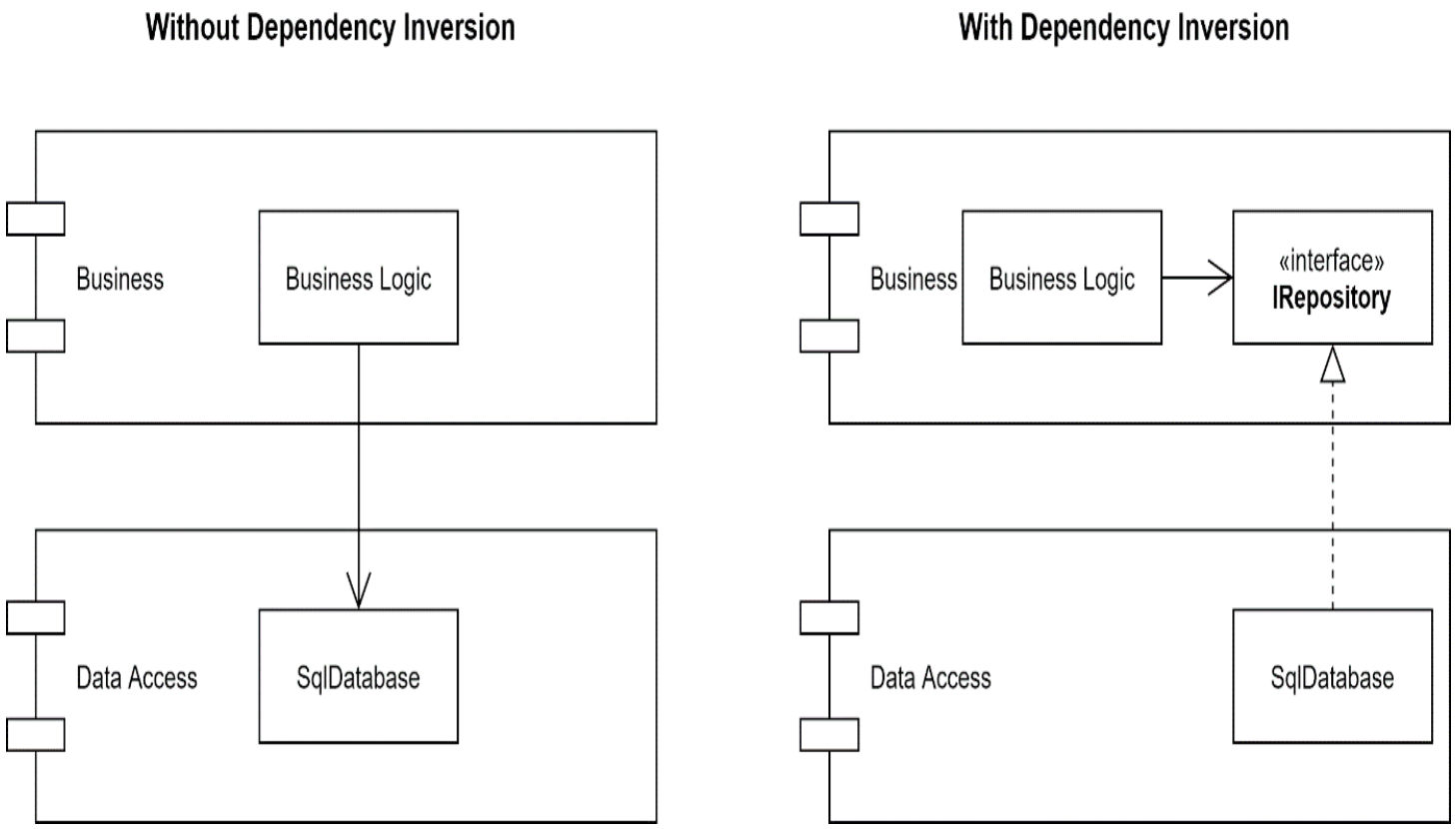
SOLID Principles

- Single-Responsibility
- Open-Closed Principle
- Liskov Substitution Principle
- Interface Segregation Principle
- Dependency Inversion Principle



Dependency Inversion Principles (DIP)

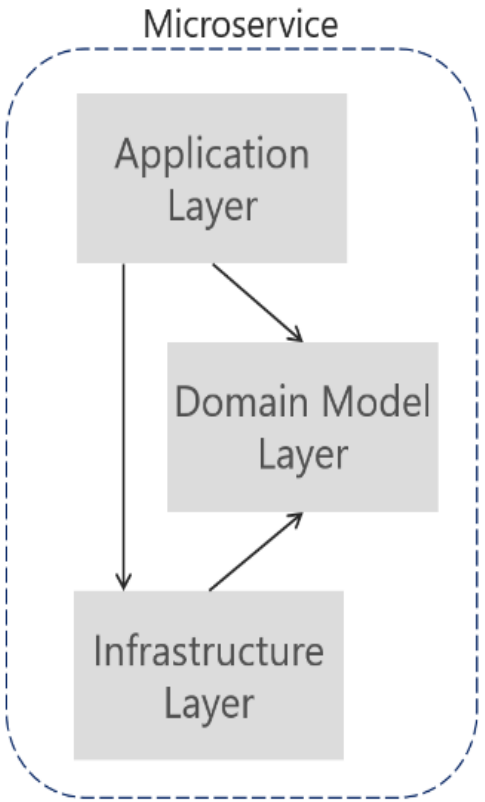
- Upper-level layer uses Lower-level layer
- Re-usability without touching code
- Reusable modules



Domain Driven Design (DDD)

- Ubiquitous Language
- Entitiy & Value Object
- Aggregate Root (AR)
- Bounded Context

Dependencies between Layers in a Domain-Driven Design service



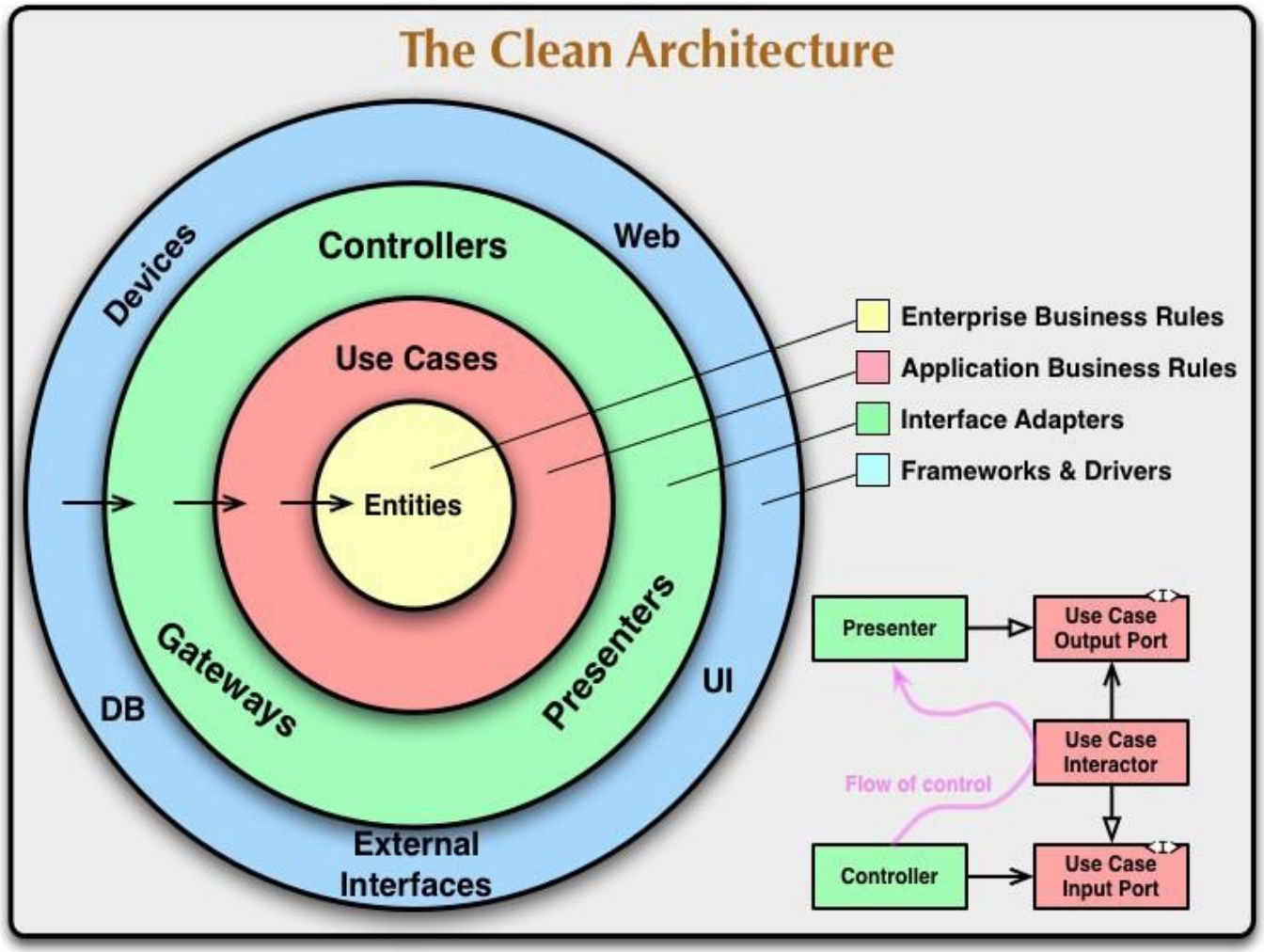
- Depends on the Domain-Model Layer so it can:
  - Use entity objects
  - Use Repository Interfaces/Contracts
- Depends on the Infrastructure Layer (thru DI) so it can:
  - Use Repository implementation classes, ideally through DI

- Ideally, it must NOT take dependency on any other layer
- It implements:
  - Domain Entities, Aggregate-Roots and Value-Objects
  - Repository Contracts/Interfaces (to be used in DI)

- Depends on the Domain-Model Layer so it can:
  - Use entity objects.
    - Like EF updating a database through mapped entities
- Direct dependency on infrastructure frameworks like EF Core or any other database, cache or infrastructure API

