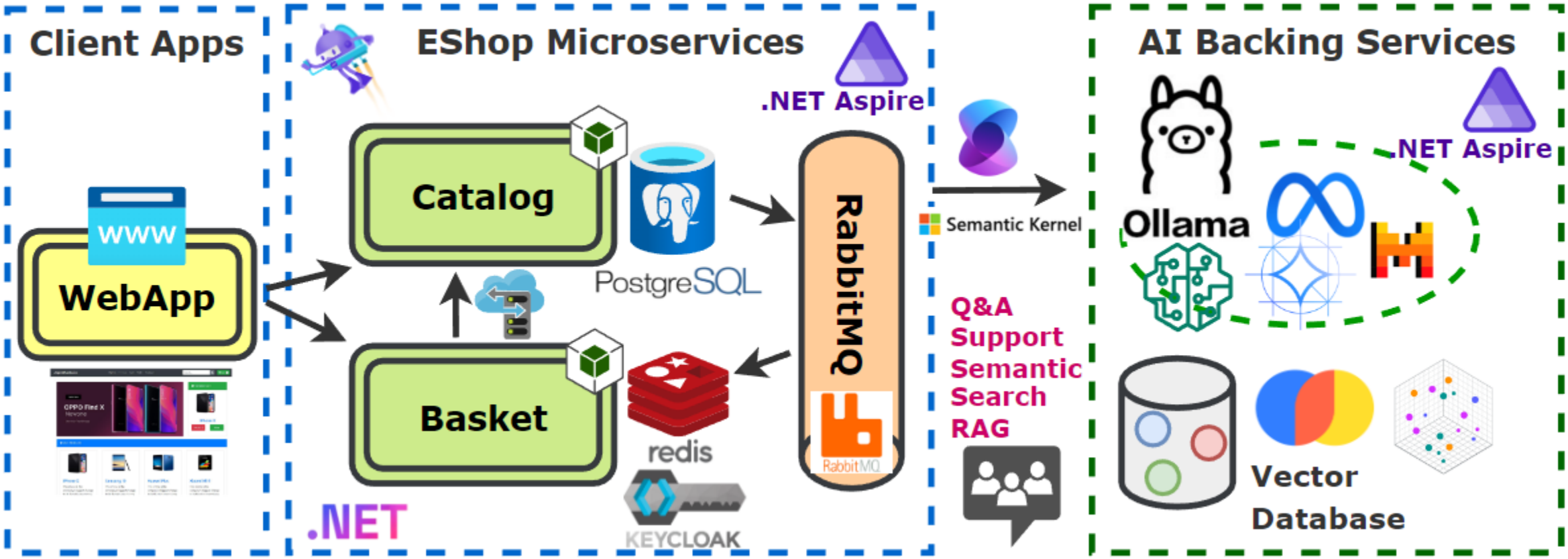
.NET Aspire and GenAI Develop Distributed Architectures

Develop EShop Catalog and Basket microservices integrate with PostgreSQL, Redis, RabbitMQ, Keycloak, Ollama and Semantic Kernel to Create Intelligent E-Shop Solutions

Mehmet Ozkaya

.NET Distributed Architectures w/ Aspire & GenAI



.NET Aspire & GenAI Distributed Architectures

Cloud-Native Distributed Architecture

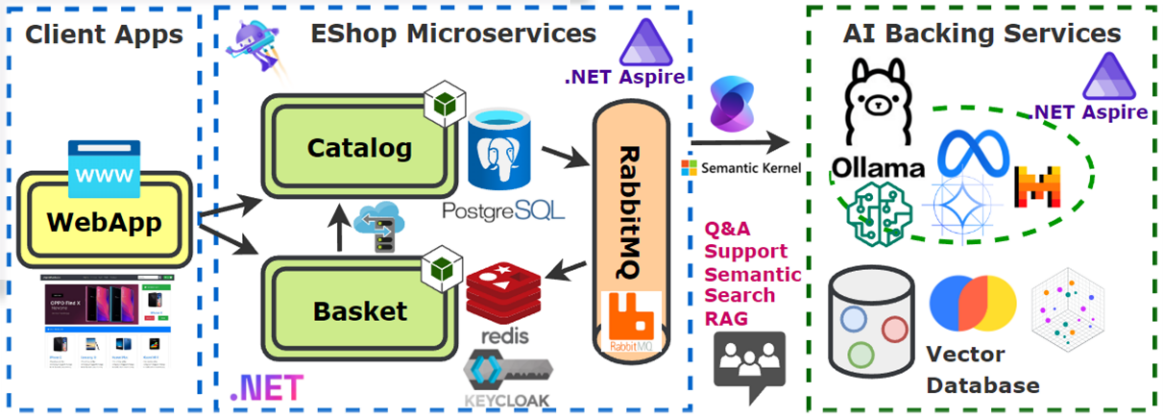
.NET Aspire Framework

E-Shop Microservices

Deploy to Azure Container Apps

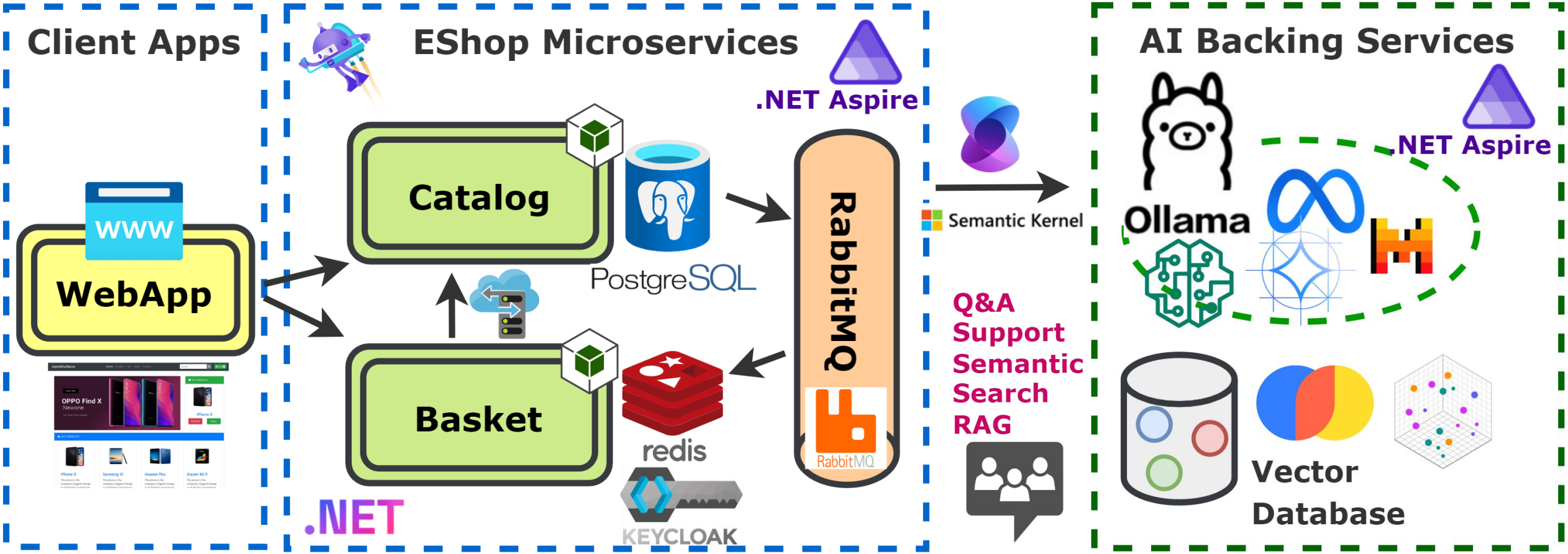
.NET GenAI Development

Use Cases: Customer Support Q&A using Ollama
Product Semantic Search leveraging a VectorDB



Course Project: EShop Microservices with AI-Powered Capabilities using LLMs

Azure Container Apps



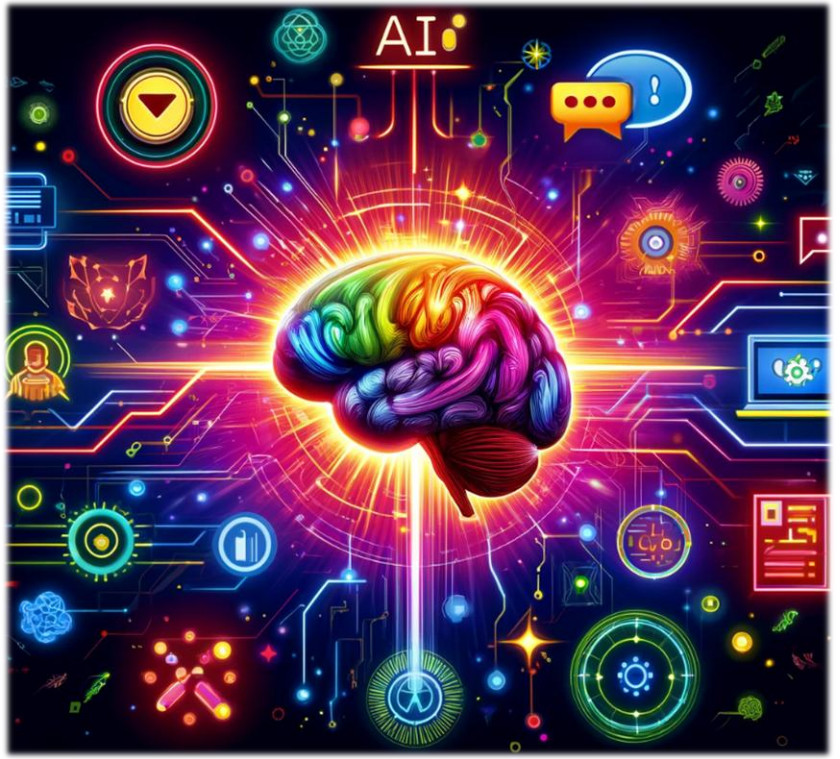
Why Learn AI-Powered Distributed Architectures ?

Growing demand for scalable, intelligent applications

Modern apps aren’t just about serving static data

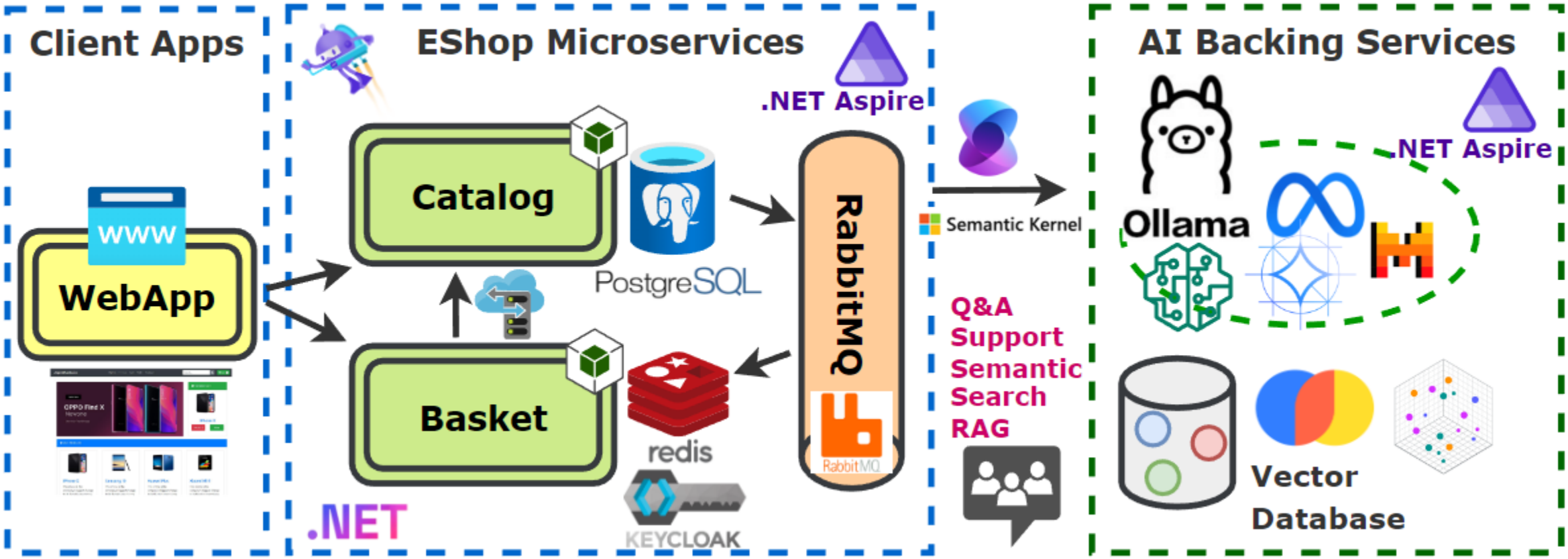
AI is transforming how we interact with software

Combining the power of microservices with AI



The course is very practical, about 95%+ of the lessons will involve you coding

.NET Distributed Architectures w/ Aspire & GenAI

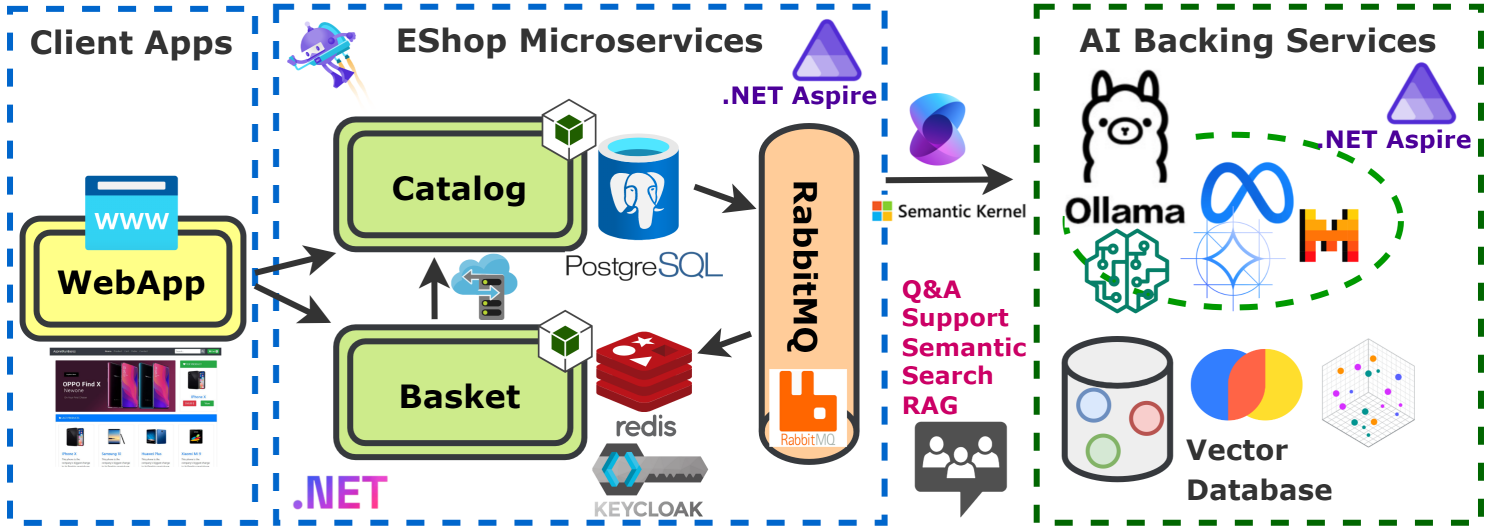


Prerequisites – What You Should Know

[.NET 9 (Includes .NET Aspire by default)]

[Visual Studio 2022 or Later]

[Docker Desktop]



Slide Source Code on GitHub

All the code written and demonstrated in this course is available on GitHub

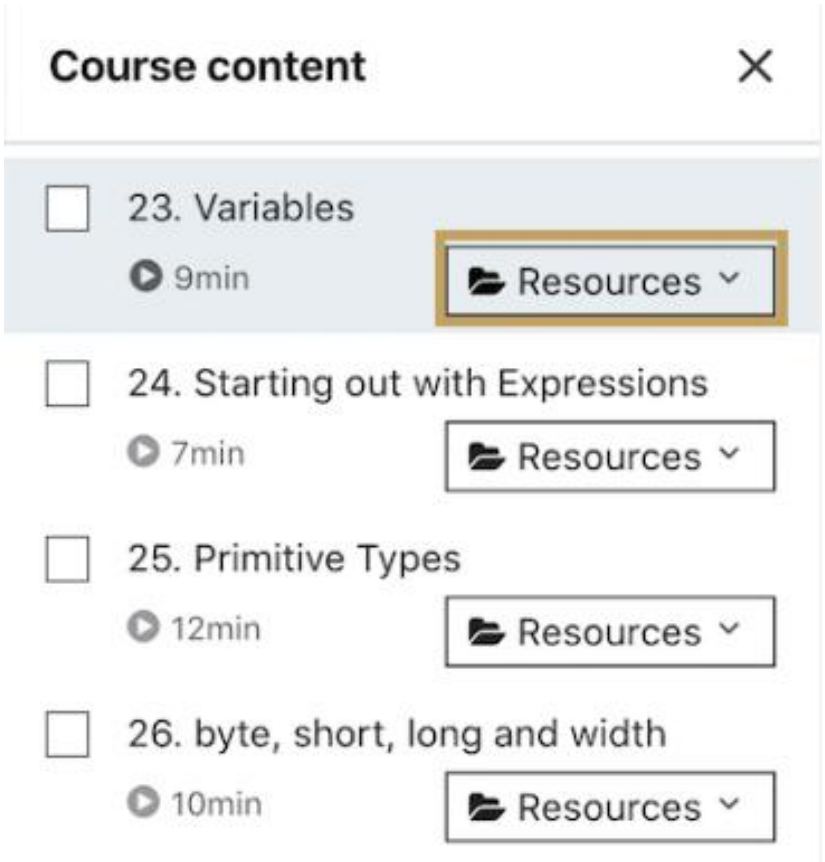
- https://github.com/mehmetozkaya/eshop-distributed

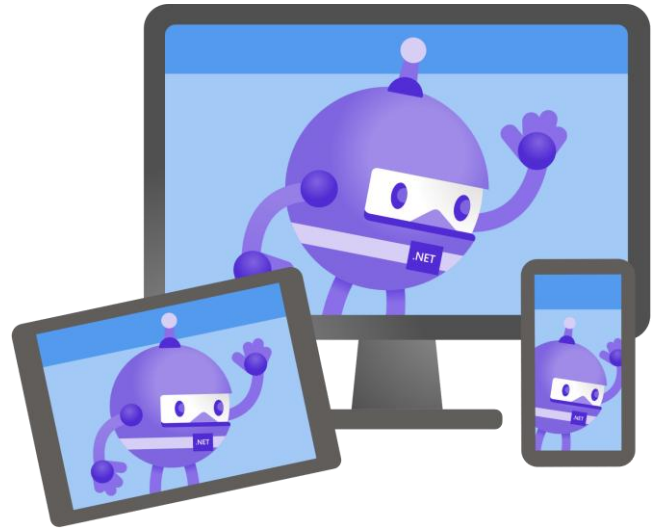
Download the source codes section-by-section

- https://github.com/mehmetozkaya/eshop-distributed-sections

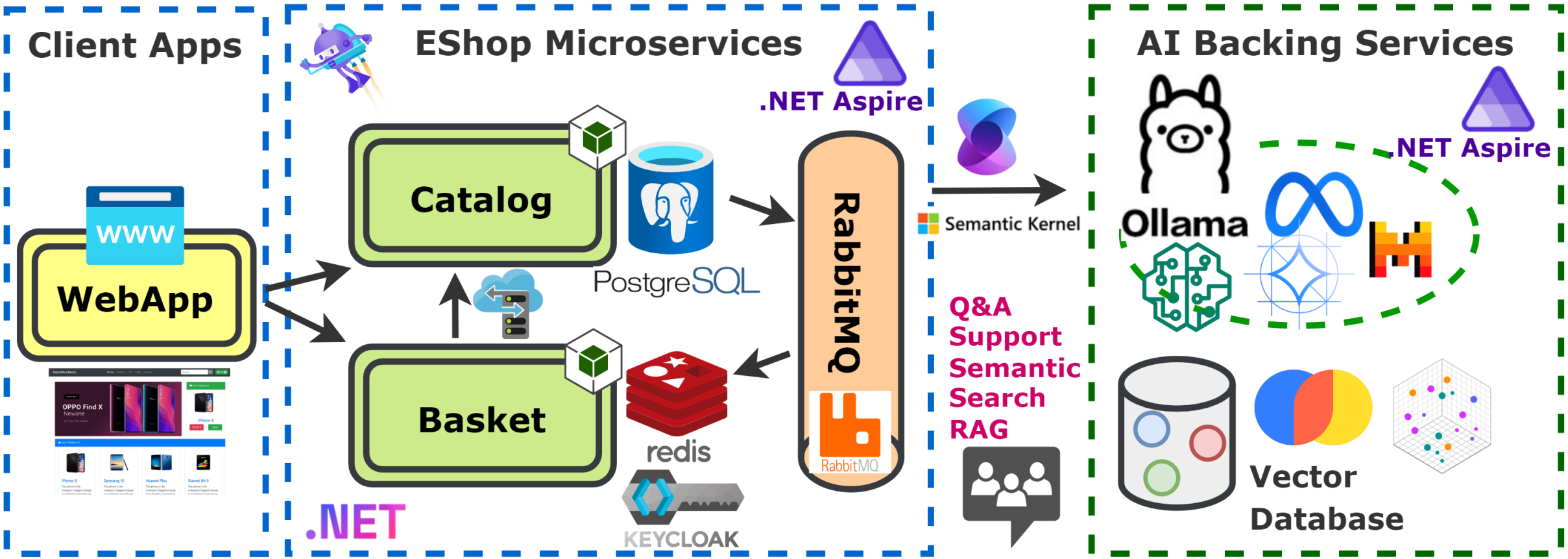
Course Slides and All Resources

Download Course Slides and **All Resources** from **This Lecture Resources**

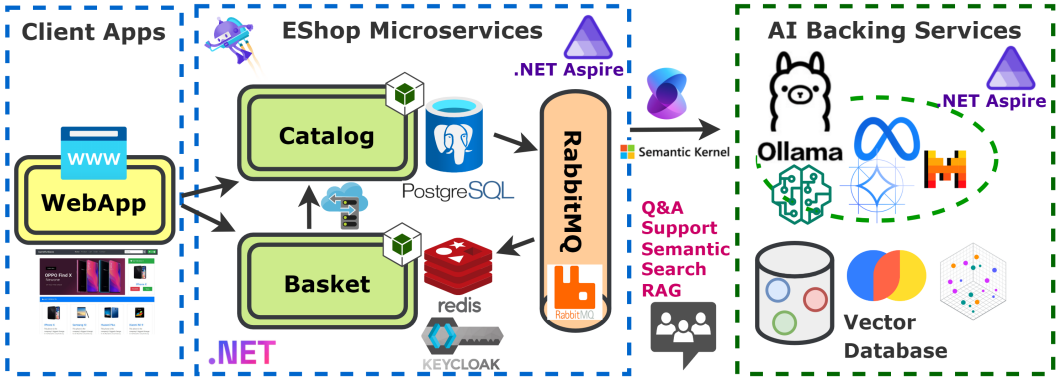




Course Project: EShop Microservices with AI-Powered Capabilities using LLMs



Course Project: EShop Microservices with AI-Powered Capabilities using LLMs



1. Clone the Repository
2. Start Docker Desktop
3. Setting Up in Visual Studio
4. Review .NET Aspire Project and EShop Microservices
5. Review AppHost Project – Cloud-Native Backing Services
6. Run the AppHost Project
7. Review Aspire Dashboard and Resources
8. Blazor Web App Test
9. API Test (Need Keycloak Realm Configurations !!)
10. Messaging and Event-Driven Patterns with RabbitMQ

What is Cloud-Native Distributed Architectures ?

What is Cloud-Native Distributed Architectures ?

Backing Services for Cloud-Native Architectures

Using LLMs and VectorDBs as Cloud-Native Backing Services in Microservices Architecture

Mehmet Ozkaya

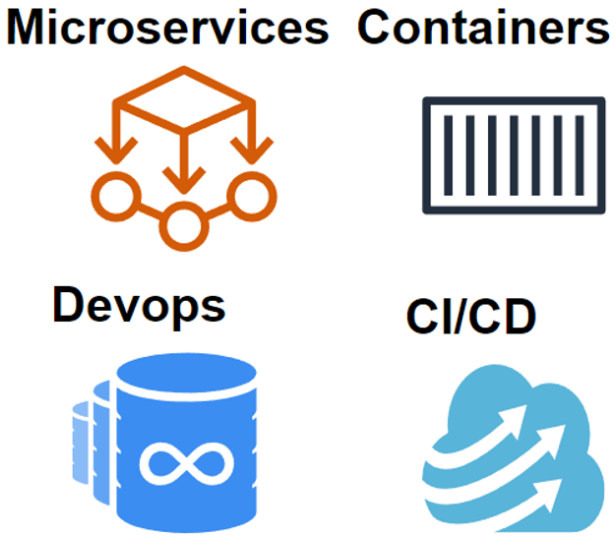
Defining «Cloud-Native»

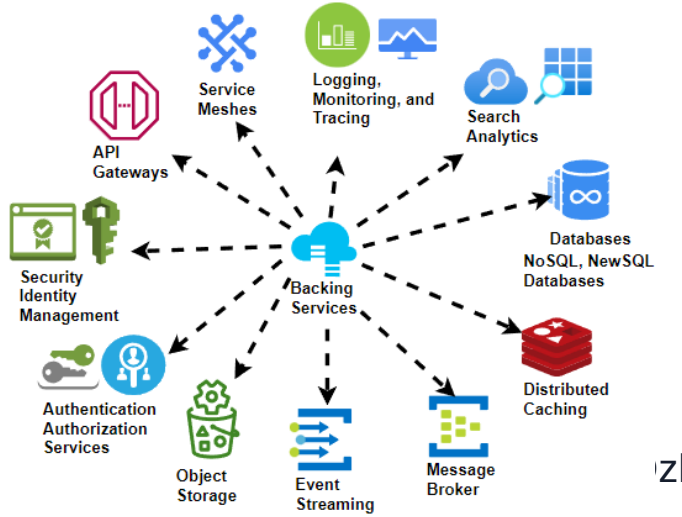
Apps **designed from** the **ground up** to **leverage cloud platforms** and **practices**

Designed to **benefit** from **elasticity**, **on-demand resources**, and **automated scaling**

Breaking down large apps into **smaller units** and **packaging** them in **containers** for consistency

Relying on **DevOps pipelines** for **continuous integration**, **delivery** and **deployment**, ensuring **rapid**, **reliable releases**





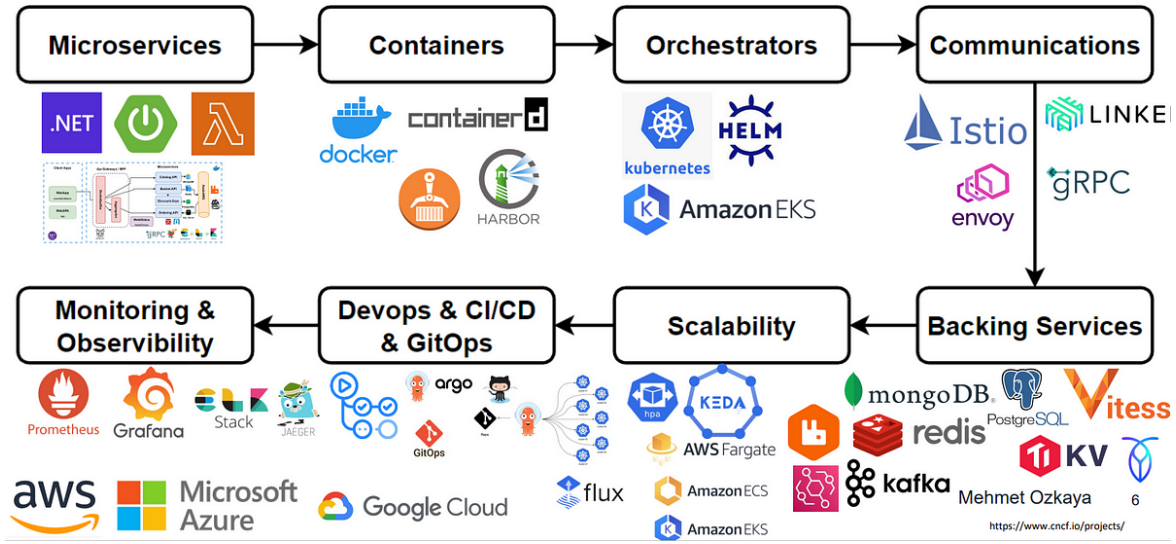
What Makes an Architecture “Distributed”?

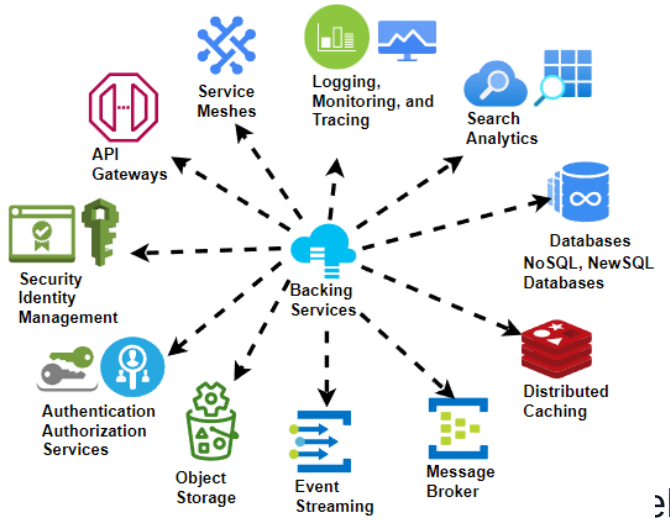
Functionality is **spread across multiple independent services** that communicate over a network

Each service can be **developed**, **deployed** and **scaled independently**

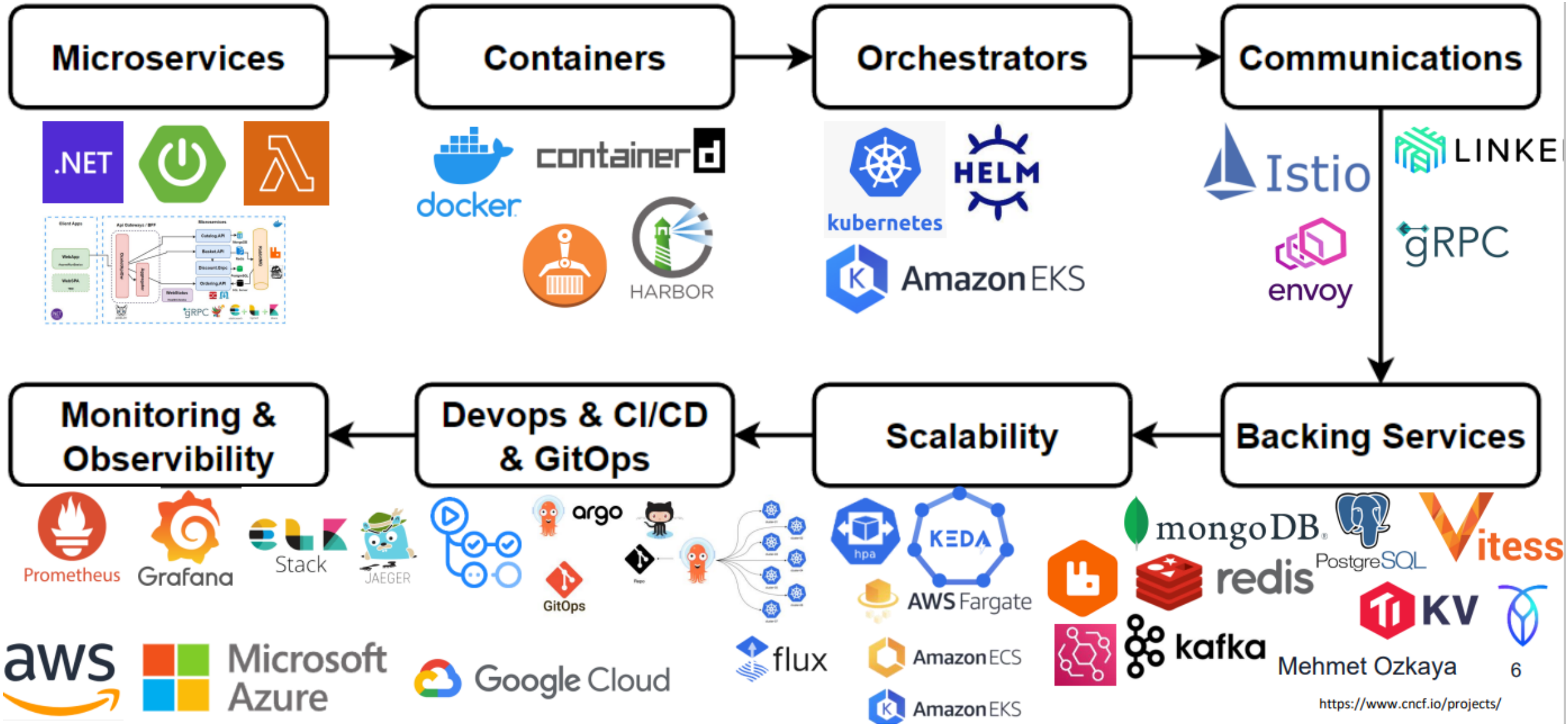
If **one service fails**, **others can continue operating**, reducing system-wide downtime

Services **handling heavy loads** can be **scaled up (or down)** without affecting others





Key Characteristics of Cloud-Native Distributed Apps



Why Choose Cloud-Native Distributed ?

