

# Section 1

## Understanding Microservices

Instructor: Phạm Quang Anh Kiệt

@email: [kietpham.dev@gmail.com](mailto:kietpham.dev@gmail.com)

@facebook/rickykiet83



# Introduction

- ▶ Microservices is a hot trend in the technology section
- ▶ Netflix, Google, Twitter have been used microservices-based architecture
- ▶ It can be extremely daunting to start, however, for the larger enterprise, each modules can be developed with their own history and purpose

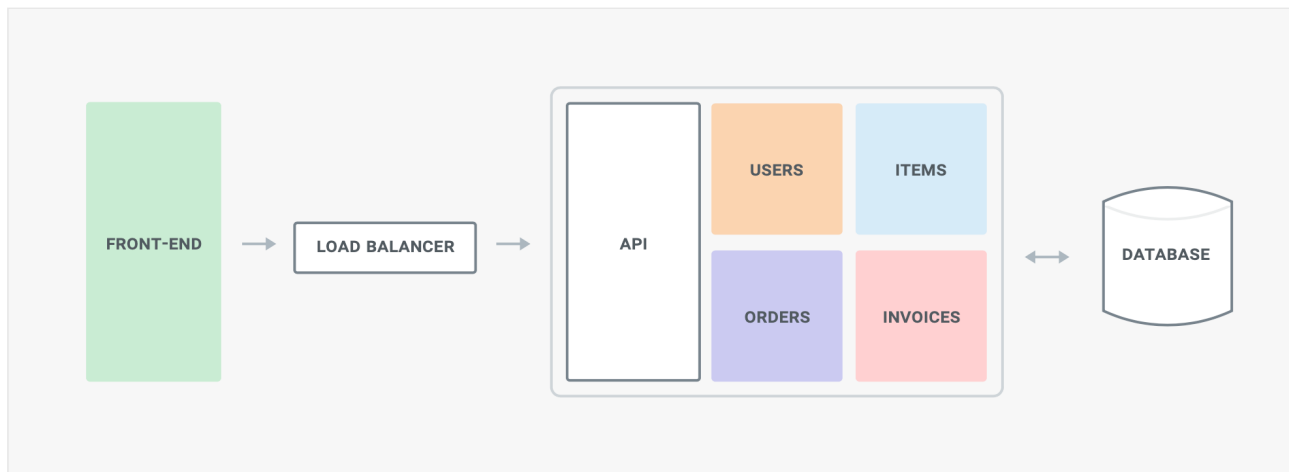
# Advantages of Microservices

1. **Agility:** Componentization and distributed functionality empower developers to iterate and deploy continuously, autonomous of other business units and application teams.
2. **Freedom of Options:** Developers can independently pick their preferred framework (language, structure) to construct and convey functionality more rapidly.
3. **Resiliency:** Microservices are designed for failure with redundancy and isolation in mind, which in turn makes applications more robust.
4. **Efficiency:** There can be significant savings for the enterprise that decouples functionality and adopts microservices.

# Monolithic vs Microservices

## ► Monolithic:

- Easy to understand
- It's great when the codebase and the team working on it are both relatively small
- A fast way to develop a product and get it into market quickly
- No other dependencies.



# Microservices

- ▶ Able to be **built independently**
- ▶ Able to be **deployed independently**
- ▶ Implementation detail will be taken care by the specific team working on that specific feature.
- ▶ Implementations of other components (services) work with interfaces, or APIs.
- ▶ One “big” specific thing tend to become much smaller => “microservices”

# Microservices

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



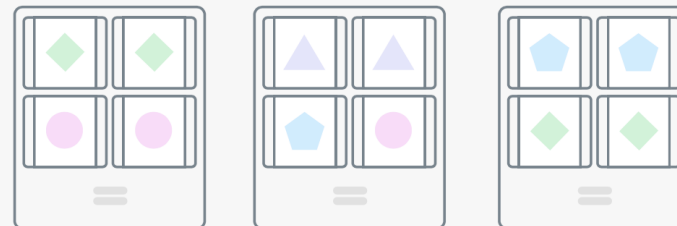
... and scales by replicating the monolith on multiple servers.



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



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



# Microservices Pros and Cons

## ► Pros:

- Better architecture for large applications
- Better agility in the long term
- Easy to learn
- Isolation for scalability and damage control