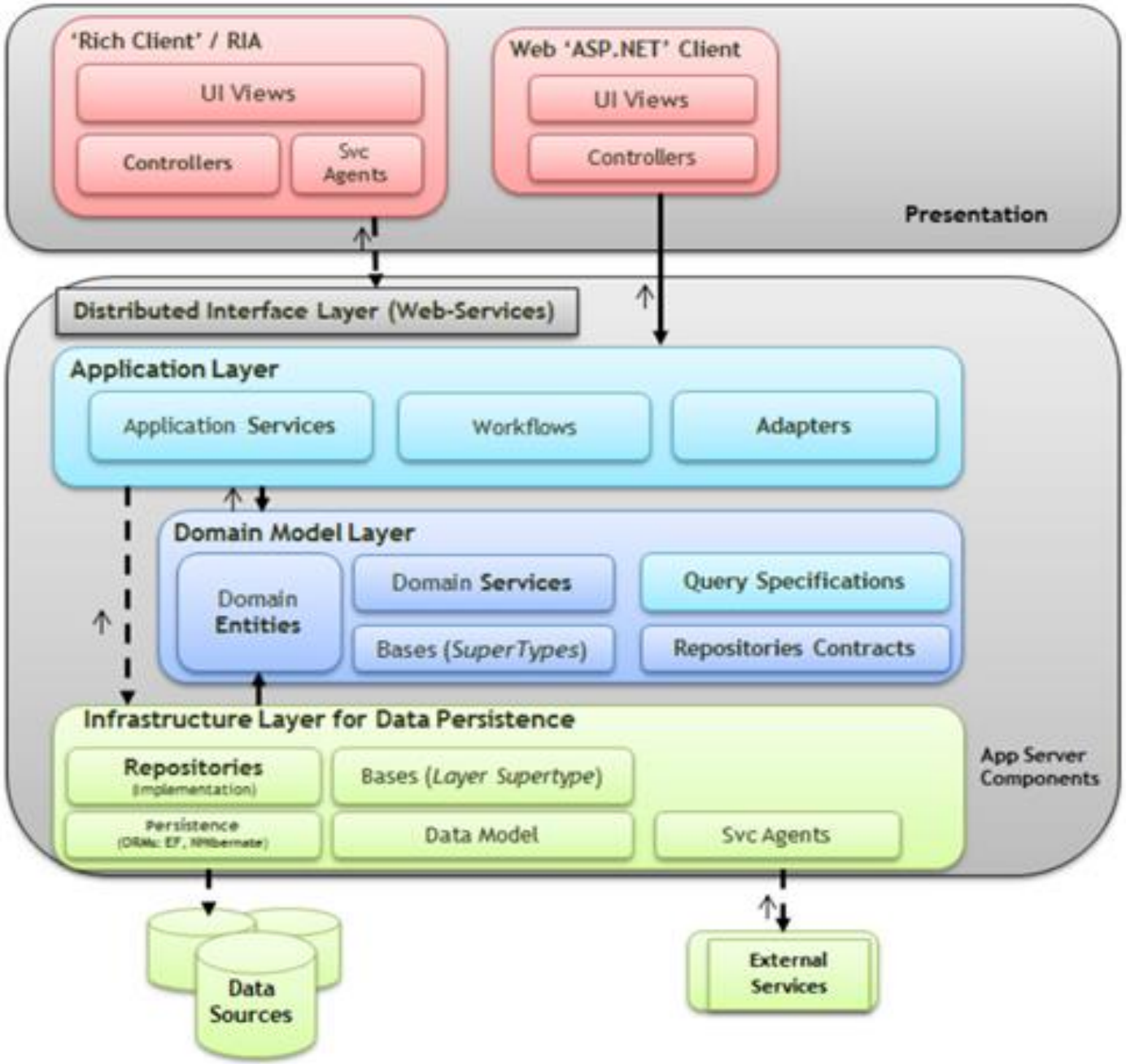
Clean Architecture (aka Ports and Adaptors)

- Hexagonal Architecture
- Onion Architecture



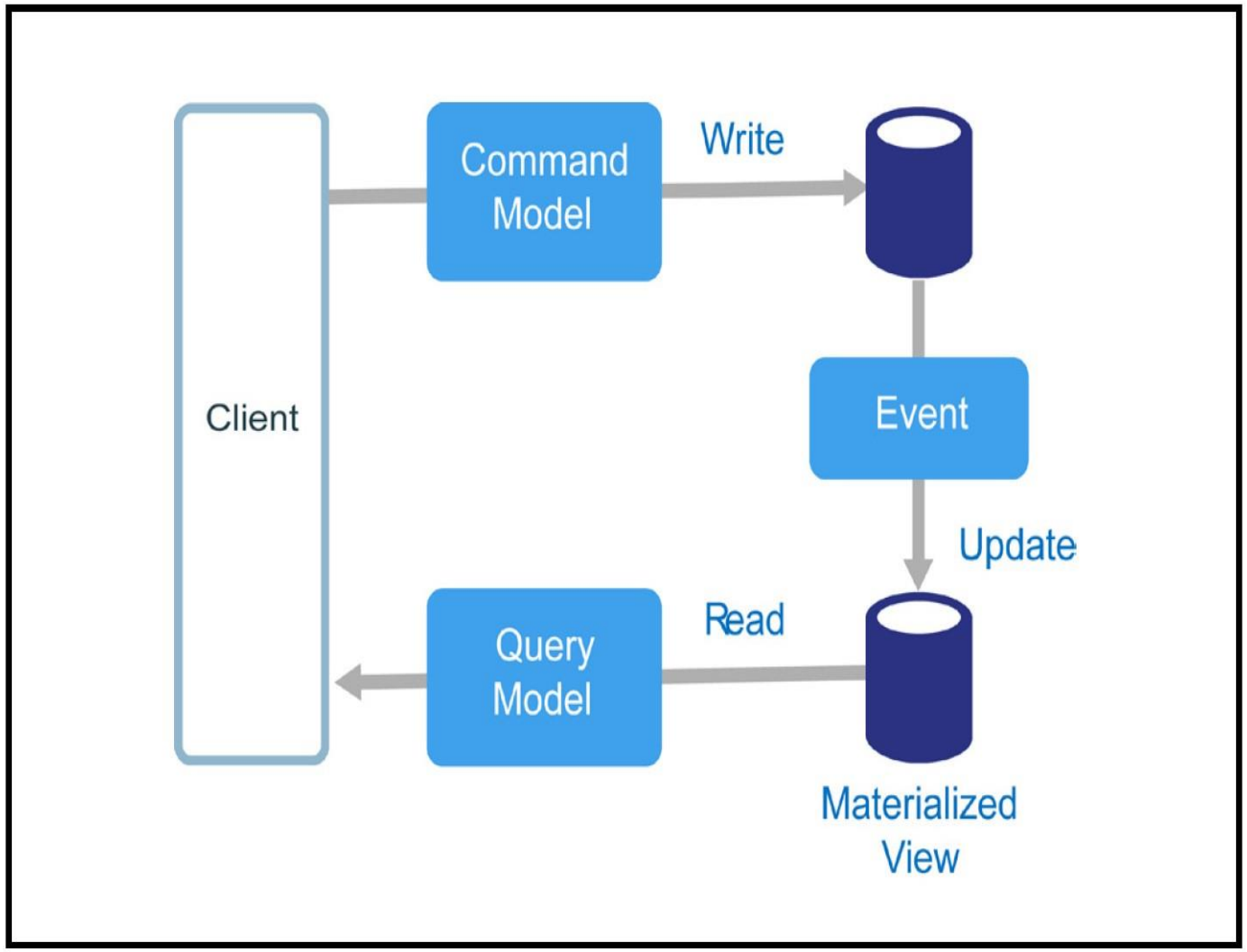
Clean Architecture (aka Ports and Adaptors)

- Core Layer
- Application Layer
- Infrastructure Layer
- Presentation Layer



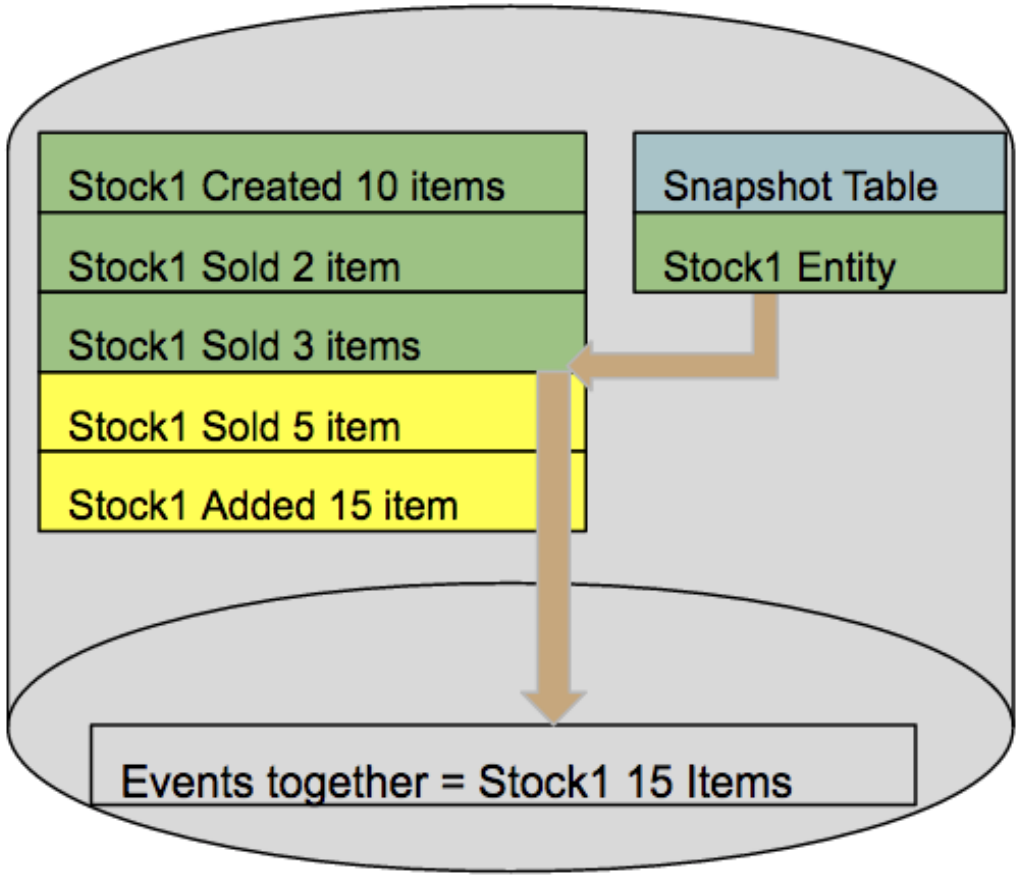
CQRS Design Pattern

- Separation of Commands and Query Responsibility
- CQS (Command Query Separation)
- Commands – CommandHandlers
- Query – QueryHandlers