Scalability & Agility
Scale only the services that need it, add new services when necessary, and quickly adapt to changing demands

Resilience & Fault Tolerance
Failures in one part of the system don’t bring down the entire application

Faster Time-to-Market
Independent teams can develop, test, and deploy services in parallel, speeding up innovation

What is Backing Services for Cloud-Native Architectures

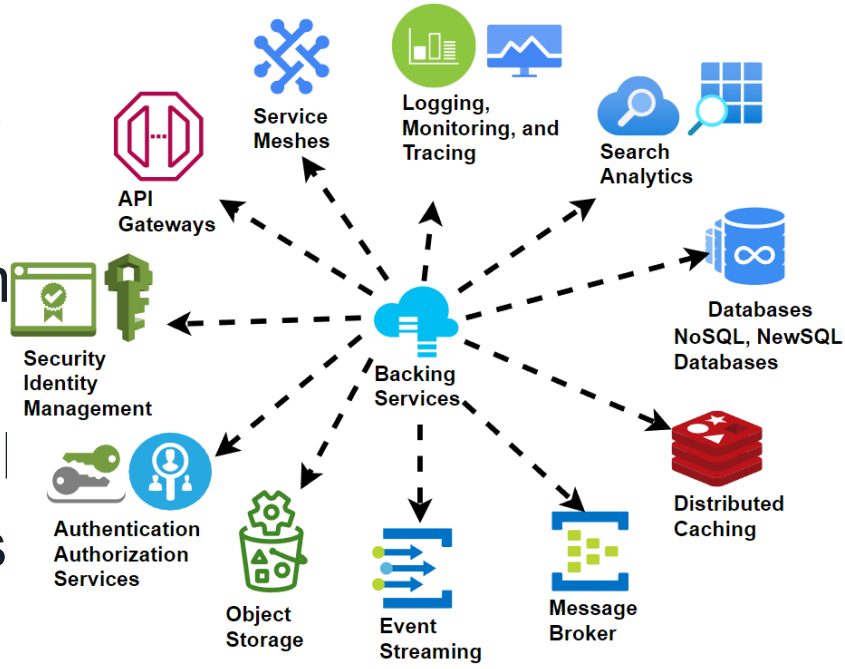
Backing services are the **external components** that applications depend on for their operation

Provide support for **various functionalities**, such as **data storage**, **messaging**, **caching** and **authentication**

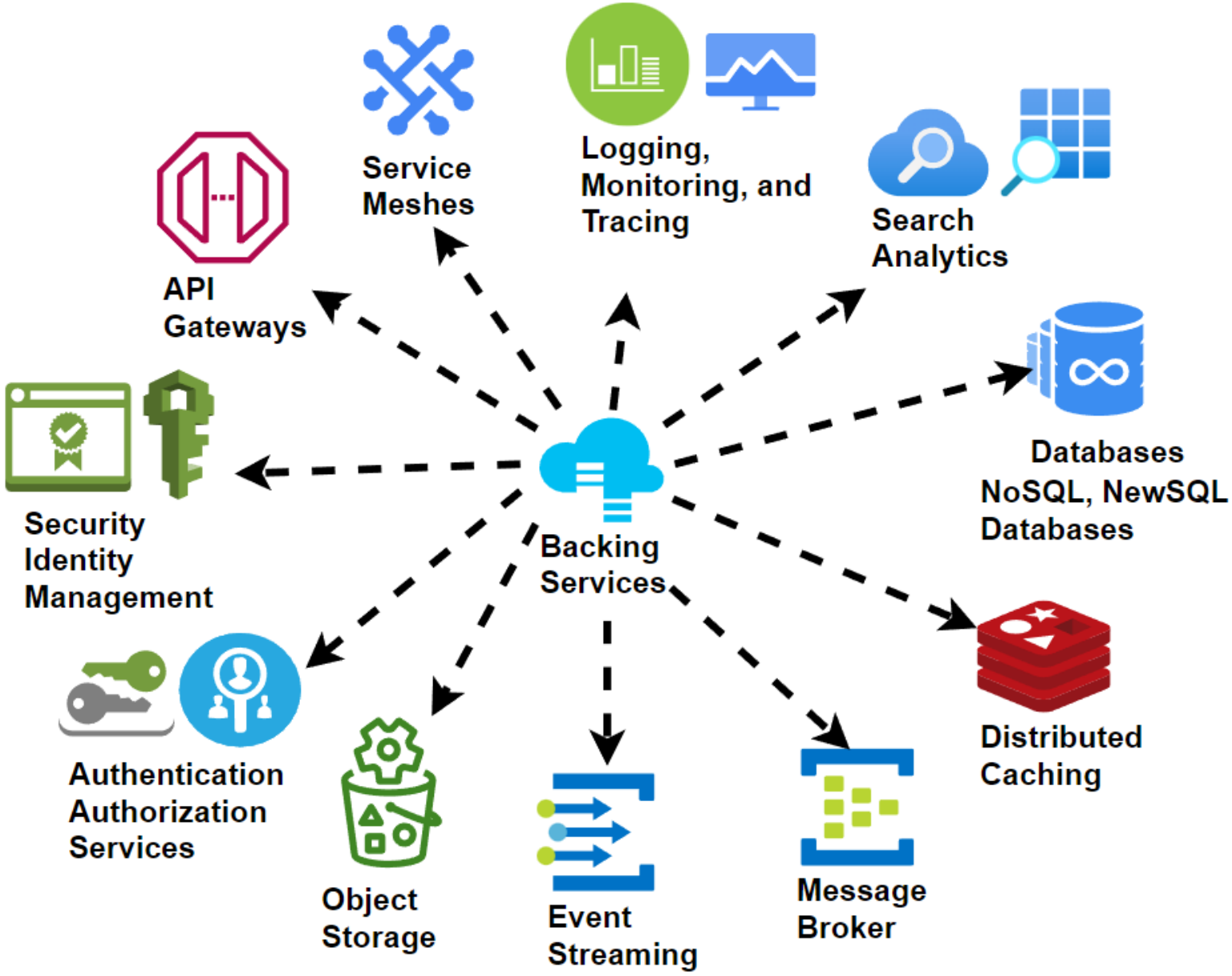
Backing services like databases, messaging systems, and caching services are **treated as attached resources**

Backing services are external to the application and can be **swapped** or **replaced without changing** the application's core logic

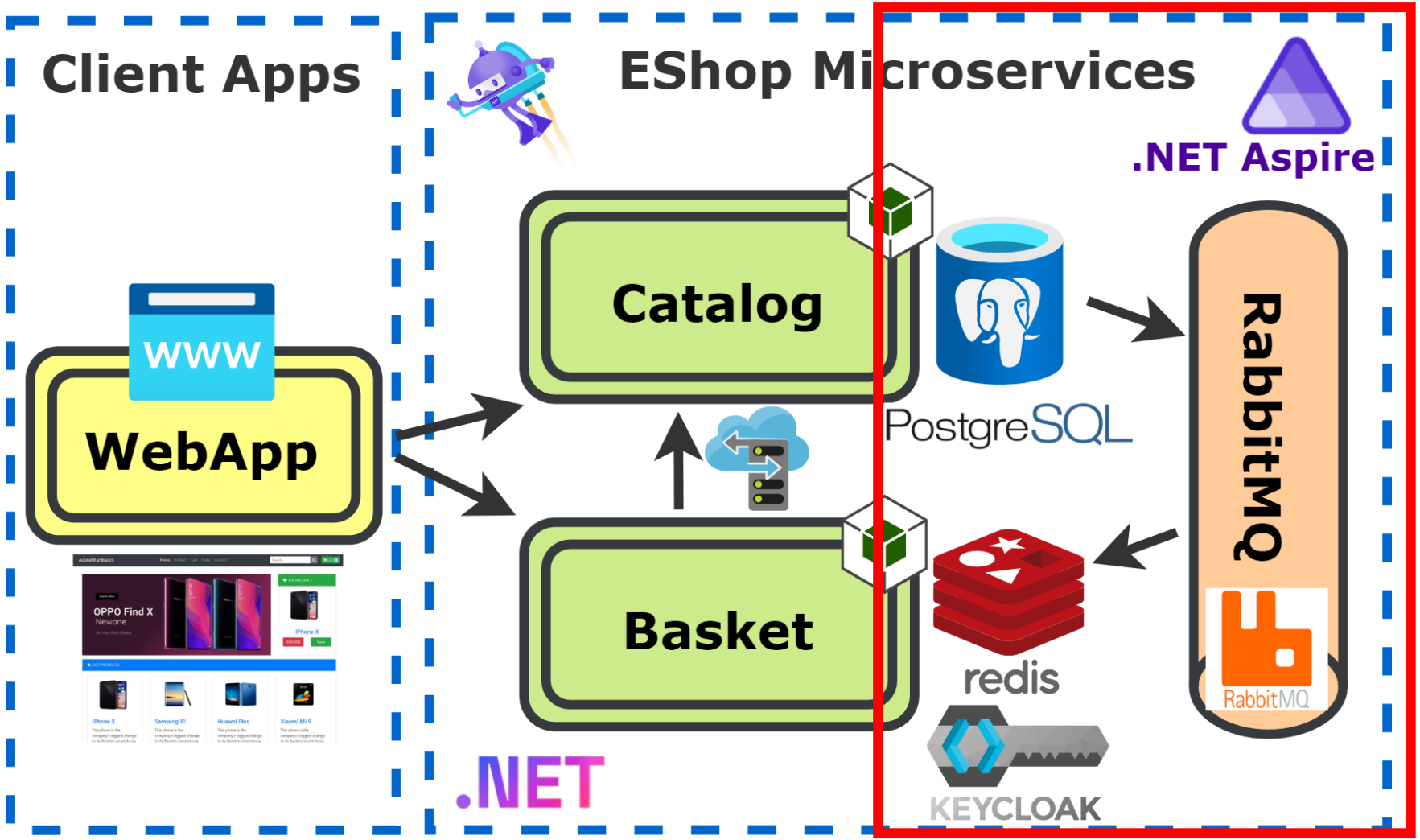
Decoupled from the application themselves, promoting flexibility, scalability, and easier maintenance



Backing Services for Cloud-Native Architectures

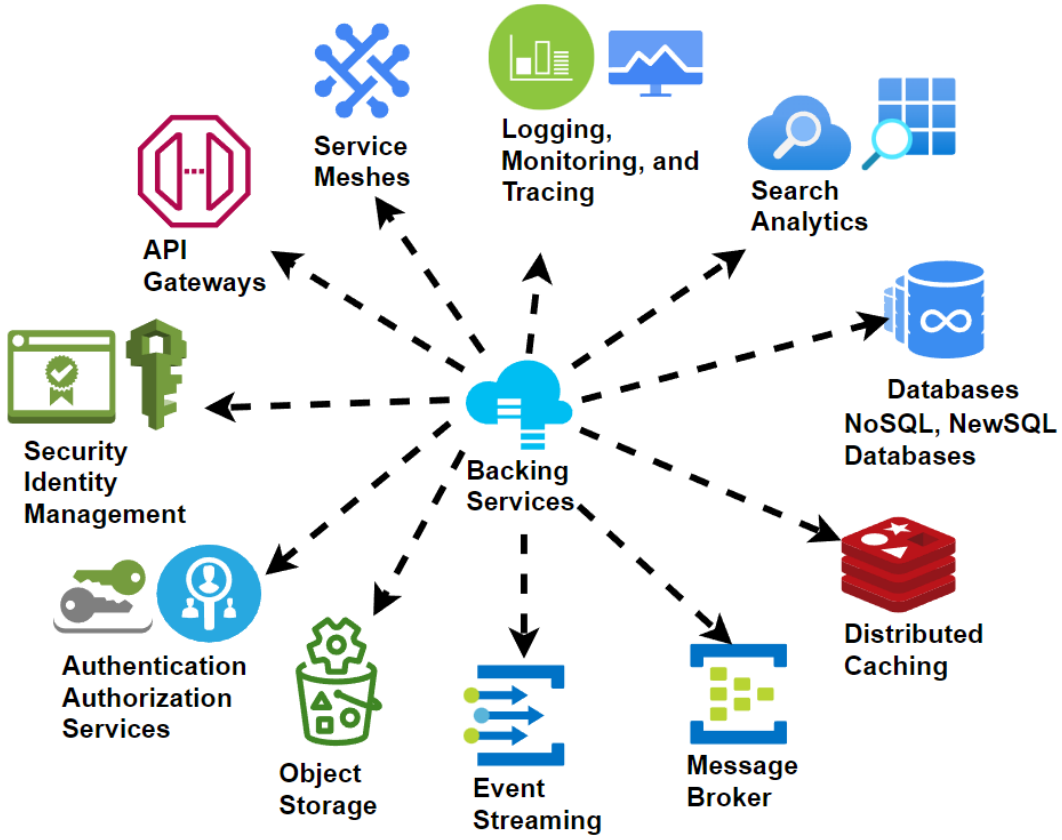


EShop Microservices Backing Services

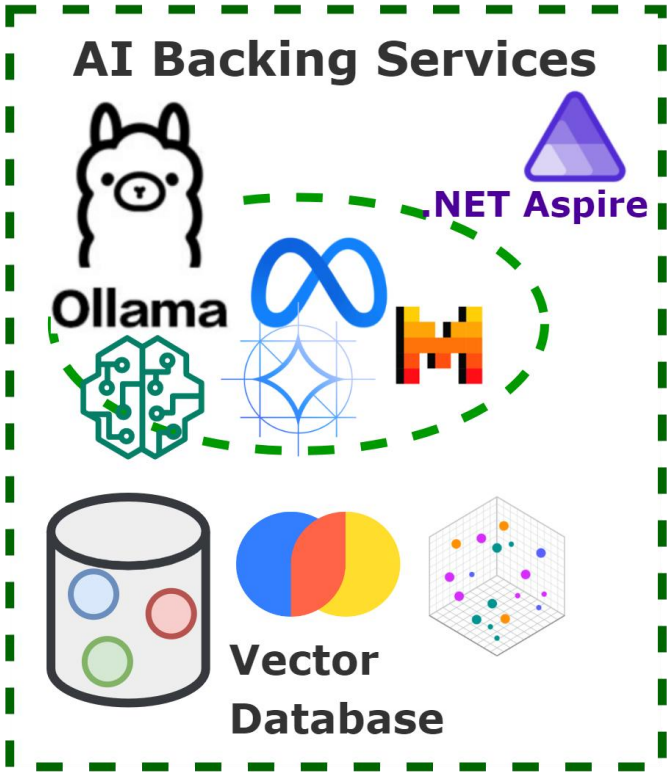


The New Era of Cloud-Native Services

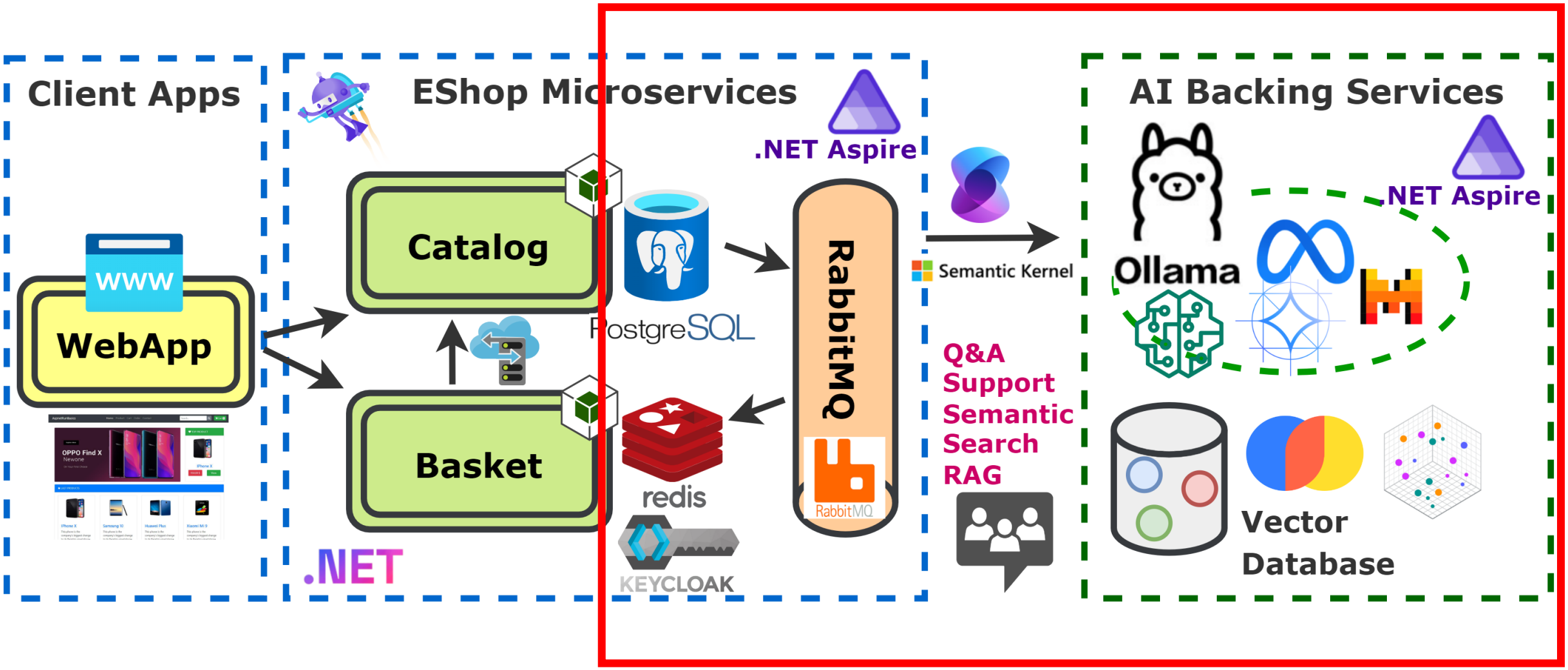
Traditional Backing Services



AI Backing Services



Use LLMs and VectorDBs as Cloud-Native Backing Service

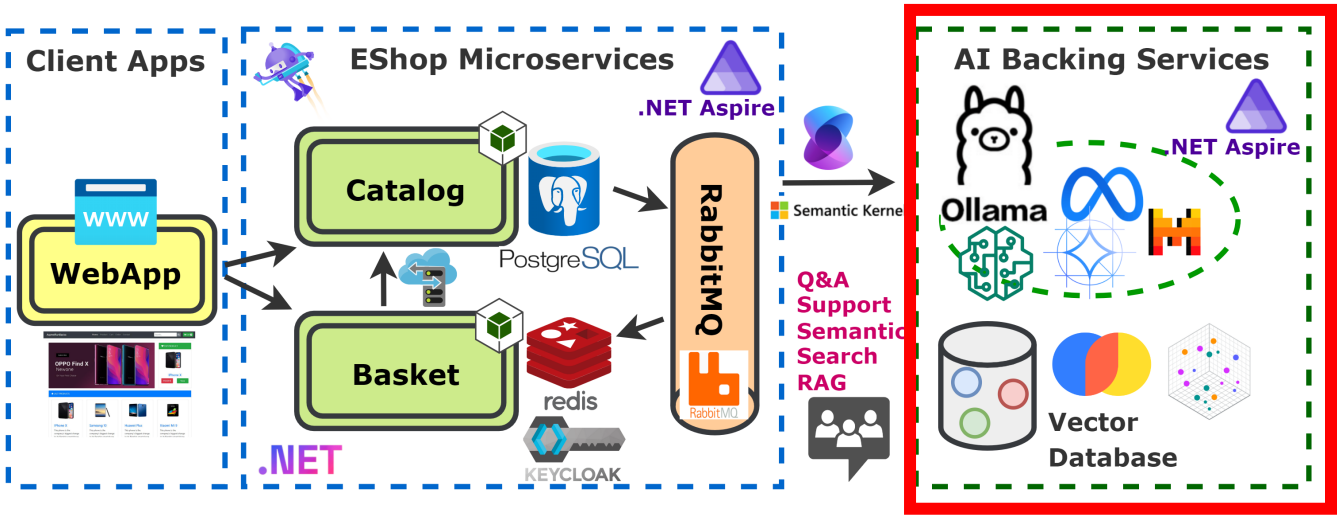


AI as a Backing Service (LLMs and VectorDBs)

LLM Backing Service
EShop uses Ollama, which enables local inference of Llama or Phi models, Ollama runs entirely on your infrastructure

Vector Databases
Specialized datastore for embeddings used in semantic search and AI workflows

Integration Points
Semantic Kernel and Microsoft.Extentions.AI library provide to connect AI Services from Microservices



.NET Aspire - Distributed App Development Framework

.NET Aspire Core Concepts: Orchestration, Integrations

.NET Aspire Orchestration - Automate Container Management

.NET Aspire Integration - Built-in Connectors

.NET Aspire Service Discovery - Finding Each Other

Mehmet Ozkaya

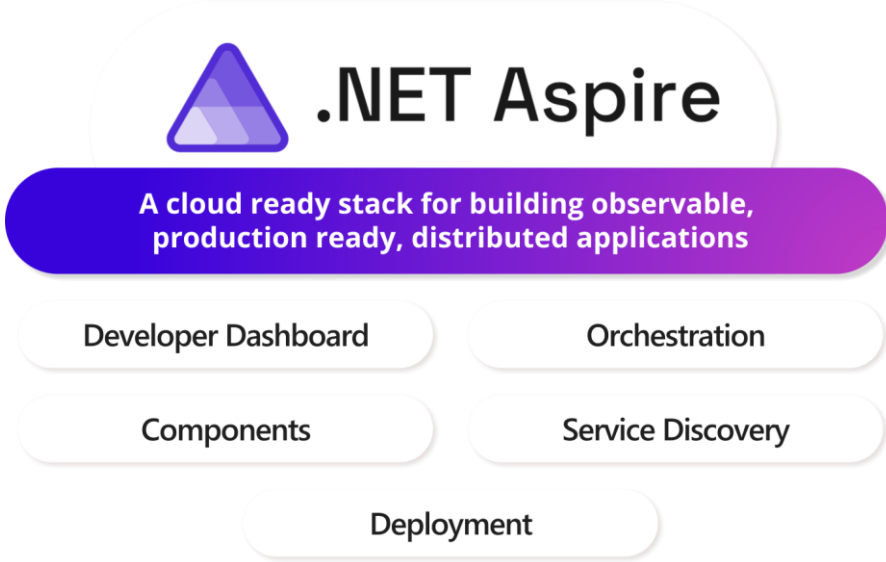
What is .NET Aspire ?

Cloud-native development framework built on top of the **.NET platform** to **streamline** and **standardize** the **process** of **creating distributed**, **microservices-based** apps

Leverages the **best practices** and **features** of .NET

Adding **scaffolding**, **orchestration** and **preconfigured integrations** for common cloud-backed services

Collection of **templates**, **libraries** that **simplify** the process of **building microservices** and **orchestrating** them in **containerized environments**



Key Features of .NET Aspire

Prebuilt Templates and Scaffolding
Project templates that jump-start your microservice creation

Built for Cloud-Native Architectures
Integrates with Docker and container orchestration platforms like ACA or K8s

Infrastructure as Code & Environment Management
Tooling that handle spinning up backing services for local development or test env

Integration with Common Backing Services
Supports popular databases (PostgreSQL, MongoDB), caches (Redis), and message brokers (RabbitMQ, Azure Service Bus)

Modular & Extensible Architecture
Modular structure to integrate AI services, logging platforms, or monitoring solutions

Why .NET Aspire ?

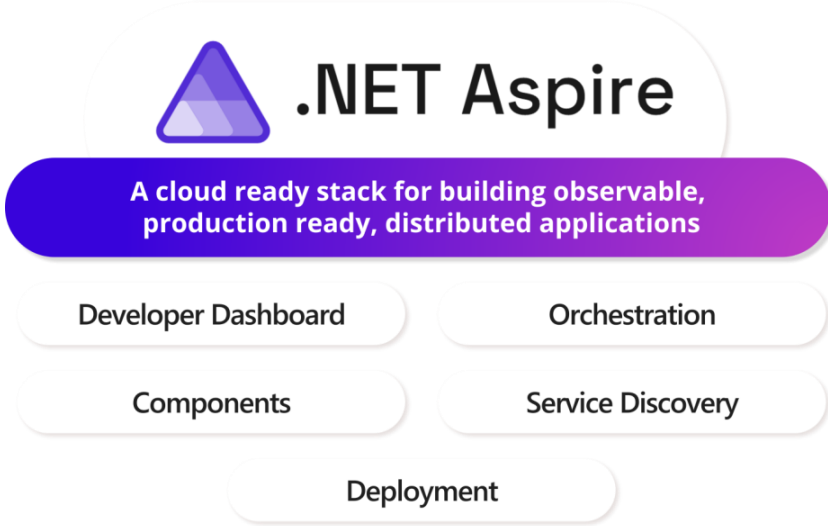
Standard templates and **tooling** help teams **quickly stand up** new **microservices**

Built-in patterns for **logging**, **configuration** and **container orchestration**

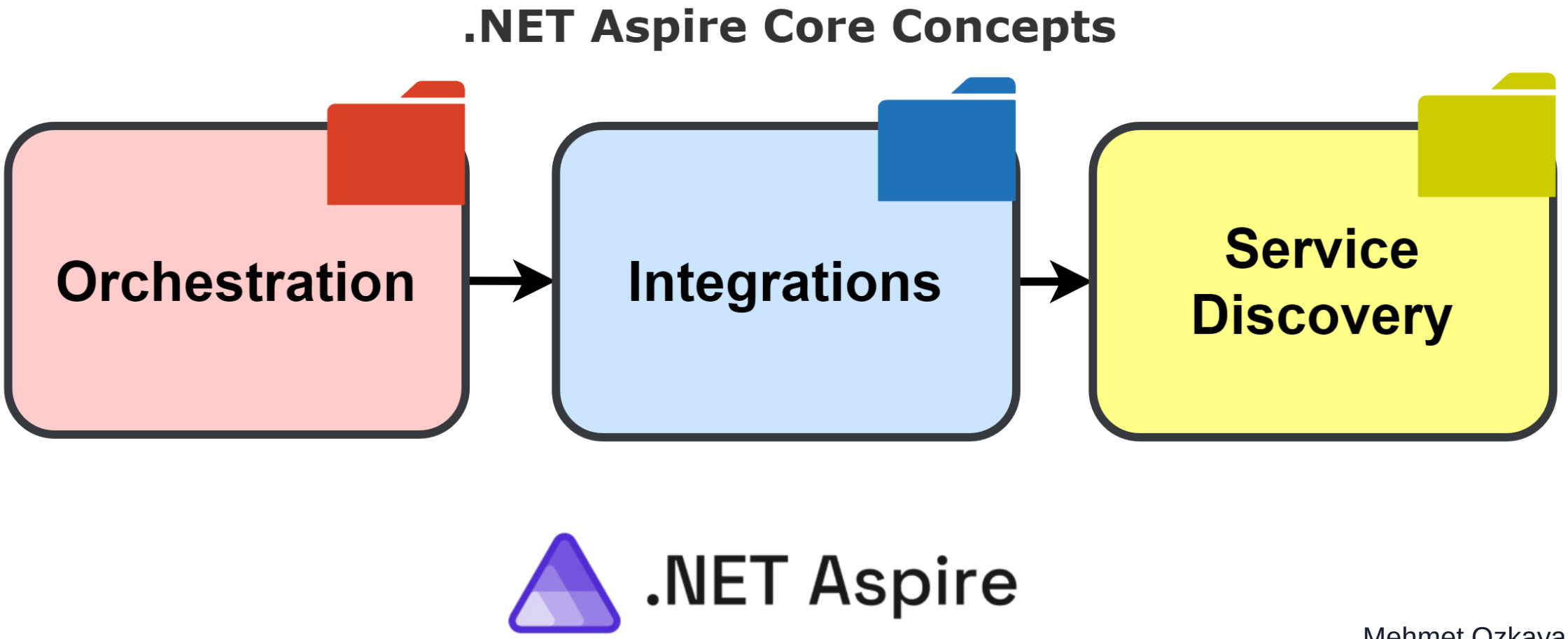
Event-driven communication, **containerization**, and **horizontal scalability** from **day one**

Connects readily to **external services** (databases, message brokers, identity providers) with **minimal extra plumbing**

Faster development of **cloud-native microservices,** combining **the power of the .NET platform** with **proven patterns** and **out-of-the-box integrations**



.NET Aspire Concepts: Orchestration, Integrations and Service Discovery



Orchestration in .NET Aspire

.NET Aspire

Automating Container Management
Automate container-based deployments without requiring manual Dockerfiles or complex docker-compose setups for each microservice

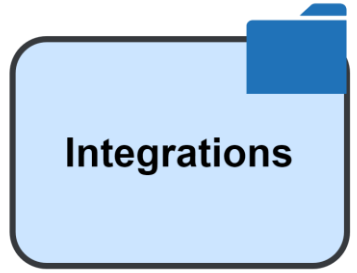
Environment Awareness
Uses environment configuration (local, test, production) to decide what services to start and how to configure them

Service Dependencies
If a microservice needs to wait for a database to spin up, Aspire can coordinate that

Scalable Infrastructure
Straightforward to scale individual microservices, whether you're using Docker, Kubernetes or ACA, Aspire manages the foundational scripts and tooling

Integrations in .NET Aspire

.NET Aspire



Built-in Connectors
Prebuilt integrations with common cloud and on-premises services, such as databases (PostgreSQL, MongoDB), caches (Redis), message brokers (RabbitMQ, Azure Service Bus), and identity providers (Keycloak)

Cross-Cutting Features
Provides default setups for logging, monitoring, and telemetry, saving time and promoting consistent patterns across different microservices

Modular & Extensible
Adding a new service—like AI-based features or a vector database—requires minimal adjustments thanks to Aspire's pluggable design

Service Discovery in .NET Aspire

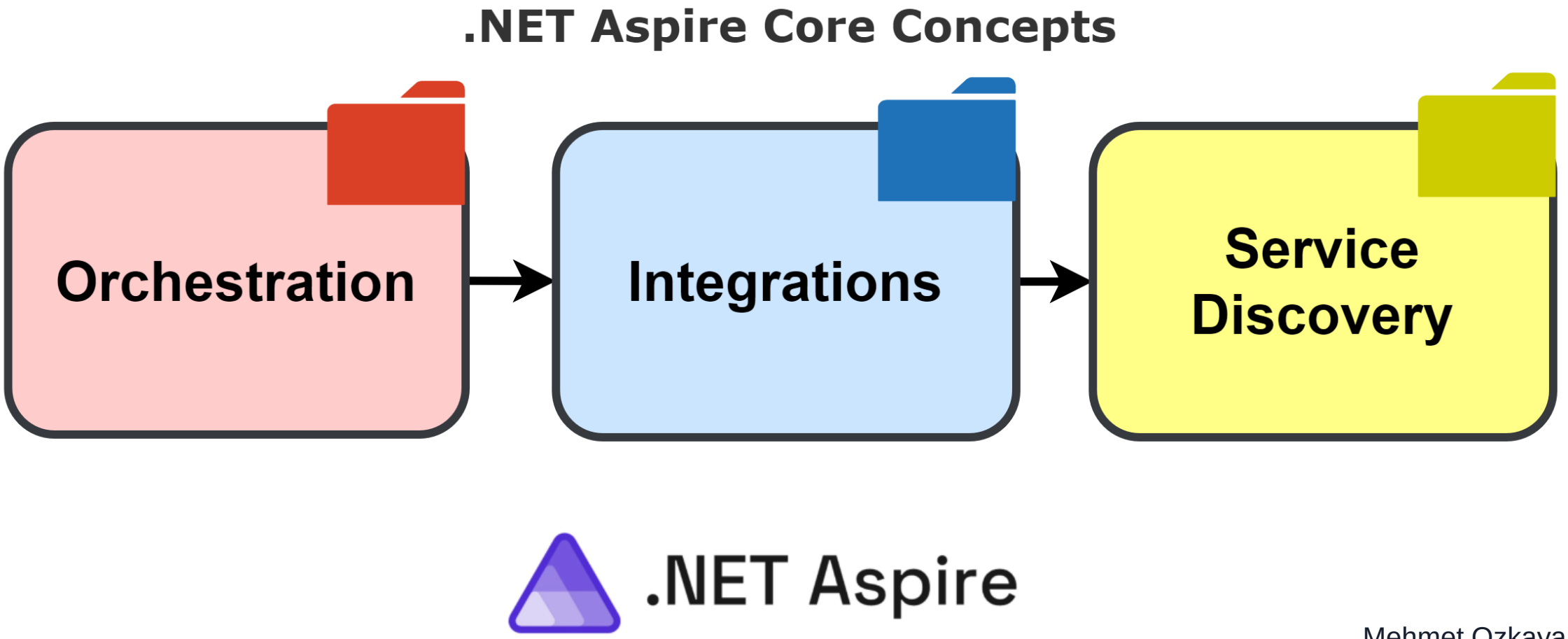
.NET Aspire

Dynamic Routing
Service endpoints can change dynamically—for instance, when scaling up or down, keep track of microservice locations, avoiding the need for hard-coded URLs or IP addresses

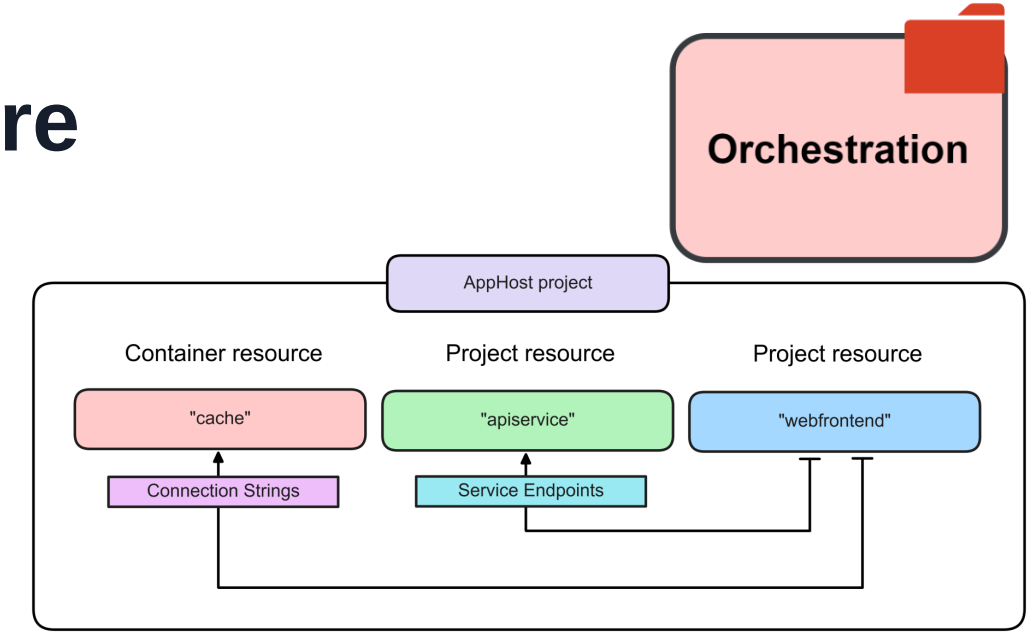
Automatic Registration
When each microservice starts, it registers itself in Aspire's service registry (or a backing service registry), making it discoverable to other services

Resilience & Scaling
If one instance of a service becomes unavailable, calls are routed to another instance automatically, increasing resiliency and ensuring uninterrupted service

.NET Aspire Concepts: Orchestration, Integrations and Service Discovery



Terms of Orchestration in .NET Aspire



App Model
The heart of your distributed application

App Host / Orchestrator
The project responsible for running all resources

Resource
Any piece of your app (databases, microservices, containers)

Integration
NuGet packages that model or configure services

Reference
Declares dependencies between resources

Defining the App Model of Orchestration

App Model

The heart of your distributed application

```
var builder =
DistributedApplication.CreateBuilder(args);
// Add resources
// Express dependencies

builder.Build().Run();
```

App Host / Orchestrator Project

App Host / Orchestrator

The project responsible for running all resources

```
<Project Sdk="Microsoft.NET.Sdk">
    <Sdk Name="Aspire.AppHost.Sdk" Version="9.0.0" />

    <PropertyGroup>
        <OutputType>Exe</OutputType>
        <TargetFramework>net9.0</TargetFramework>
        <IsAspireHost>true</IsAspireHost>
    </PropertyGroup>

    <ItemGroup>
        <PackageReference Include="Aspire.Hosting.AppHost" Version="9.0.0"
/>  </ItemGroup>
</Project>
```

Coding Example – Redis + Two Projects


```
var builder = DistributedApplication.CreateBuilder(args);

var cache = builder.AddRedis("cache");

var apiService = builder.AddProject<Projects.AspireApp_ApiService>
("apiservice");
builder.AddProject<Projects.AspireApp_Web>("webfrontend")
       .WithExternalHttpEndpoints()
       .WithReference(cache)
       .WaitFor(cache)
       .WithReference(apiService)
       .WaitFor(apiService);

builder.Build().Run();
```

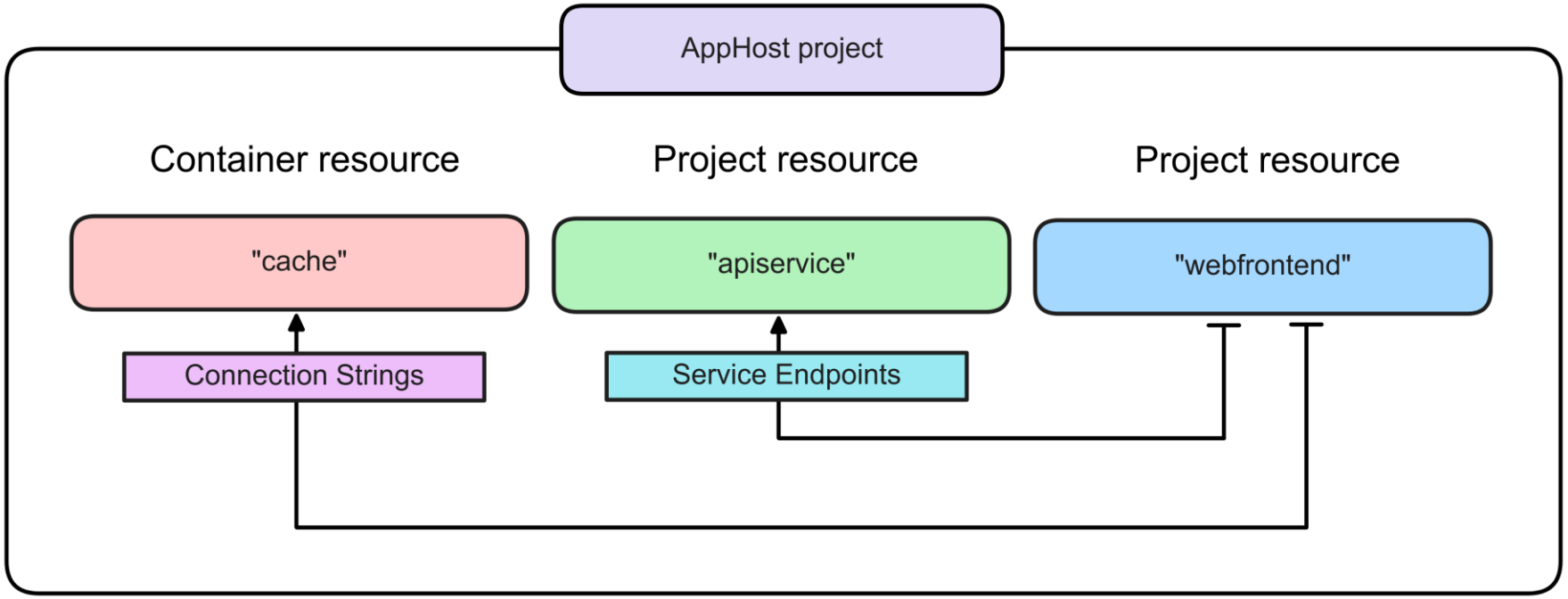
Built-In Resource Types of Orchestration

AddProject – ProjectResource

AddContainer – ContainerResource

AddExecutable – ExecutableResource

AddParameter – ParameterResource



References & Dependencies


```
var builder =
DistributedApplication.CreateBuilder(args);
var cache = builder.AddRedis("cache");

builder.AddProject<Projects.AspireApp_Web>
        ("webfrontend")
       .WithReference(cache);
```

Waiting for Resources

Orchestration

```
var postgres = builder.AddPostgres("postgres");
var postgresdb =
postgres.AddDatabase("postgresdb");
builder.AddProject<Projects.AspireApp_ApiService>
        ("apiservice")
       .WithReference(postgresdb)
       .WaitFor(postgresdb);
```

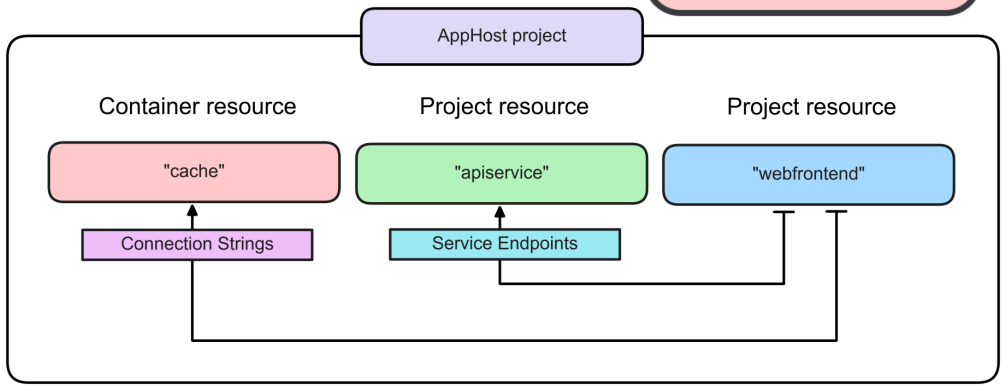
Putting It All Together - Orchestration in .NET Aspire

Define your App Model in an **App Host project (.AppHost)**

Add & Configure Resources—like microservice projects, caches, and databases—using built-in integrations

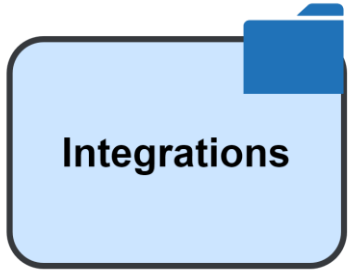
Reference & WaitFor to express dependencies and correct startup order

Build & Run: .NET Aspire automatically launches the containers, orchestrates networking, and injects environment variables