## ► Cons:

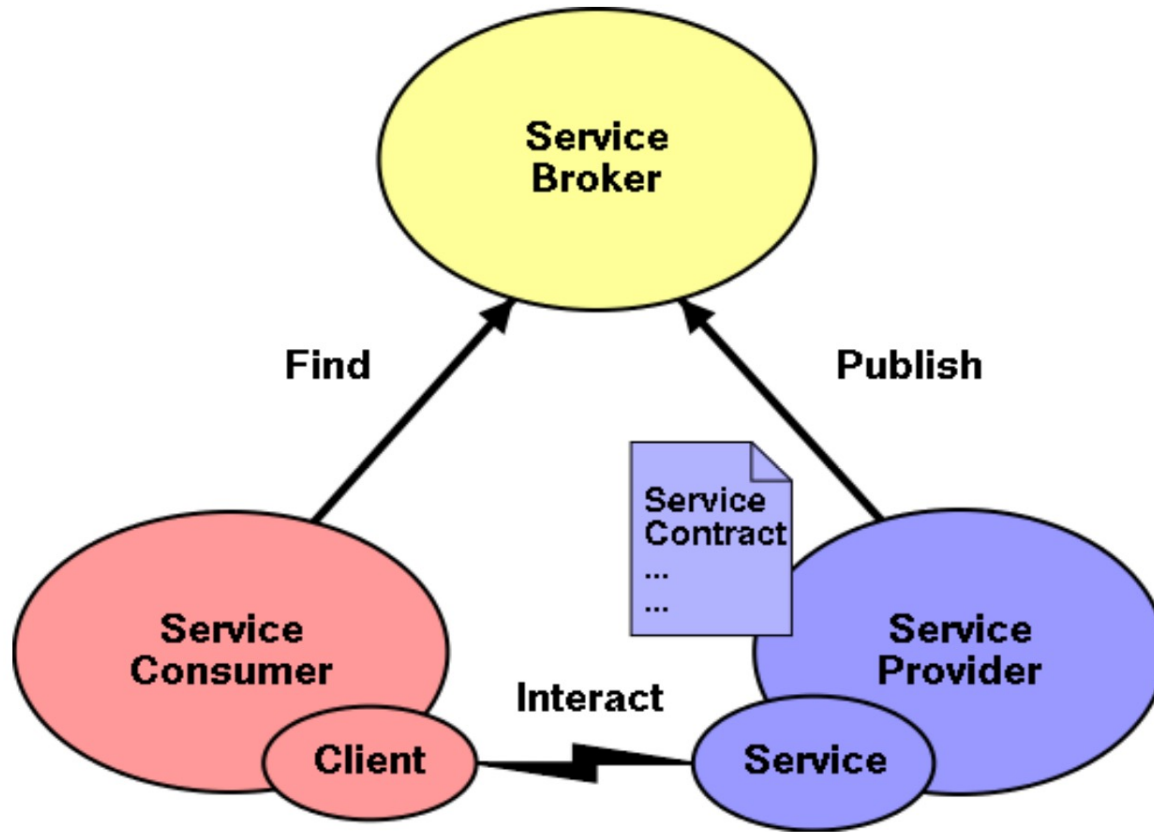
- More moving parts
- Complex infrastructure requirements
- Consistency and availability
- Harder to test

# Service-oriented architecture (SOA)

- ▶ *“Service-oriented architecture (SOA) is a type of software design that makes software components reusable using service interfaces that use a common communication language over a network.”*
- ▶ In briefly, SOA integrates software components that have been separately deployed and maintained and allows them to communicate and work together to form software applications across different systems.



# Service-oriented architecture (SOA)



# Microservices architecture principles

## 1. A microservices has a single concern.

- ▶ Should do one thing and one thing only = Single object responsibility
- ▶ Easier to maintain and scale

## 2. A microservice is a discrete

- ▶ Must clear boundaries separating it from its environment.
- ▶ Must be well-encapsulated
- ▶ Development: Isolated from all other microservices
- ▶ Production: It becomes part of a larger application after deployment

# Microservices architecture principles

## 3. A microservices is transportable.

- ▶ Can be moved from one runtime environment to another
- ▶ Easier to use in an automated or declarative deployment process.

## 4. A microservice carries its own data

- ▶ Should have its own data storage that is isolated from all other microservices.
- ▶ Shared with other microservices by a public interface
- ▶ The common problem is data redundancy.

# Microservices architecture principles

## 5. A microservice is ephemeral

- ▶ It can be created, destroyed, and replenished on demand
- ▶ The standard operating expectation is that microservices come and go all the time, sometimes due to system failure and sometimes due to scaling demands.

# Microservice communication

## 1. Synchronous protocol

- ▶ HTTP/HTTPS
- ▶ The client sends a request and waits for a response from the service
- ▶ Thread is blocked
- ▶ The client code can only continue its task when it receives the HTTP server response.

## 2. Asynchronous protocol

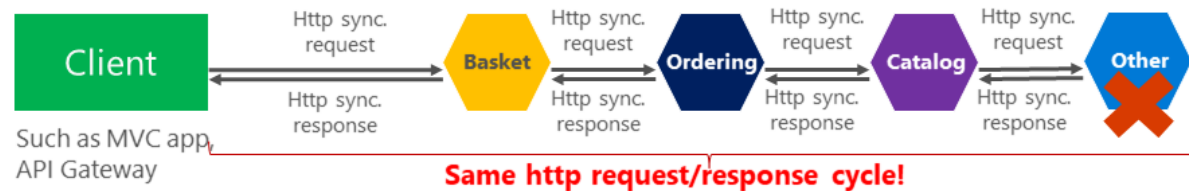
- ▶ AMQP (a protocol supported by many OS and cloud environments)
- ▶ Asynchronous messages
- ▶ The client send message and doesn't wait for a response.
- ▶ RabbitMQ or Kafka is a message queue