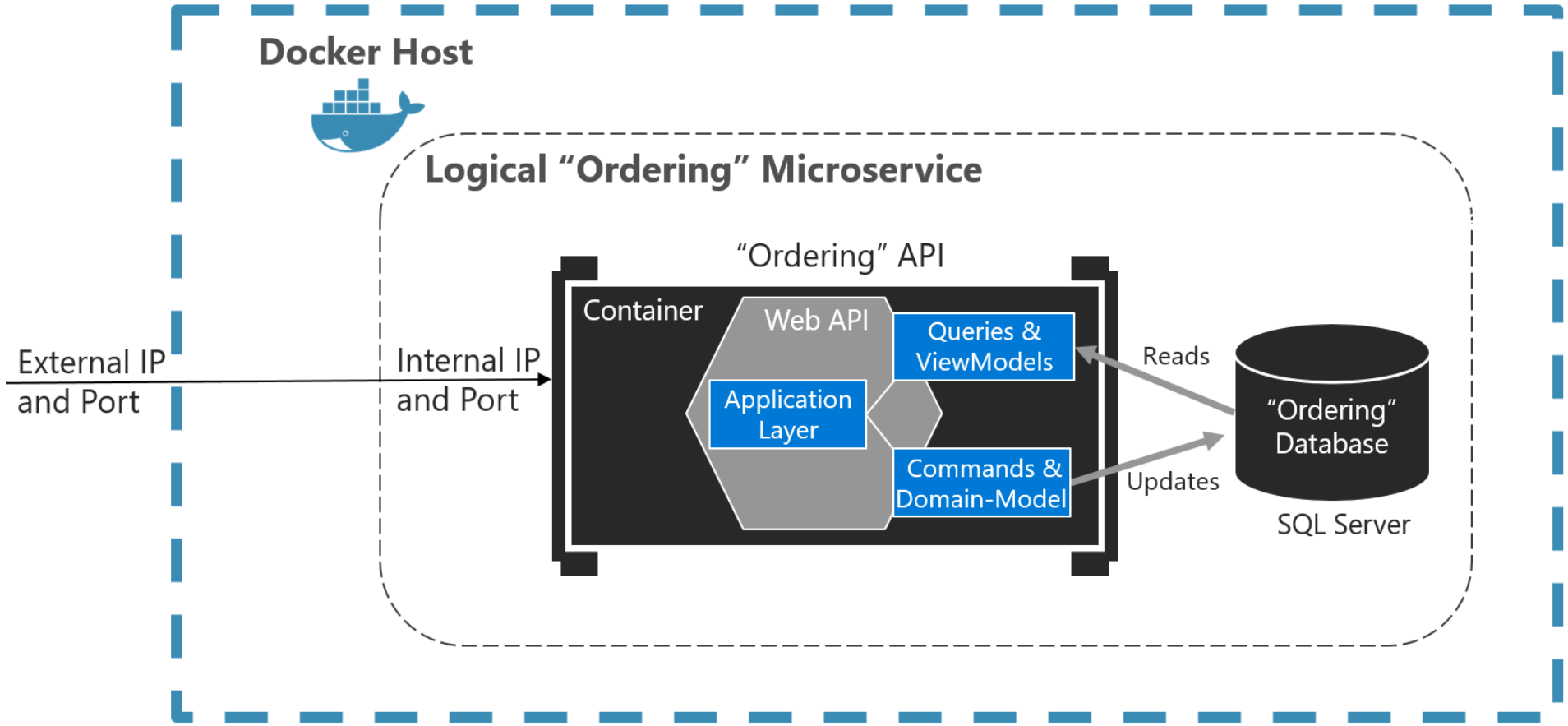
Event Sourcing

- Accumulating Events
- Assets are not recorded
- Events affect the state of asset
- Sum of the Events