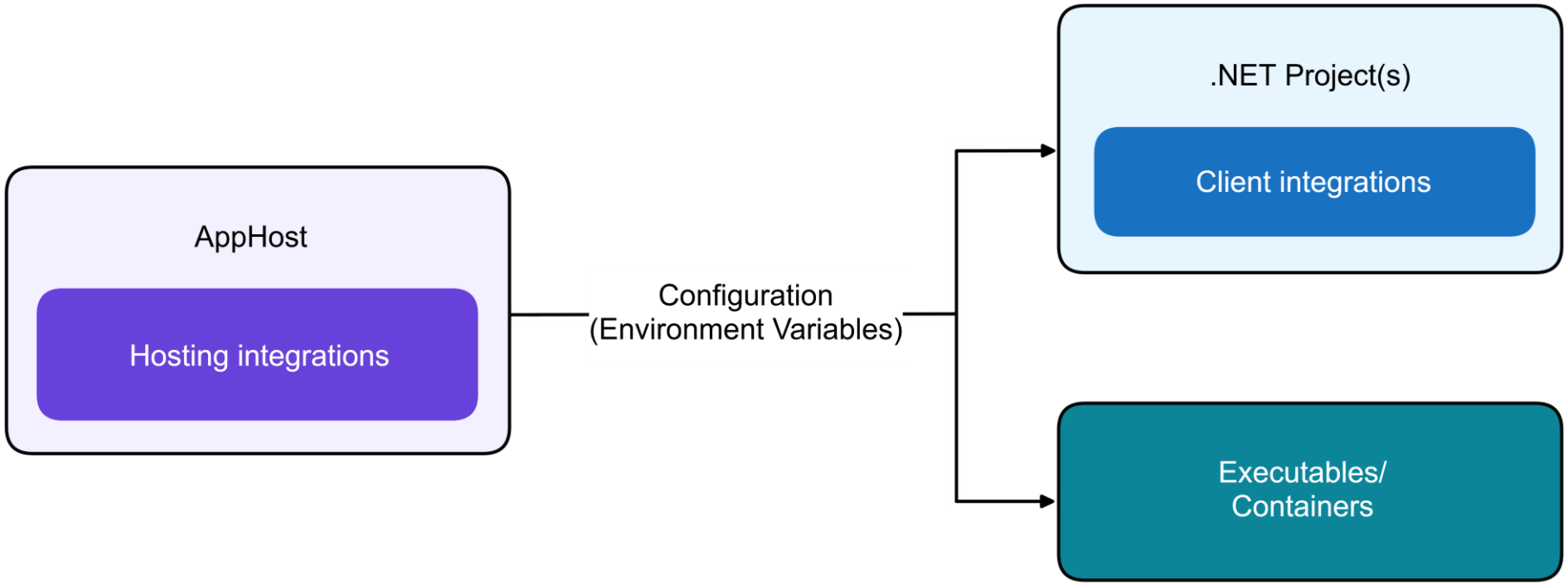


.NET Aspire Integrations

.NET Aspire



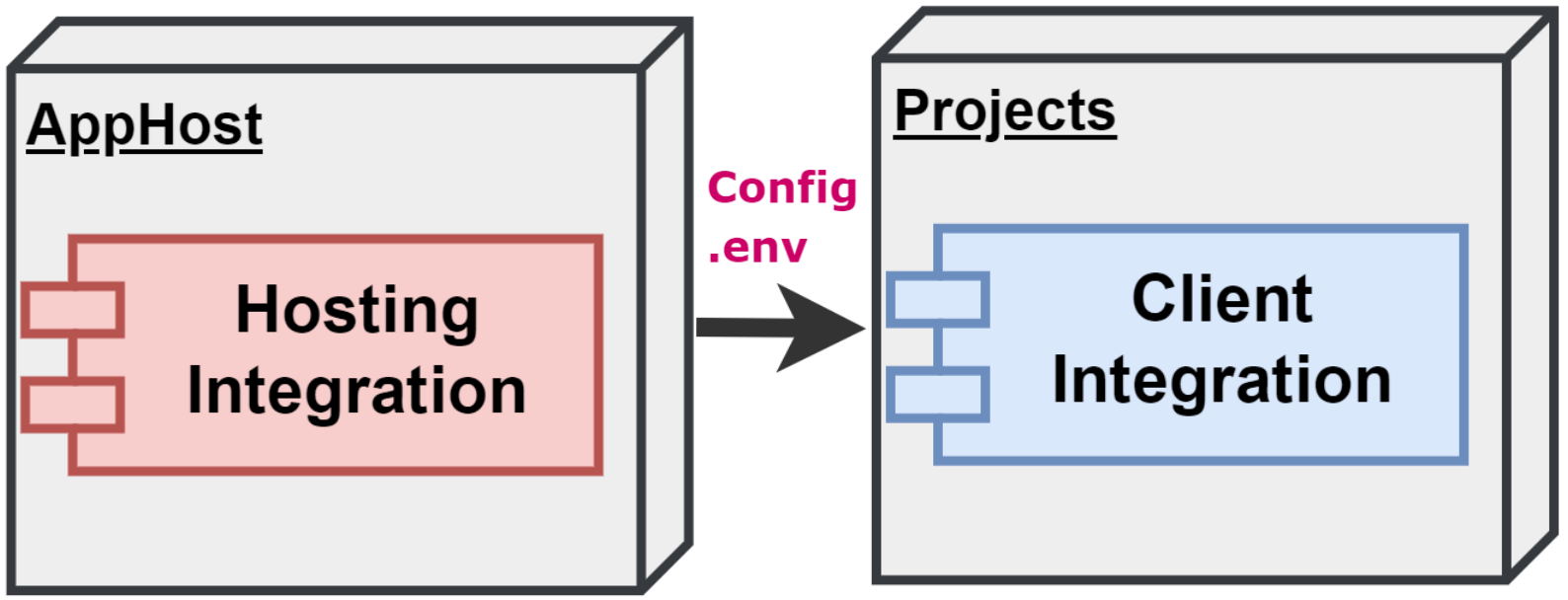
Curated NuGet packages designed to make it **simple** to **integrate** your **microservices** with **external resources**—such as **caches**, **databases**, or **message brokers**



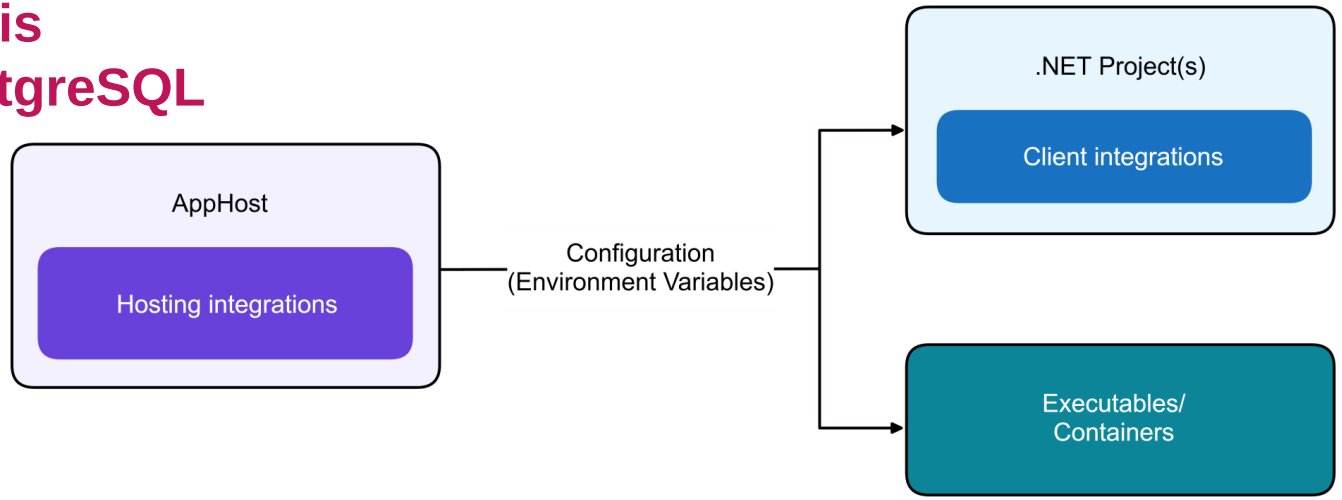
Hosting vs. Client Integrations

.NET Aspire

Integrations

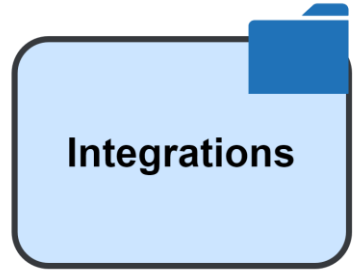


Aspire.Hosting.Redis
Aspire.Hosting.PostgreSQL



Aspire.StackExchange.Redis
Aspire.Npgsql.EntityFrameworkCore.PostgreSQL

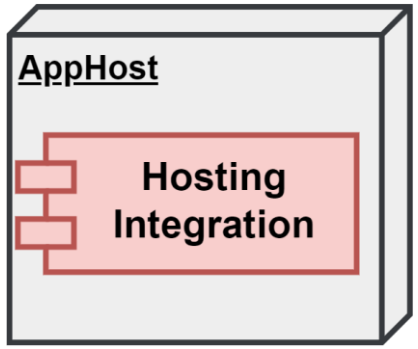
Hosting Integrations – Provisioning Resources



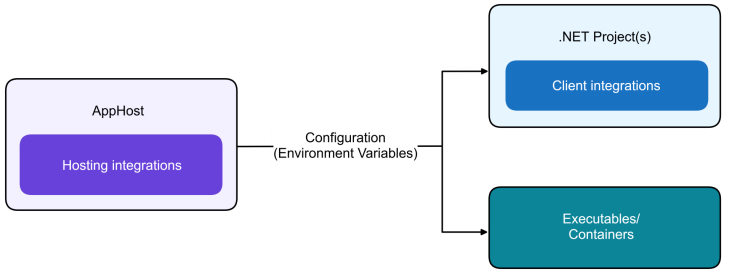
Extend the **IDistributedApplicationBuilder** interface, adding methods like **AddRedis("name")** or **AddPostgres("db")**

Provision container-based resources automatically

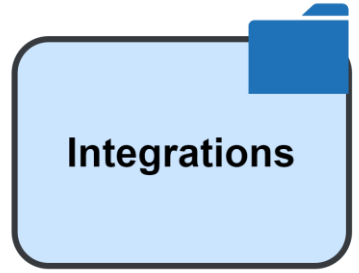
Use **environment variables** to **pass connection strings** to other projects that depend on them




```
var builder =
DistributedApplication.CreateBuilder(args);
// Hosting integration call:
var cache = builder.AddRedis("cache");

// This sets up a Redis container named "cache"
```



Client Integrations – Connecting to Resources

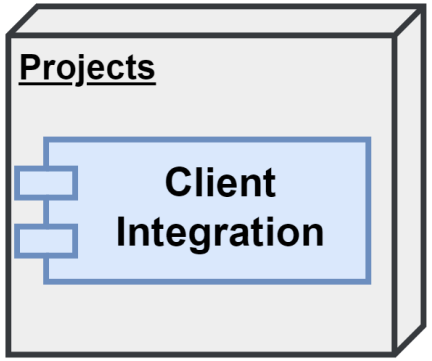


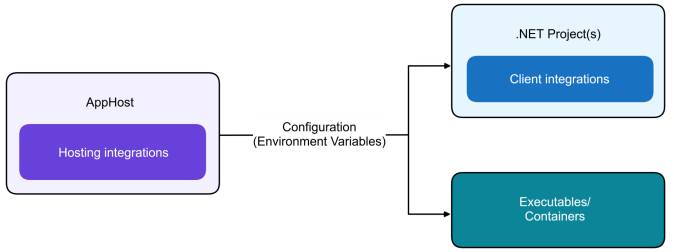
Extend **IHostApplicationBuilder** in Consuming Projects;
builder.AddRedisDistributedCache(«cache»)

Add **Health Checks**, **Logging**, **Telemetry**

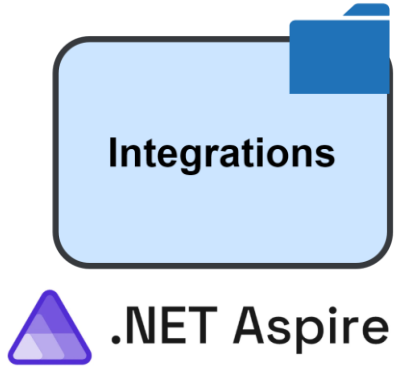
Example: Aspire.StackExchange.Redis

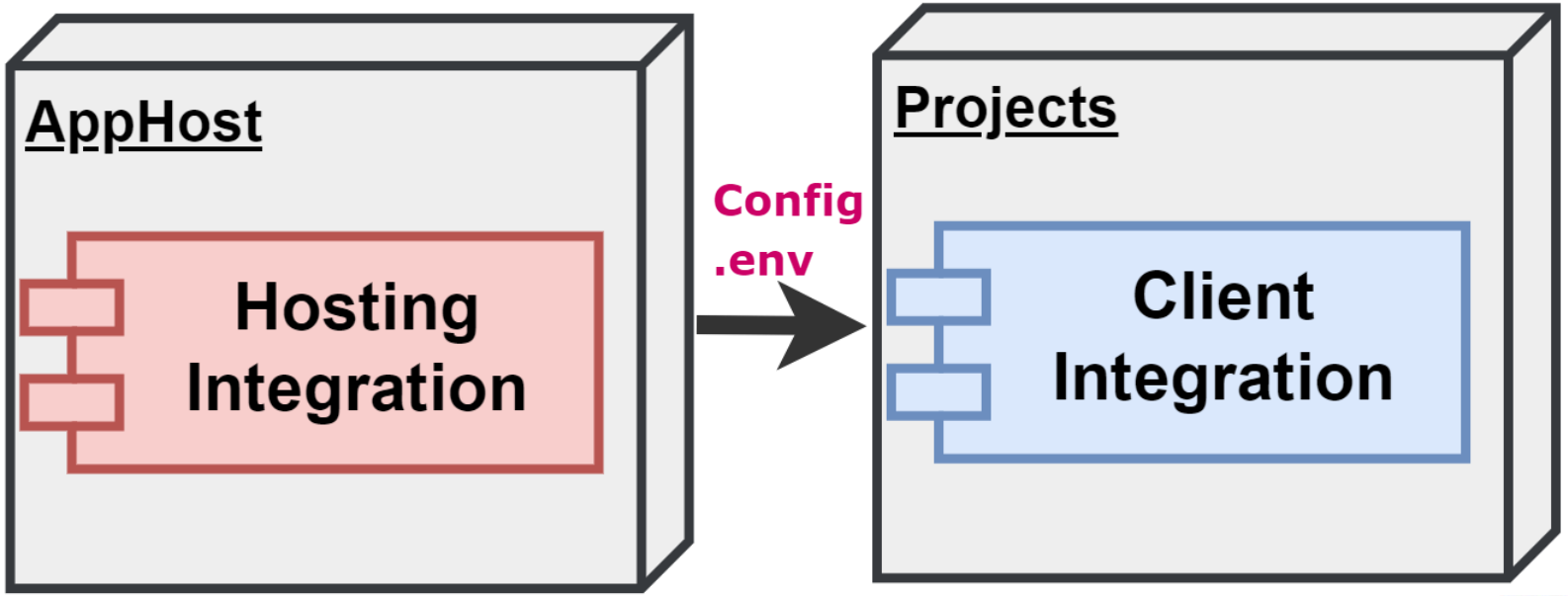
```
var builder = WebApplication.CreateBuilder(args);

// AddServiceDefaults is often called automatically in Aspire-based
// projects, but you can call it explicitly, too.
builder.AddServiceDefaults();

// This might be from Aspire.StackExchange.Redis
builder.AddRedisDistributedCache(connectionName: "cache");
```





Relationship Hosting vs. Client Integrations

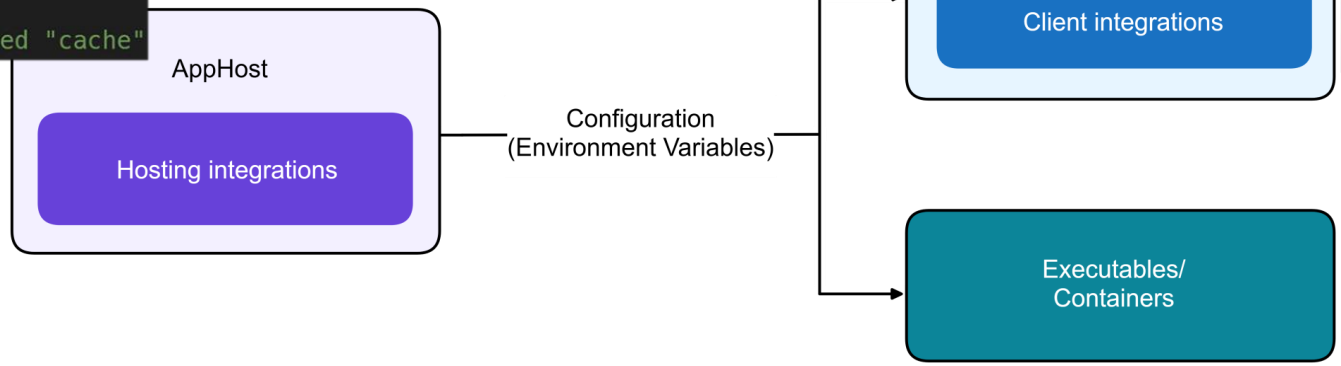




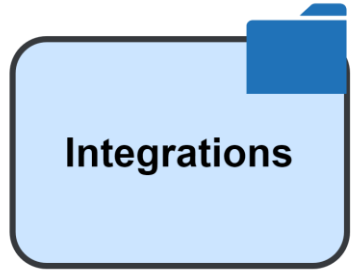

```
var builder =
DistributedApplication.CreateBuilder(args);
// Hosting integration call:
var cache = builder.AddRedis("cache");

// This sets up a Redis container named "cache"
```

Observability
Logging, tracing and metrics
Health Checks
Resiliency

```
var builder = WebApplication.CreateBuilder(args);

// AddServiceDefaults is often called automatically
builder.AddServiceDefaults();

// This might be from Aspire.StackExchange.Redis
builder.AddRedisDistributedCache(connectionName:
"cache");
```



Official Integrations



Redis, PostgreSQL, RabbitMQ, Kafka, Keycloak

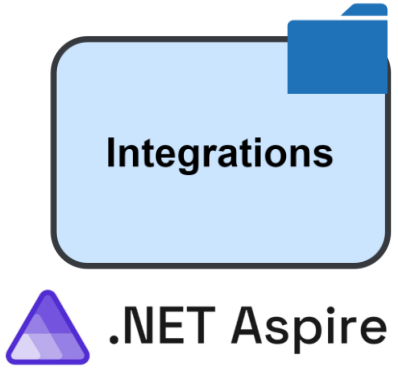
Azure & AWS Packages

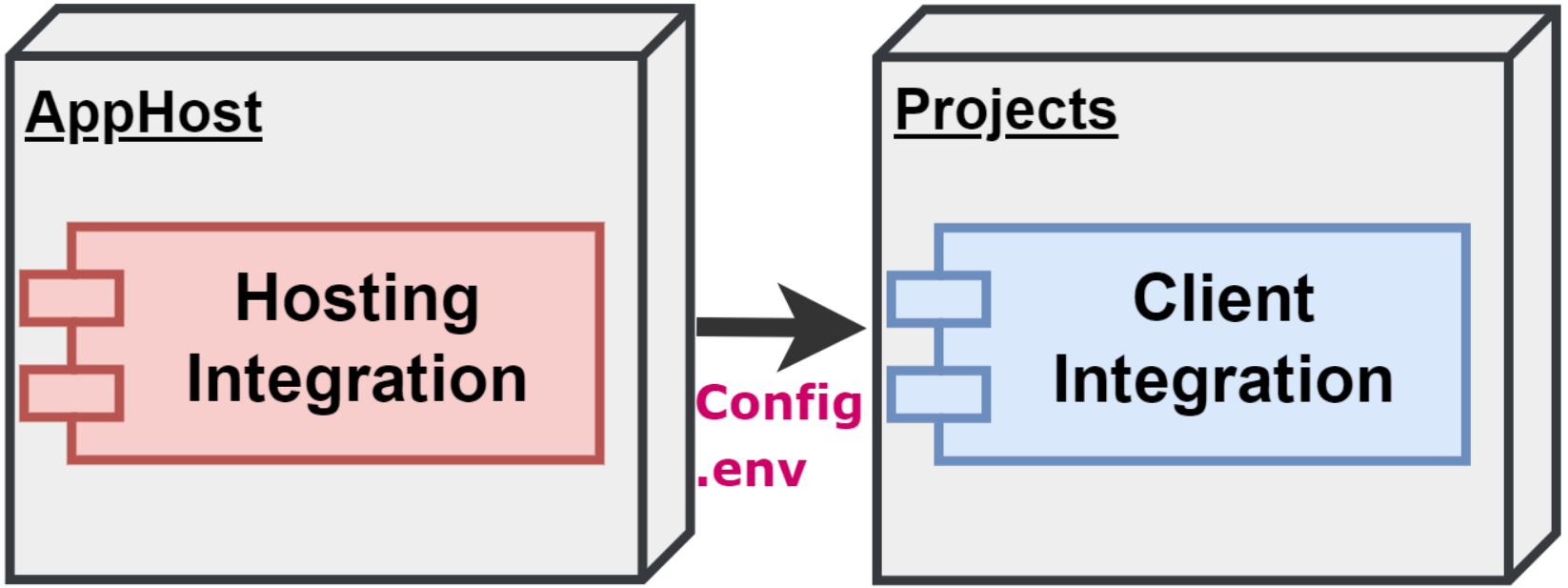
Community Toolkit

Official Integrations Table

Integration docs and NuGet packages	Description
- Learn more: Apache Kafka - Hosting: Aspire.Hosting.Kafka - Client: Aspire.Confluent.Kafka	A library for producing and consuming messages from an Apache Kafka broker.
- Learn more: Dapr - Hosting: Aspire.Hosting.Dapr - Client: N/A	A library for modeling Dapr as a .NET Aspire resource.
- Learn more: Elasticsearch - Hosting: Aspire.Hosting.Elasticsearch - Client: Aspire.Elastic.Clients.Elasticsearch	A library for accessing Elasticsearch databases.
- Learn more: Keycloak - Hosting: Aspire.Hosting.Keycloak - Client: Aspire.Keycloak.Authentication	A library for accessing Keycloak authentication.
- Learn more: Milvus - Hosting: Aspire.Hosting.Milvus - Client: Aspire.Milvus.Client	A library for accessing Milvus databases.
- Learn more: MongoDB Driver - Hosting: Aspire.Hosting.MongoDB - Client: Aspire.MongoDB.Driver	A library for accessing MongoDB databases.

Conclusion: Hosting vs. Client Integrations






```
var builder =
DistributedApplication.CreateBuilder(args);
// Hosting integration call:
var cache = builder.AddRedis("cache");

// This sets up a Redis container named "cache"
```

```
var builder = WebApplication.CreateBuilder(args);

// AddServiceDefaults is often called automatically
builder.AddServiceDefaults();

// This might be from Aspire.StackExchange.Redis
builder.AddRedisDistributedCache(connectionName:
"cache");
```

.NET Aspire Service Discovery

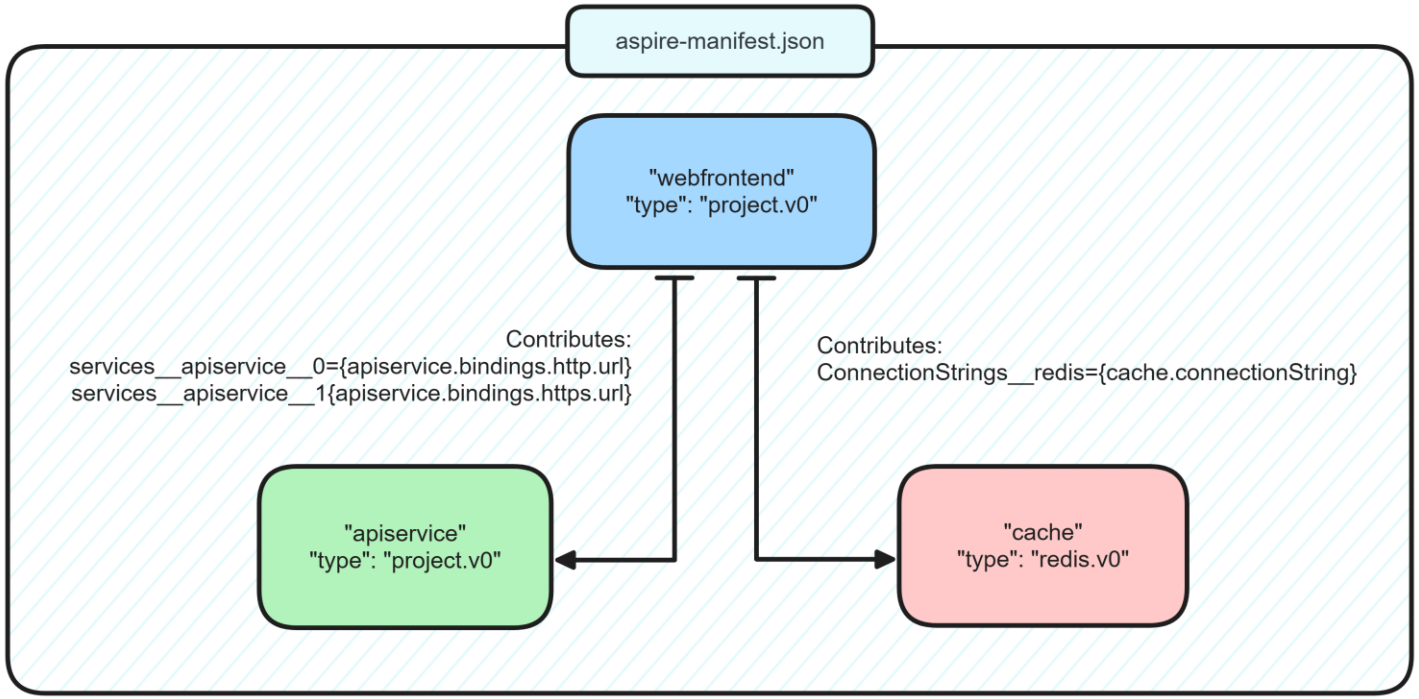
.NET Aspire

Service Discovery

Microservices frequently **scale** or move around, **Service discovery** is how those **microservices locate** each other

Service A needs to **call Service B**'s REST endpoint—but can't rely on a **hard-coded host**/port

Rely on **configuration** that's **injected automatically**, making each service **discoverable** by **name**



Implicit Service Discovery by Reference

.NET Aspire

Service Discovery

```
var builder = DistributedApplication.CreateBuilder(args);

var cache = builder.AddRedis("cache");

var catalog = builder.AddProject<Projects.CatalogService>
("catalog");
var basket = builder.AddProject<Projects.BasketService>("basket");

var frontend = builder.AddProject<Projects.MyFrontend>("frontend")
  .WithReference(cache)
  .WithReference(catalog)
  .WithReference(basket);
```

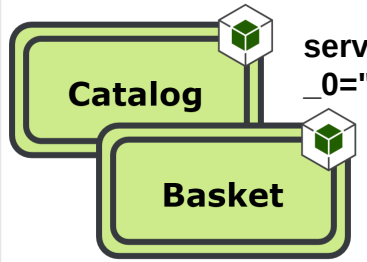
Built-in resource types

.NET Aspire

Service Discovery

Built-in resource types

Method	Resource type	Description
AddProject	ProjectResource	A .NET project, for example, an ASP.NET Core web app.
AddContainer	ContainerResource	A container image, such as a Docker image.
AddExecutable	ExecutableResource	An executable file, such as a Node.js app.
AddParameter	ParameterResource	A parameter resource that can be used to express external parameters.



services__apiservice__http__0="http://localhost:5455"

ConnectionStrings__cache="localhost:62354"

Implicit Service Discovery by Reference

.NET Aspire

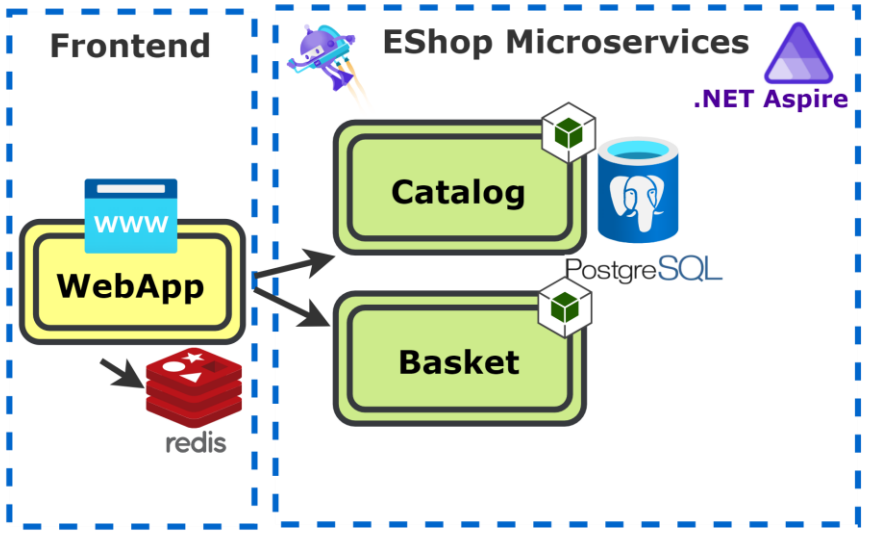
Service Discovery

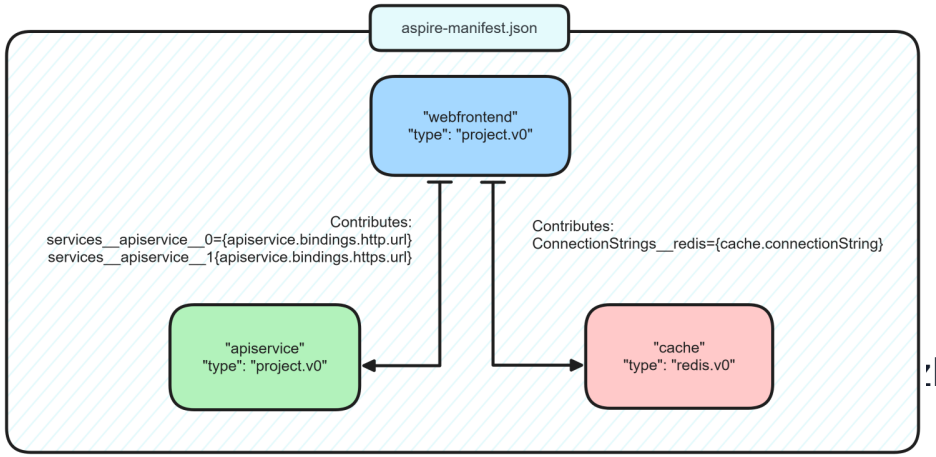
Containers→injects→connection string

Projects→injects→microservice endpoints

```
var builder = DistributedApplication.CreateBuilder(args);

var cache = builder.AddRedis("cache");

var catalog = builder.AddProject<Projects.CatalogService>
("catalog");
var basket = builder.AddProject<Projects.BasketService>("basket");

var frontend = builder.AddProject<Projects.MyFrontend>("frontend")
  .WithReference(cache)
  .WithReference(catalog)
  .WithReference(basket);
```

```
WithReference(cache)
  ConnectionStrings__cache="localhost:62354"

WithReference(apiservice)
  services__apiservice__http__0="http://localhost:5455"

 services__apiservice__https__0="https://localhost:7356"
```





:kaya

Configuring Container Endpoints

.NET Aspire

Service Discovery

```
var builder = DistributedApplication.CreateBuilder(args);

var customContainer = builder.AddContainer("myapp", "mycustomcontainer")
                             .WithHttpEndpoint(port: 9043, name: "endpoint");

var endpoint = customContainer.GetEndpoint("endpoint");

var apiservice = builder.AddProject<Projects.AspireApp_ApiService>
("apiservice")           .WithReference(endpoint);
```

Implicit Service Discovery by Reference

.NET Aspire

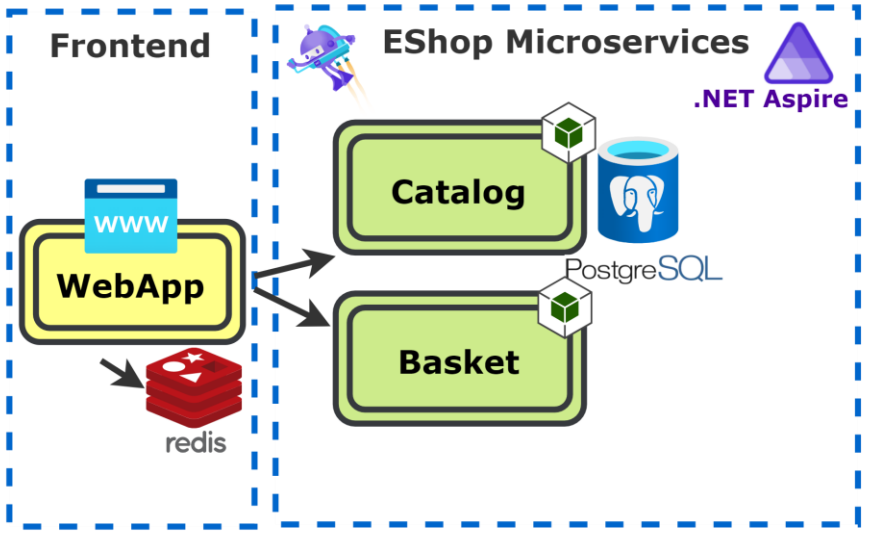
Service Discovery

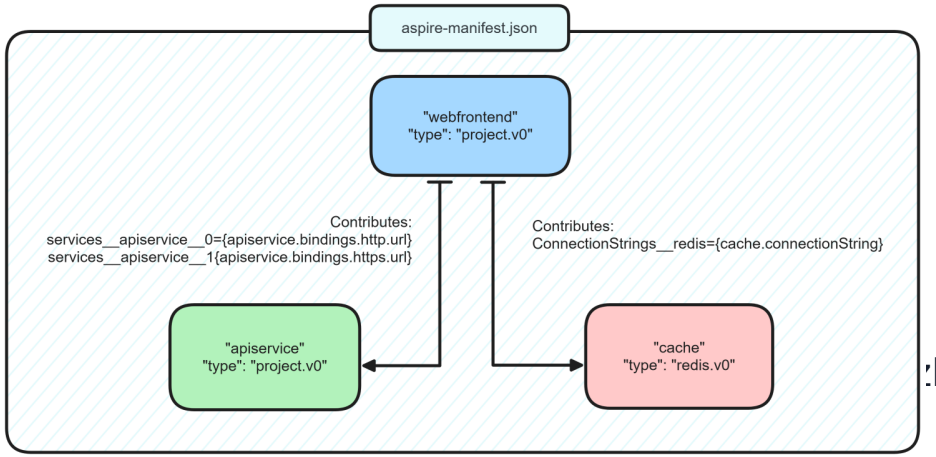
Containers→injects→connection string

Projects→injects→microservice endpoints

```
var builder = DistributedApplication.CreateBuilder(args);

var cache = builder.AddRedis("cache");

var catalog = builder.AddProject<Projects.CatalogService>
("catalog");
var basket = builder.AddProject<Projects.BasketService>("basket");

var frontend = builder.AddProject<Projects.MyFrontend>("frontend")
  .WithReference(cache)
  .WithReference(catalog)
  .WithReference(basket);
```

```
WithReference(cache)
  ConnectionStrings__cache="localhost:62354"

WithReference(apiservice)
  services__apiservice__http__0="http://localhost:5455"

 services__apiservice__https__0="https://localhost:7356"
```





:kaya

Building Your First .NET Aspire Application

Start the .NET Aspire Project for Redis-Api-Frontend Architecture

Run Aspire App and Explore the .NET Aspire dashboard

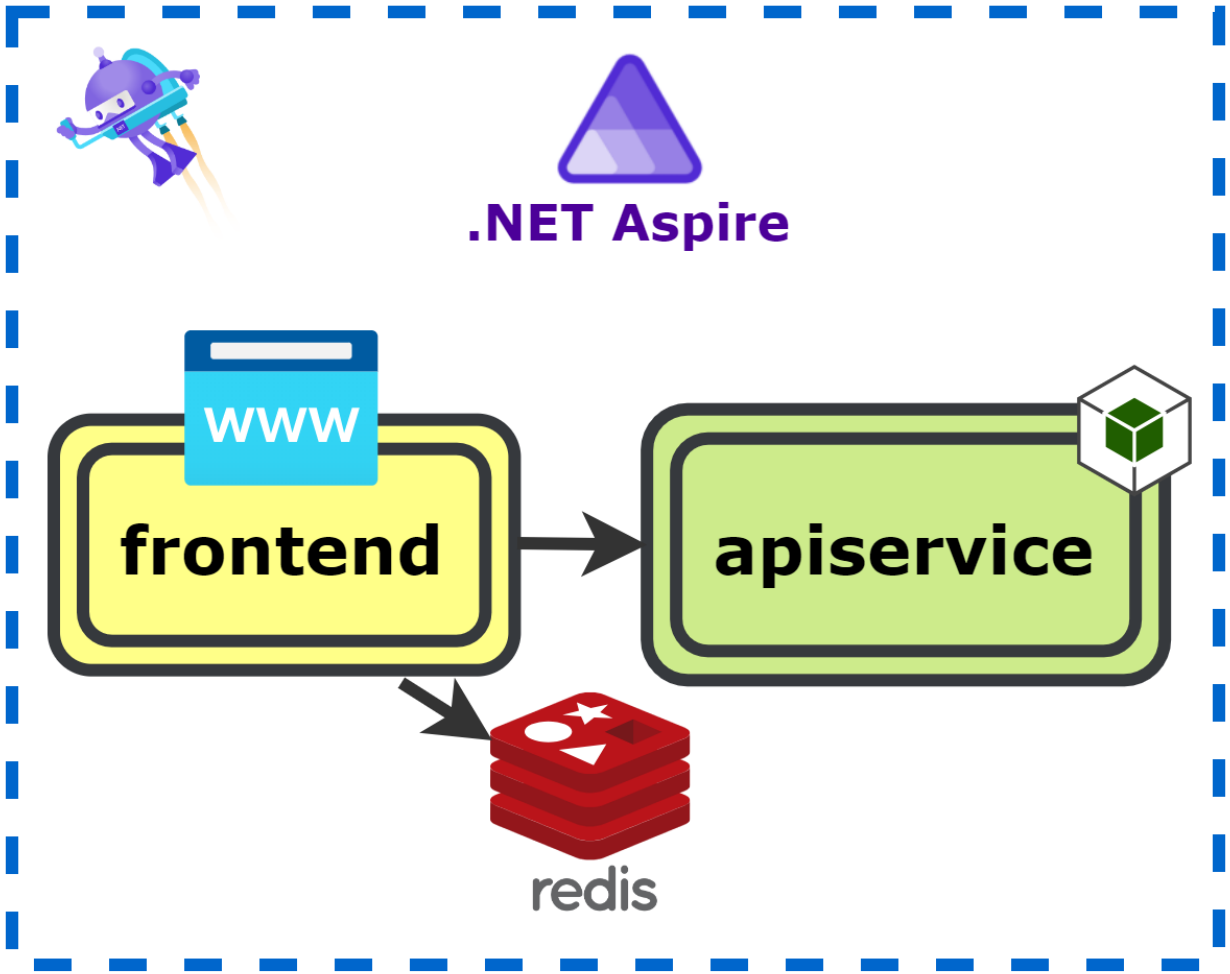
Understanding .NET Aspire Project Organization