# Microservice communication

## Synchronous vs. async communication across microservices

### Anti-pattern

**Synchronous**  
all request/response cycle



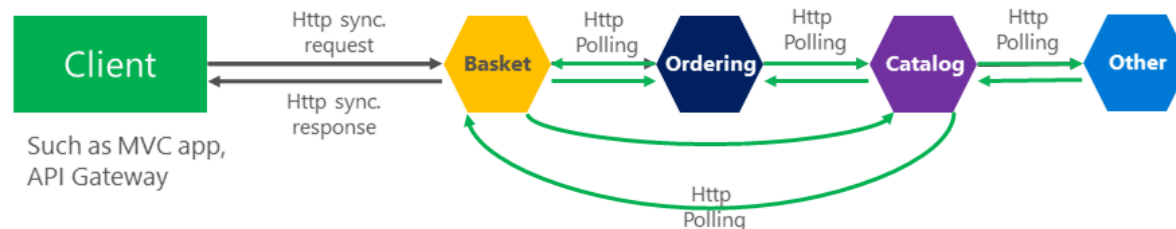
### Asynchronous

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

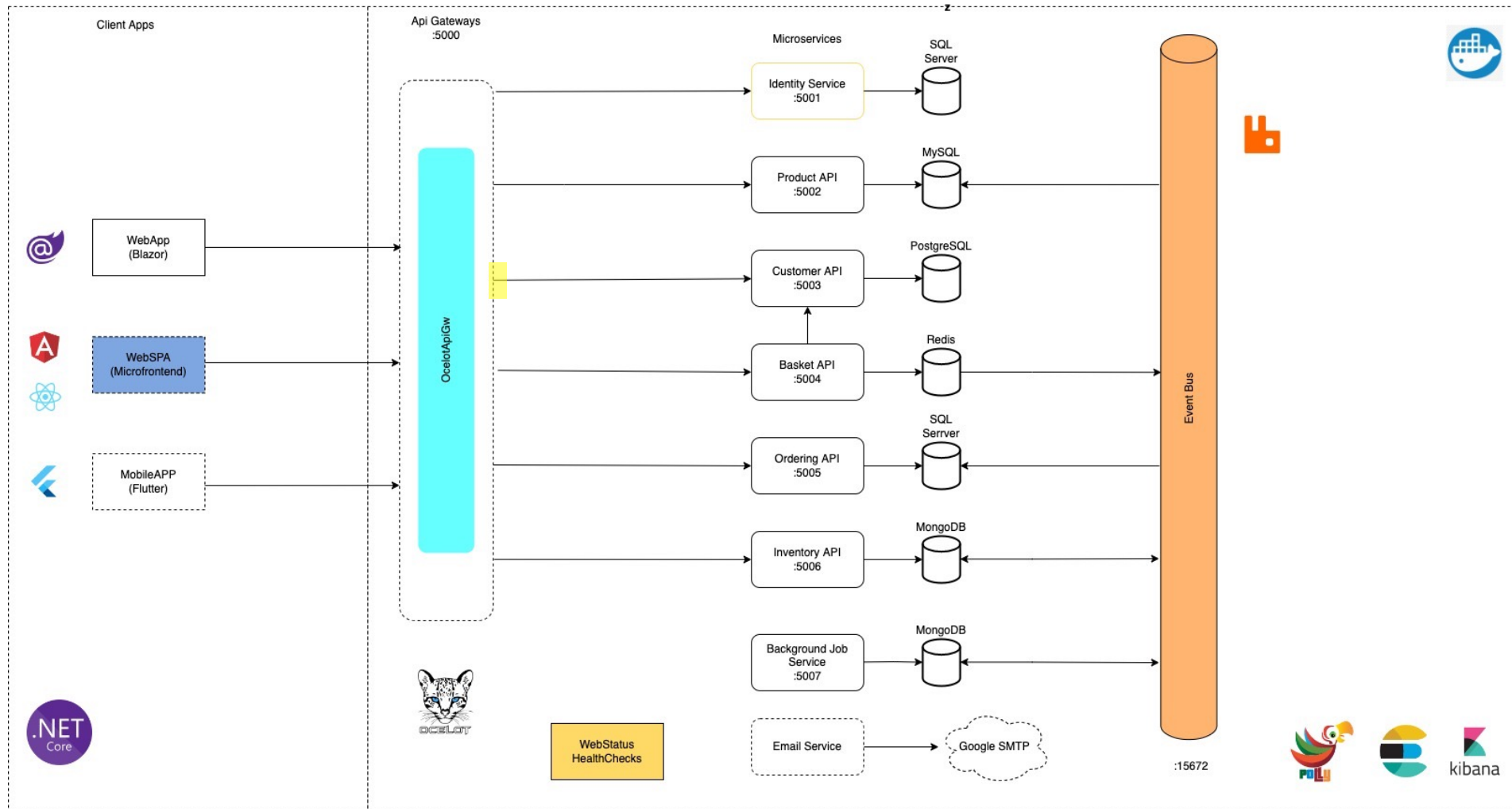


### "Asynchronous"

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



# Tedu aspnetcore Microservices project



aspnetcore-microservices.sln

Solution

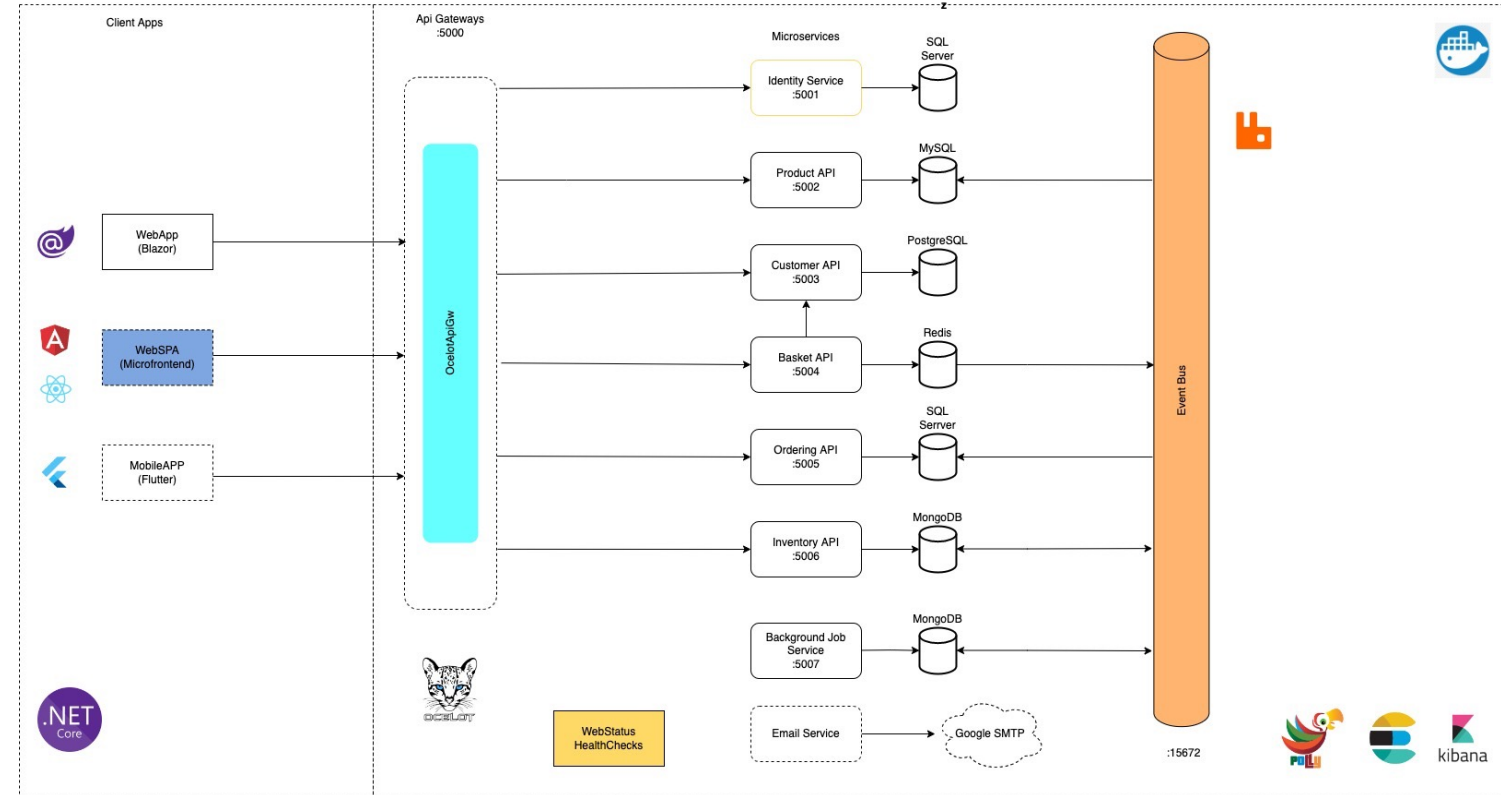
aspnetcore-microservices · 16 projects

Solution Items

- .dockerignore
- docker-compose.yml
- docker-compose.override.yml
- global.json
- README.md
- Tedu-Microservices-Solution\_Architect.jpg

src · 16 projects

- ApiGateways · 1 project
  - OcelotApiGw
- BuildingBlocks · 5 projects
  - Common.Logging
  - Contracts
  - EventBus.Messages
  - Infrastructure
  - Shared
- Services · 9 projects
  - Basket · 1 project
  - Customer · 1 project
  - Inventory · 1 project
  - Ordering · 4 projects
  - Product · 1 project
  - ScheduledJob · 1 project
- WebApps · 1 project





# Solution exploration

- ▶ **Building Blocks:** Including class libraries which defines interfaces, contracts, shared and common methods.
  - ▶ **Common.Logging:** Logging system with Serilog and elasticsearch.
  - ▶ **Contracts:** The blue print of the system, where we can define the common interfaces as: Repository, UnitOfWork... to define our contracts for the whole system.
  - ▶ **EventBus.Message:** Event Bus Message system, AMQP, standardize communication across microservices.
  - ▶ **Infrastructure:** Class library implements from Contracts interface.
  - ▶ **Shared:** Sharing resources, common variables, configurations across microservices.

# Solution exploration

- ▶ Services: Including the microservices of the system.
  - ▶ Basket: Basket API with Redis
  - ▶ Customer: Customer Minimal API with PostgreSQL
  - ▶ Ordering: Ordering API with Clean Architecture and SQL Server
  - ▶ Product: Product API with MySQL
  - ▶ Inventory: Inventory API with MongoDB
  - ▶ ScheduledJob: Hangfire API with MongoDB, background tasks

# Solution exploration

- ▶ WebApps:
  - ▶ WebHealthStatus MVC, presentation health check system.
  - ▶ Microfrontend Client App (not included in this course)