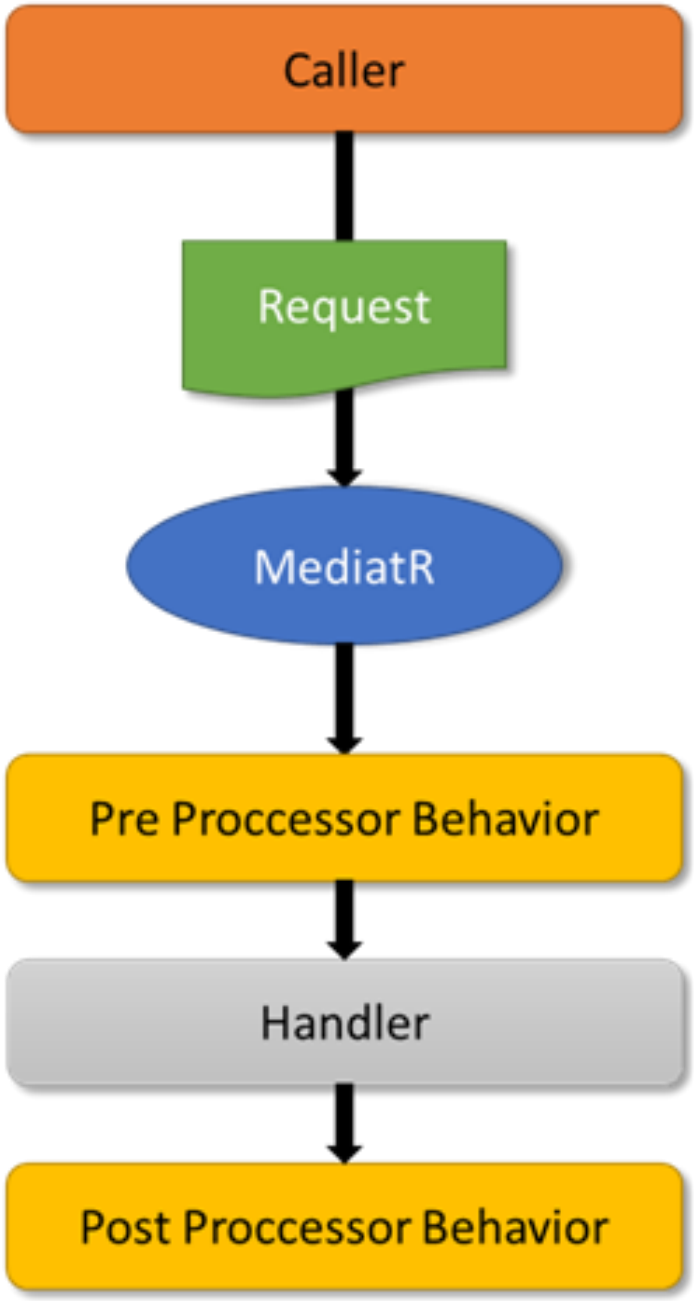
Simplified CQRS and DDD microservice

High level design

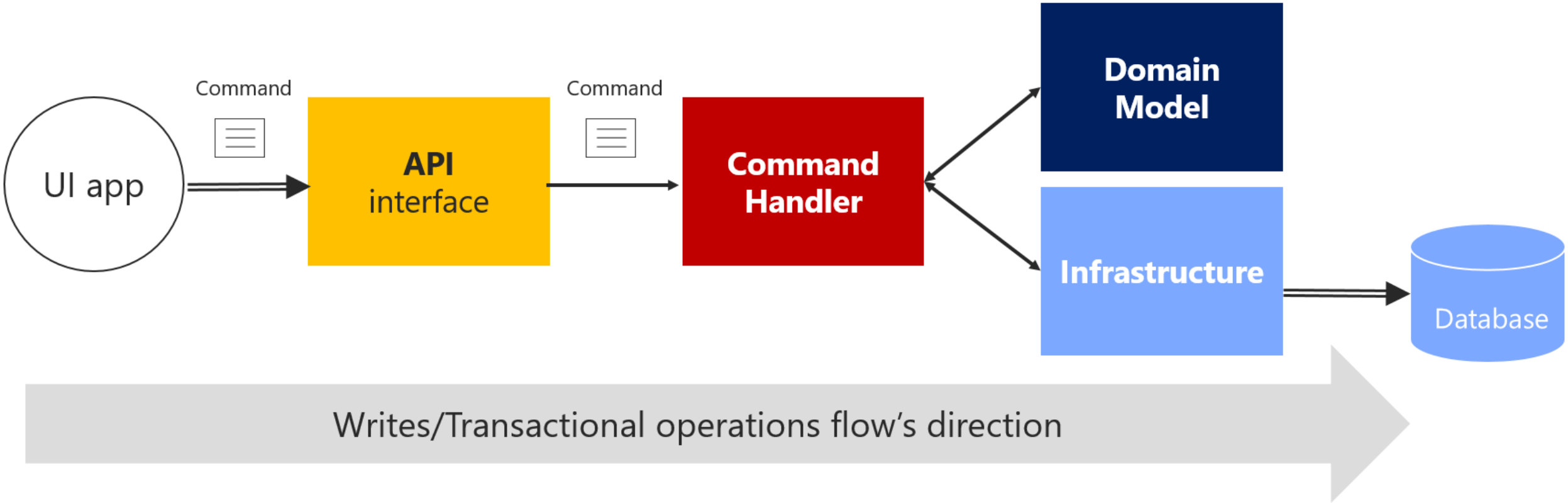


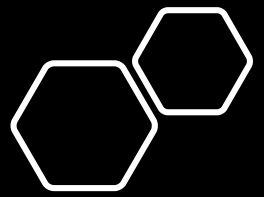
Mediator Design Pattern

- MediatR Nuget Package



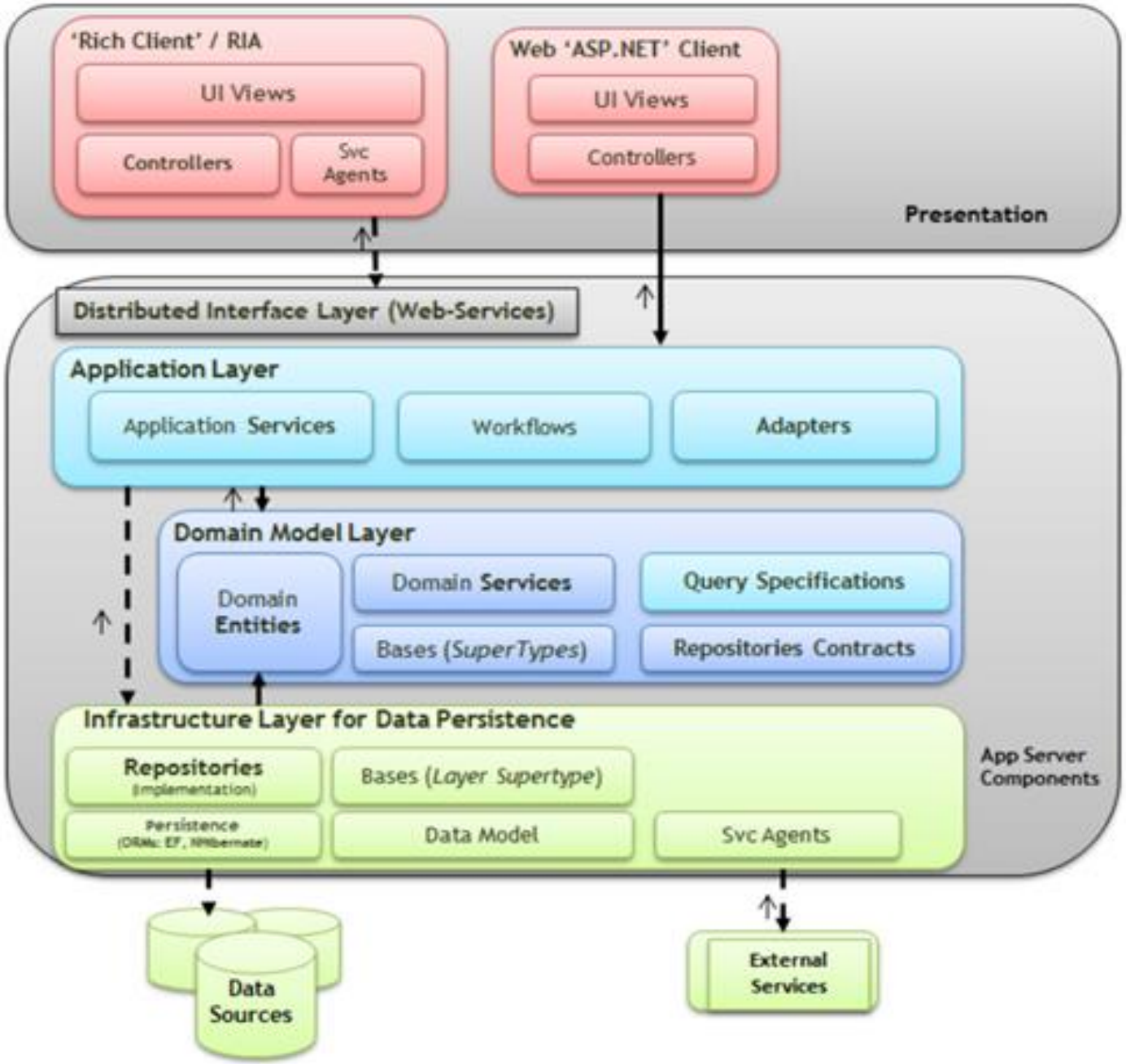
High level “Writes-side” in CQRS





Code Structure of Ordering Microservices

- Ordering.Core Layer
- Ordering.Infrastructure Layer
- Ordering.Application Layer
- Ordering.API Layer



Nuget Package of Ordering Microservices

- Ordering.Infrastructure Layer

Microsoft.EntityFrameworkCore by Microsoft
Entity Framework Core is a lightweight and extensible version of the popular Entity Framework data access technology.

Microsoft.EntityFrameworkCore.Design by Microsoft
Shared design-time components for Entity Framework Core tools.

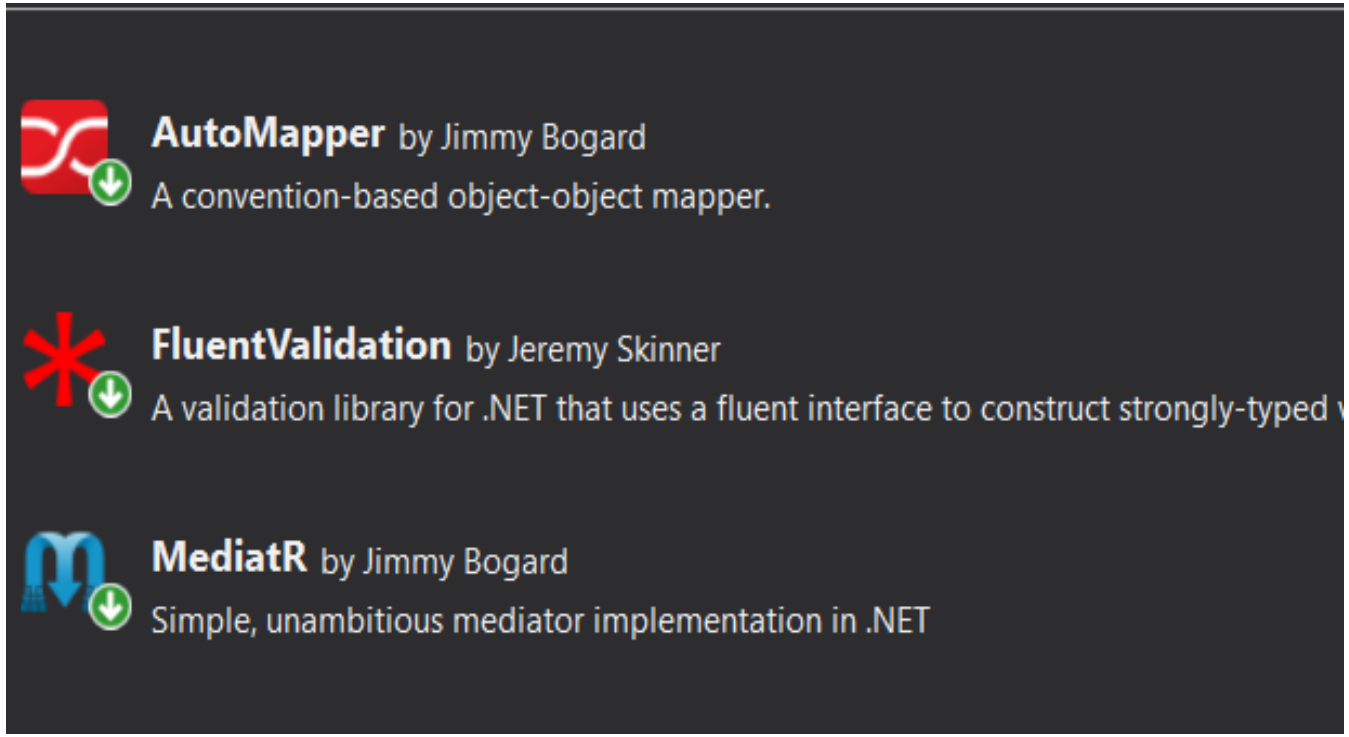
Microsoft.EntityFrameworkCore.InMemory by Microsoft
In-memory database provider for Entity Framework Core (to be used for testing purposes).

Microsoft.EntityFrameworkCore.SqlServer by Microsoft
Microsoft SQL Server database provider for Entity Framework Core.

Microsoft.EntityFrameworkCore.Tools by Microsoft
Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.

Nuget Package of Ordering Microservices

- Ordering.Application Layer



Nuget Package of Ordering Microservices

- Ordering.API Layer

AutoMapper.Extensions.Microsoft.DependencyInjection by Jimmy Bogard
AutoMapper extensions for ASP.NET Core

MediatR by Jimmy Bogard
Simple, unambitious mediator implementation in .NET

MediatR.Extensions.Microsoft.DependencyInjection by Jimmy Bogard
MediatR extensions for ASP.NET Core

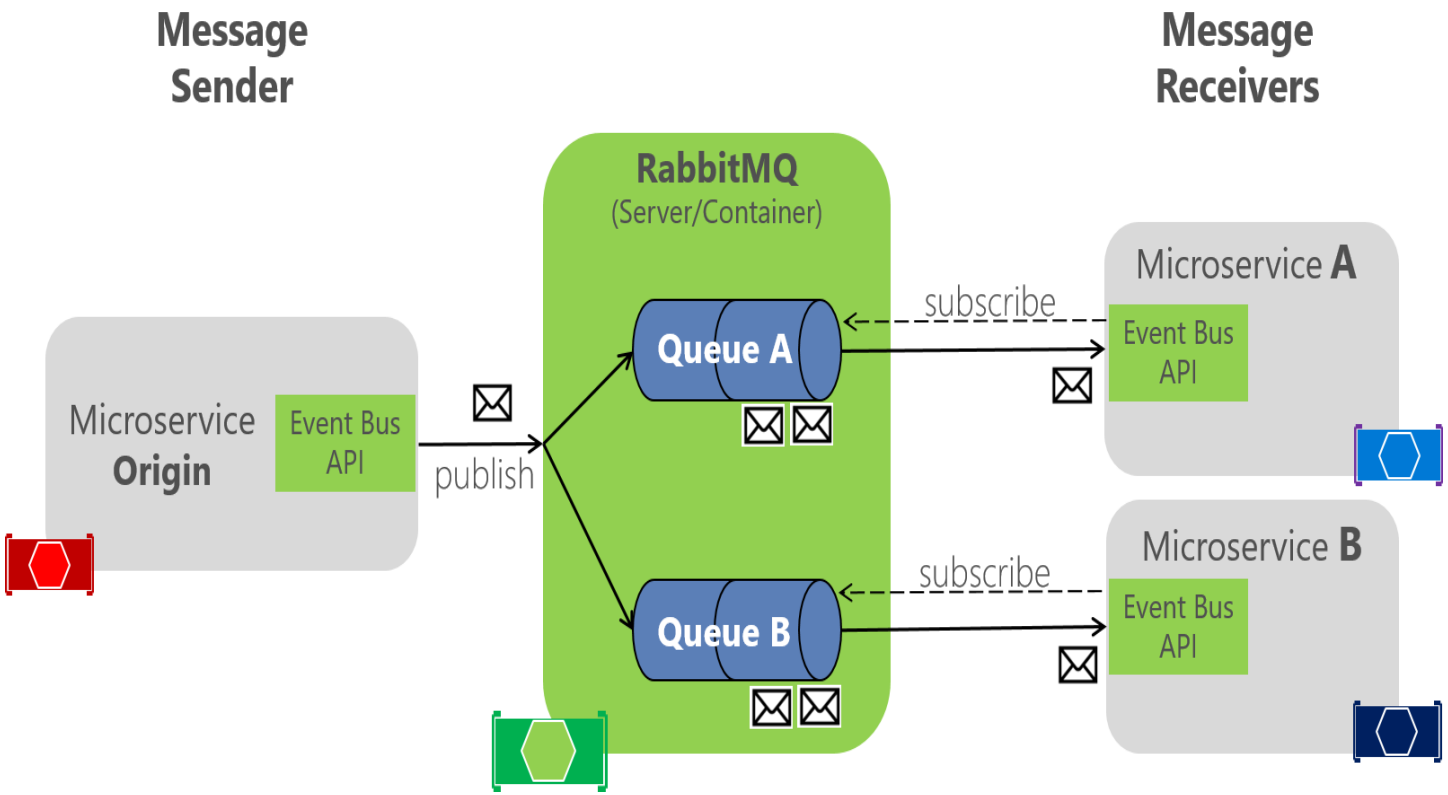
Microsoft.EntityFrameworkCore.Design by Microsoft
Shared design-time components for Entity Framework Core tools.

Microsoft.VisualStudio.Azure.Containers.Tools.Targets by Microsoft
Targets files to enable the Visual Studio Tools for Containers.