Examine .NET Aspire Host Project

Examine .NET Aspire Service Defaults Project

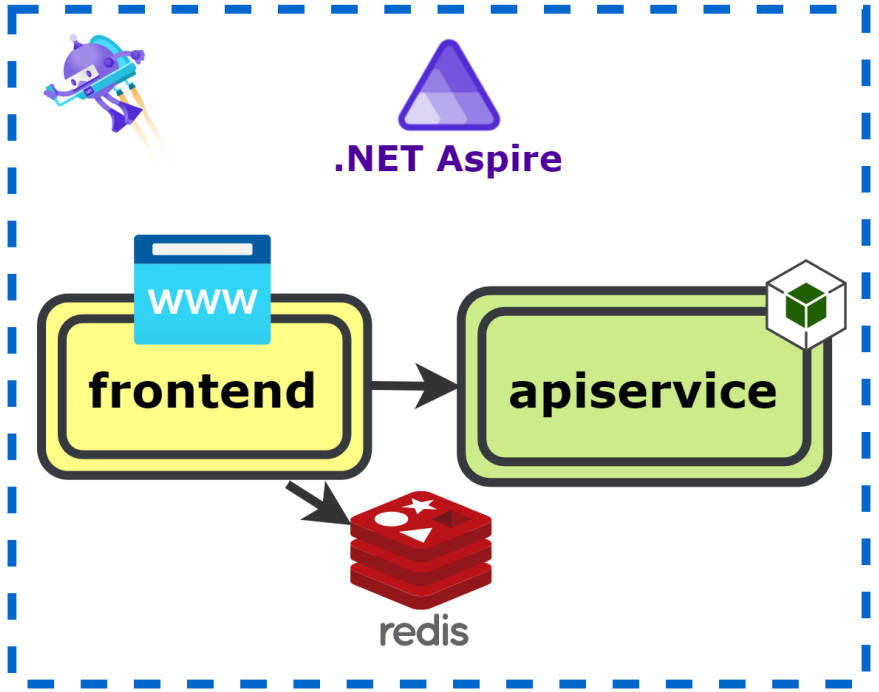
Mehmet Ozkaya

Start the .NET Aspire Project for Redis-Api-Frontend



Start the .NET Aspire Project for Redis-Api-Frontend

1. Create a brand-new .NET Aspire Starter App
2. Check Redis as our caching solution
3. Build an API project within the solution and see how Aspire handles service discovery
4. Connect everything—API, Frontend, and Redis—under one solution, all orchestrated by Aspire



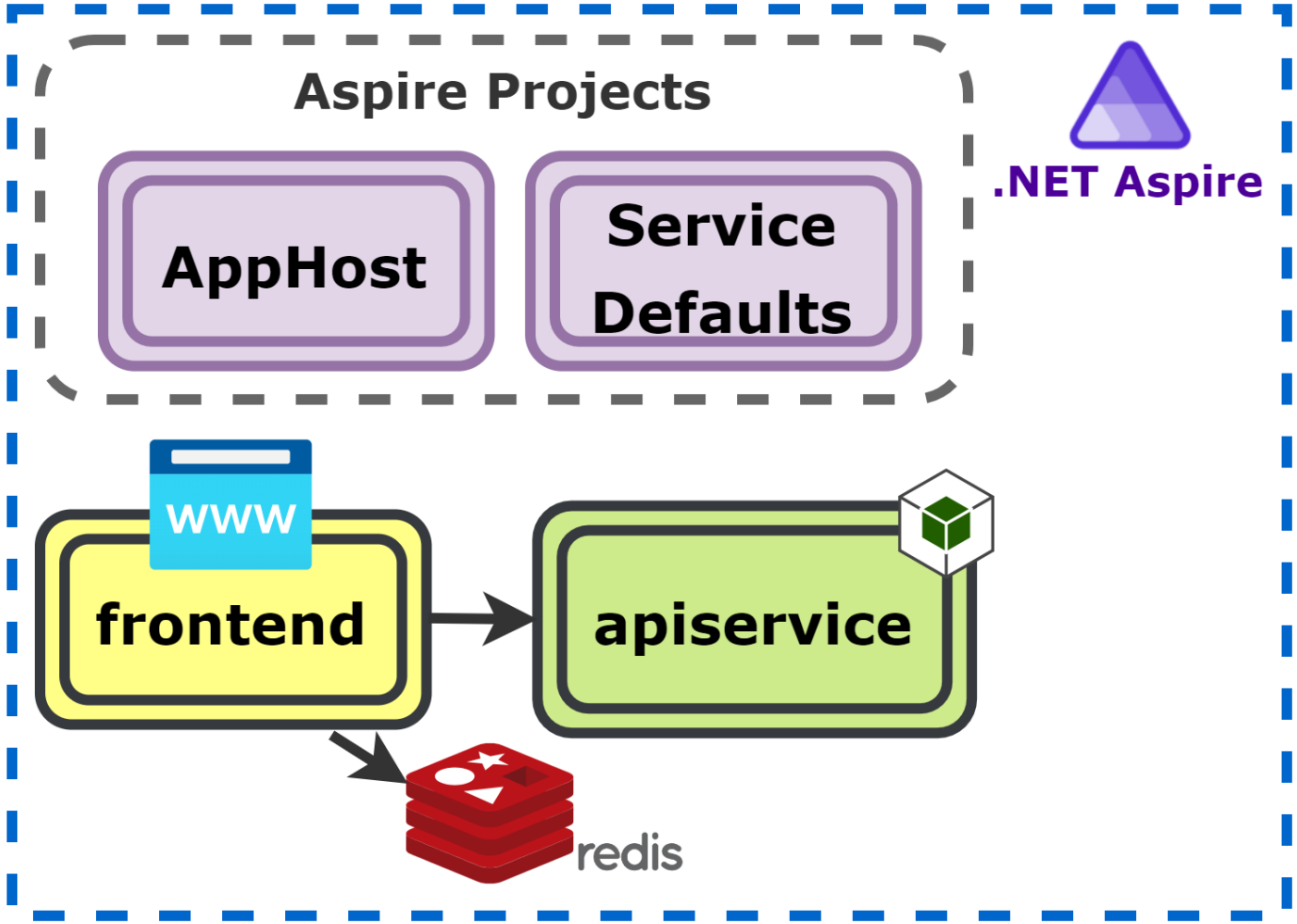
Start the .NET Aspire Project for Redis-Api-Frontend

AspireSample.AppHost

AspireSample.ServiceDefaults

AspireSample.ApiService

AspireSample.Web

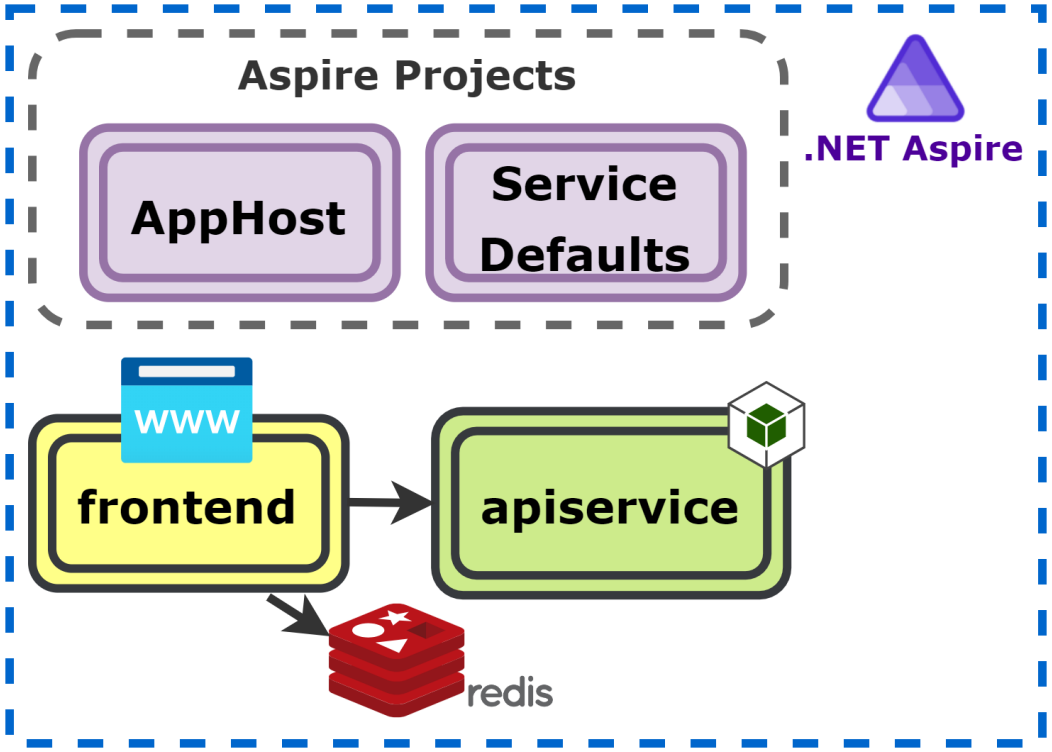


AspireSample.AppHost

Central Hub for **All Projects** & **Services**

Program.cs Defines **Orchestration**

Must Be **Set** as the **Startup Project**



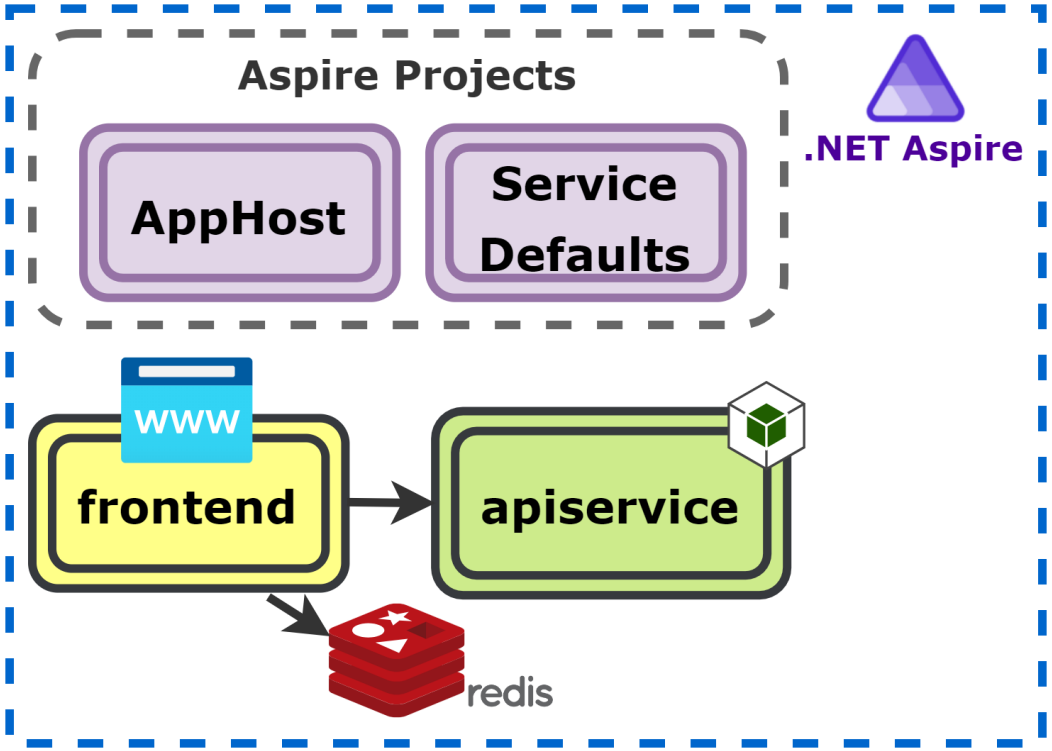
AspireSample.ServiceDefaults

Centralized Cross-Cutting Concerns

Logging, **Tracing**, **Health Checks**

Service Discovery defaults

Referenced by **ApiService** & **Web**

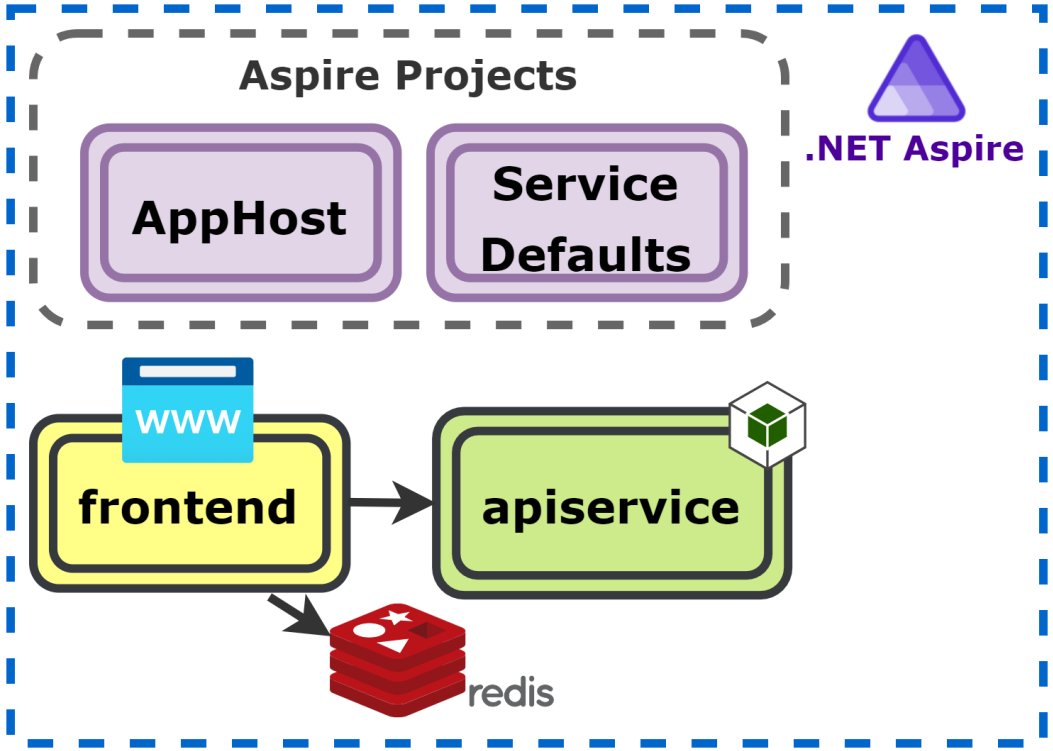


AspireSample.ApiService

Exposes endpoints that the Web project can call

Depends on the shared **ServiceDefaults** for **cross-cutting** concerns

Typical ASP.NET Core app, **minimal** set of **Routes** for **data retrieval**



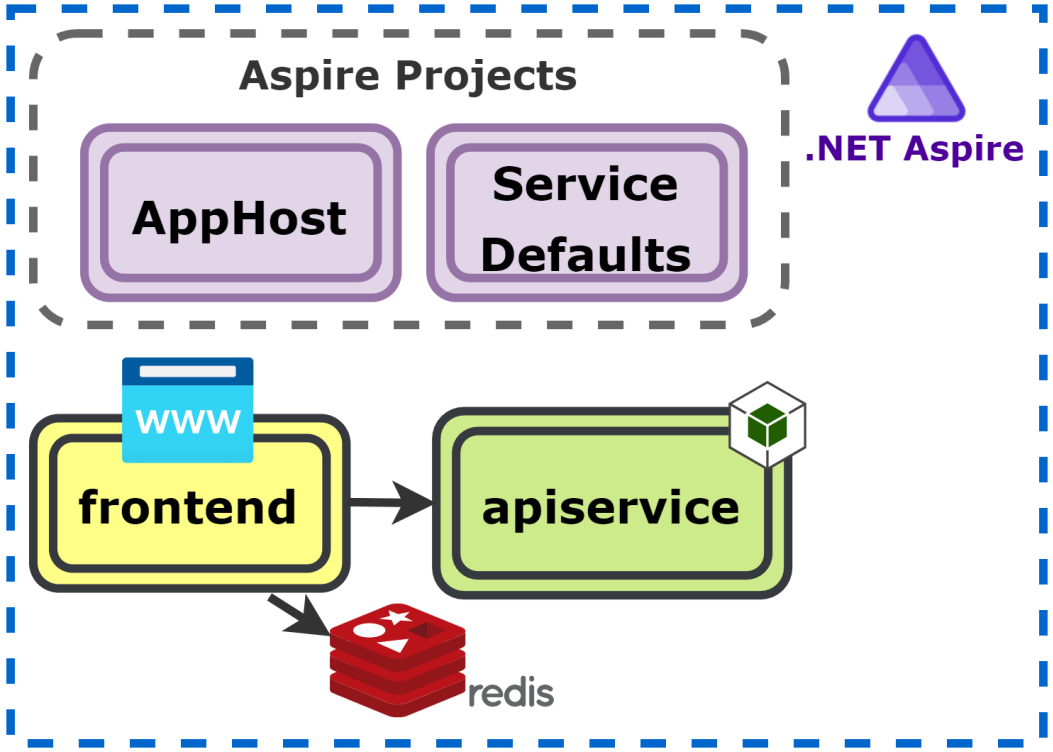
AspireSample.Web

Houses **UI components**, **layouts** and **pages**

Depends on the shared **ServiceDefaults** for **cross-cutting** concerns

Communicates with the **ApiService**—strongly typed **HTTP client** to fetch data

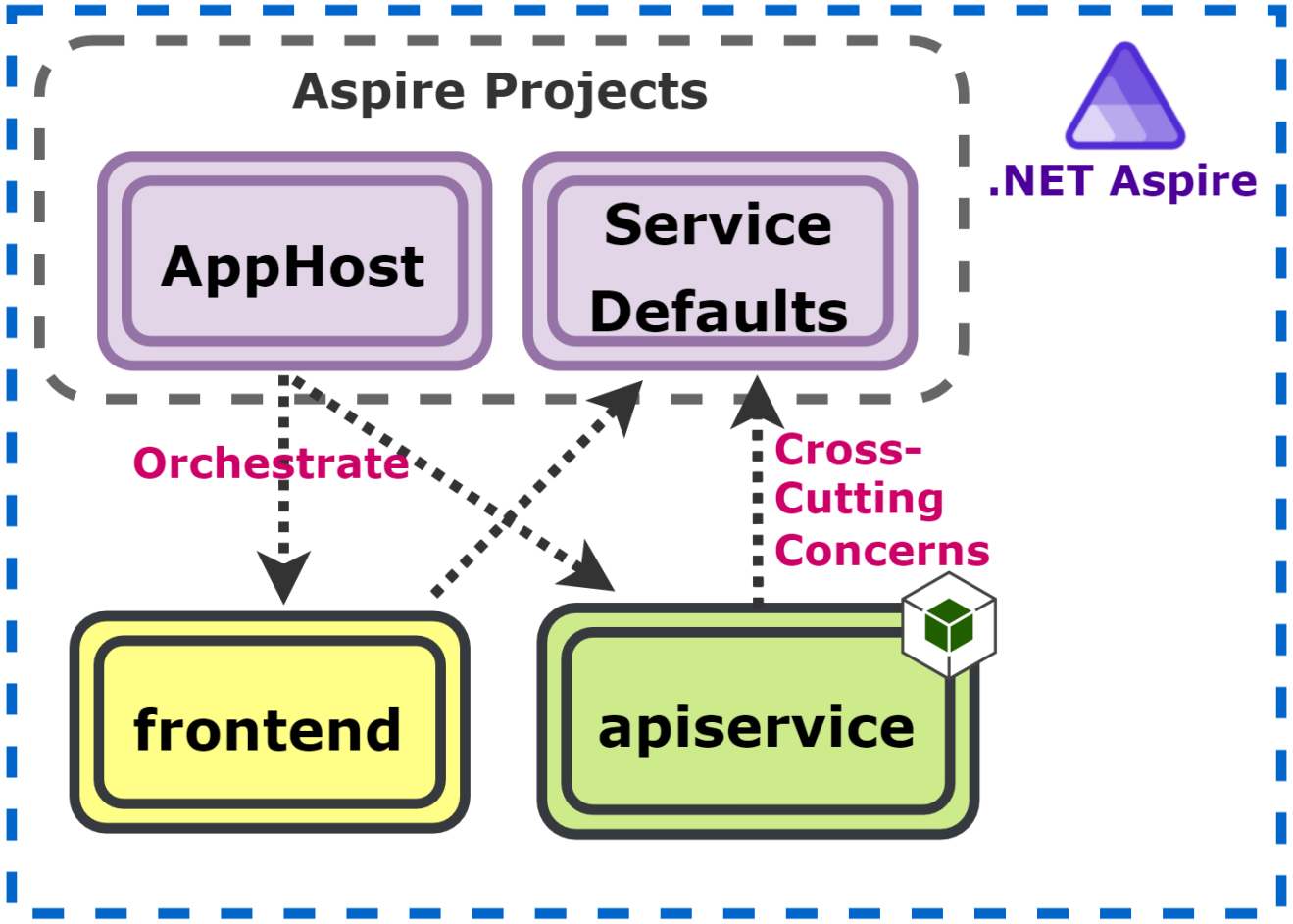
Discovered via .NET Aspire's **built-in service discovery**



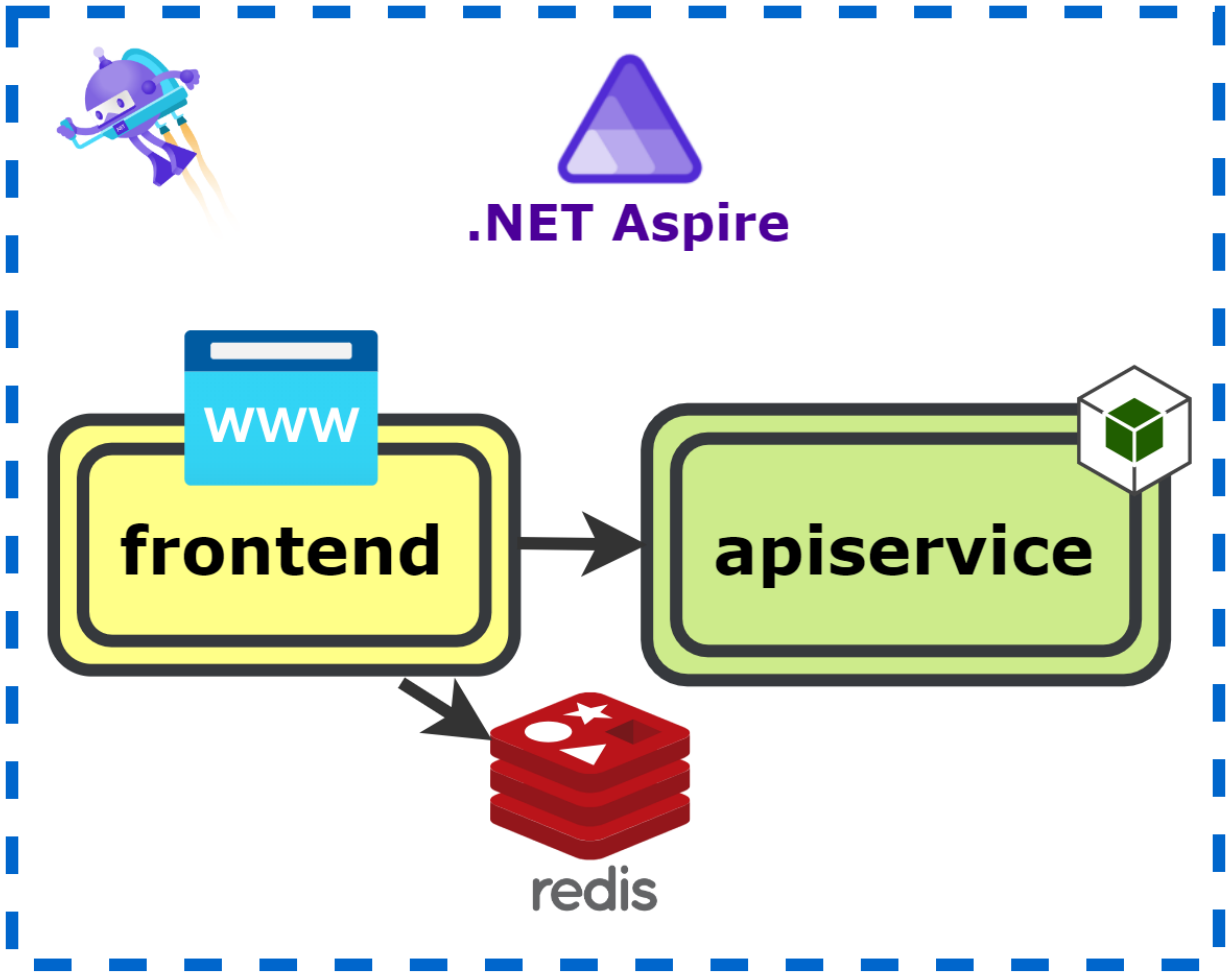
Project Dependencies

AppHost → references ApiService & Web for orchestration

ApiService/Web → reference ServiceDefaults for cross-cutting concerns



Start the .NET Aspire Project for Redis-Api-Frontend



Remember: Service Discovery

.NET Aspire

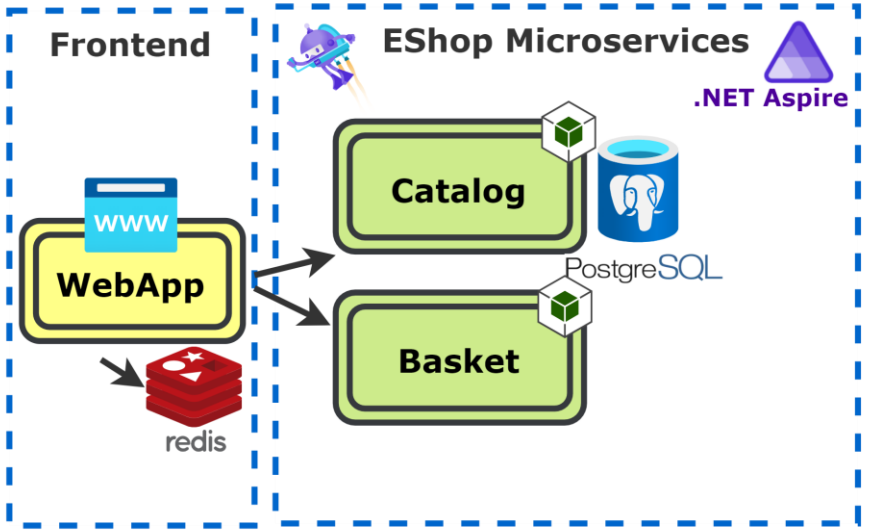
Service Discovery

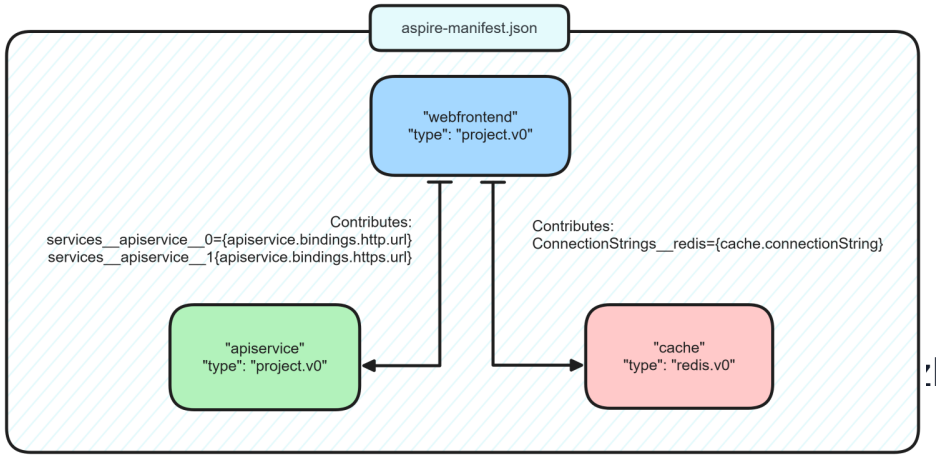
Containers→injects→connection string

Projects→injects→microservice endpoints

```
var builder = DistributedApplication.CreateBuilder(args);

var cache = builder.AddRedis("cache");

var catalog = builder.AddProject<Projects.CatalogService>
("catalog");
var basket = builder.AddProject<Projects.BasketService>("basket");

var frontend = builder.AddProject<Projects.MyFrontend>("frontend")
  .WithReference(cache)
  .WithReference(catalog)
  .WithReference(basket);
```

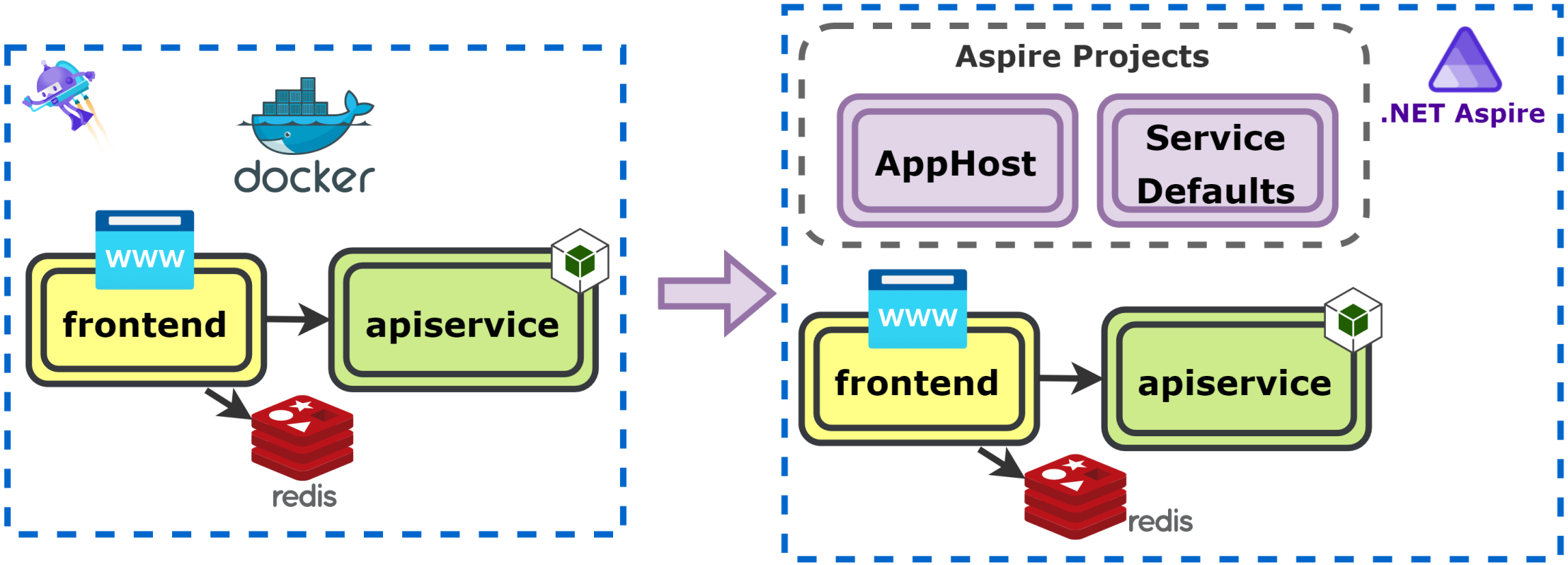
```
WithReference(cache)
  ConnectionStrings__cache="localhost:62354"

WithReference(apiservice)
  services__apiservice__http__0="http://localhost:5455"

 services__apiservice__https__0="https://localhost:7356"
```





:kaya

Adding .NET Aspire to Existing .NET Applications



Building EShop Distributed Microservices Architecture with .NET Aspire

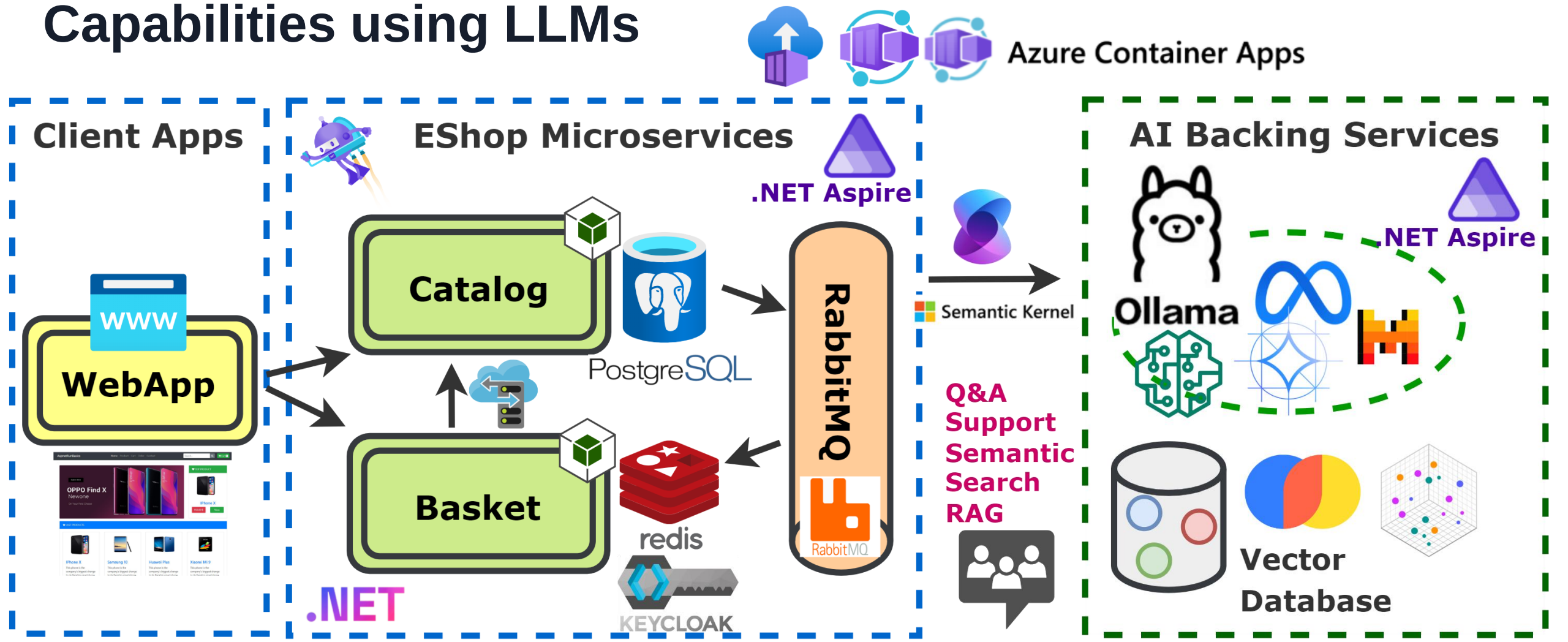
Create New Blank Solution and Organize Solution Folder Structure

Create .NET Aspire Projects: AppHost (Orchestrator), ServiceDefaults(Shared config)

Step by Step Development Plan for Building EShop Distributed Architecture with .NET Aspire

Mehmet Ozkaya

Course Project: EShop Microservices with AI-Powered Capabilities using LLMs



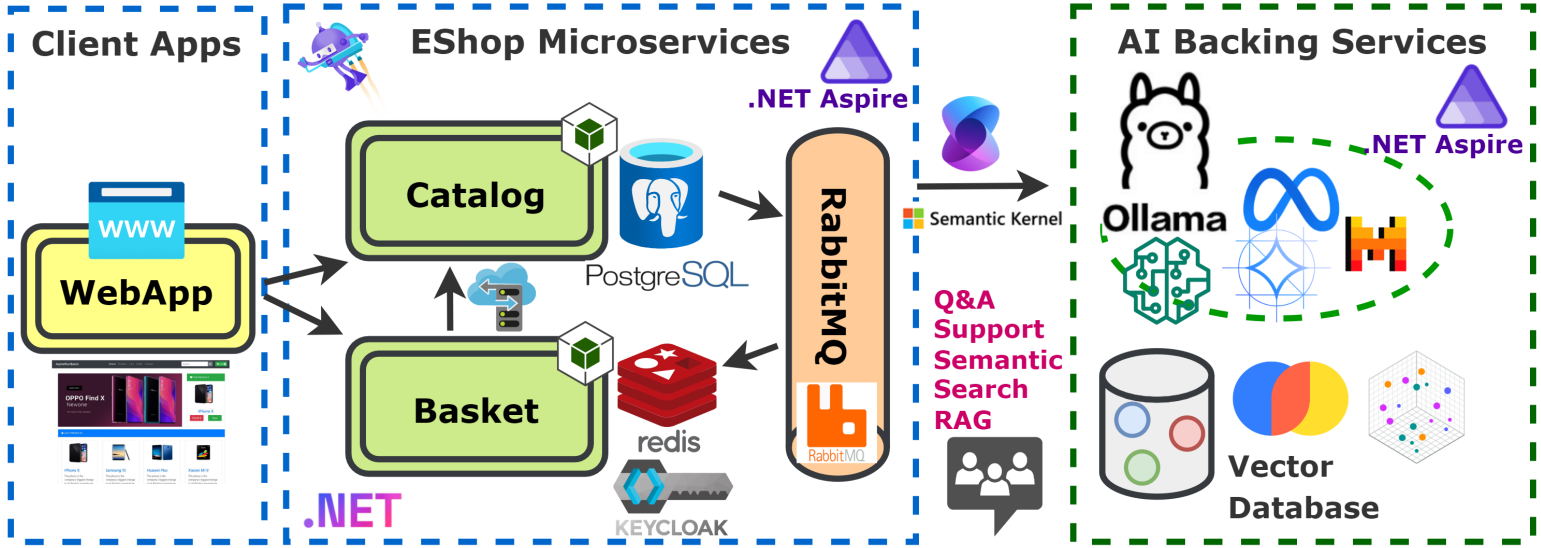
During this section

Create **New Blank Solution**

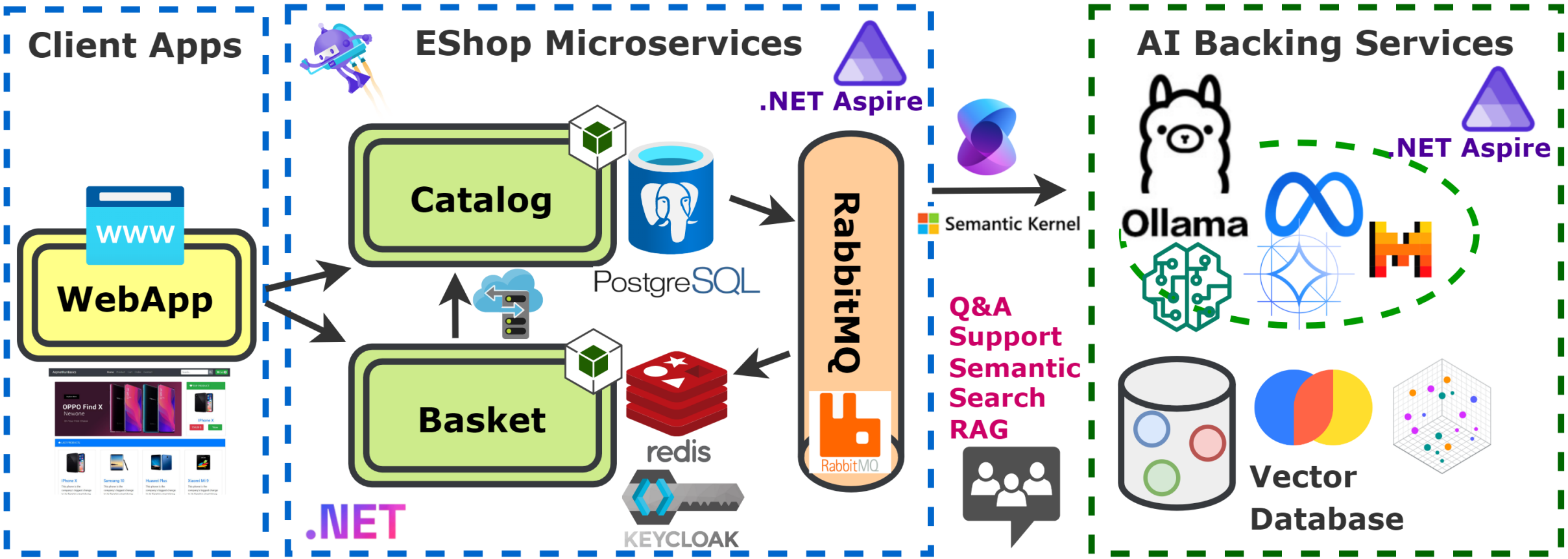
Create **Visual Studio Solution Structure**: **aspire**, **services**, **frontend**