# Section 2

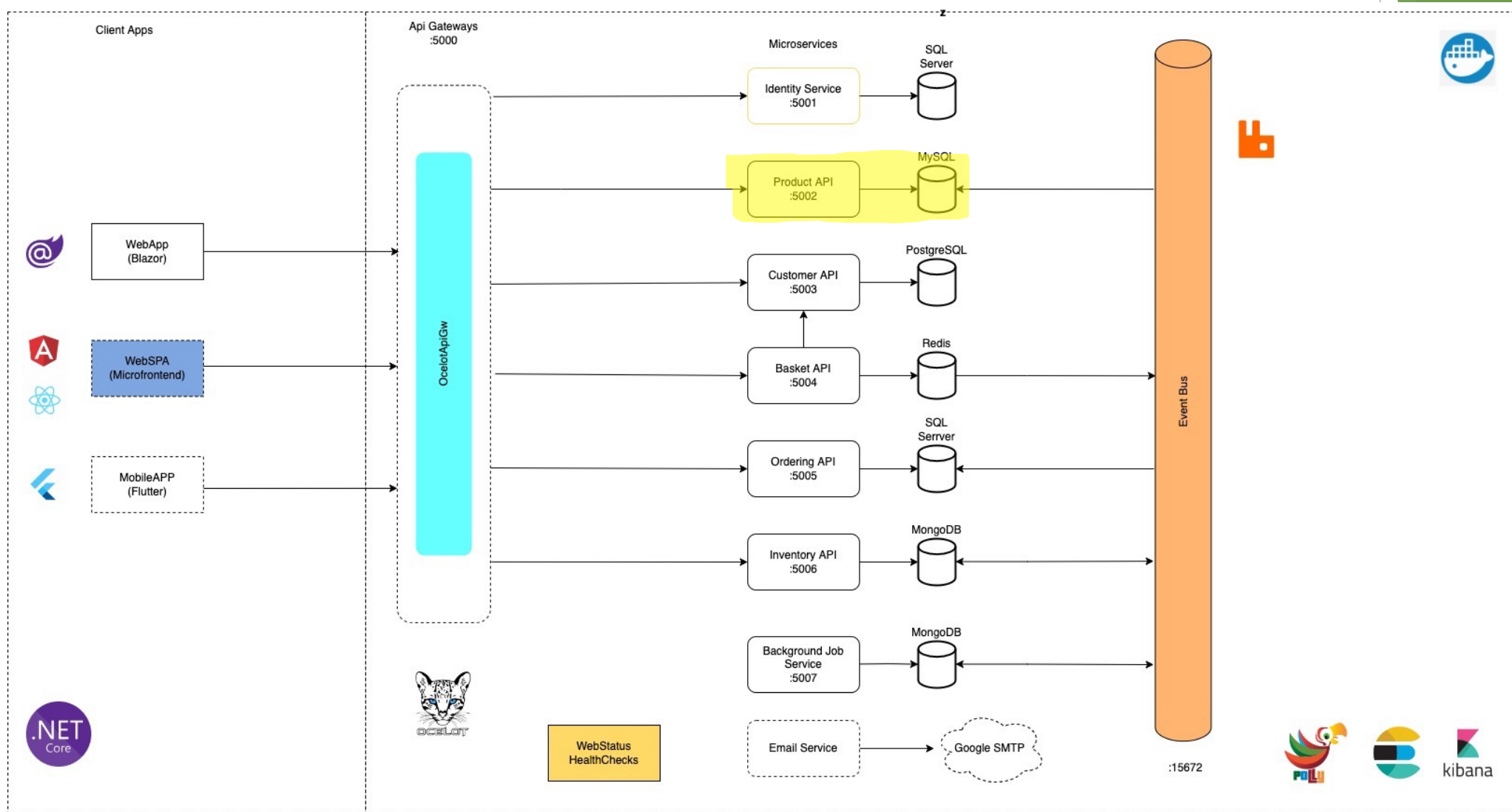
## Product API with MySQL

Instructor: Phạm Quang Anh Kiệt

@email: [kietpham.dev@gmail.com](mailto:kietpham.dev@gmail.com)

@facebook/rickykiet83





# Section 3

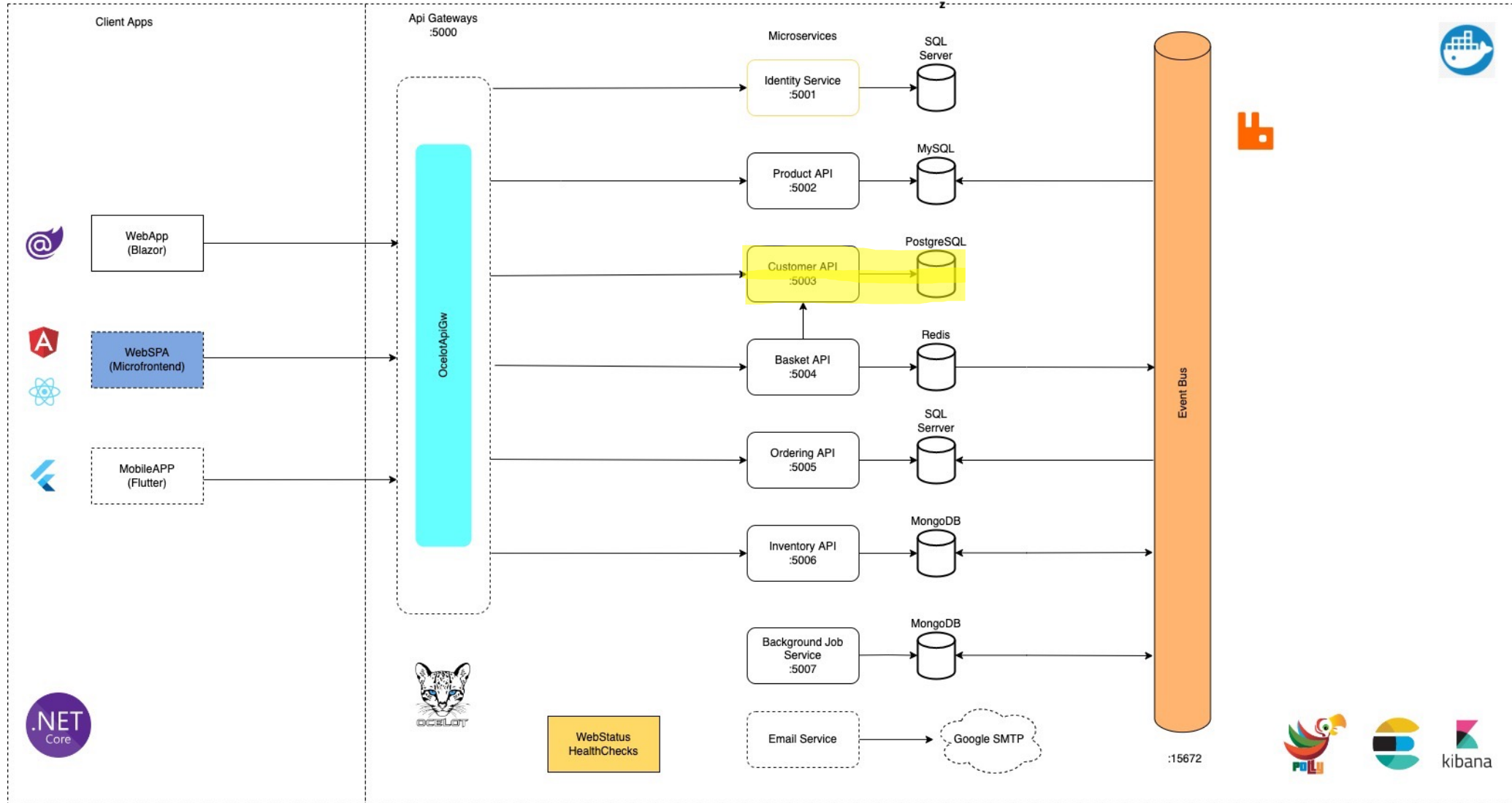
## Customer API with Minimal API & PostgreSQL