Swashbuckle.AspNetCore by Swashbuckle.AspNetCore
Swagger tools for documenting APIs built on ASP.NET Core

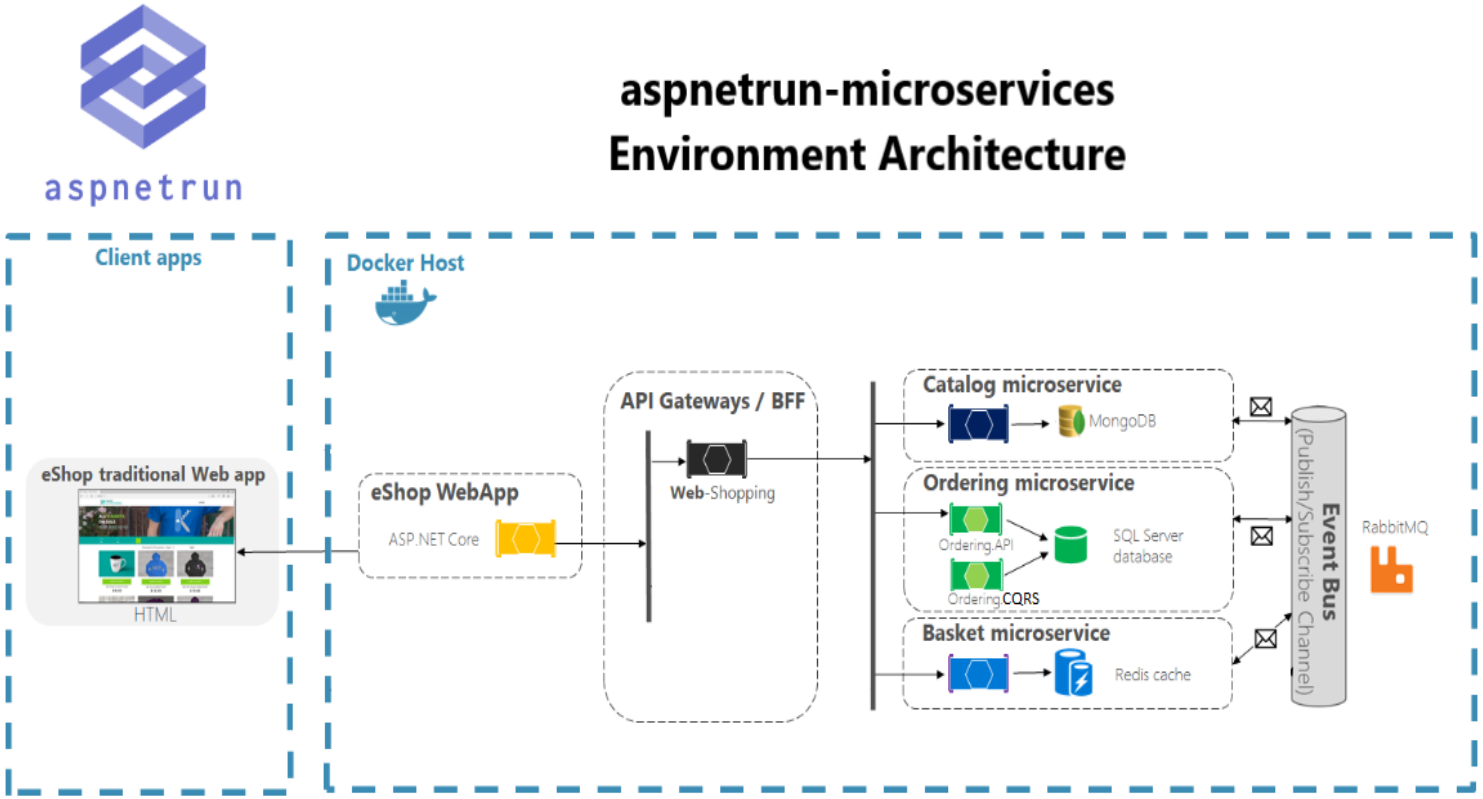
Analysis & Design of EventBus Class Library

- Create RabbitMQ Connection
- Create BasketCheckout Event
- Develop Basket Microservices as Producer of BasketCheckout Event
- Develop Ordering Microservices as Consumer of BasketCheckout Event



Docker Operations for Basket Microservices

- Dockerfile creation
- Dockerfile commands
- Docker-compose file creation
- Docker-compose file commands

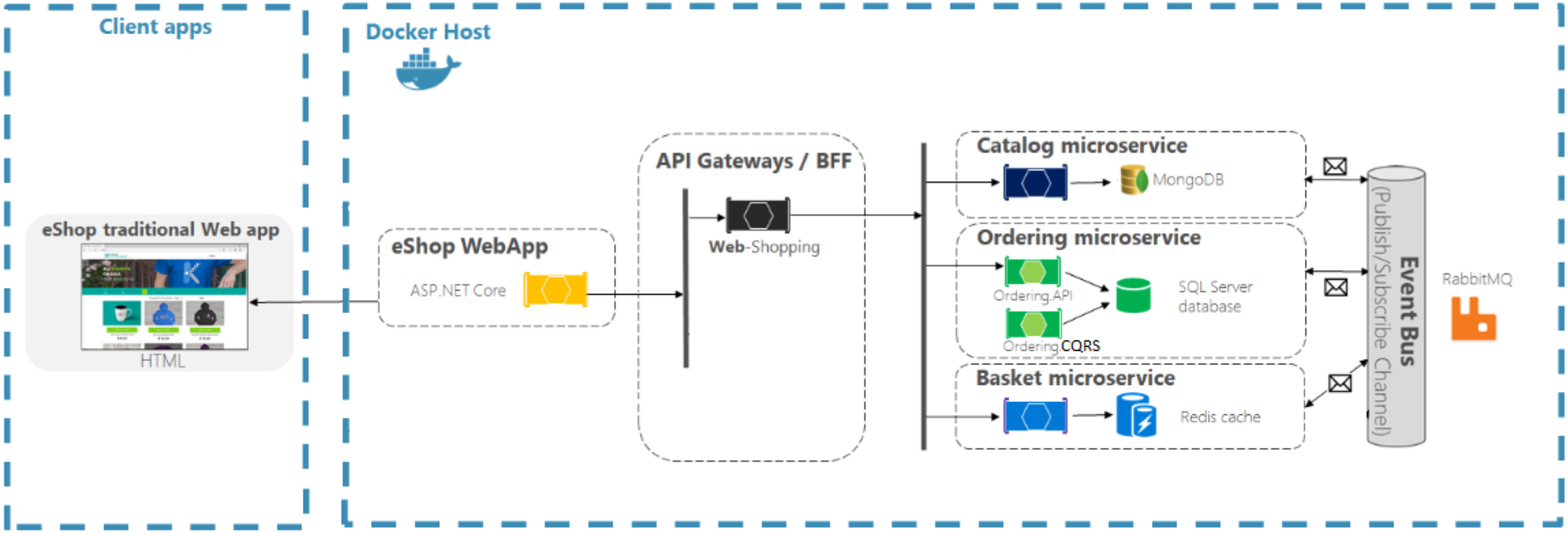


BUILDING OCELOT API GATEWAY MICROSERVİCES

- ASP.NET Core **Blank Web** application
- **Ocelot Routing, UpStream, DownStream**
- **Ocelot** Configuration
- **Dockerfile** implementation
- **Docker-compose** integration