Create **.NET Aspire Projects**

Make **Step by Step Development Plan**



Course Project: EShop Microservices with AI-Powered Capabilities using LLMs



Step by Step Development Plan

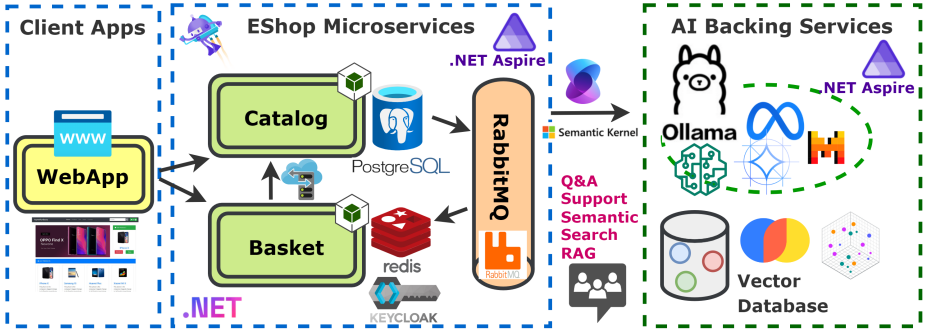
Setting Up Core Microservices
Catalog and Basket Microservices Development w/ Backing Services

Microservices Communications with .NET Aspire
Synchronous and Asynchronous Messaging between Catalog & Basket using RabbitMQ Message Broker

Authentication & Security
Secure Basket Endpoints w/ Keycloak using OpenID Connect Jwt Tokens

Frontend Development
Blazor WebApp Products page development

Deployment to Azure Container Apps
.NET Aspire project to deploy ACA using azd up/down



Adding GenAI Features
Integrate Ollama LLMs and Vector Store for Semantic Search

Develop Catalog Microservice with PostgreSQL orchestrate in .NET Aspire

Domain and Technical Analysis of Catalog Microservice

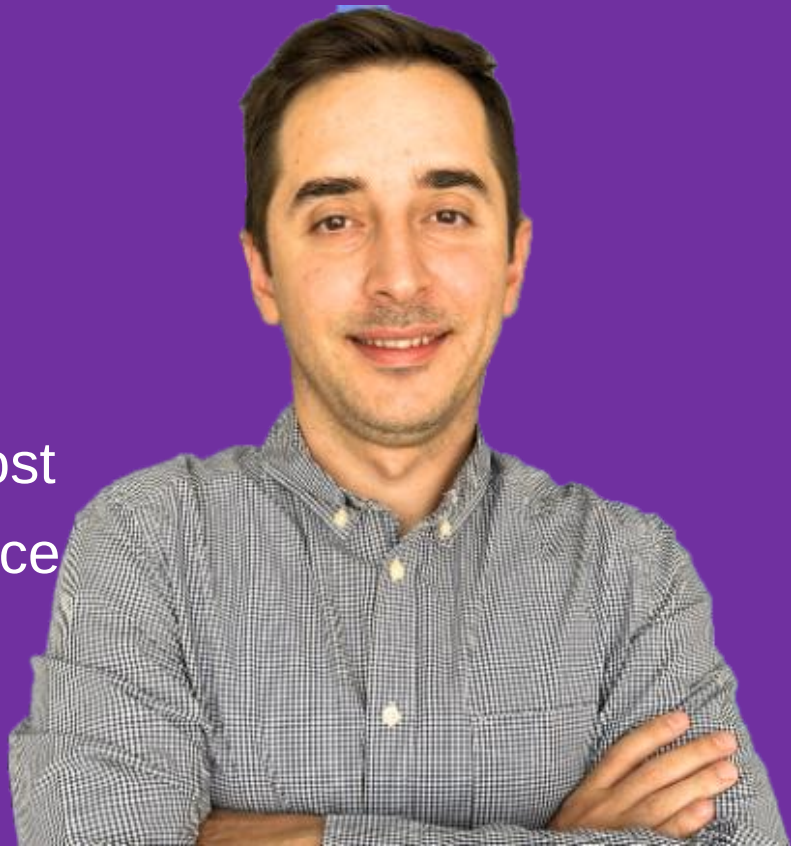
Create Catalog Microservice Web API and Add .NET Aspire

Catalog Hosting Integration: Add PostgreSQL Resource in AppHost

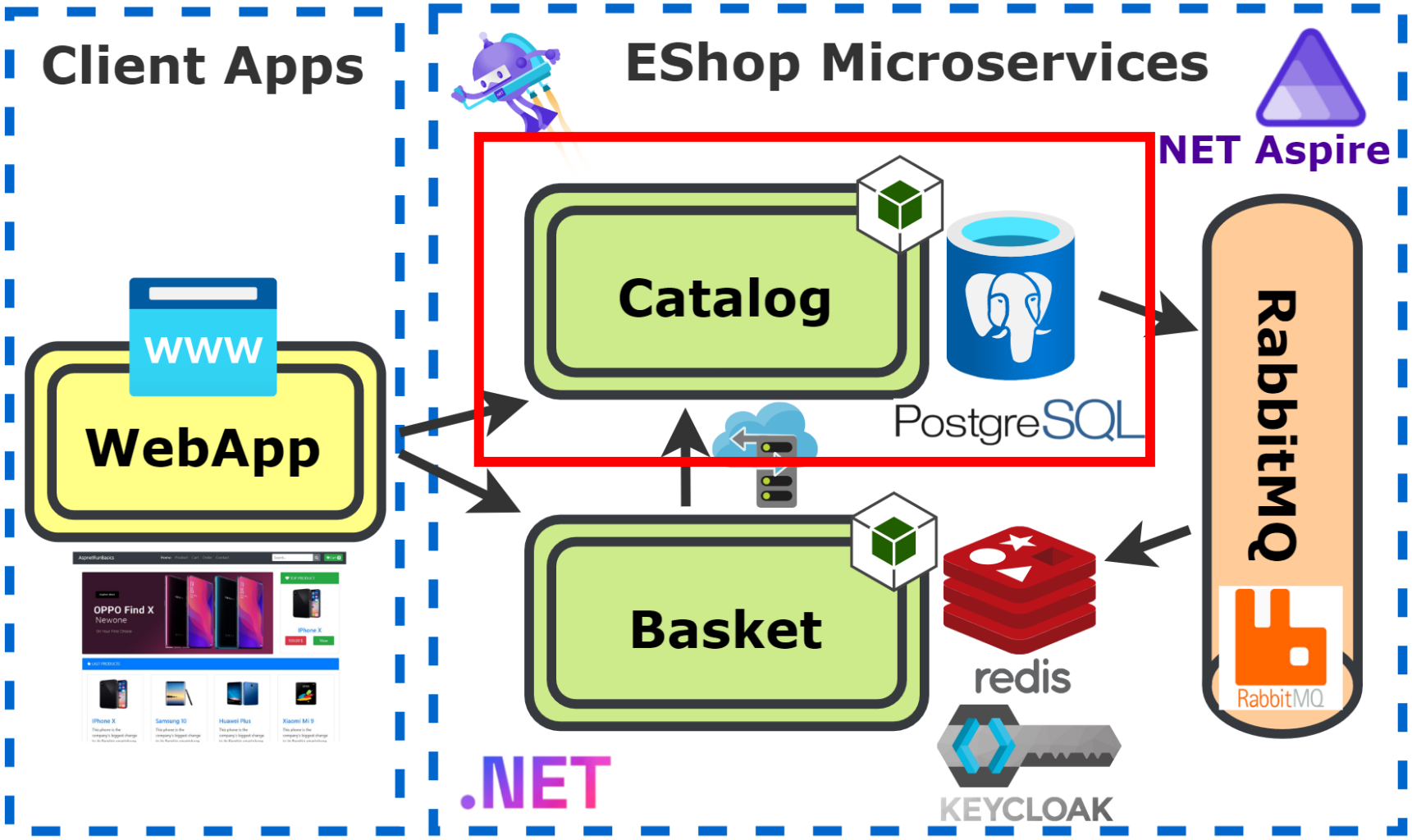
Catalog Client Integration: Connect PostgreSQL DB in Microservice

Develop and Inject EF Core ProductDbContext.cs in Data Layer

Mehmet Ozkaya



Catalog Microservice with PostgreSQL



Catalog Microservice with PostgreSQL

Domain Analysis of Catalog Microservices: Models, UCs, Rest Apis, Databases

Technical Analysis of Catalog Microservices: Architectures, Patterns, Libraries, Folders

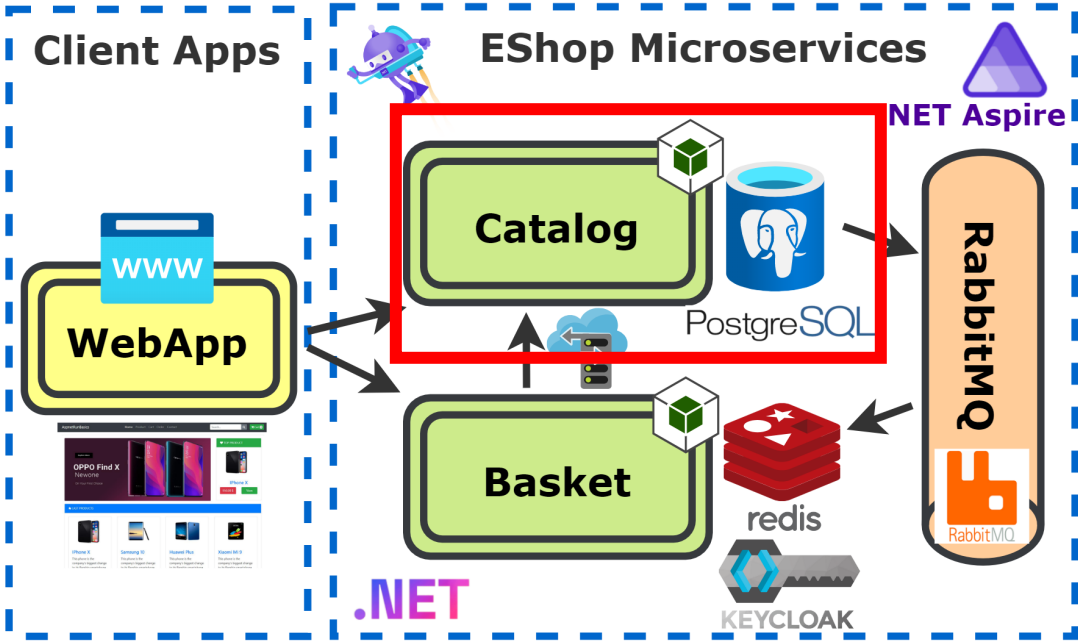
Develop **Product Domain Entity Models**

Hosting Integration(Backing Services)

Client Integration(EF Core Connection)

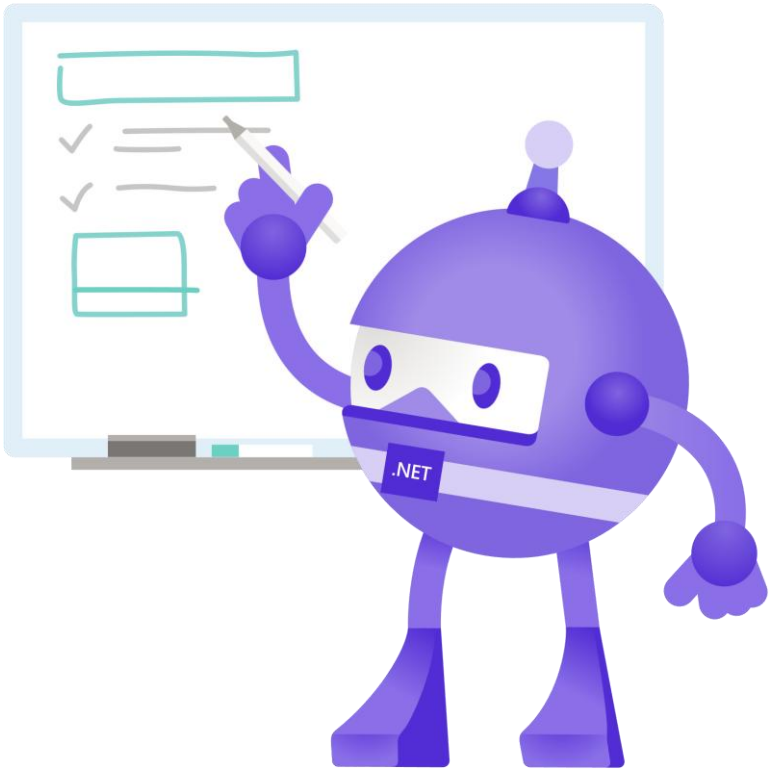
Catalog **Application Use Case** Development

Develop **Catalog API Endpoints** Exposing Minimal Api Endpoints

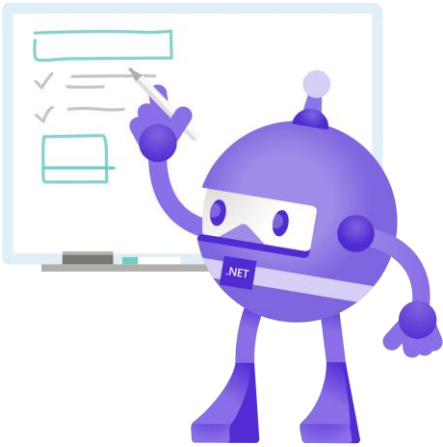


Domain Analysis of Catalog Microservice

1. Domain Models
2. Application Use Cases
3. Rest API Endpoints
4. Underlying Data Structures



1. Domain Models of Catalog Microservice



- Primary domain model is '**Product**'

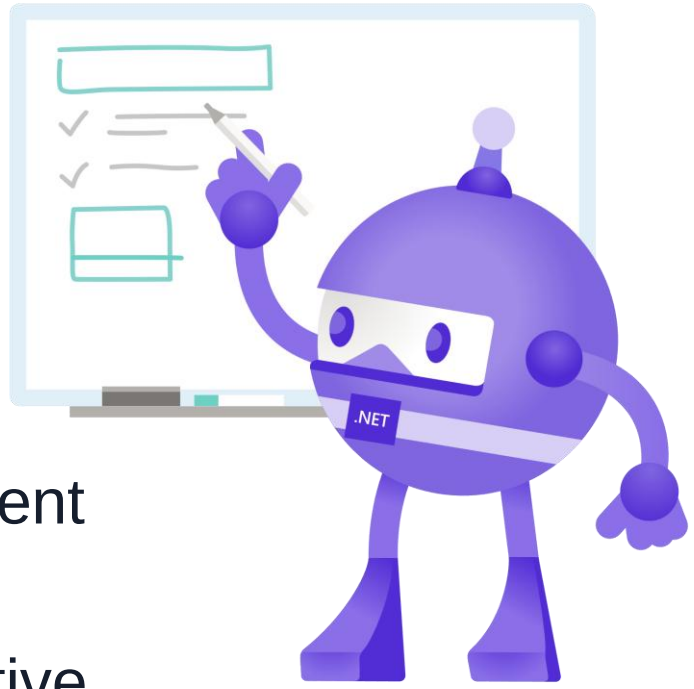
```
public class Product
{
    public int Id { get; set; }
    public string Name { get; set; } = default!;
    public string Description { get; set; } = default!;
    public decimal Price { get; set; }
    public string ImageUrl { get; set; } = default!;
}
```

- Consider Domain event for **ProductPriceChanges**, leading to integration events.

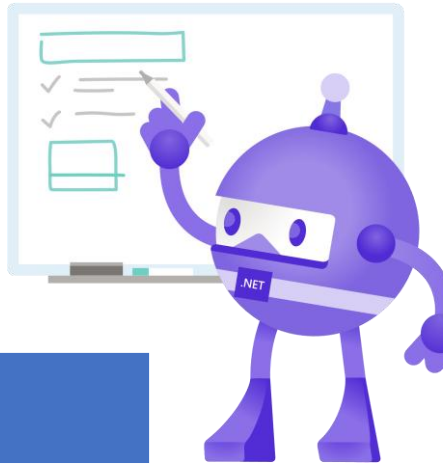
2. Application Use Cases of Catalog Microservice

CRUD Operations:

- Listing Products
- Get Product with Product Id
- Create new Product
- Update Product
- Delete Product
- If the price has changed, raise a **ProductPriceChanged** event that will lead to an integration event.
- Along with this design our APIs according to REST perspective.



3. Rest API Endpoints of Catalog Microservices

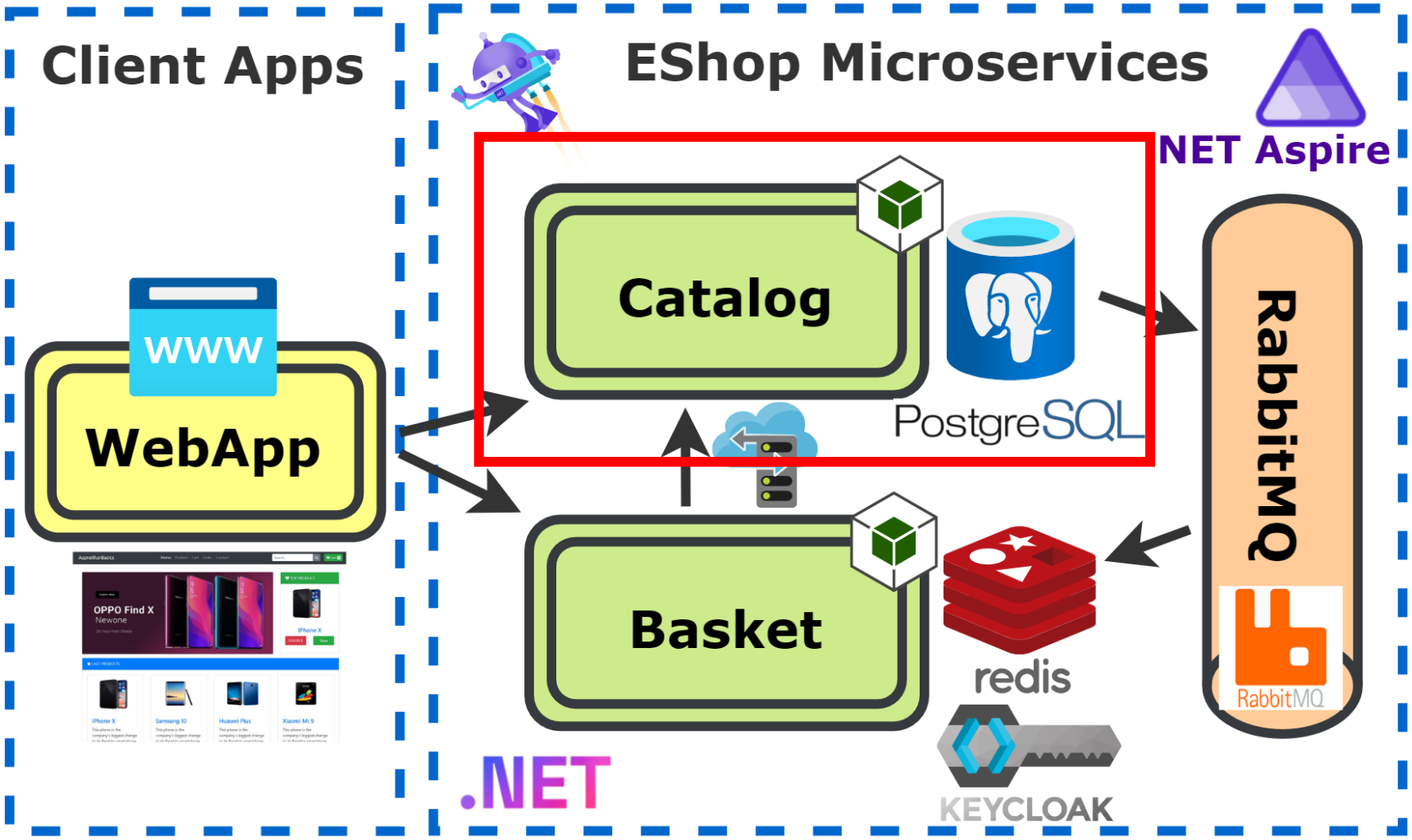


Method	Request URI	Use Cases
GET	**/products**	List all products
GET	**/products/{id}**	Fetch a specific product
POST	**/products**	Create a new product
PUT	**/products/{id}**	Update a product
DELETE	**/products/{id}**	Remove a product

4. Underlying Data Structures of Catalog Microservices

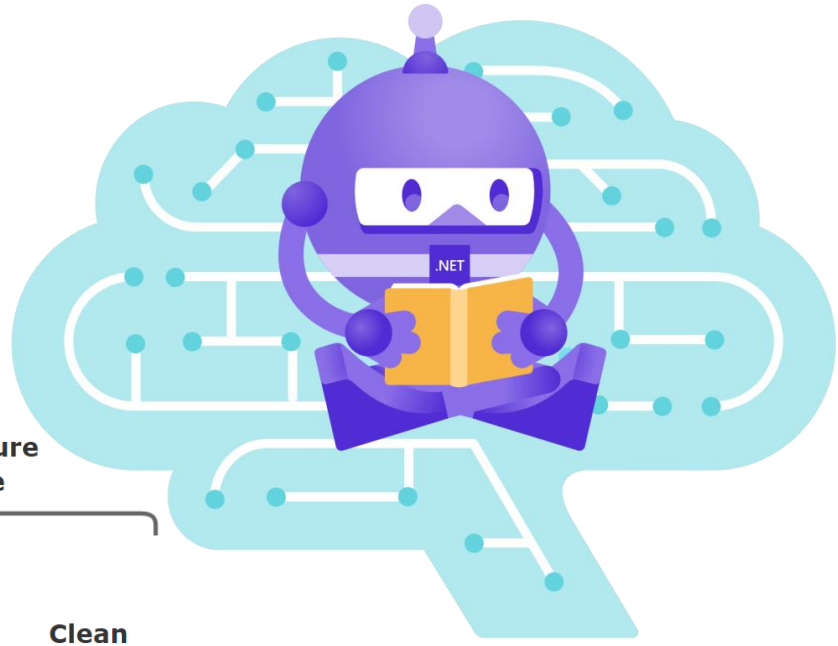
- **Catalog Microservice** will be stored in a **PostgreSQL** relational database.
- Use **Entity Framework Core's Code-First approach** for **database migrations** and **entity configurations**.
- **Configure** our **entities** and their **relationships** using **EF Core**.

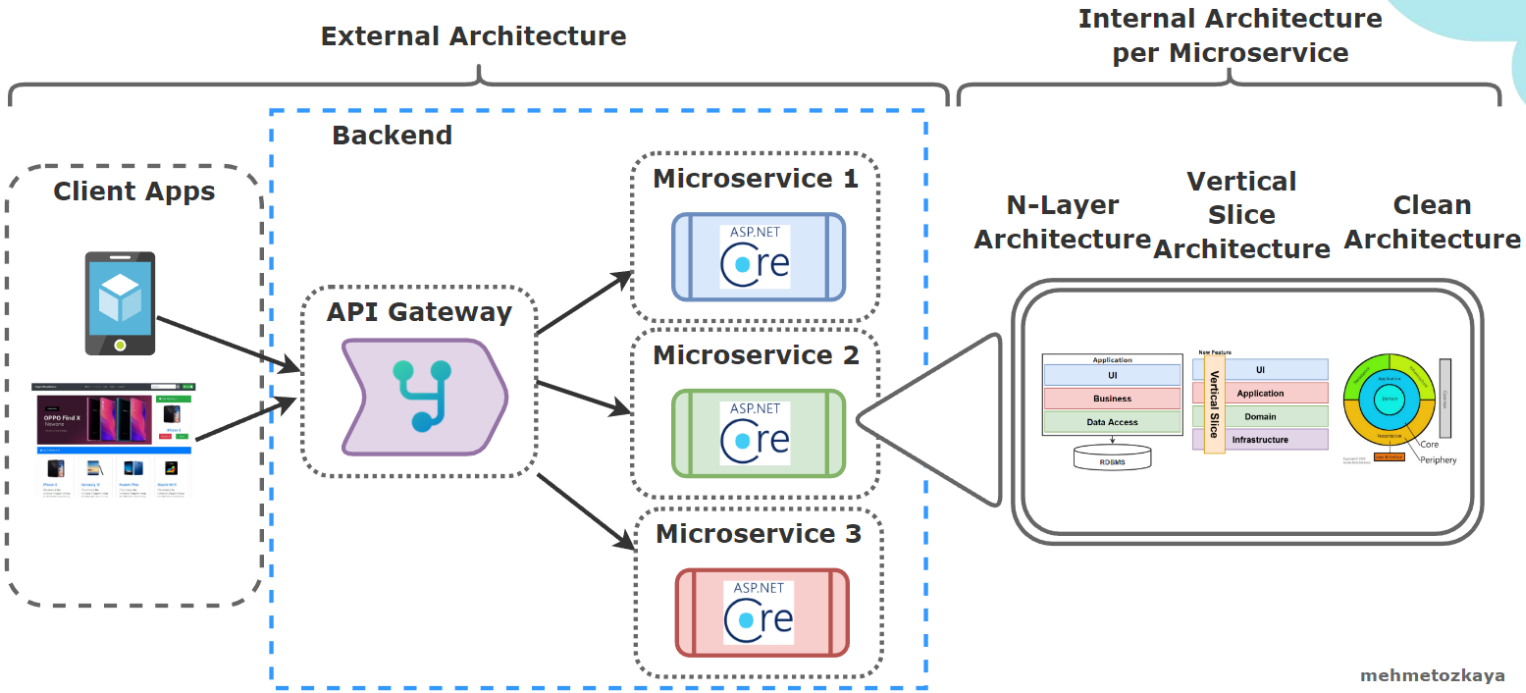
Catalog Microservice with PostgreSQL



Technical Analysis of Catalog Microservice

1. Application Architecture Style
2. Patterns & Principles
3. Libraries Nuget Packages
4. Project Folder Structure

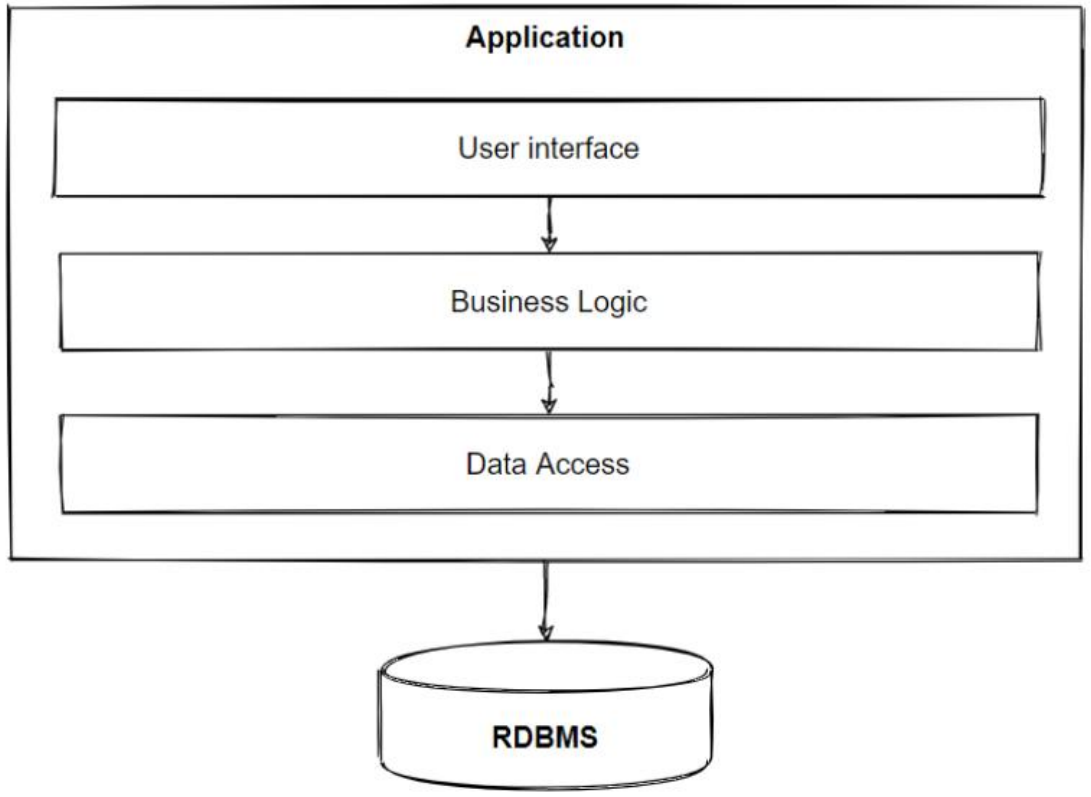


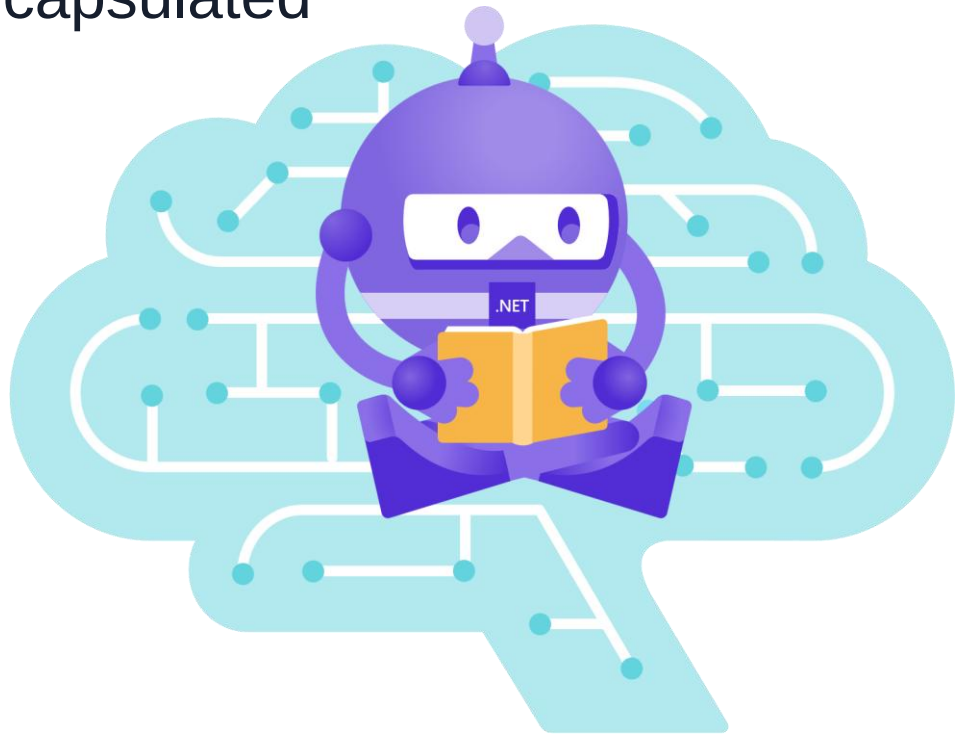


1. Application Architecture Style of Catalog Microservice

Classical N-Layer Architecture

Organizes our code into layer folders, each feature encapsulated in a folder





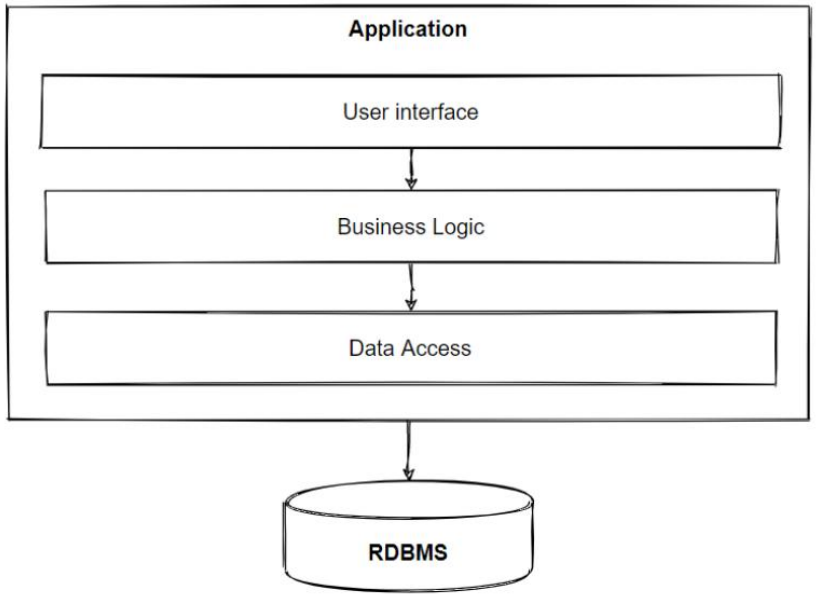
N-Layer(Layered) Architecture

Domain Layer
Stores and Handles domain entities and domain events

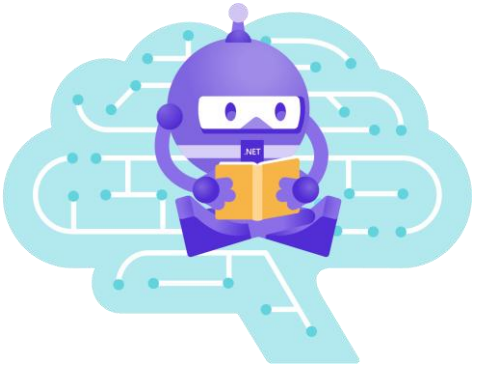
Data Access Layer
Manages interactions with the database or other storage systems

Business Logic Layer
Processes the application's core business logic and rules

Presentation Layer
Displaying data to users and capturing user input



2. Patterns & Principles of Catalog Microservice



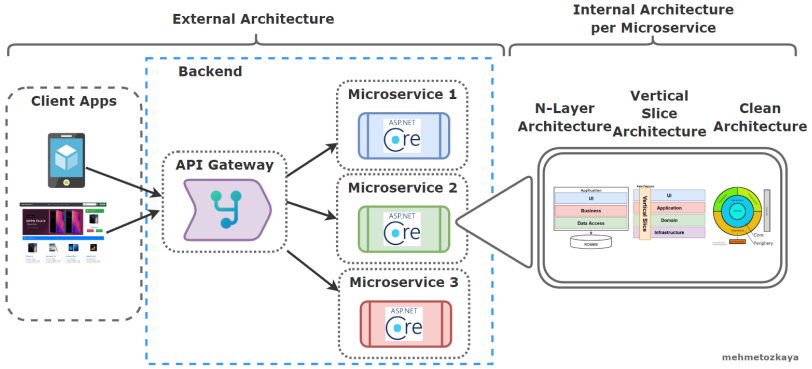
KISS principles and **use less patterns** in our **microservices**

Focus on integrations and **orchestrations** with **.NET Aspire**

DI in ASP.NET Core: Dependency Injection is a core feature, allowing us to inject dependencies.

Minimal APIs and Routing in ASP.NET 8: ASP.NET Minimal APIs simplify endpoint definitions, providing lightweight syntax for routing and handling HTTP requests.

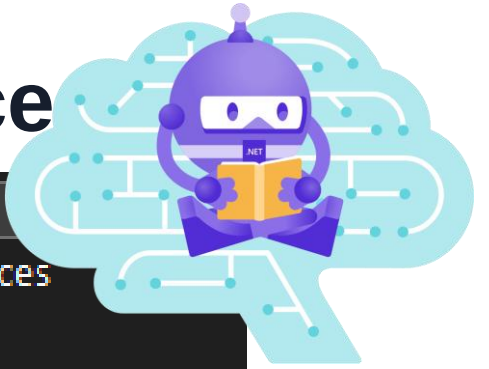
ORM Pattern: Object-Relational Mapping abstracts database interactions, work with database objects using high-level codes.



3. Libraries Nuget Packages of Catalog Microservic

- **Aspire Hosting Packages for Orchestration:** Aspire.Hosting.PostgreSQL
- **Aspire EF Core for PostgreSQL Interaction: (Client Integration) :** Aspire.Npgsql.EntityFrameworkCore.PostgreSQL
- EF Core provides a robust ORM for interacting with PostgreSQL, enabling us to manage database operations with ease.