Instructor: Phạm Quang Anh Kiệt

@email: [kietpham.dev@gmail.com](mailto:kietpham.dev@gmail.com)

@facebook/rickykiet83





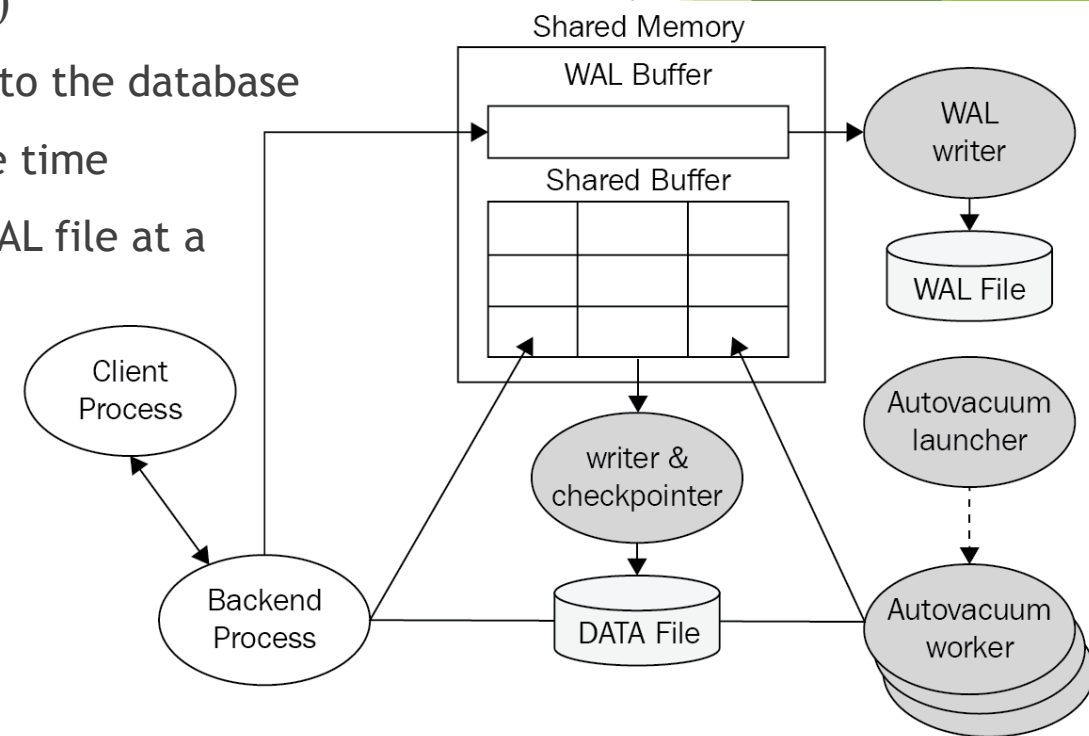
# An overview of PostgreSQL

- ▶ Released in 1994
- ▶ PostgreSQL is an **object-relational database management system (ORDBMS)**
- ▶ A popular **Database as a Service (DBaaS)**
- ▶ PostgreSQL can be delivered as DBaaS on many clouds, such as **Amazon Web Services (AWS)**, Google Cloud SQL, Microsoft Azure, Heroku, and EnterpriseDB Cloud.
- ▶ Open source **license is free**, so developers can easily operate as many databases as they wish without any cost.



# The PostgreSQL architecture

- ▶ Shared memory:
  - ▶ Minimize DISK I/O
  - ▶ Access very large buffers (tens or hundreds of gigabytes) worth) quickly.
  - ▶ The reduction of **write-ahead log (WAL)** (Nhật ký ghi trước)
  - ▶ The WAL buffer is a buffer that temporarily stores changes to the database
  - ▶ Minimize contention when many users access it at the same time
  - ▶ The contents stored in the WAL buffer are written to the WAL file at a predetermined point in time.



# Standout Features:

- ▶ Complex query
- ▶ Trigger
- ▶ View
- ▶ Integrity transactions
- ▶ Multi-version concurrency control (Kiểm tra truy cập đồng thời đa phiên bản)
- ▶ Parallel query
- ▶ Types: JSON/JSONB, XML, Key-Value
- ▶ Point-in-time-recovery - PITR)
- ▶ Authentication: GSSAPI, SSPI, LDAP, SCRAM-SHA-256, Certificate
- ▶ Columns/Rows security
- ▶ Index: B-tree, Multicolumn, Expression, Partial
- ▶ Advanced Index: GiST, SP-Gist, KNN Gist, GIN, BRIN, Bloom filters

# PgAdmin4

- ▶ <https://www.pgadmin.org/>
- ▶ Download desktop version at: <https://www.pgadmin.org/download/>
- ▶ Or using docker at: <http://localhost:5050> (docker-compose-override.yml file)

```
pgadmin:
  container_name: pgadmin
  environment:
    - PGADMIN_DEFAULT_EMAIL=admin@tedu.com.vn
    - PGADMIN_DEFAULT_PASSWORD=admin1234
  restart: always
  ports:
    - "5050:80"
  volumes:
    - pgadmin_data:/root/.pgadmin
```