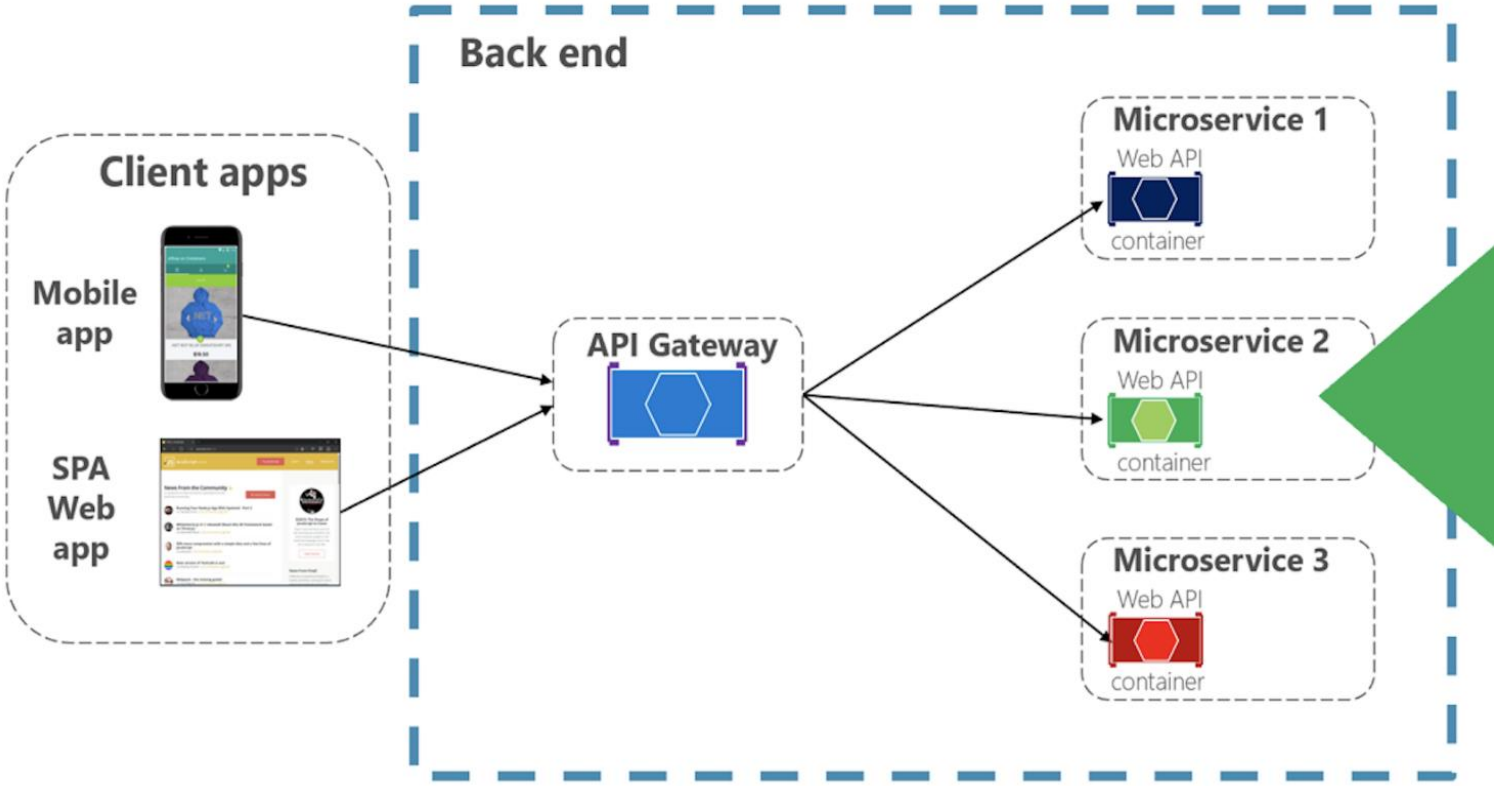
aspnetrun

aspnetrun-microservices Environment Architecture



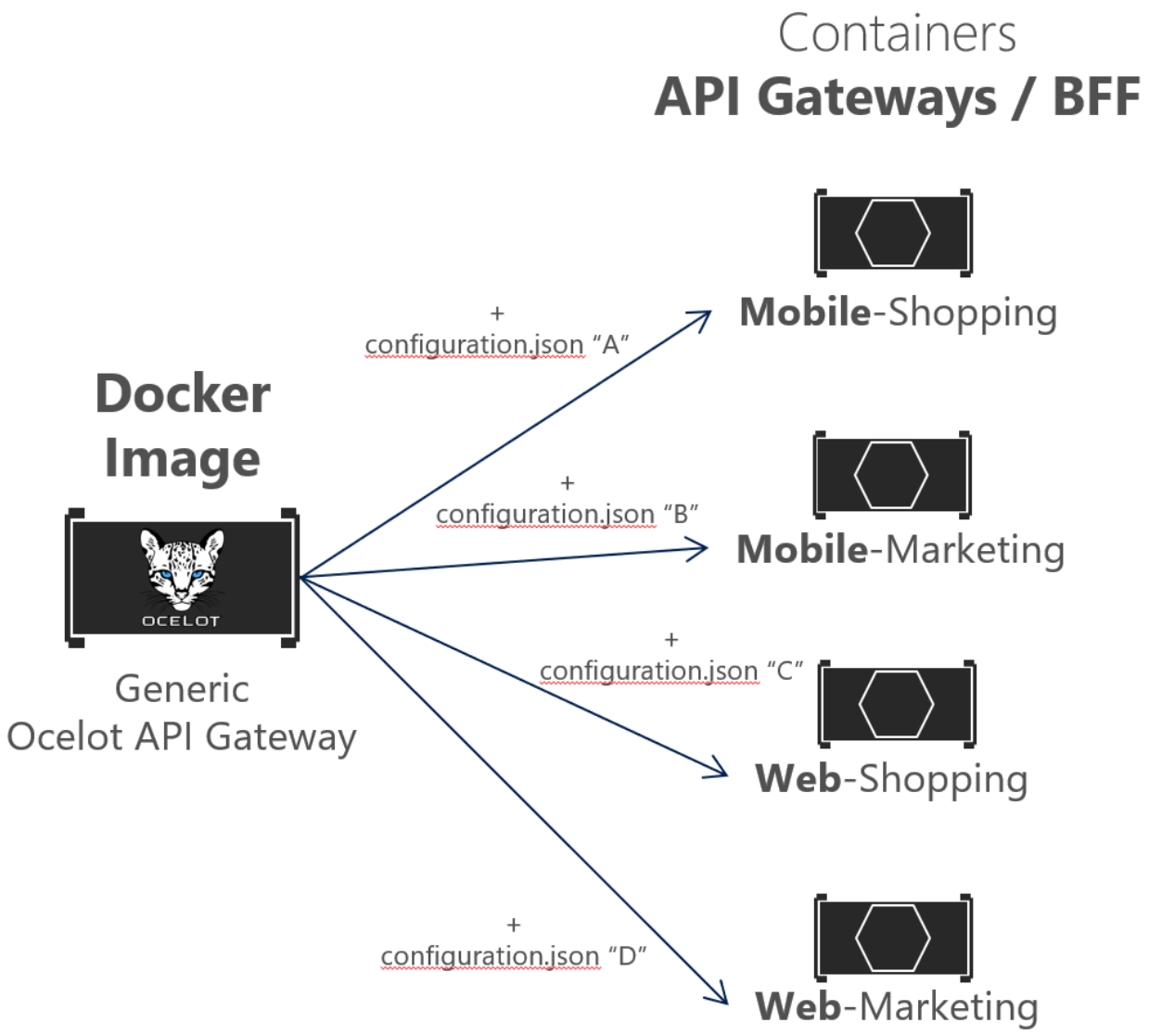
API Gateway Design Pattern

- Routing
- Data Aggregator
- Protocol Abstraction
- Centralized Error Management



Ocelot API Gateway

- Lightweight API Gateway
- .NET Core based API Gateway
- Open Source
- Fast & Scalable



Analysis & Design of API Gateway Microservices

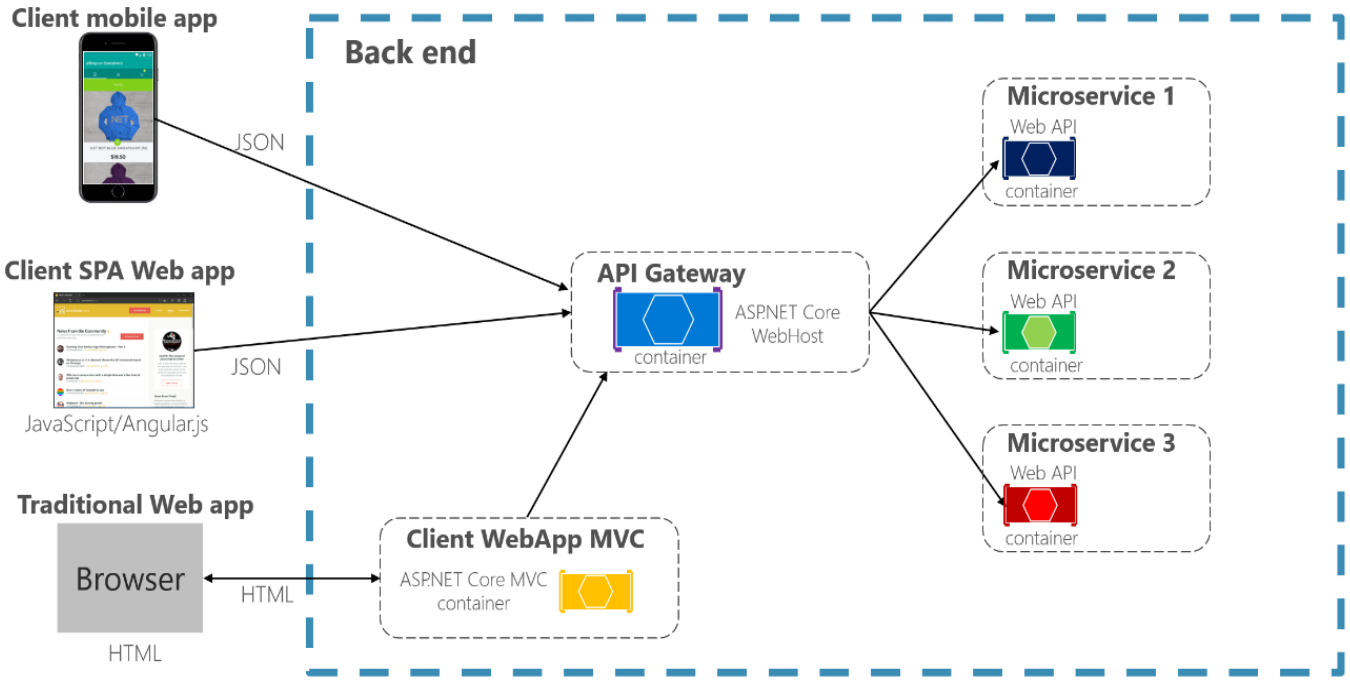
- Route Catalog APIs with /Catalog path
- Route Basket APIs with / Basket path
- Route Ordering APIs with /Ordering path

Method	Request URI	Use Case
GET/POST	/Catalog	Route /api/v1/Catalog apis
GET	/Catalog/{id}	Route /api/v1/Catalog apis
GET/POST	/Basket	Basket /api/v1/Basket apis
POST	/Basket/Checkout	Basket /api/v1/Basket apis
GET	/Order	Order /api/v1/Order apis

Architecture of API Gateway Microservices

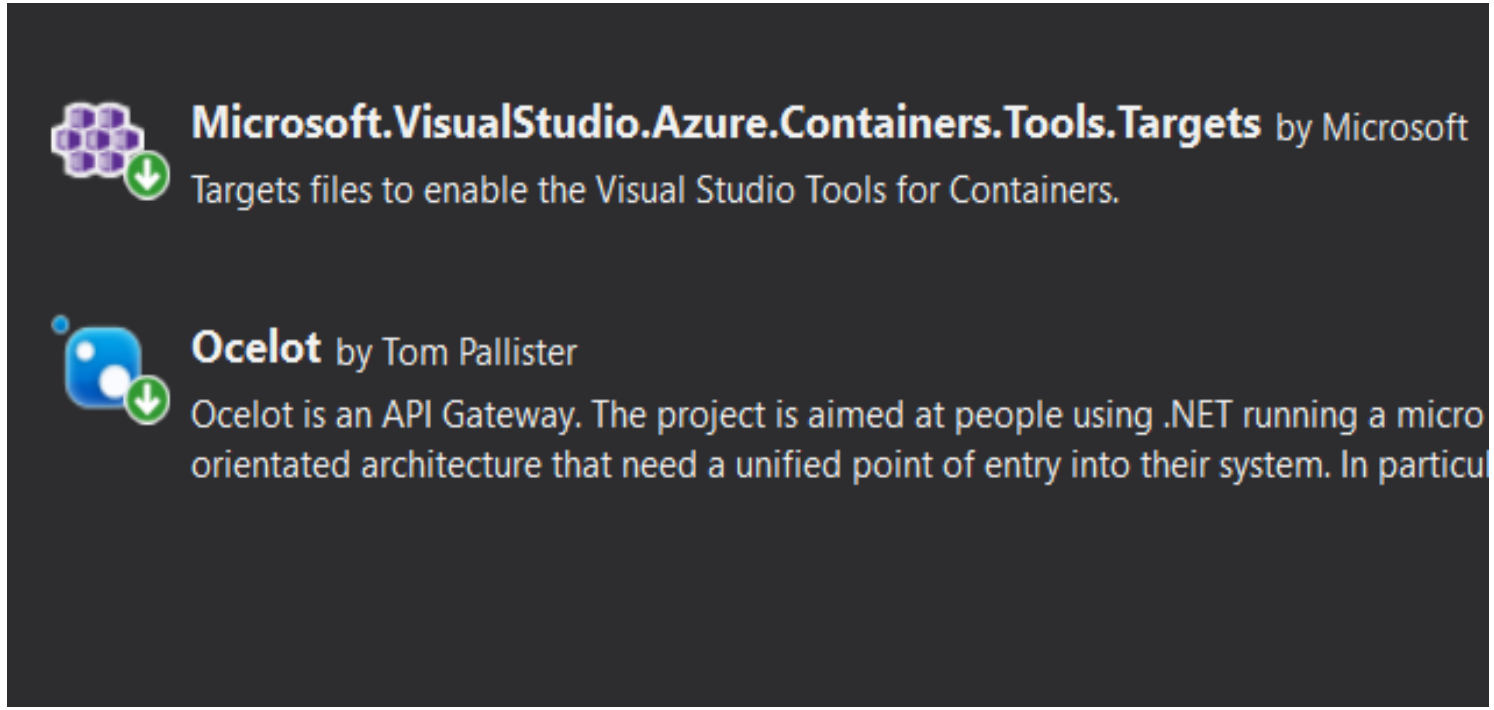
- Blank Web Application
- Ocelot Nuget Package
- Json Configurations