4. Project Folder Structure of Catalog Microservice



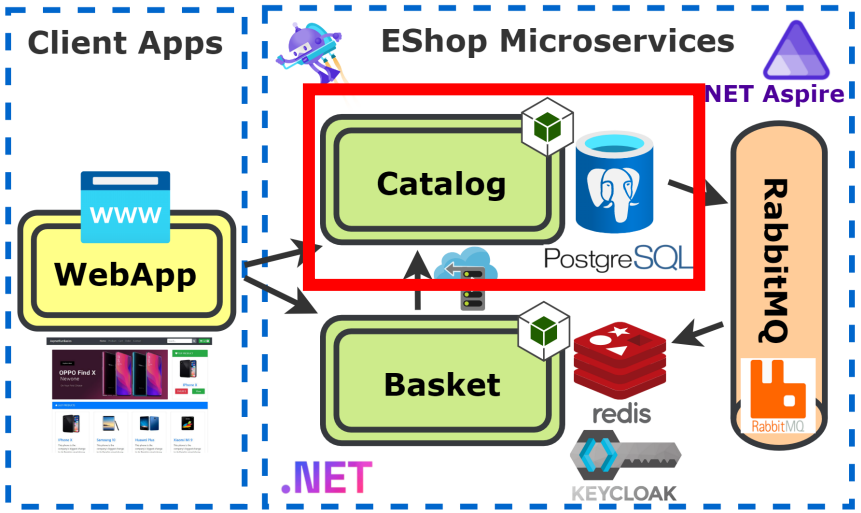
Organized into **Models**, **Data, Services** and **Endpoints**

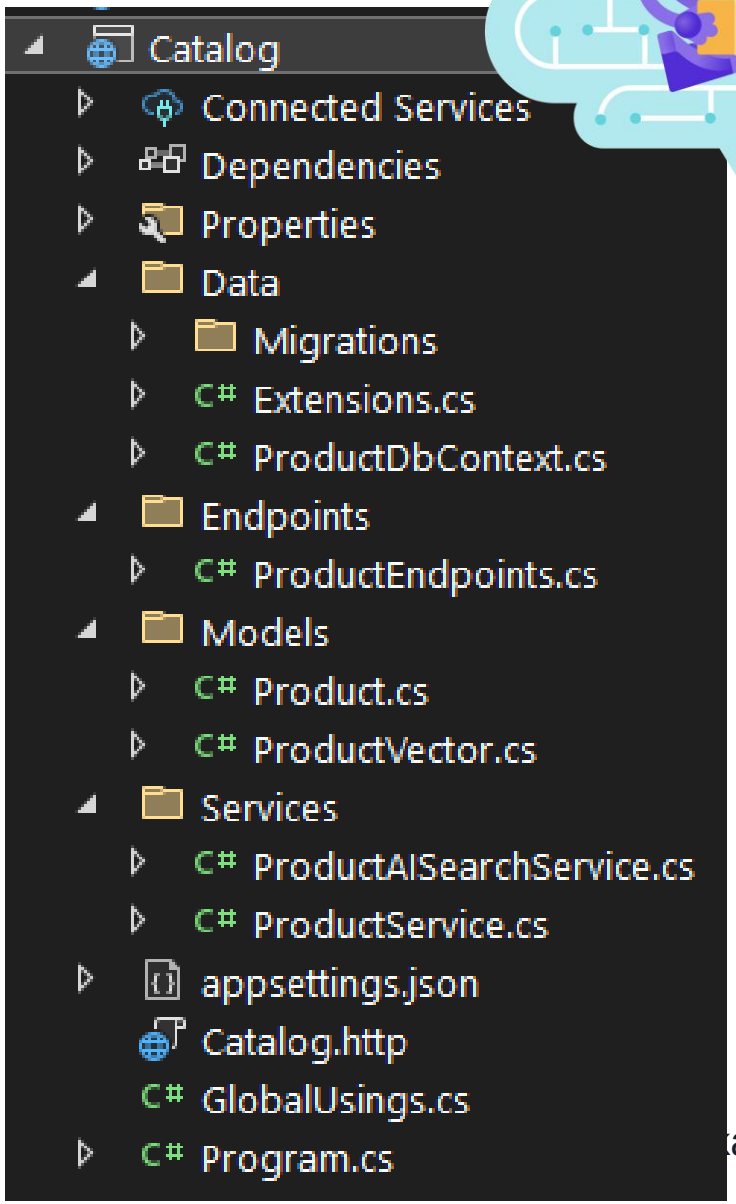
Models holds our **domain entities** (e.g., Product)

Data includes **DbContext** classes for data access logic

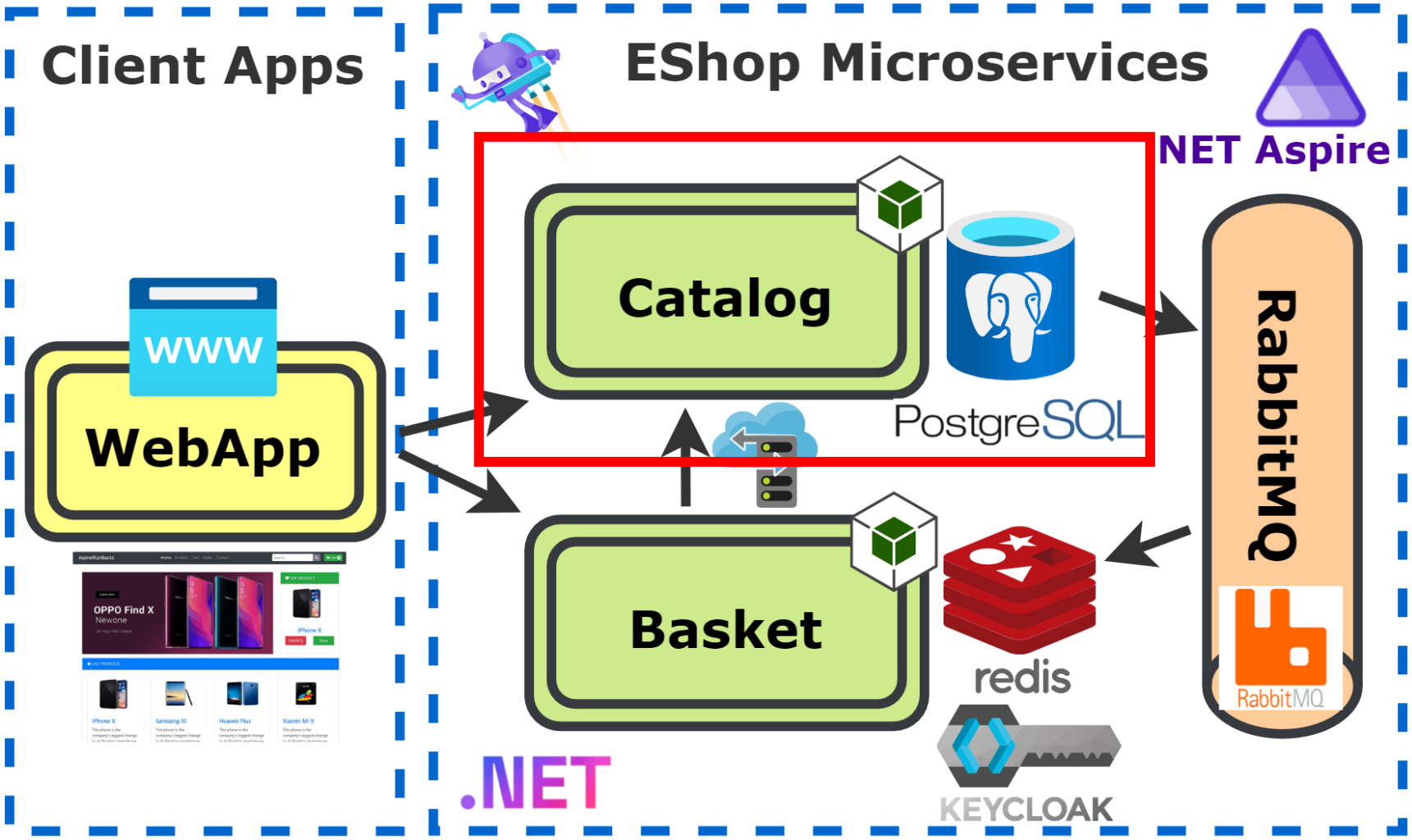
Services contains **business logic**

Endpoints minimal API endpoints that call into the services layer





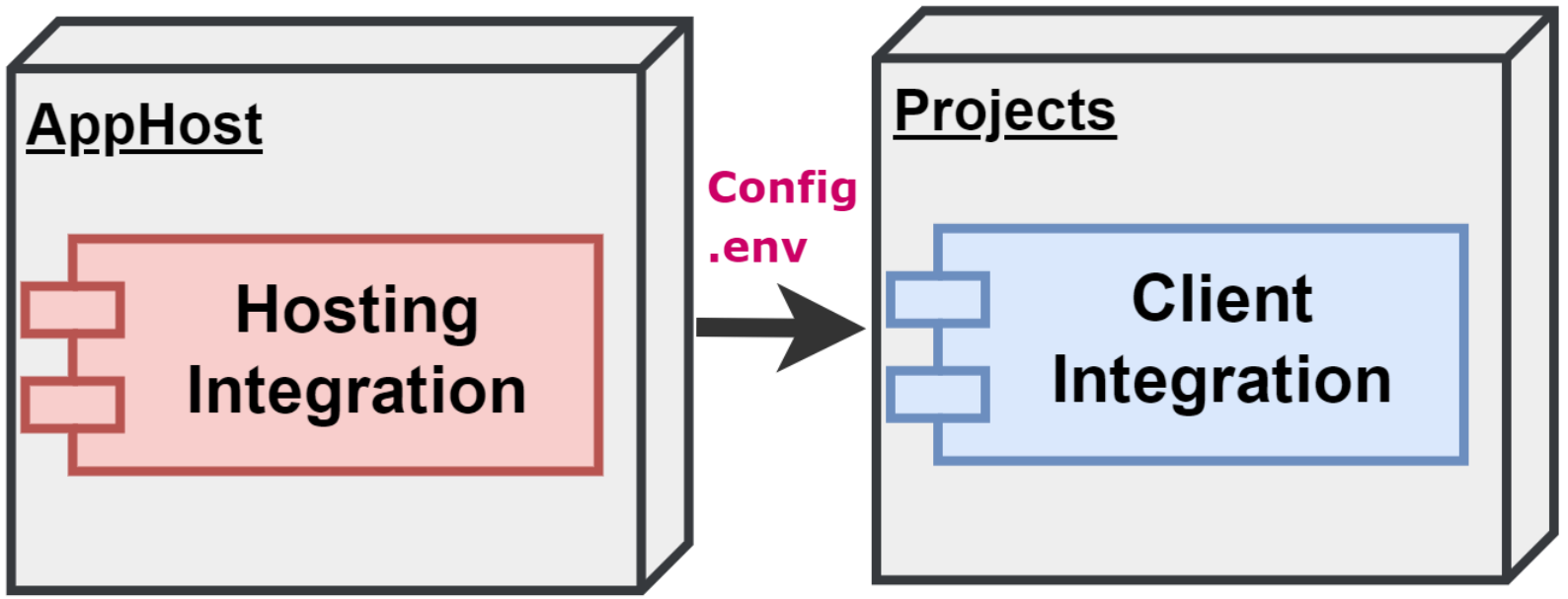
Catalog Microservice with PostgreSQL

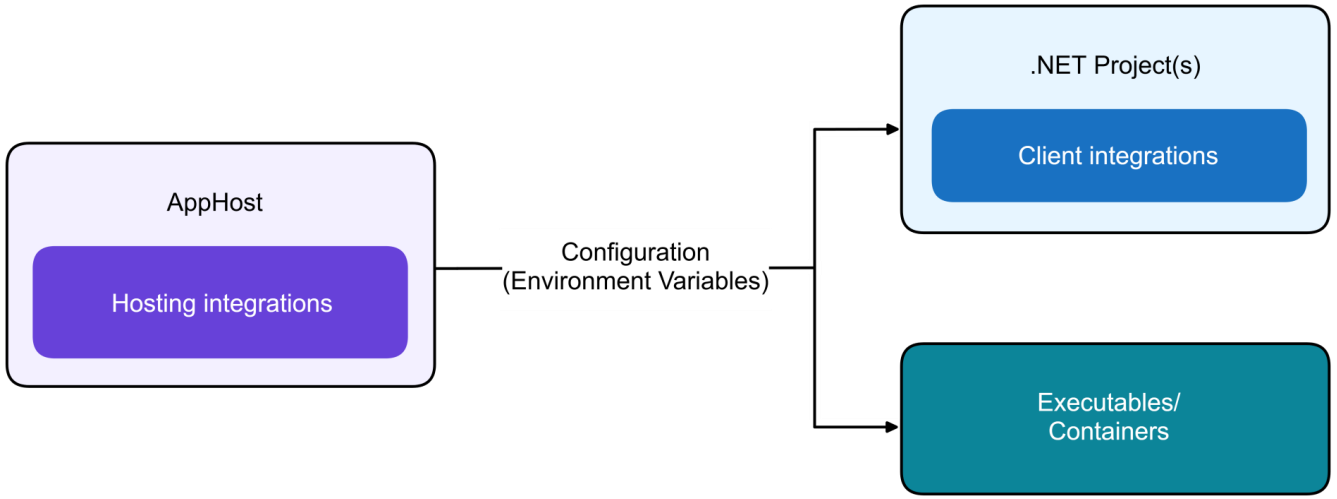


Hosting vs. Client Integrations

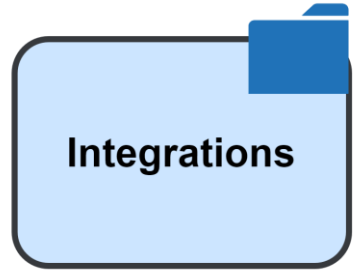
.NET Aspire

Integrations





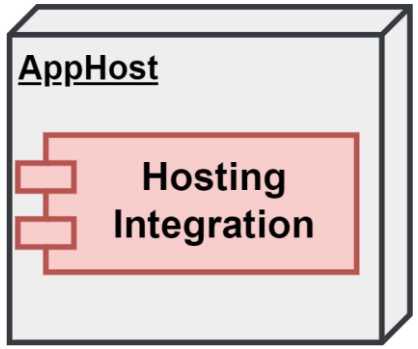
Hosting Integrations – Provisioning Resources



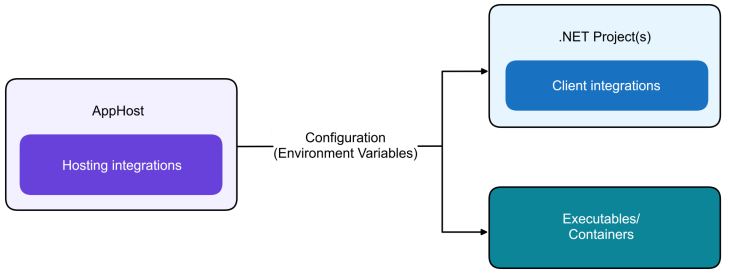
Extend the **IDistributedApplicationBuilder** interface, adding methods like **AddRedis("name")** or **AddPostgres("db")**

Provision container-based resources automatically

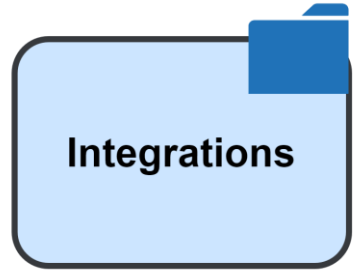
Use **environment variables** to **pass connection strings** to other projects that depend on them




```
var builder =
DistributedApplication.CreateBuilder(args);
// Hosting integration call:
var cache = builder.AddRedis("cache");

// This sets up a Redis container named "cache"
```



Client Integrations – Connecting to Resources

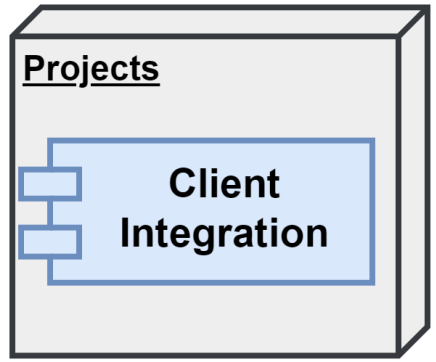


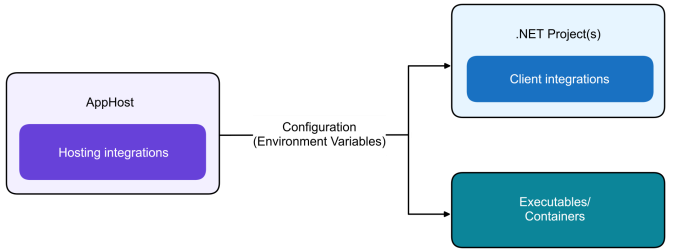
Extend **IHostApplicationBuilder** in Consuming Projects;
builder.Services.AddStackExchangeRedis(...)

Add **Health Checks**, **Logging**, **Telemetry**

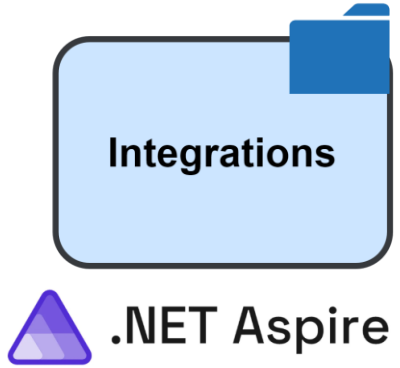
Example: Aspire.StackExchange.Redis

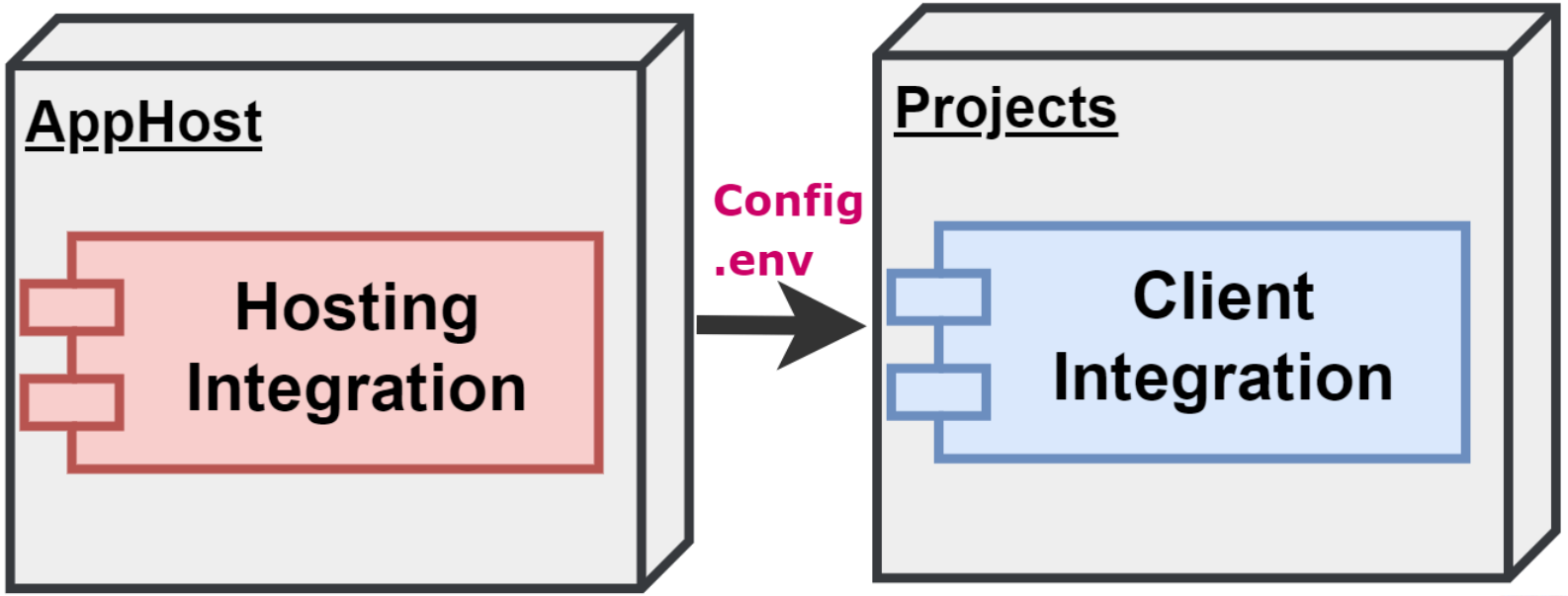
```
var builder = WebApplication.CreateBuilder(args);

// AddServiceDefaults is often called automatically in Aspire-based
// projects, but you can call it explicitly, too.
builder.AddServiceDefaults();

// This might be from Aspire.StackExchange.Redis
builder.AddRedisDistributedCache(connectionName: "cache");
```





Relationship Hosting vs. Client Integrations

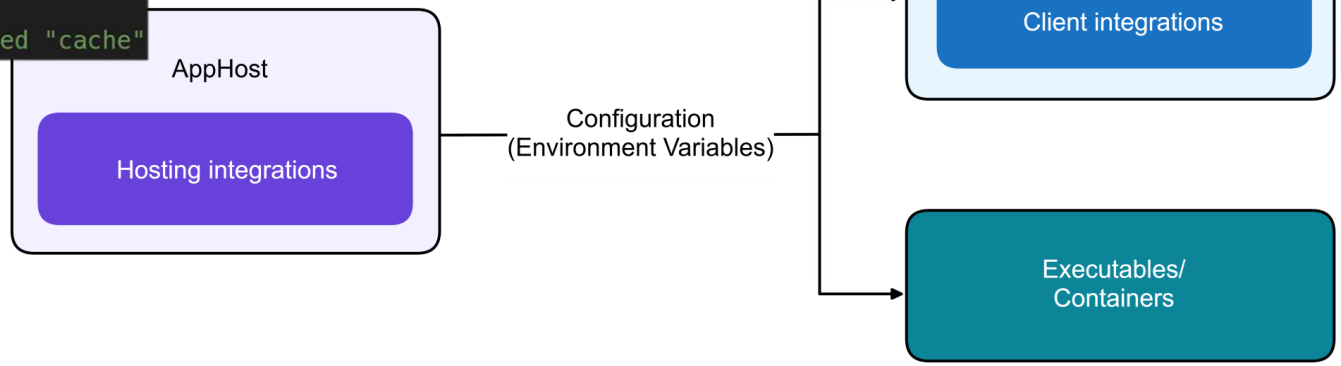




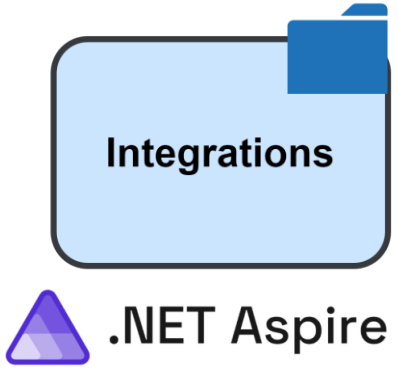

```
var builder =
DistributedApplication.CreateBuilder(args);
// Hosting integration call:
var cache = builder.AddRedis("cache");

// This sets up a Redis container named "cache"
```

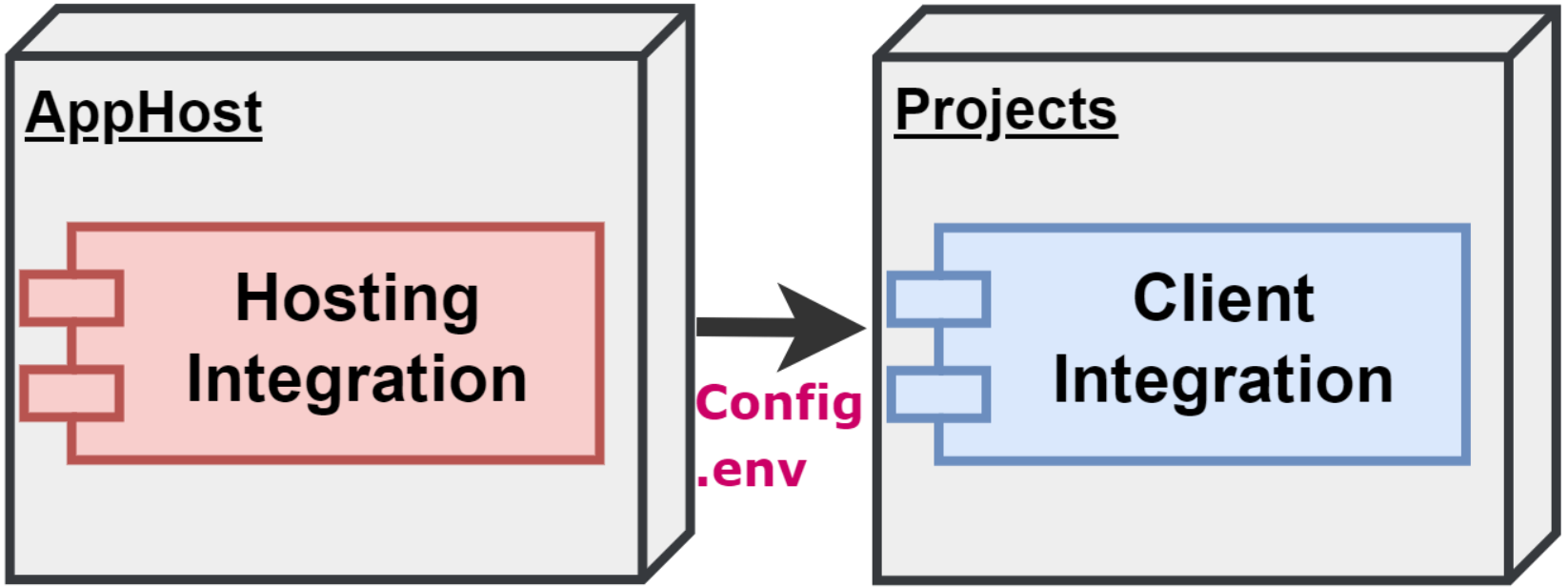
Observability
Logging, tracing and metrics
Health Checks
Resiliency

```
var builder = WebApplication.CreateBuilder(args);

// AddServiceDefaults is often called automatically
builder.AddServiceDefaults();

// This might be from Aspire.StackExchange.Redis
builder.AddRedisDistributedCache(connectionName:
"cache");
```



Conclusion: Hosting vs. Client Integrations

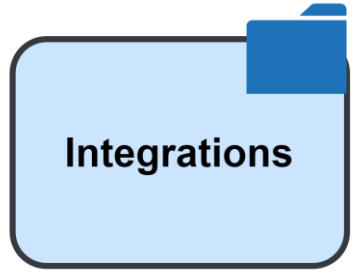





```
var builder =
DistributedApplication.CreateBuilder(args);
// Hosting integration call:
var cache = builder.AddRedis("cache");

// This sets up a Redis container named "cache"
```

```
var builder = WebApplication.CreateBuilder(args);

// AddServiceDefaults is often called automatically
builder.AddServiceDefaults();

// This might be from Aspire.StackExchange.Redis
builder.AddRedisDistributedCache(connectionName:
"cache");
```

Official Integrations



Redis, PostgreSQL, RabbitMQ, Kafka, Keycloak

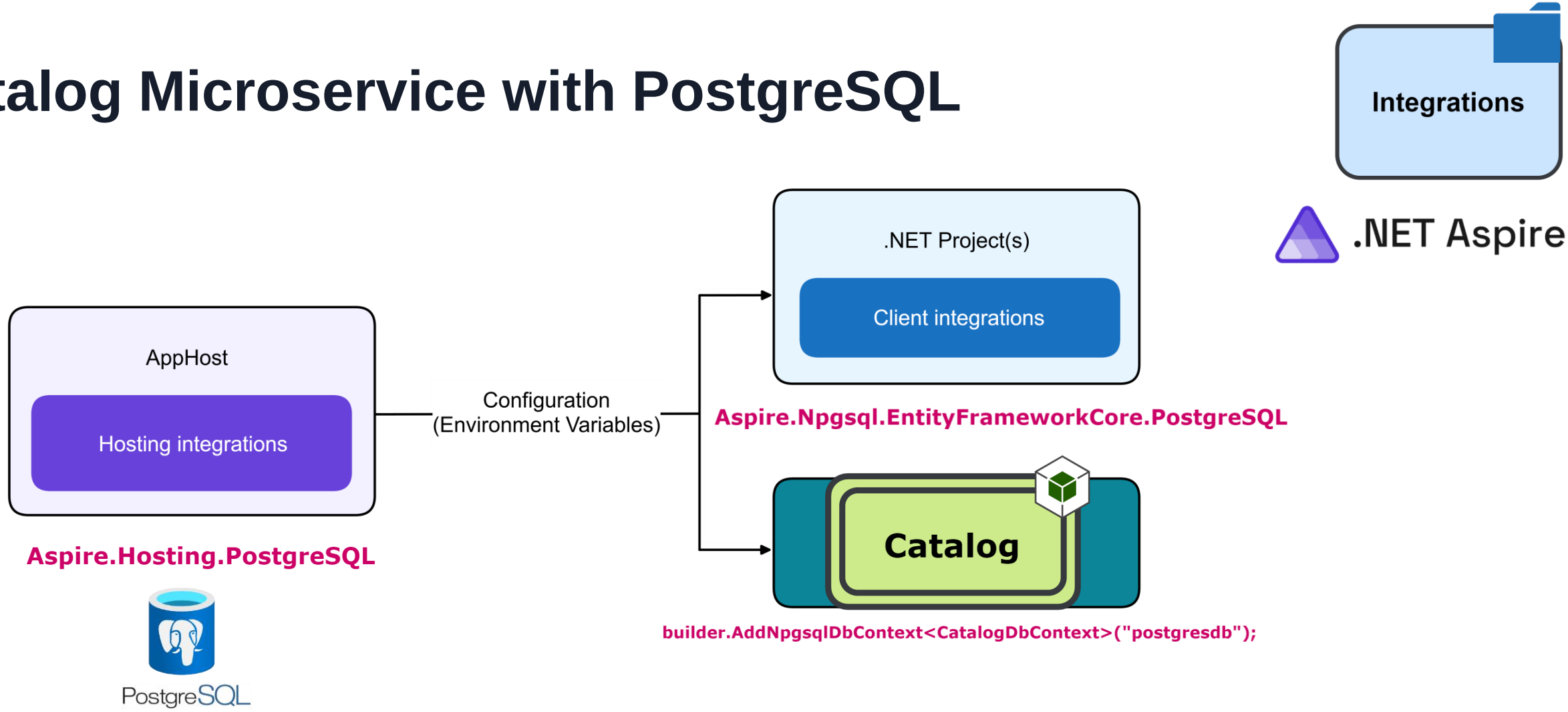
Azure & AWS Packages

Community Toolkit

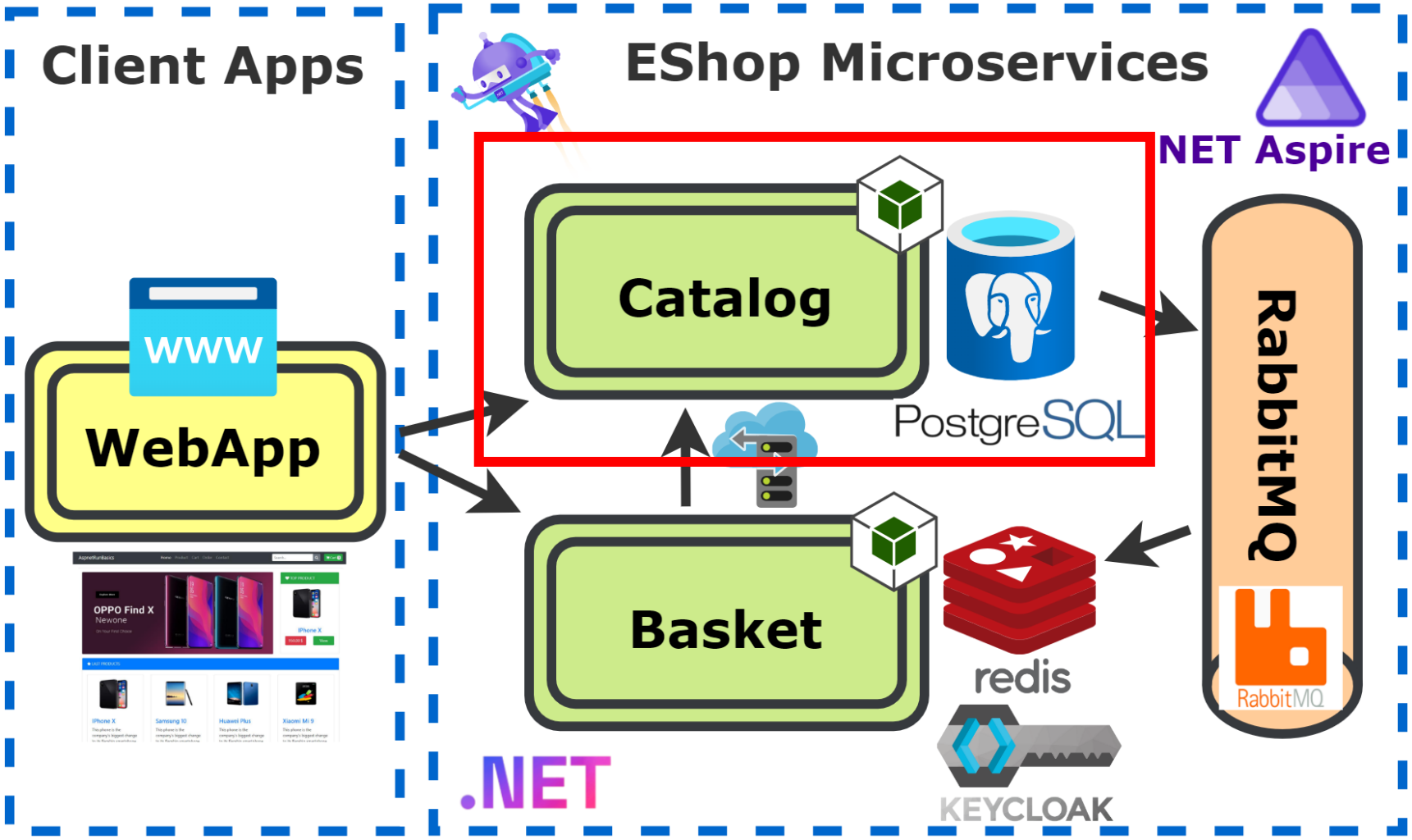
Official Integrations Table

Integration docs and NuGet packages	Description
- Learn more: Apache Kafka - Hosting: Aspire.Hosting.Kafka - Client: Aspire.Confluent.Kafka	A library for producing and consuming messages from an Apache Kafka broker.
- Learn more: Dapr - Hosting: Aspire.Hosting.Dapr - Client: N/A	A library for modeling Dapr as a .NET Aspire resource.
- Learn more: Elasticsearch - Hosting: Aspire.Hosting.Elasticsearch - Client: Aspire.Elastic.Clients.Elasticsearch	A library for accessing Elasticsearch databases.
- Learn more: Keycloak - Hosting: Aspire.Hosting.Keycloak - Client: Aspire.Keycloak.Authentication	A library for accessing Keycloak authentication.
- Learn more: Milvus - Hosting: Aspire.Hosting.Milvus - Client: Aspire.Milvus.Client	A library for accessing Milvus databases.
- Learn more: MongoDB Driver - Hosting: Aspire.Hosting.MongoDB - Client: Aspire.MongoDB.Driver	A library for accessing MongoDB databases.