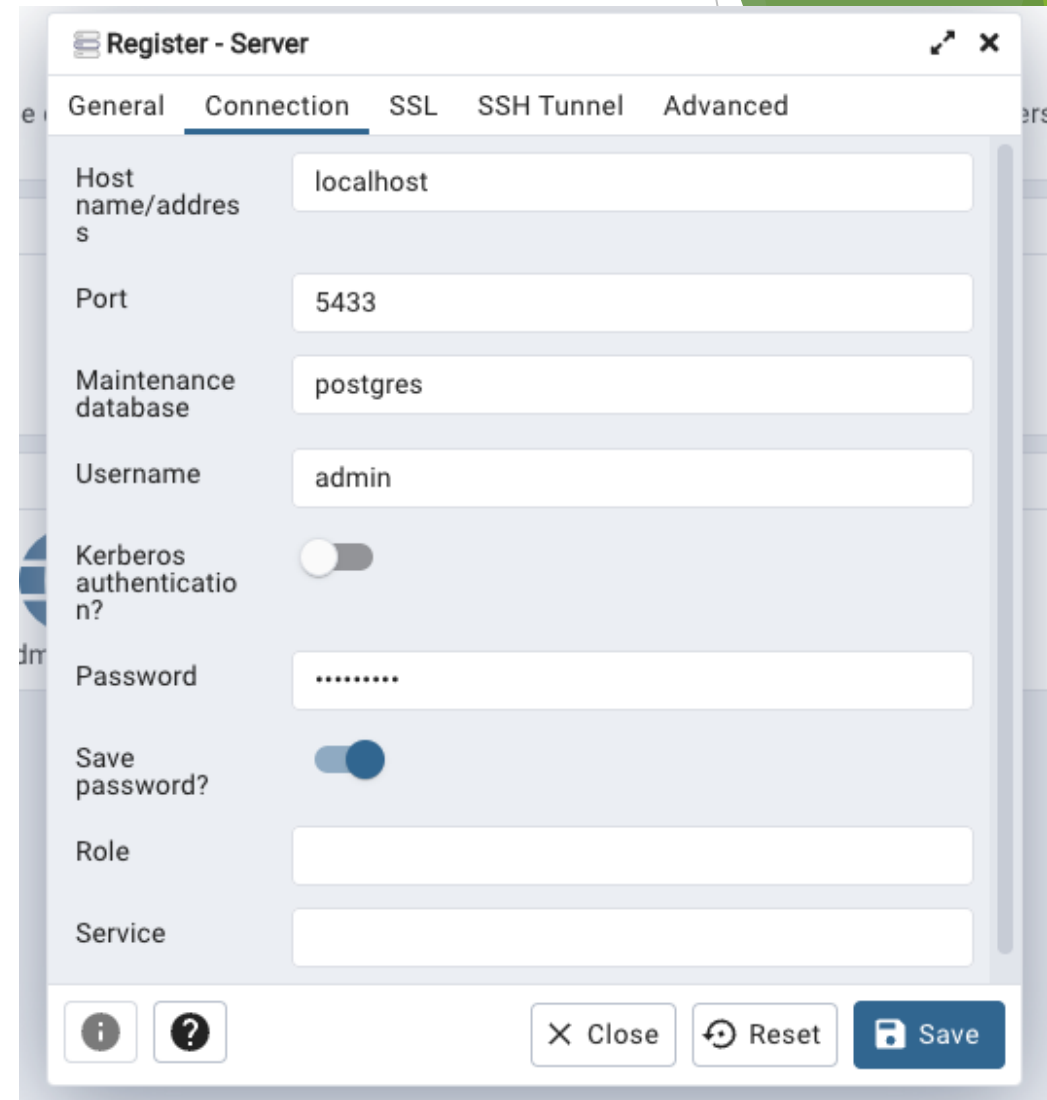
# PgAdmin4

- Add new Server

The screenshot shows the 'Register - Server' dialog box in PgAdmin4. The 'General' tab is selected, and the 'Name' field contains 'Tedu-Microservices'. The 'Server group' is set to 'Servers'. The 'Background' and 'Foreground' checkboxes are unchecked. The 'Connect now?' toggle is turned on, and the 'Shared?' toggle is turned off. The 'Comments' field is empty. A red error message at the bottom states: 'Either Host name, Address or Service must be specified.' The dialog box has buttons for 'Close', 'Reset', and 'Save' at the bottom right.

# PgAdmin4

- ▶ Connection (docker-compose.override.yml)
- ▶ Username: admin
- ▶ Password: admin1234



The image shows the 'Register - Server' dialog box in PgAdmin4, with the 'Connection' tab selected. The dialog contains the following fields and controls:

- Host name/address:** localhost
- Port:** 5433
- Maintenance database:** postgres
- Username:** admin
- Kerberos authentication?:** ☐
- Password:** ..... (masked)
- Save password?:** ☒
- Role:** (empty field)
- Service:** (empty field)