Using a single custom **API Gateway service**



Libraries of API Gateway Microservices

- Ocelot Nuget Package

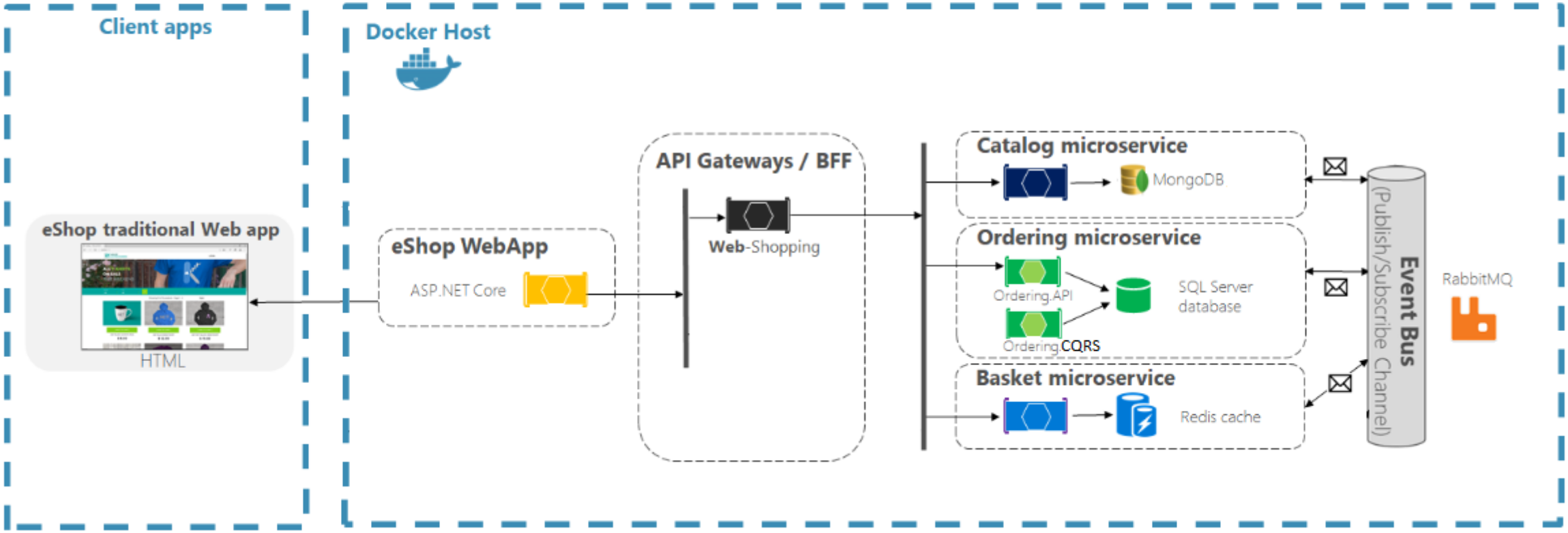


BUILDING SHOPPING WEB APPLICATION MICROSERVİCES

- ASP.NET Core **Web Project** application
- **Consume Ocelot APIs** with **HttpClientFactory**
- **Bootstrap 4 Theme** Implementation
- ASP.NET Core **Razor Tools** — View Components, Partial Views, Tag Helpers, Model Bindings and Validations, Razor Sections etc.
- **Dockerfile** implementation
- **Docker-compose** integration

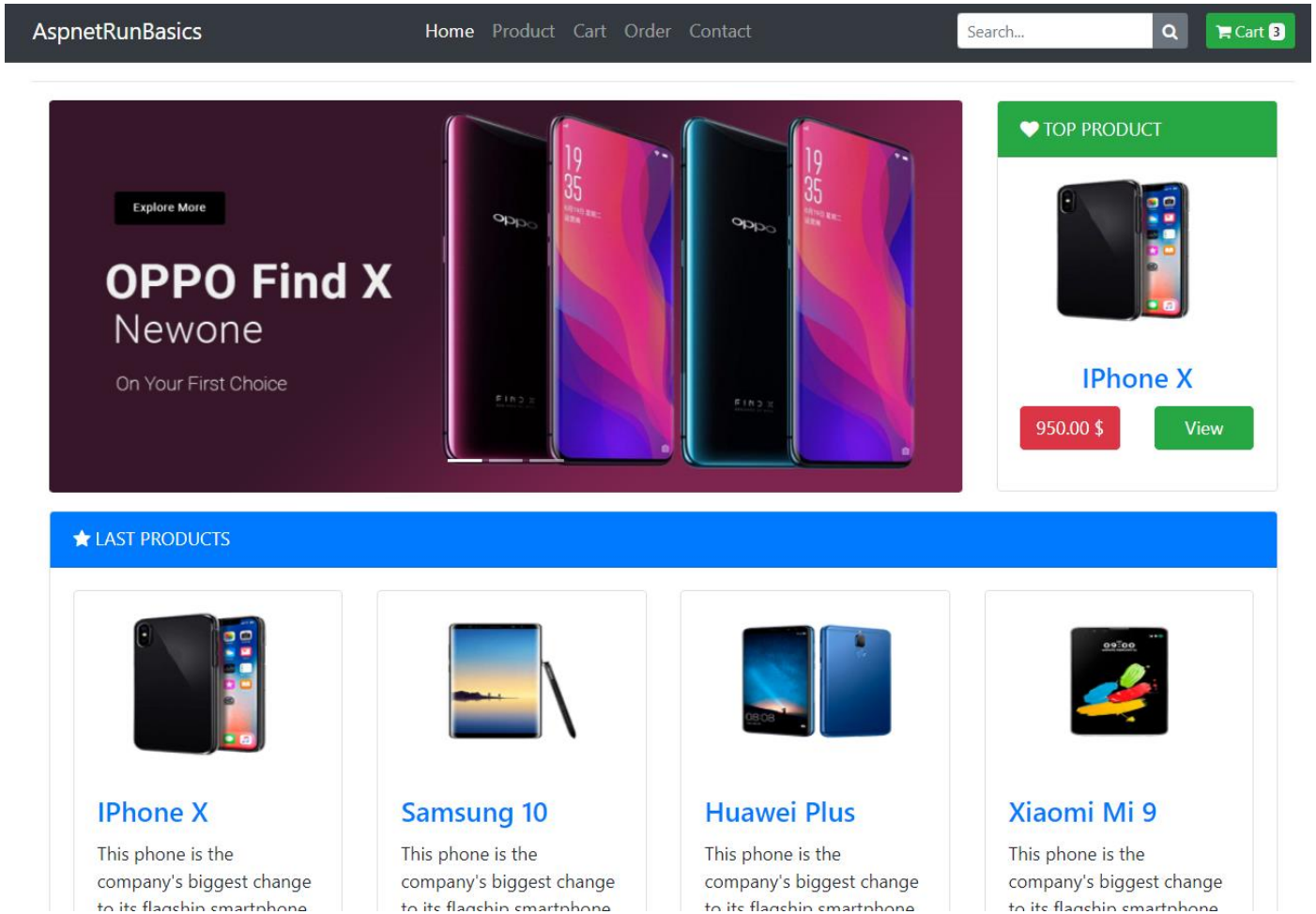
aspnetrun

aspnetrun-microservices Environment Architecture



Shopping Web Application Microservices

- Default Web Application
- Razor Templates
- HttpClientFactory
- Consume Ocelot API Gateway



Base of Shopping Web Application Microservices

- Base Application
- Github Repository of aspnetcore-basics
- Razor Pages
- Bootstrap4

aspnetrun / run-aspnetcore-basics

Unwatch 3 | Unstar 36 | Fork 3

Code | Issues 0 | Pull requests 1 | Actions | Projects 0 | Wiki | Security 0 | Insights | Settings

Implementation of Real-World example in One Solution - One Project for web application development with Asp.Net Core & EF.Core. Only one web application project which used aspnetcore components; razor pages, middlewares, dependency injection, configuration, logging. To create websites with minimum implementation of asp.net core based on HTML5, C...
https://aspnetrun.azurewebsites.net/

Edit

aspnet-core | entity-framework-core | razor-pages | e-commerce | shopping-cart | real-world-project | real-world | aspnet-core-identity | aspnet-core-extensions | aspnetcore-authorization | aspnet-core-authentication | aspnet-core-validation | aspnetcore-basic-authentication | business-rules | configuration | validation | bootstrap4 | starter-kit | starter-template | bootstrap-theme