Catalog Microservice with PostgreSQL



Catalog Microservice with PostgreSQL



Develop Basket Microservice with Redis orchestrate in .NET Aspire

Domain and Technical Analysis of Basket Microservice

Basket Hosting Integration: Add Redis Resource in AppHost

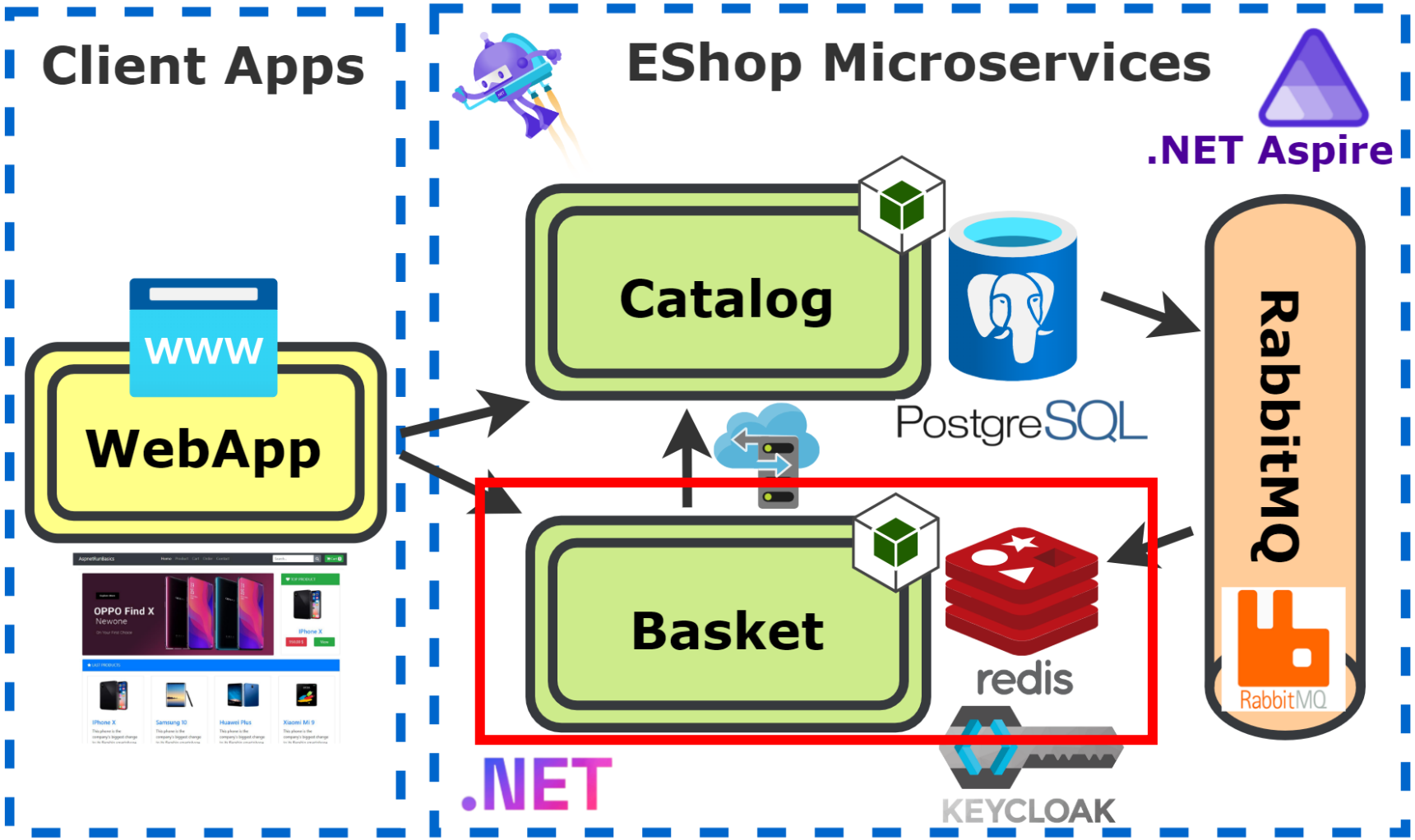
Basket Client Integration: Connect Redis Cache in Microservice

Develop BasketService class with injecting IDistributedCache

Develop Basket API Endpoints with ASP.NET Minimal APIs

Mehmet Ozkaya

Basket Microservice with Redis



Basket Microservice with Redis

Domain Analysis of Basket Microservices: Models, UCs, Rest Apis, Databases

Technical Analysis of Basket Microservices: Architectures, Patterns, Libraries, Folders

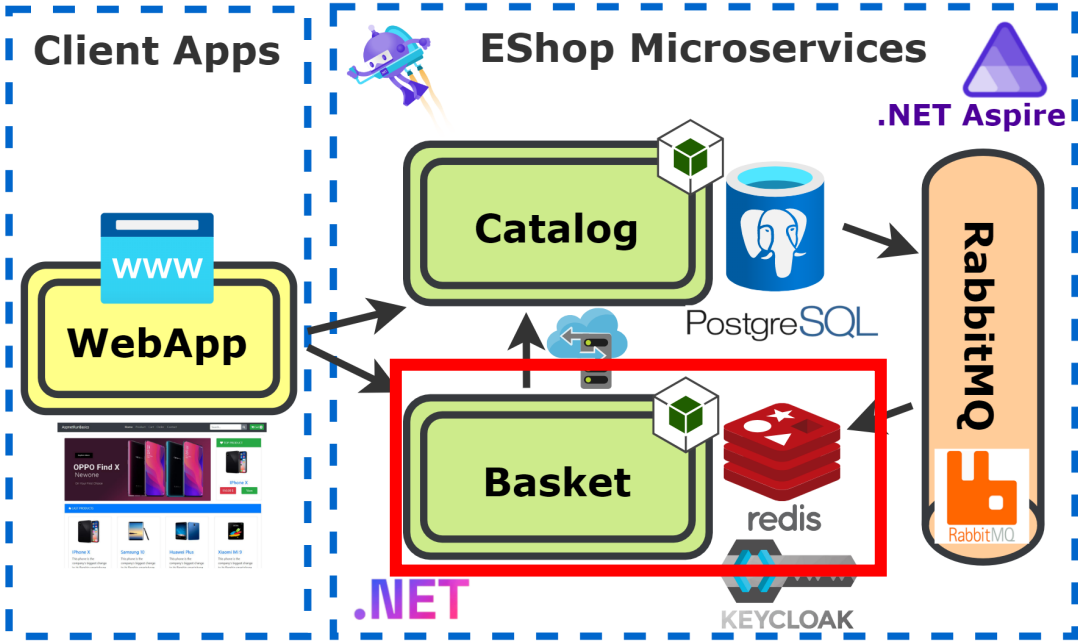
Develop **Basket Domain Entity Models**

Hosting Integration(Backing Services)

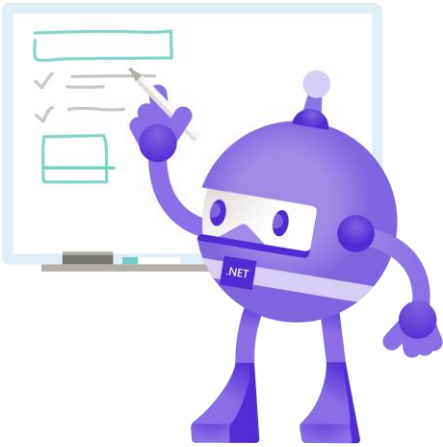
Client Integration(StackExchange Redis)

Basket **Application Use Case** Development

Develop **Basket API Endpoints** Exposing Minimal Api Endpoints



1. Domain Models of Basket Microservice



- Primary domain model is '**ShoppingCart**' and '**ShoppingCartItem**'

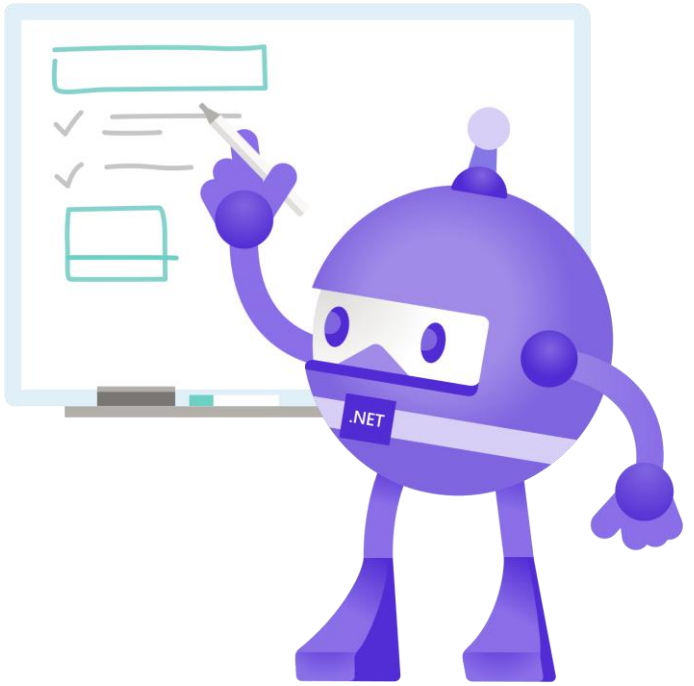
```
public class ShoppingCart
  {
      public string UserName { get; set; } = default!;
      public List<ShoppingCartItem> Items { get; set; } = new();
      public decimal TotalPrice => Items.Sum(x => x.Price * x.Quantity);
  }
```

```
public class ShoppingCartItem
  {
      public int Quantity { get; set; } = default!;
      public string Color { get; set; } = default!;
      public int ProductId { get; set; } = default!;

      // will comes from Catalog module
      public decimal Price { get; set; } = default!;
      public string ProductName { get; set; } = default!;
  }
```

2. Application Use Cases of Basket Microservice

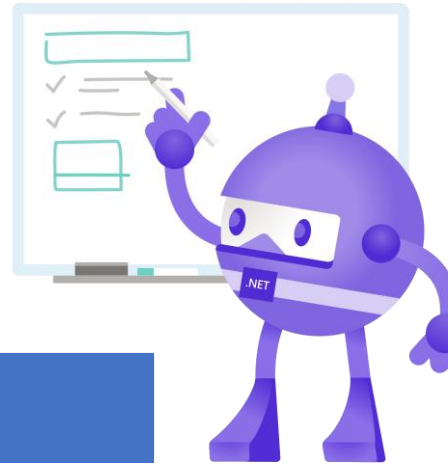
CRUD Operations:

- GetBasket
- UpdateBasket - whole basket
- DeleteBasket
- UpdateBasketItemProductPrices



- Before update SC, we should **call Catalog Microservice GetProductById** endpoint sync
- **Get latest product information** and **set Price** and **ProductName** when adding

3. Rest API Endpoints of Basket Microservice



Method	Request URI	Use Cases
GET	**/basket/{userName}**	Fetch a specific basket
POST	**/basket**	Create or update basket
DELETE	**/basket/{id}**	Remove a basket

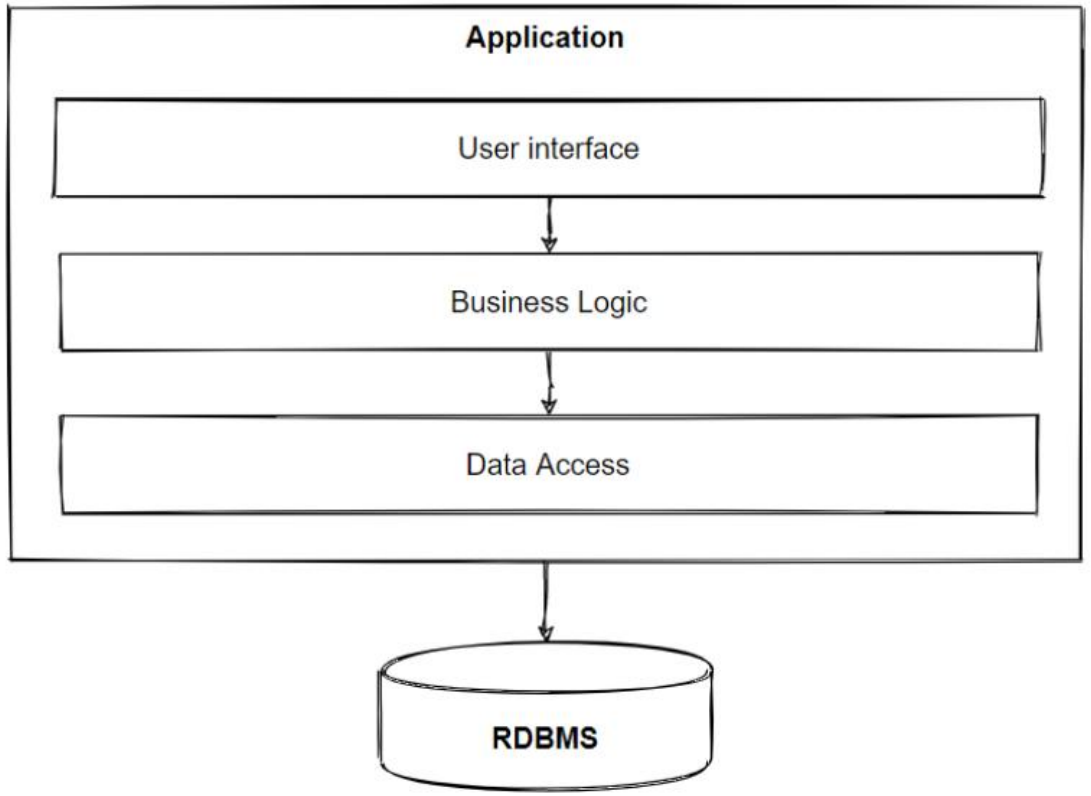
4. Underlying Data Structures of Basket Microservices

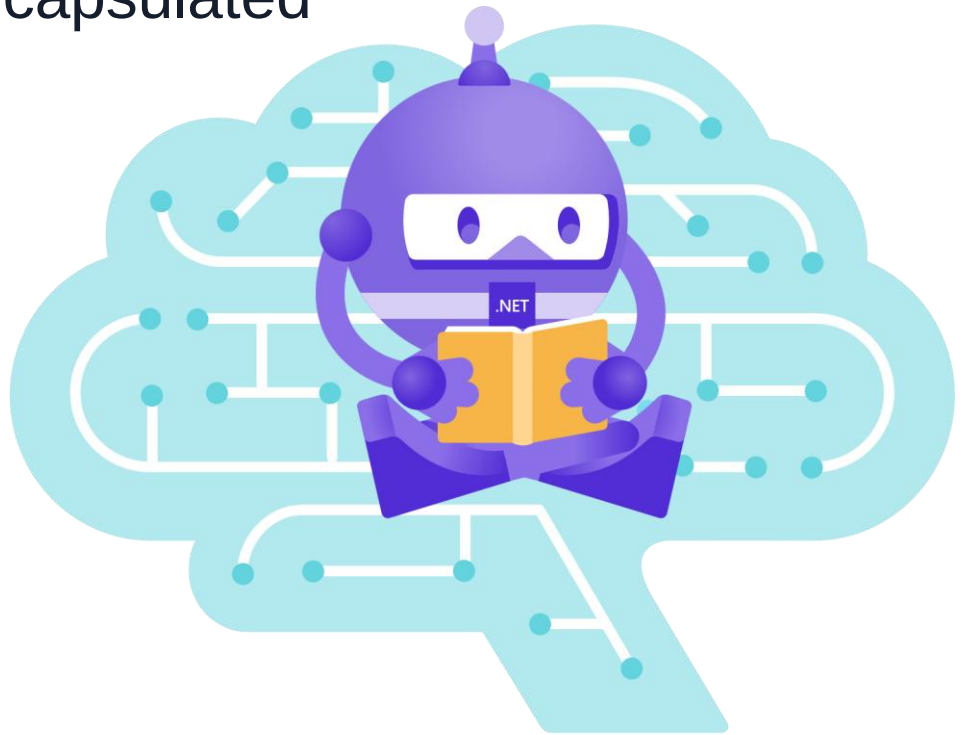
- **Basket Microservice** will be stored in a **Redis Distributed Cache**
- Use **Aspire.StackExchange.Redis.DistributedCaching**
- **IDistributedCache** cache

1. Application Architecture Style of Basket Microservice

Classical N-Layer Architecture

Organizes our code into layer folders, each feature encapsulated in a folder





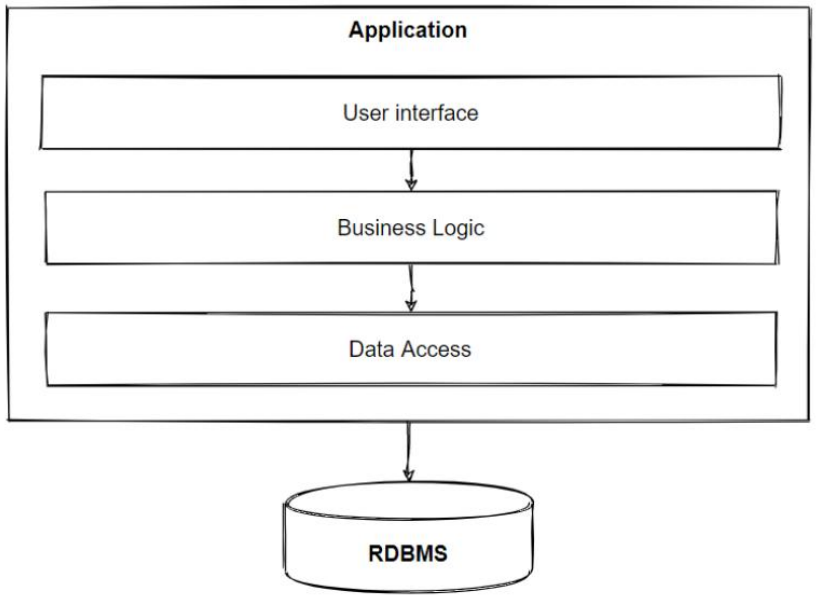
N-Layer(Layered) Architecture

Domain Layer
Stores and Handles domain entities and domain events

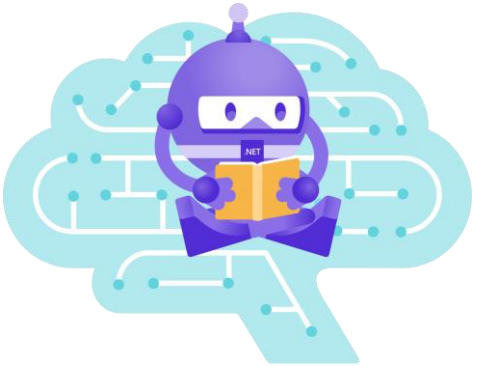
Data Access Layer
Manages interactions with the database or other storage systems

Business Logic Layer
Processes the application's core business logic and rules

Presentation Layer
Displaying data to users and capturing user input



2. Patterns & Principles of Basket Microservice



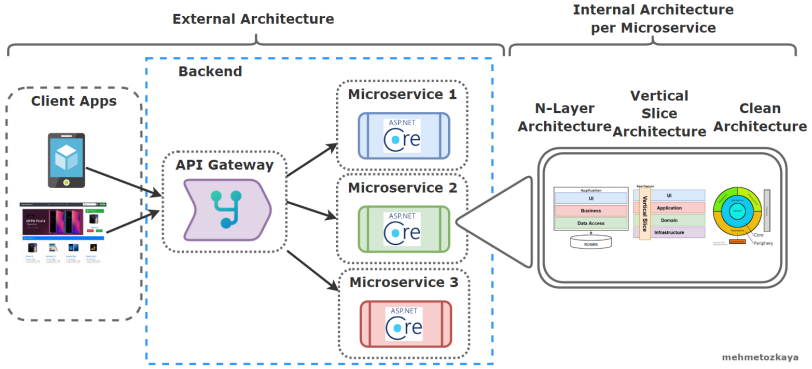
KISS principles and **use less patterns** in our **microservices**

Focus on integrations and **orchestrations** with **.NET Aspire**

DI in ASP.NET Core: Dependency Injection is a core feature, allowing us to inject dependencies.

Minimal APIs and Routing in ASP.NET 8: ASP.NET 8's Minimal APIs simplify endpoint definitions, providing lightweight syntax for routing and handling HTTP requests.

ORM Pattern: Object-Relational Mapping abstracts database interactions, work with database objects using high-level codes.



3. Libraries Nuget Packages of Basket Microservic

- **Aspire Hosting Packages for Orchestration:**
  Aspire.Hosting.Redis
- **Aspire for Redis Interaction: (Client Integration) :**
  Aspire.StackExchange.Redis.DistributedCaching

- builder.AddRedisDistributedCache(connectionName: "cache");
- public class ExampleService(IDistributedCache cache)

4. Project Folder Structure of Catalog Microservice

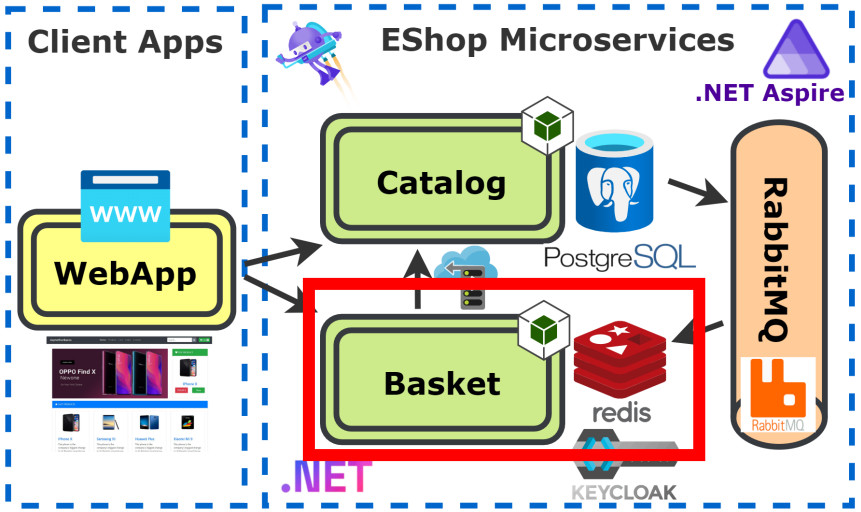
Organized into **Models, Services, Endpoints** and **ApiClients**

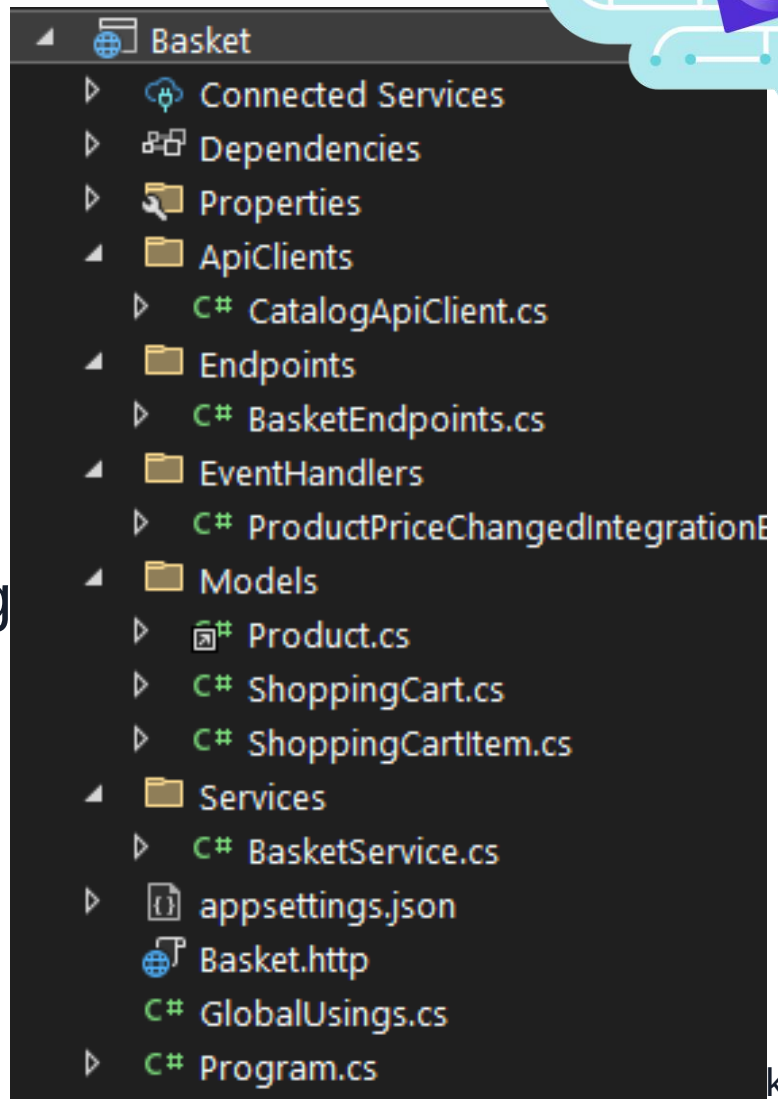
Models holds our **domain entities** (e.g. SC)

Services contains **business logic**

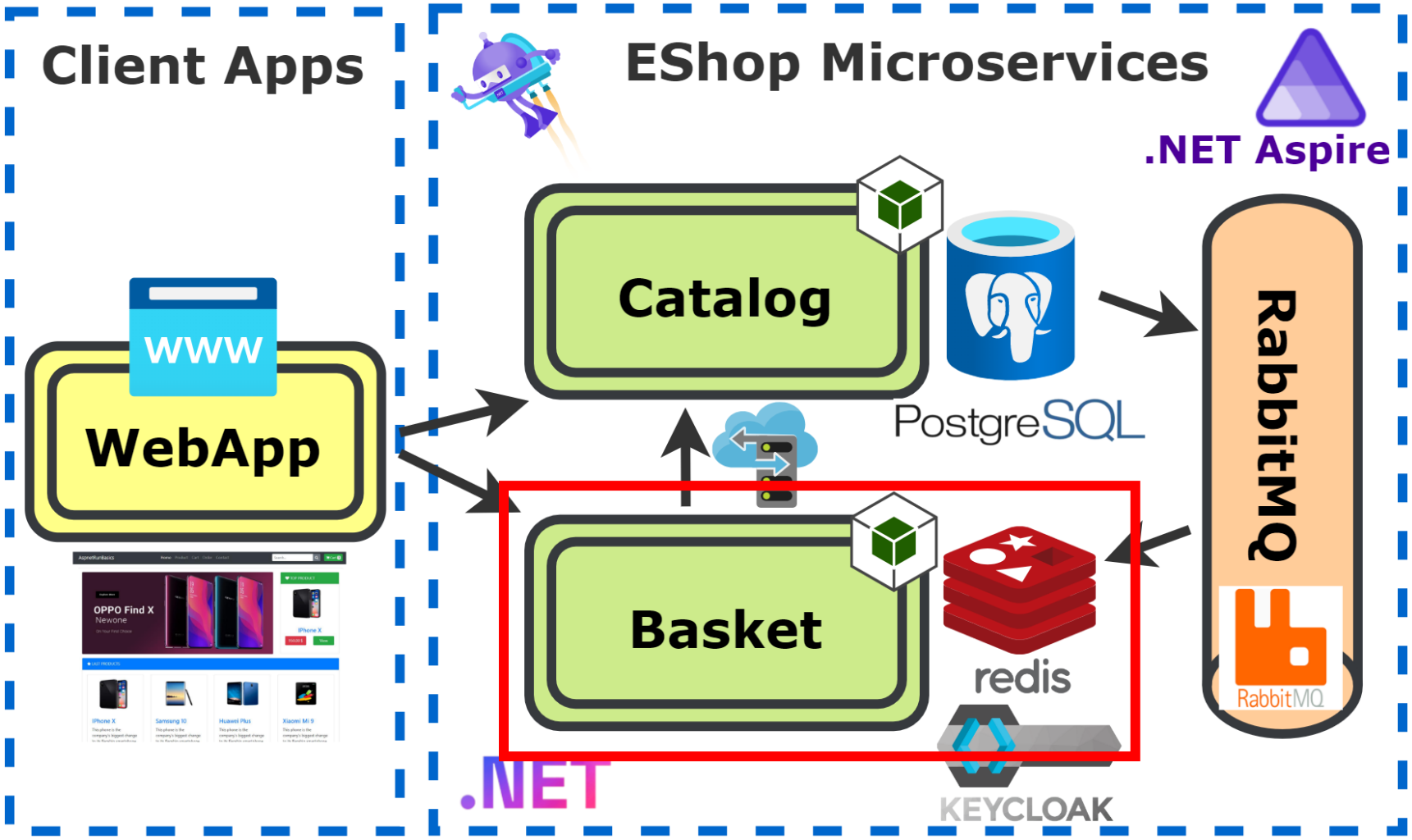
Endpoints minimal API endpoints that call into the services layer

ApiClients includes **HTTP Client** for calling Catalog

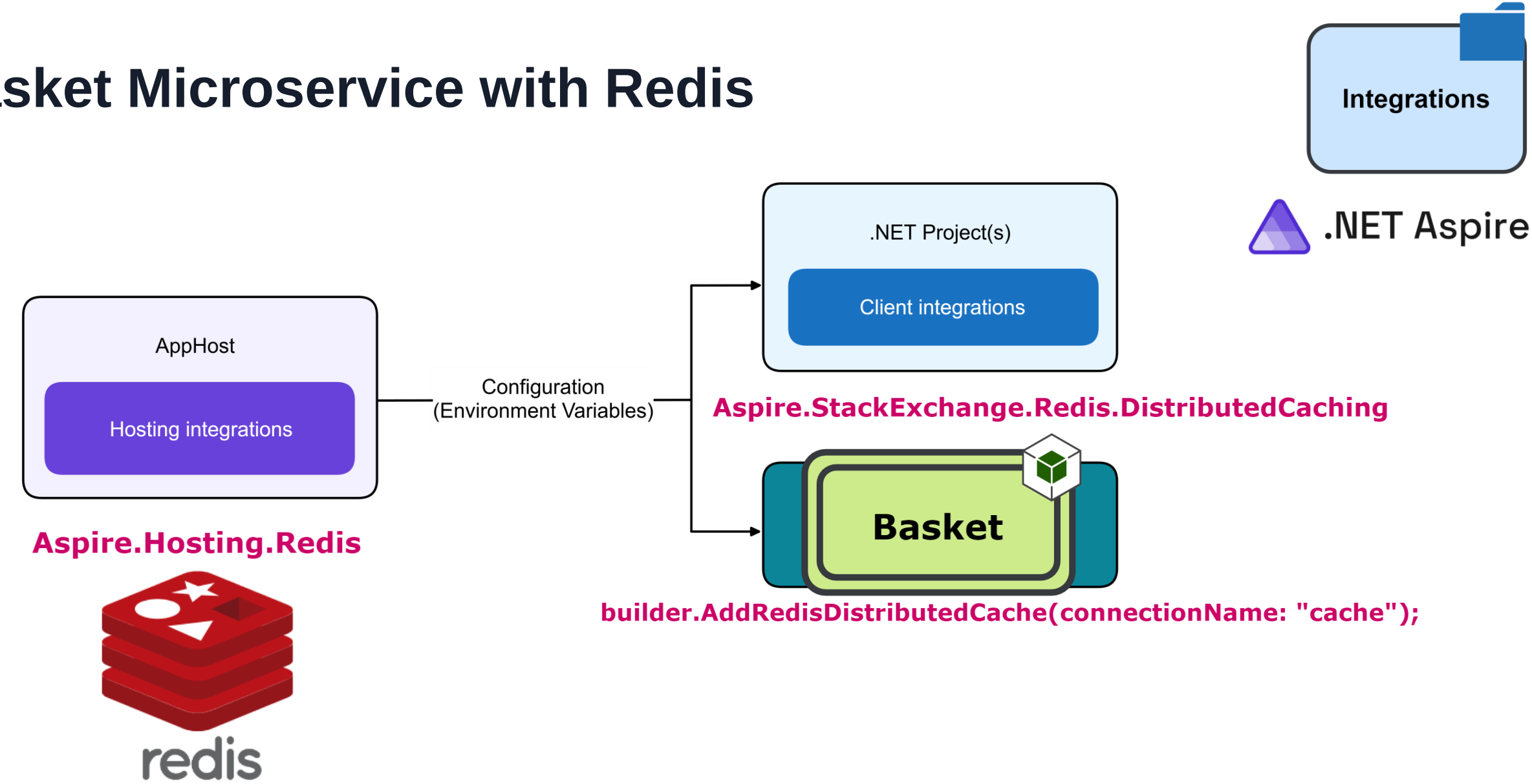




Catalog Microservice with PostgreSQL



Basket Microservice with Redis



Sync Communications between Catalog-Basket w/ .NET Aspire Service Discovery

Sync Communications Use Case: When Update Basket, Get Latest Product Prices

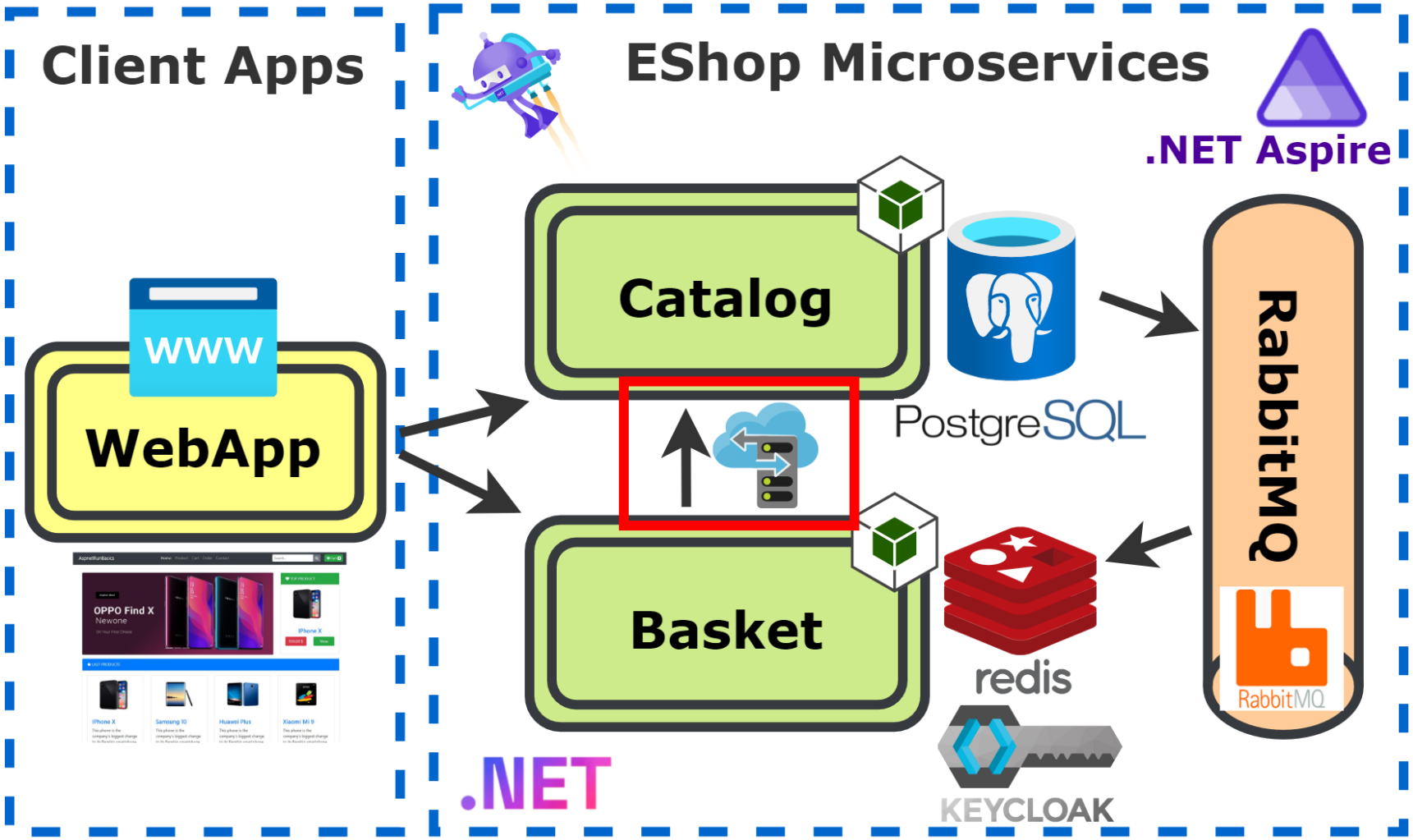
Add WithReference(catalog) into Basket for Service Discovery

Develop CatalogApiClient fro Sync Http Communication

Integrate w/ Catalog into BasketService.cs using CatalogApiClient

Mehmet Ozkaya

Sync Communications between Catalog-Basket



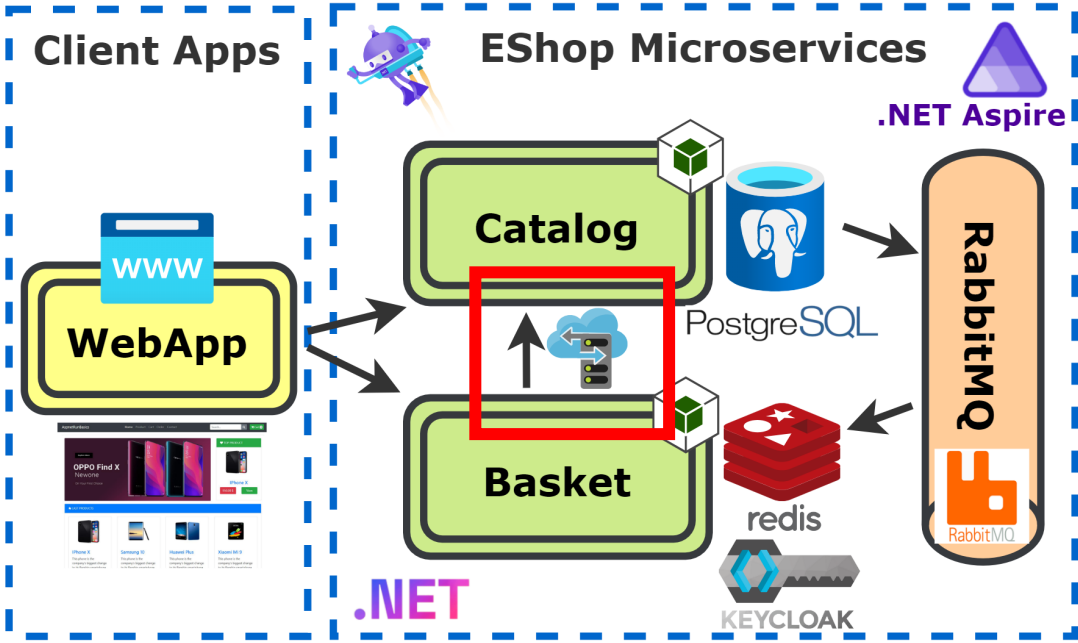
Sync Communications between Catalog-Basket

Adding .WithReference(catalog)

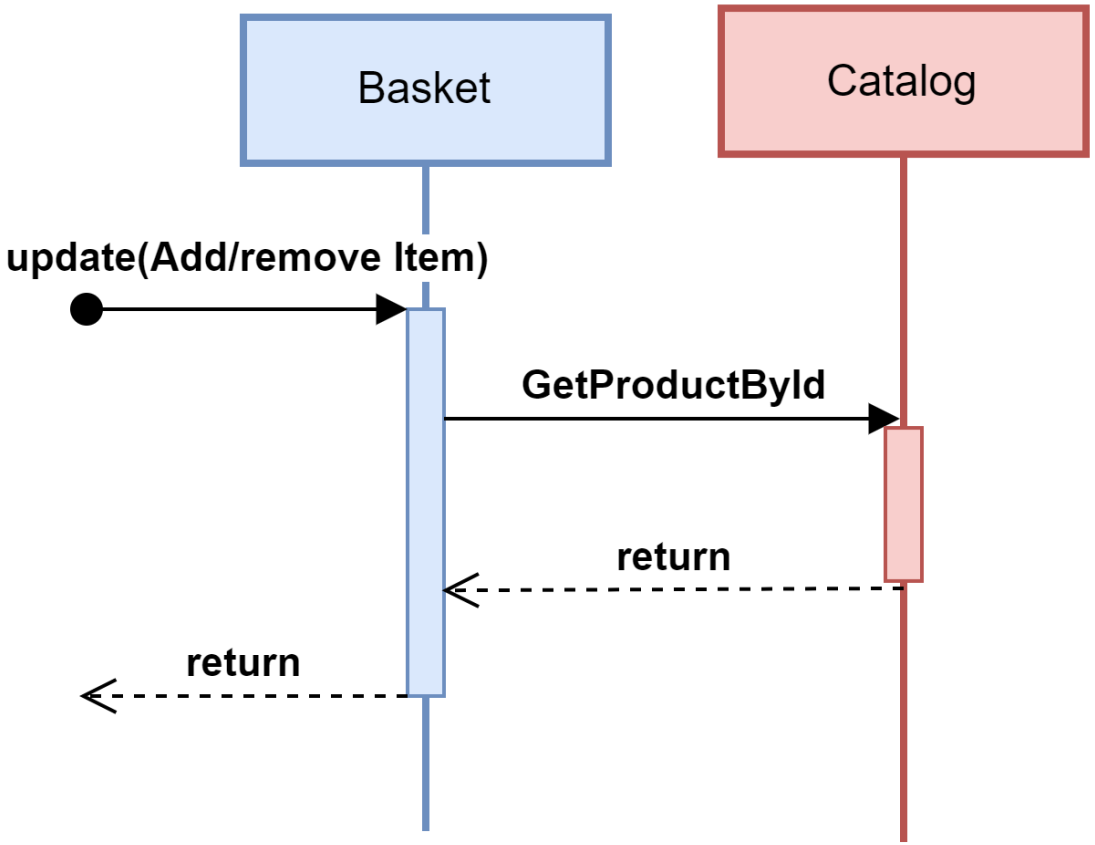
```
var basket = builder
    .AddProject<Projects.Basket>("basket")
    .WithReference(catalog)
    .WaitFor(catalog);
```