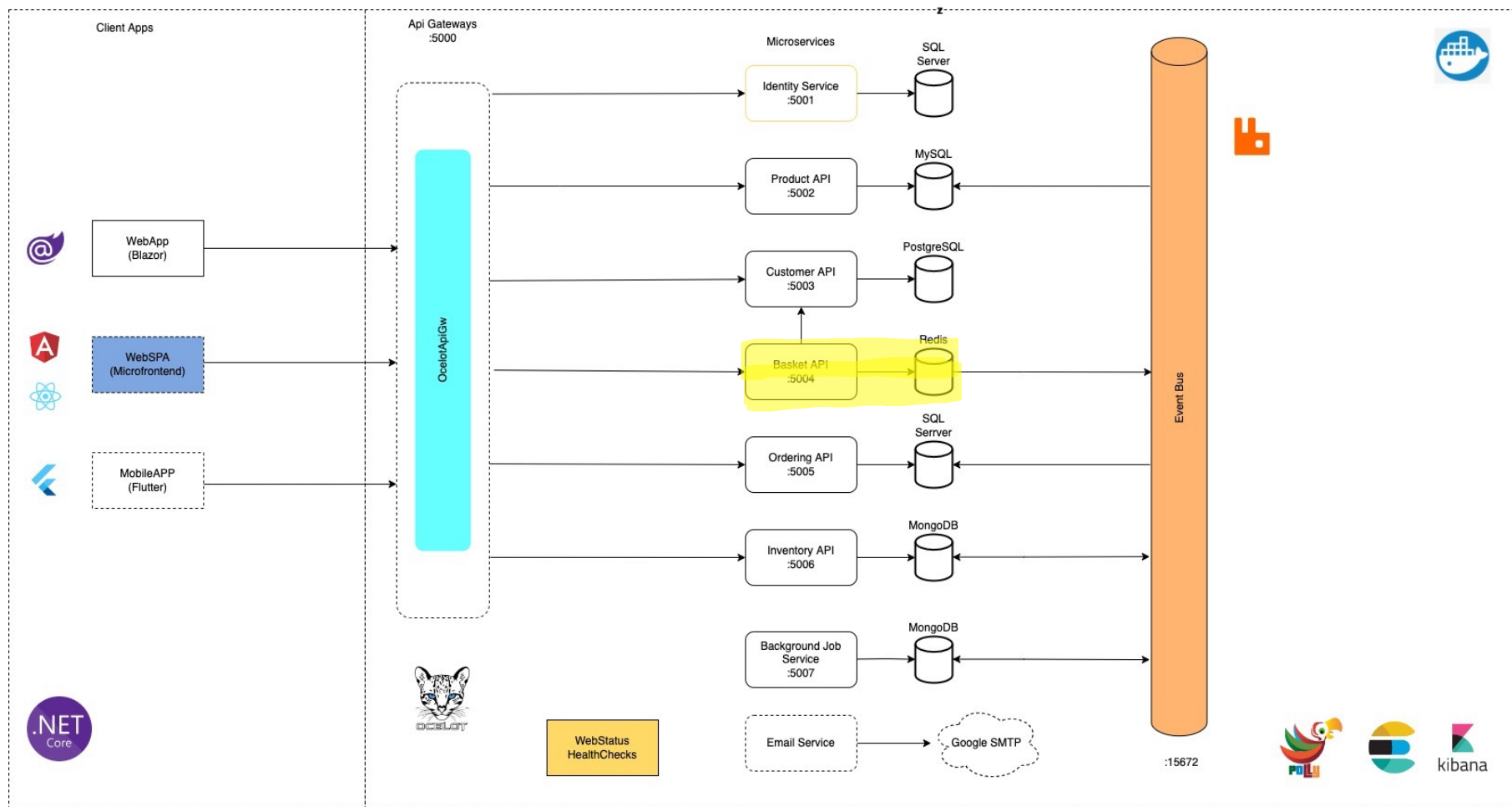
At the bottom of the dialog, there are three buttons: 'Close' (with an 'X' icon), 'Reset' (with a circular arrow icon), and 'Save' (with a floppy disk icon). There are also information and help icons on the left.

# Section 4

## Basket API with Redis



Instructor: Phạm Quang Anh Kiệt  
@email: [kietpham.dev@gmail.com](mailto:kietpham.dev@gmail.com)  
@facebook/rickykiet83



# An overview of Redis

- ▶ Redis' name comes from “**RE**mote **D**ictionary **S**erver”.
- ▶ a fast in-memory database and cache
- ▶ Open source
- ▶ Redis is often called a data structure server because its core data types are similar to those found in programming languages like: strings, lists, dictionaries (or hashes), sets, and sorted sets.
- ▶ Ideal for rapid development and fast applications, as core data structures are easily shared between processes and services.



# Primarily Use cases

## ▶ **Intelligent Caching**

- ▶ Redis is commonly used as a cache to store frequently accessed data in memory so that applications can be responsive to users.
- ▶ You can easily to configure: how long you want to keep data, and which data to evict first,

## ▶ **Data Expiration and Eviction Policies**

- ▶ Data structures in Redis can be marked with a Time To Live (TTL) set in seconds, after which they will be removed.

## ▶ **Publication and Subscription Messaging (Pub/Sub)**

- ▶ Pub/Sub messaging allows for messages to be passed to channels and for all subscribers to that channel to receive that message.

# Primarily Use cases

## ▶ Session Management

- ▶ Session state is data that captures the current status of user interaction with applications such as a website or a game.
- ▶ Session state is how apps remember user identity, login credentials, personalization information, recent actions, shopping cart, and more.
- ▶ The session state is cached data for a specific user or application that allows fast response to user actions.
- ▶ While the session is live, the application reads from and writes to the in-memory session store exclusively.

# Exploring Redis with the CLI

- ▶ Start redis server: “**redis-server**”
- ▶ Start redis cli: “**redis-cli**”
- ▶ check redis with ping command: “ping”
- ▶ Set key: “**SET** {key} {value}”
- ▶ Get key: “**GET** {key}”
- ▶ Increase value of key to 1: “**INCR** {key}”
- ▶ Decrease value of key to 1: “**DECR** {key}”
- ▶ Create a list: “**LPUSH** {key} {value1 value2 ...}”
- ▶ Retrieve the value of list index: “**LINDEX** {key} {index}”
- ▶ Retrieve the range of items: “**LRANGE** {key} 0 -1” (The -1 means ”to the end of the list”)