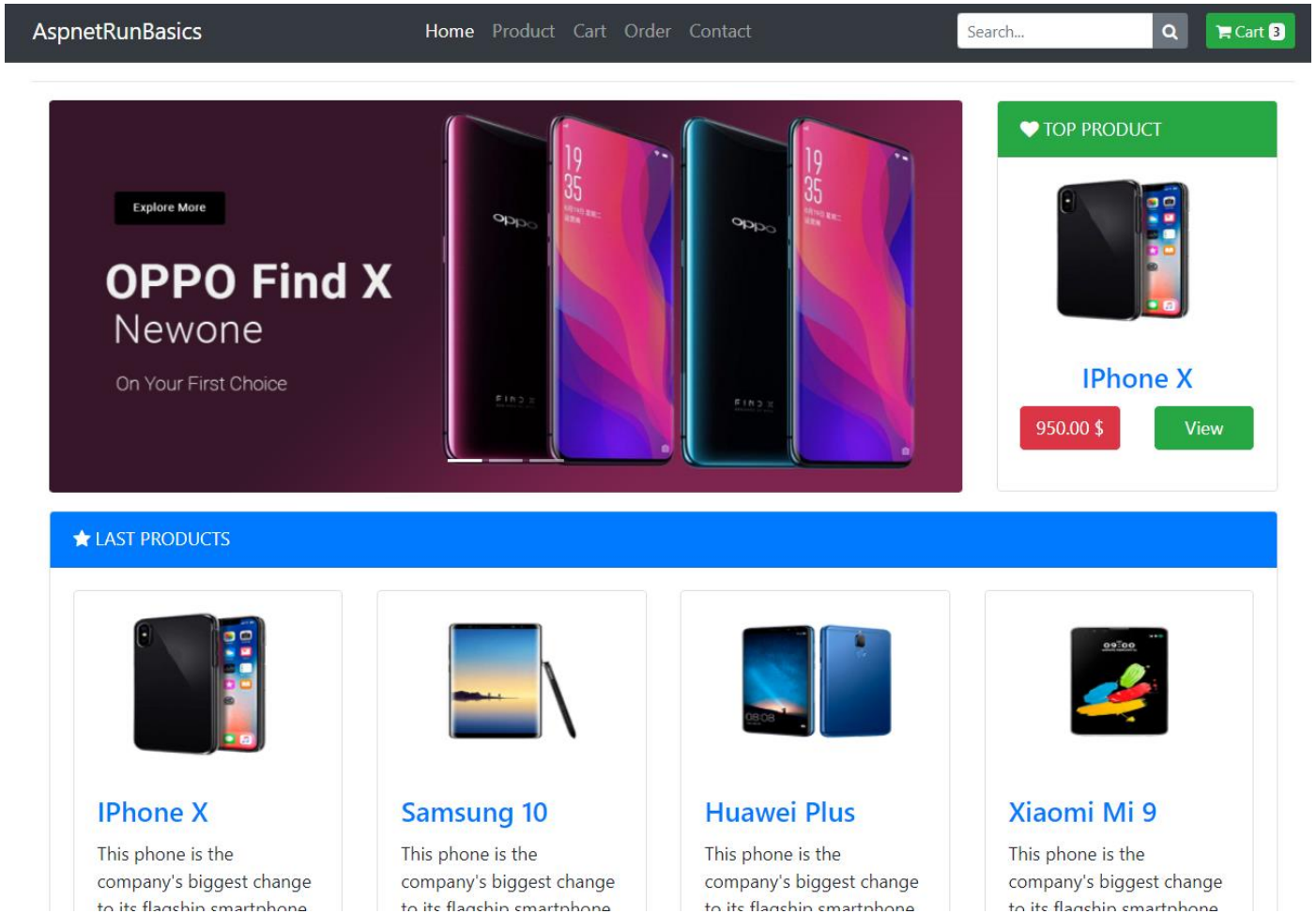
Manage topics

118 commits | 1 branch | 0 packages | 0 releases | 1 contributor | MIT

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

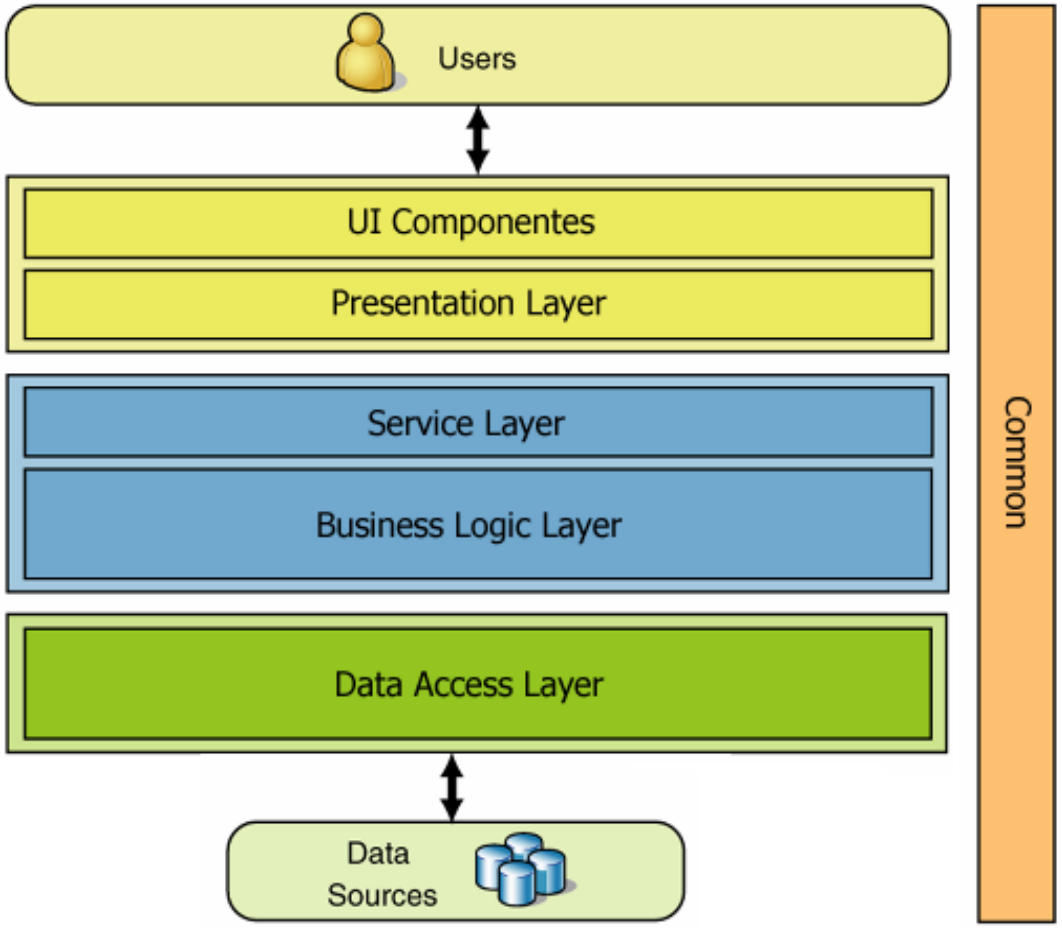
Analysis & Design of Shopping Web App Microservices

- Listing Products and Categories
- Add Product to Shopping Cart
- Checkout Order



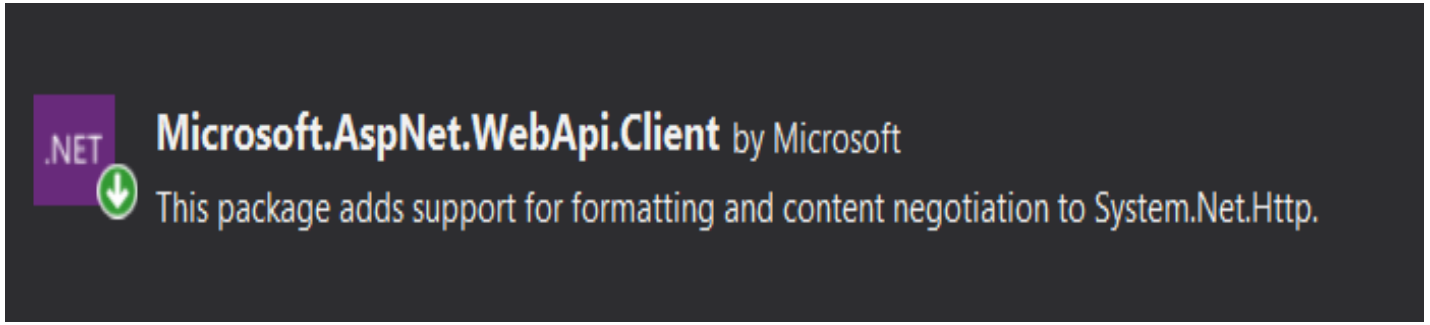
Architecture of Shopping Web App Microservices

- Data Source
- Data Access Layer
- Business Logic Layer
- Presentation Layer
- Common Layer



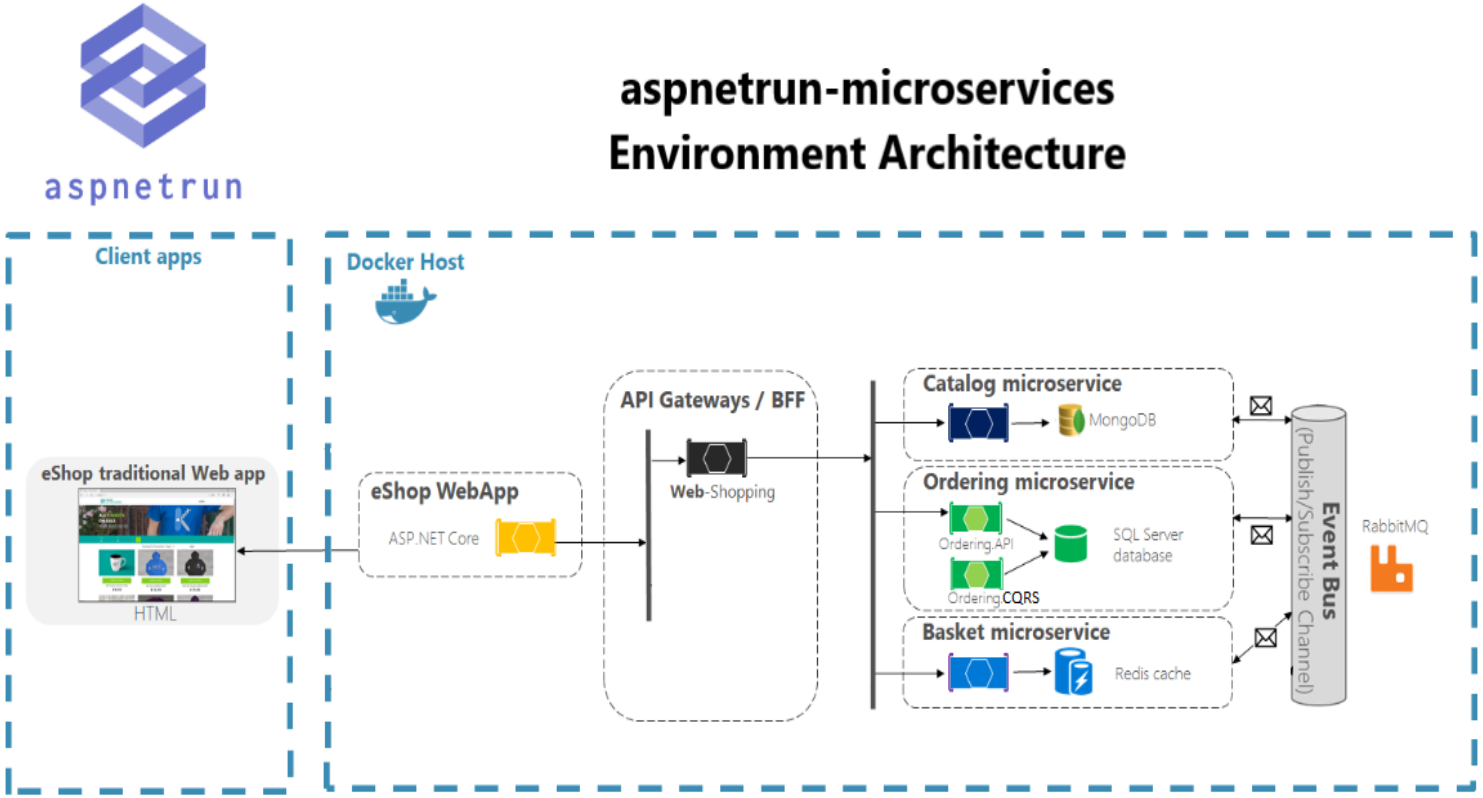
Library of Shopping Web Application Microservices

- WebApi.Client
- HTTPClientFactory



Docker Operations for Shopping Web App Microservices

- Dockerfile creation
- Dockerfile commands
- Docker-compose file creation
- Docker-compose file commands



THANKS

- **Open issue** on Github repository
- **Give a Star** on Github repository - https://github.com/aspnetrun/run-aspnetcore-microservices
- Repository and Course **evolving** according to new enhancements
- Follow me on **medium -** https://medium.com/aspnetrun
- Follow me on **twitter -** https://twitter.com/ezozkme
- Send mail for **the microservices book – ezozkme@gmail.com**