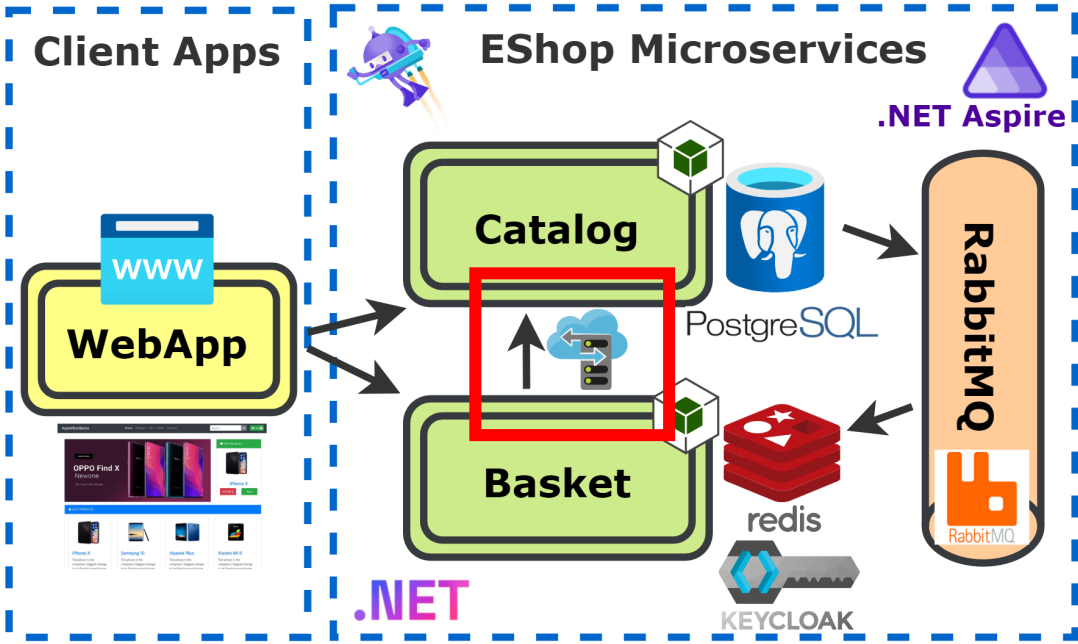
Creating a **Typed HttpClient consume Catalog** from **Basket**

```
builder.Services.AddHttpClient<CatalogApiClient>(client
=>  {
        client.BaseAddress = new("https+http://catalog");
    });
```



When Update Basket, Get Latest Product Prices






```
public async Task UpdateBasket(ShoppingCart basket)
{
    //TODO: Before update(Add/remove Item) into SC, we should call Catalog Module GetProductById method
    // Get latest product information and set Price and ProductName when adding item into SC
```

Remember: Service Discovery

.NET Aspire

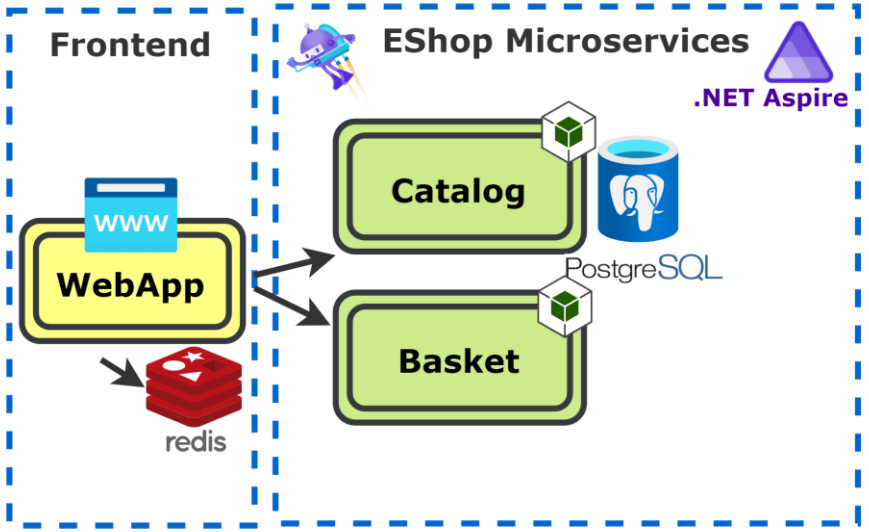
Service Discovery

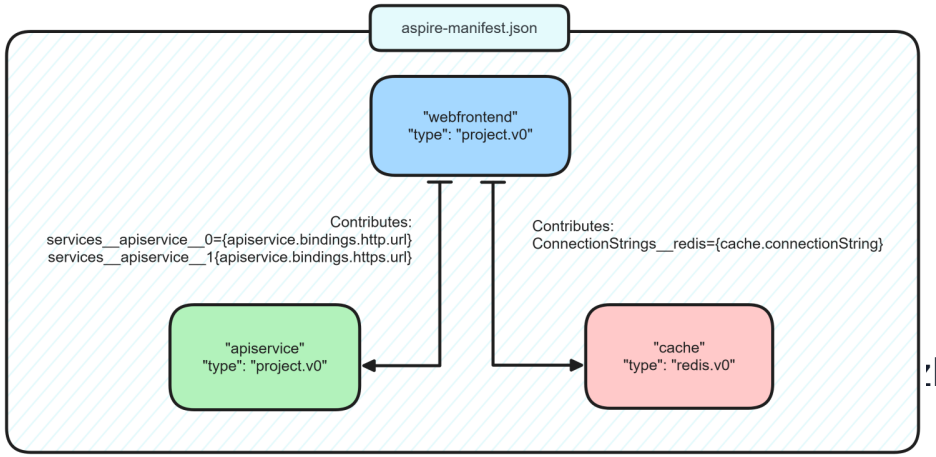
Containers→injects→connection string

Projects→injects→microservice endpoints

```
var builder = DistributedApplication.CreateBuilder(args);

var cache = builder.AddRedis("cache");

var catalog = builder.AddProject<Projects.CatalogService>
("catalog");
var basket = builder.AddProject<Projects.BasketService>("basket");

var frontend = builder.AddProject<Projects.MyFrontend>("frontend")
  .WithReference(cache)
  .WithReference(catalog)
  .WithReference(basket);
```

```
WithReference(cache)
  ConnectionStrings__cache="localhost:62354"

WithReference(apiservice)
  services__apiservice__http__0="http://localhost:5455"

 services__apiservice__https__0="https://localhost:7356"
```





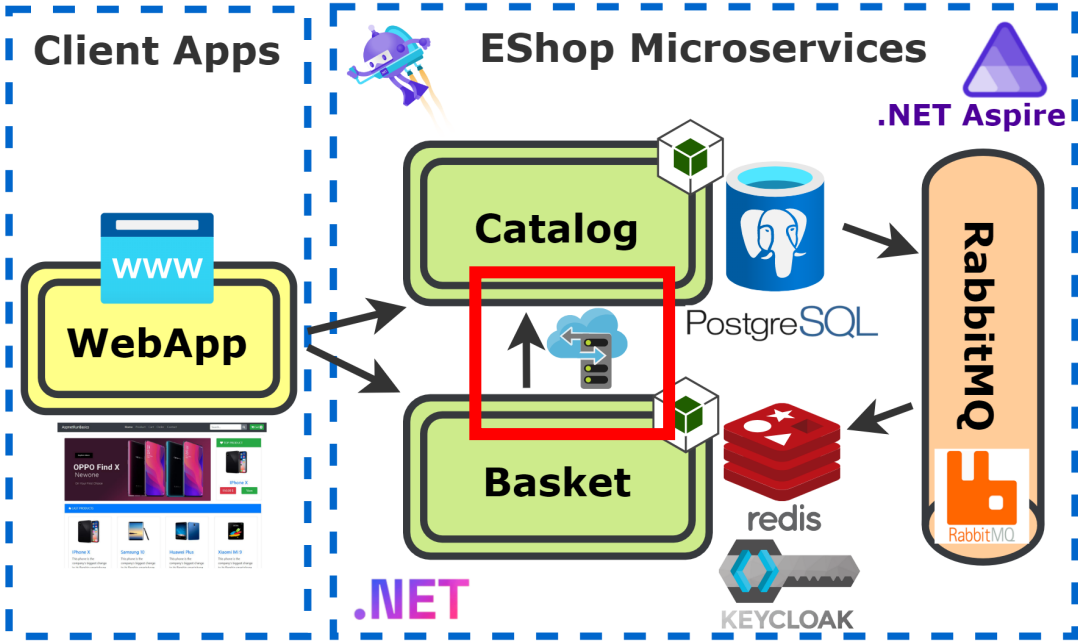
:kaya

When Update Basket, Get Latest Product Prices

.WithReference(catalog) for Basket → Catalog

AddHttpClient<CatalogClient> referencing "catalog"

Synchronous calls via strongly typed methods: await catalogClient.**GetProductByIdAsync**(productId);



Goto BasketService – UpdateBasket Method

```
public async Task UpdateBasket(ShoppingCart basket)
{
    //TODO: Before update(Add/remove Item) into SC, we should call Catalog Module GetProductById method
    // Get latest product information and set Price and ProductName when adding item into SC
```

Async Communications w/ RabbitMQ & MassTransit orchestrate .NET Aspire

RabbitMQ Hosting Integration .NET Aspire

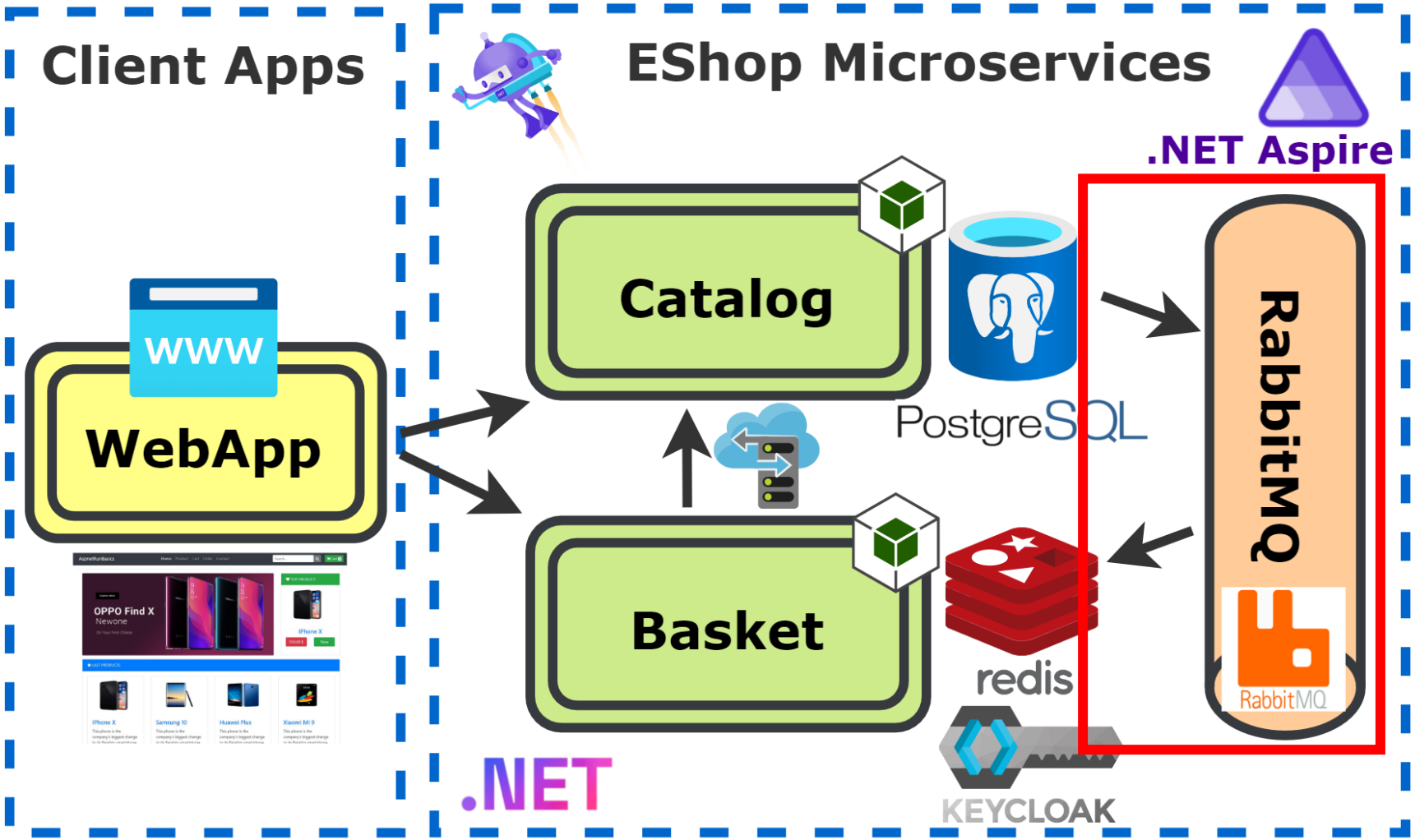
Create Shared Messaging Folders and Classes for ProductPriceChangedIntegrationEvent

Catalog ms Publish ProductPriceChanged Integration Event

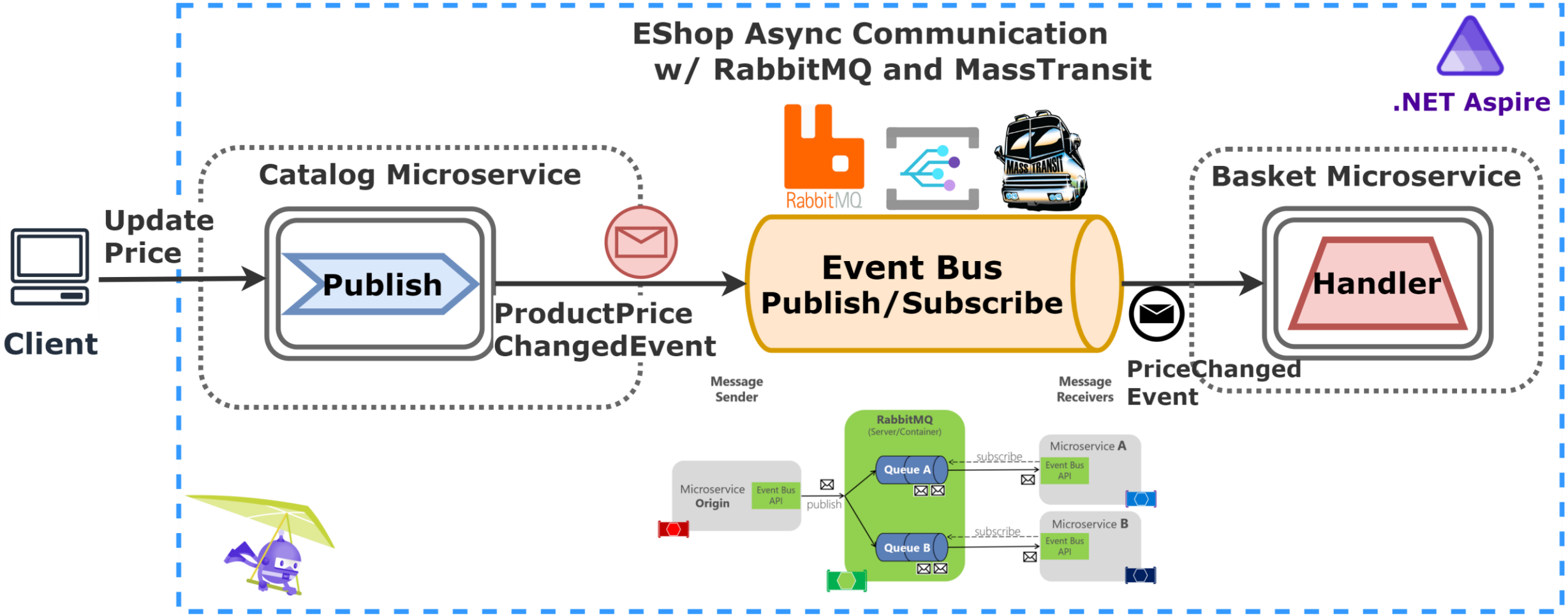
Basket Subscribe and consume ProductPriceChanged Integration Event

Mehmet Ozkaya

Async Communications w/ RabbitMQ & MassTransit



Async Communications w/ RabbitMQ & MassTransit



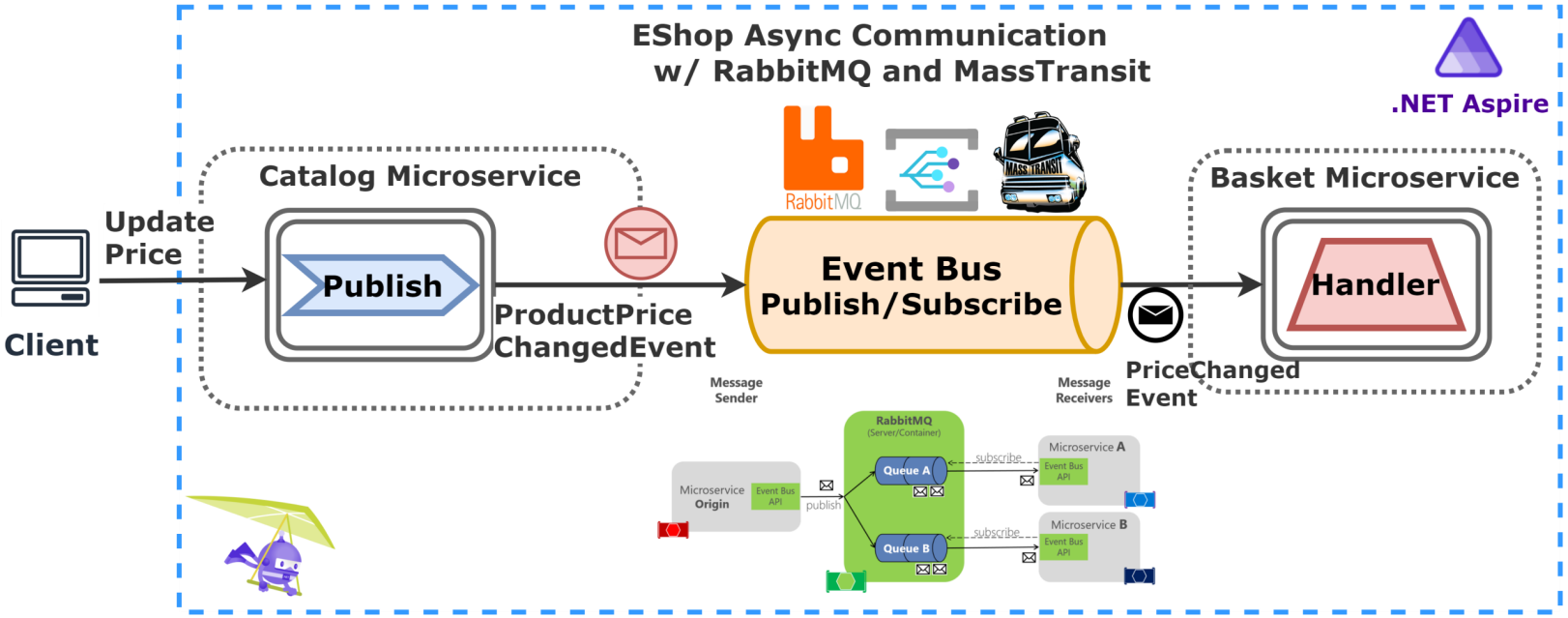
Steps for Async Communication

Hosting Integration (.AddRabbitMq("messagebroker")

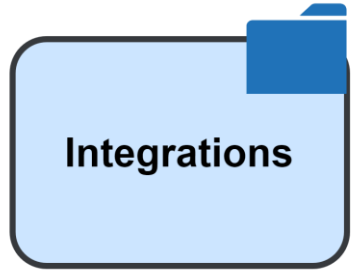
Client Integration

Shared Messaging Events

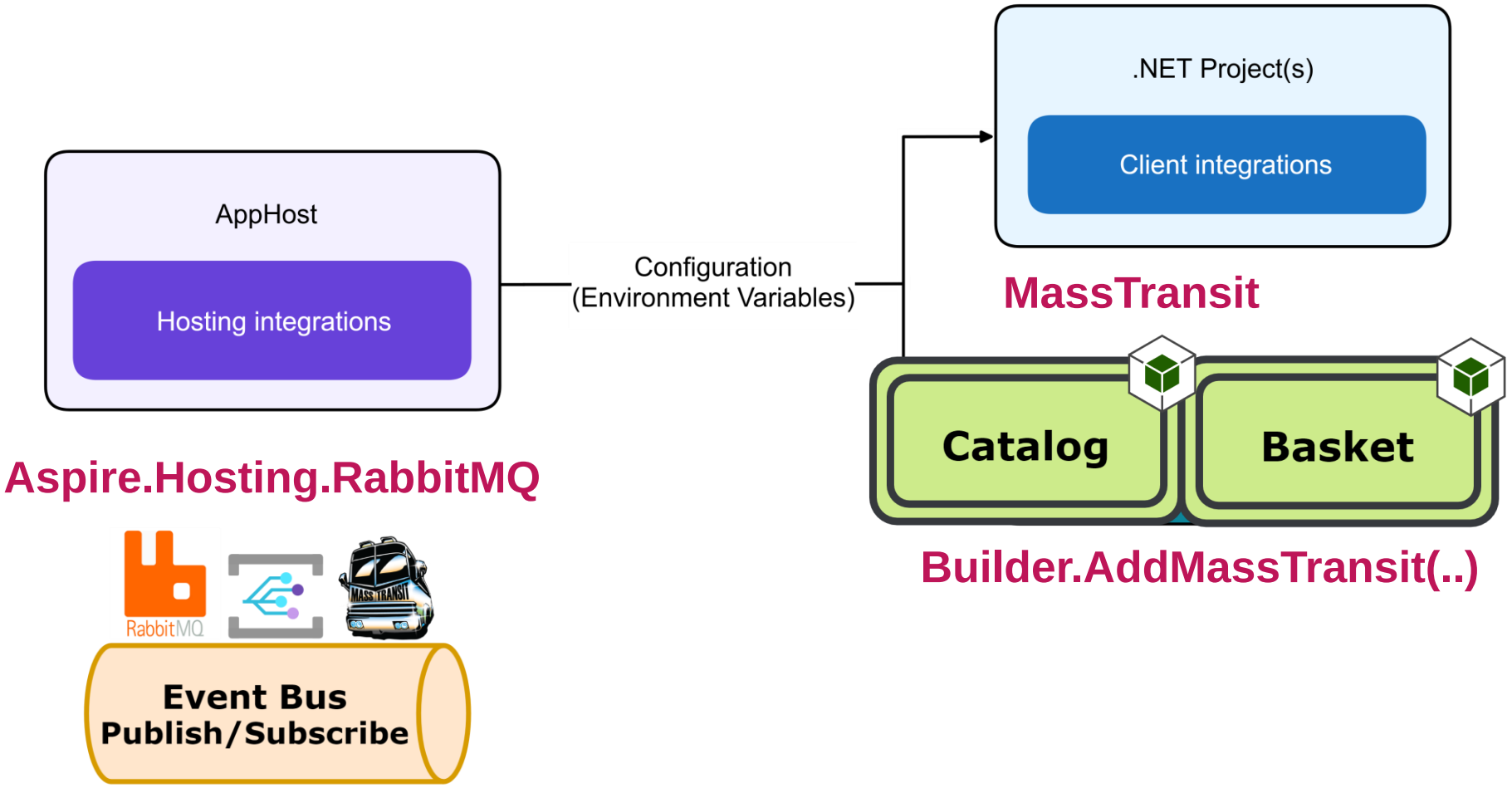
MassTransit



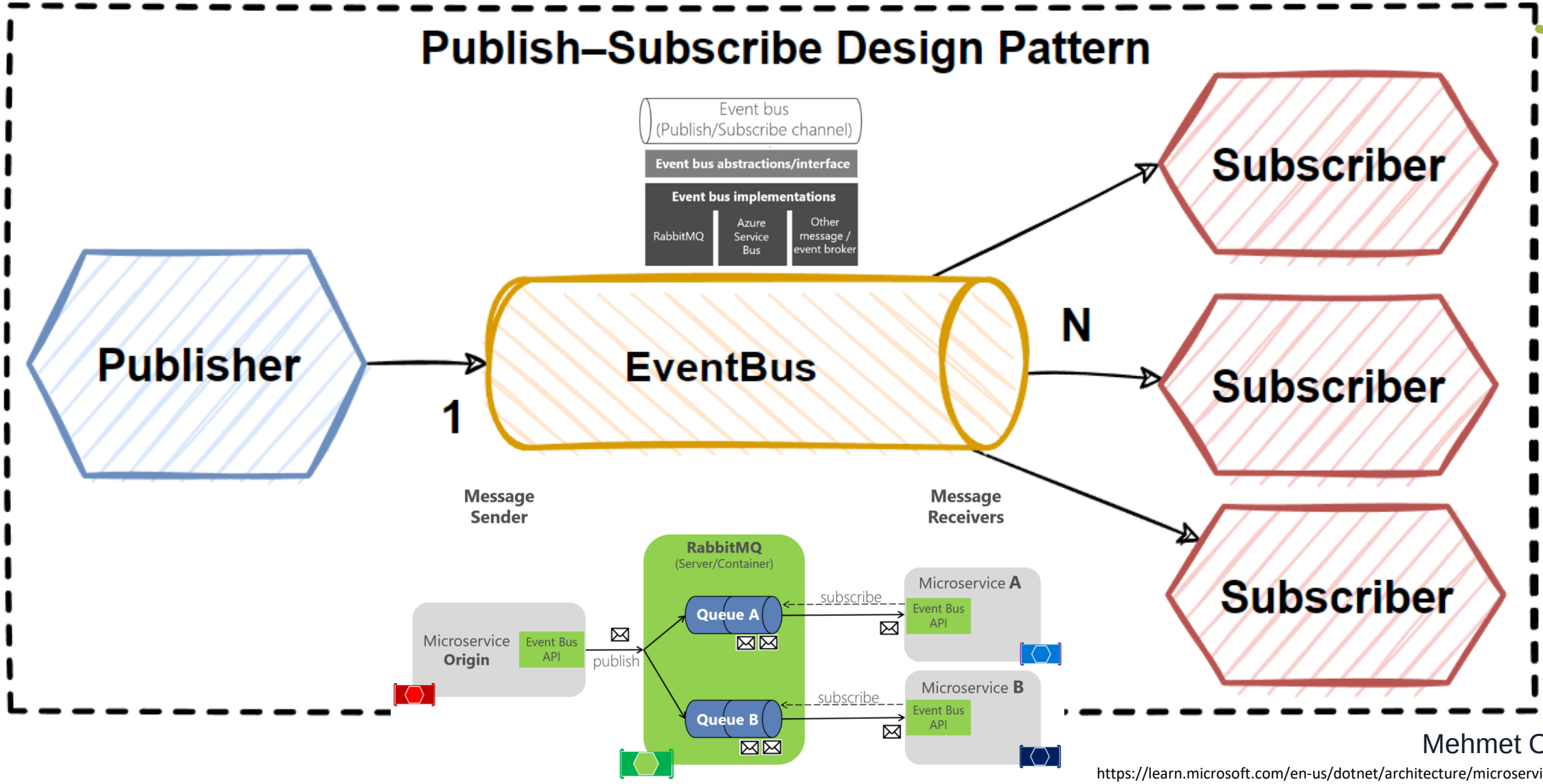
Async Communications w/ RabbitMQ & MassTransit



.NET Aspire

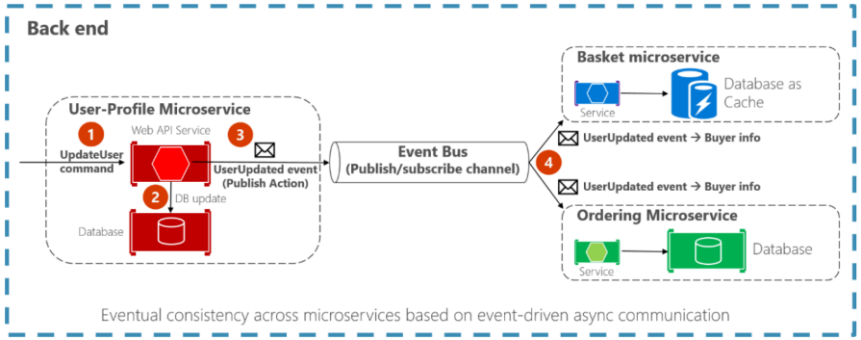


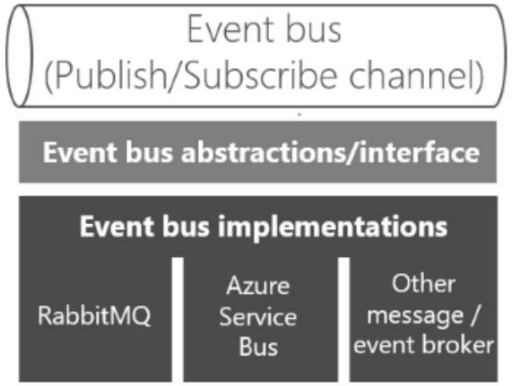
Pub/Sub RabbitMQ Architecture

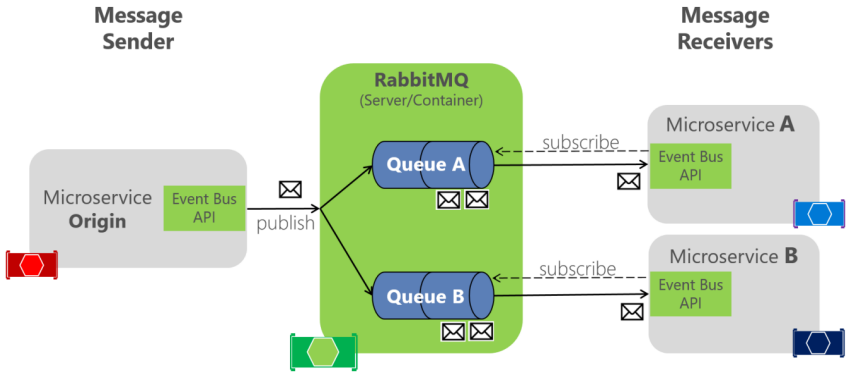


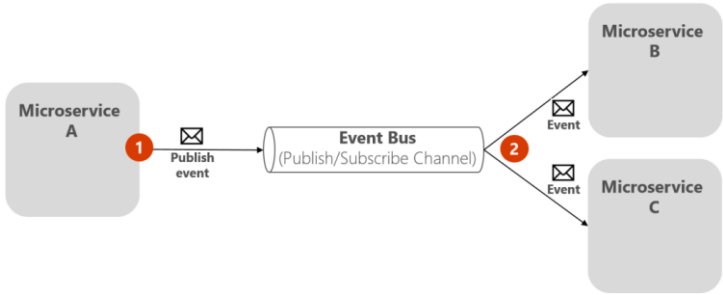
https://learn.microsoft.com/en-us/dotnet/architecture/microservices

Microservices Async Communication w/ RabbitMQ & MassTransit



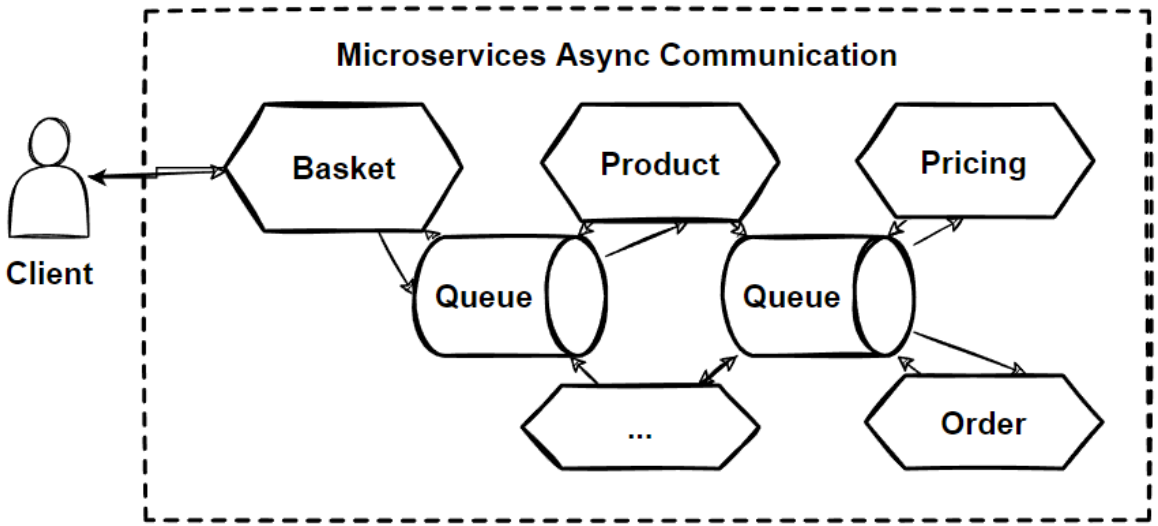




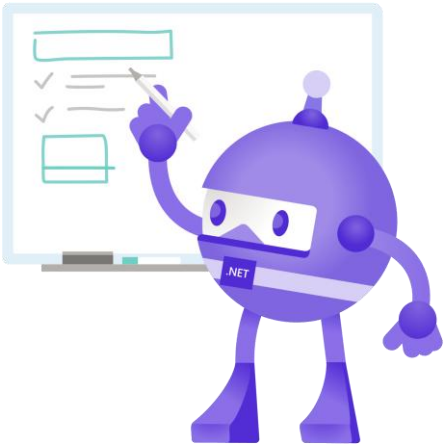


Why Use Asynchronous Communication?

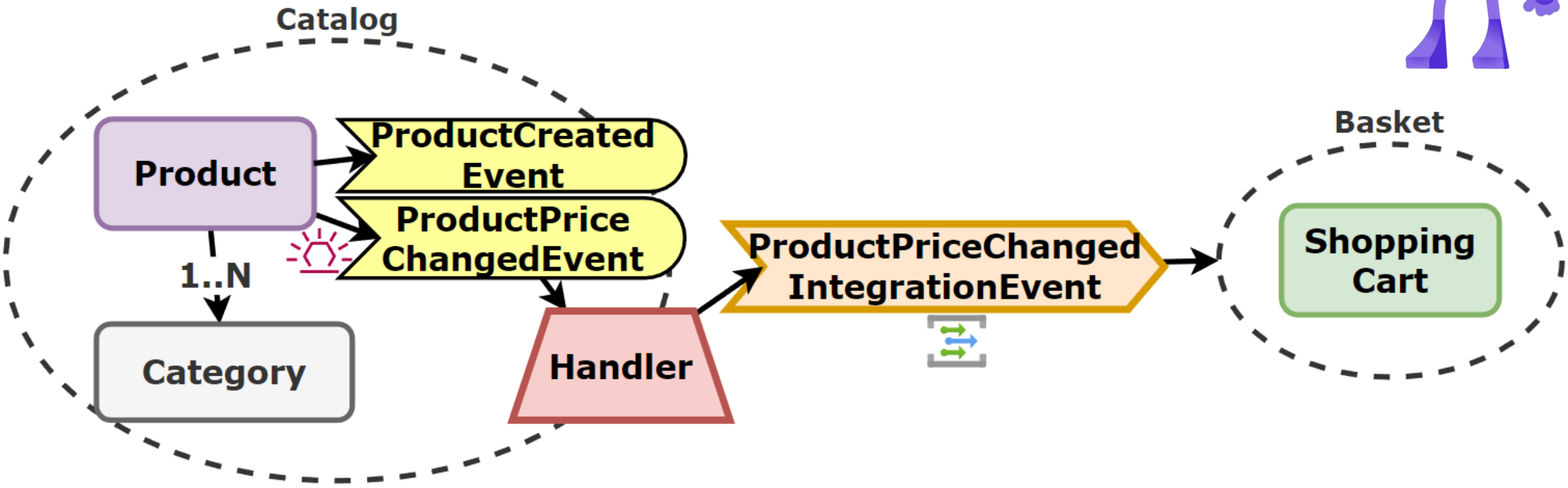
- **Decoupling:** Asynchronous communication decouples the sender and receiver, allowing them to operate independently.
- **Scalability:** By not waiting for responses, the system can handle more requests and scale effectively.
- **Resilience:** If a module fails, messages can be retried or processed later, improving fault tolerance.
- **Responsiveness:** Users receive immediate feedback, even if the operation takes time to complete.



1. Event Models of ProductPriceChangedEvent



- Primary event model is '**ProductPriceChangedEvent**'

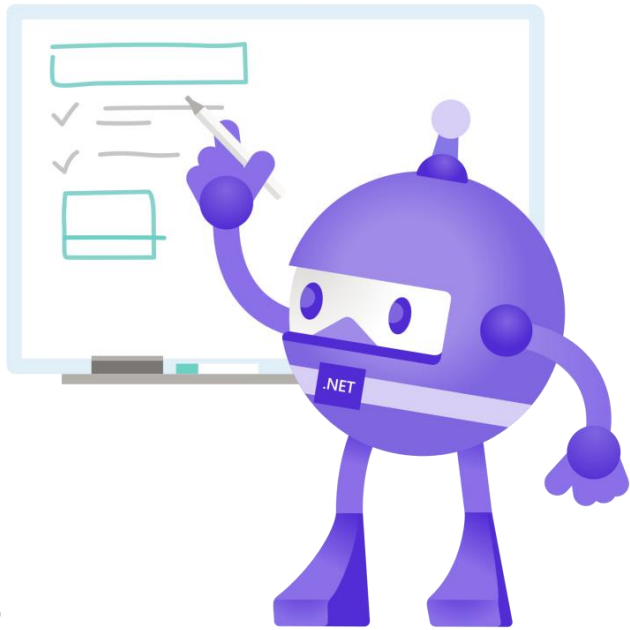


- Consider Domain event for ProductPriceChanged, leading to integration events.

2. Application Use Cases between Catalog & Basket

Async Cases for Catalog:

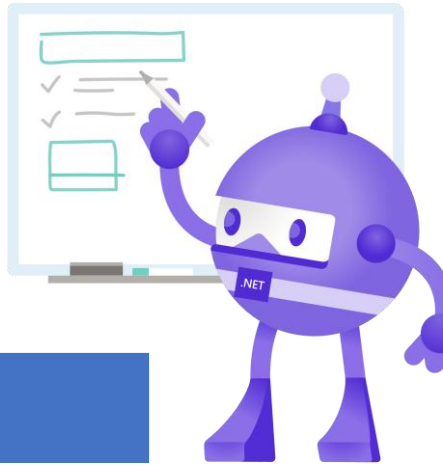
- **Update Product Price:** publish domain event that lead to an integration event that contains the updated product information.



Async Cases for Basket:

- **Basket Update Price:** Consume ProductPriceChangedEvent integration event from RabbitMQ using MassTransit.
  - Get existing basket with changed productId
  - Set ProductName and Price from incoming integration event
  - Update all basket shopping carts according to changed product information

3. Rest API Endpoints of Catalog Module



Method	Request URI	Use Cases
POST	**/products**	Update Product Price

RabbitMQ AMQP Protocol

- AMQP: Consume ProductPriceChangedEvent Event from RabbitMQ using MassTransit

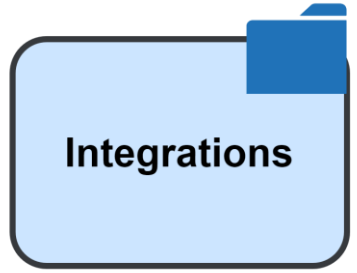
4. Underlying Data Structures of Async Communications

For Async Communicatin Between Modules has 1 Backing Services:

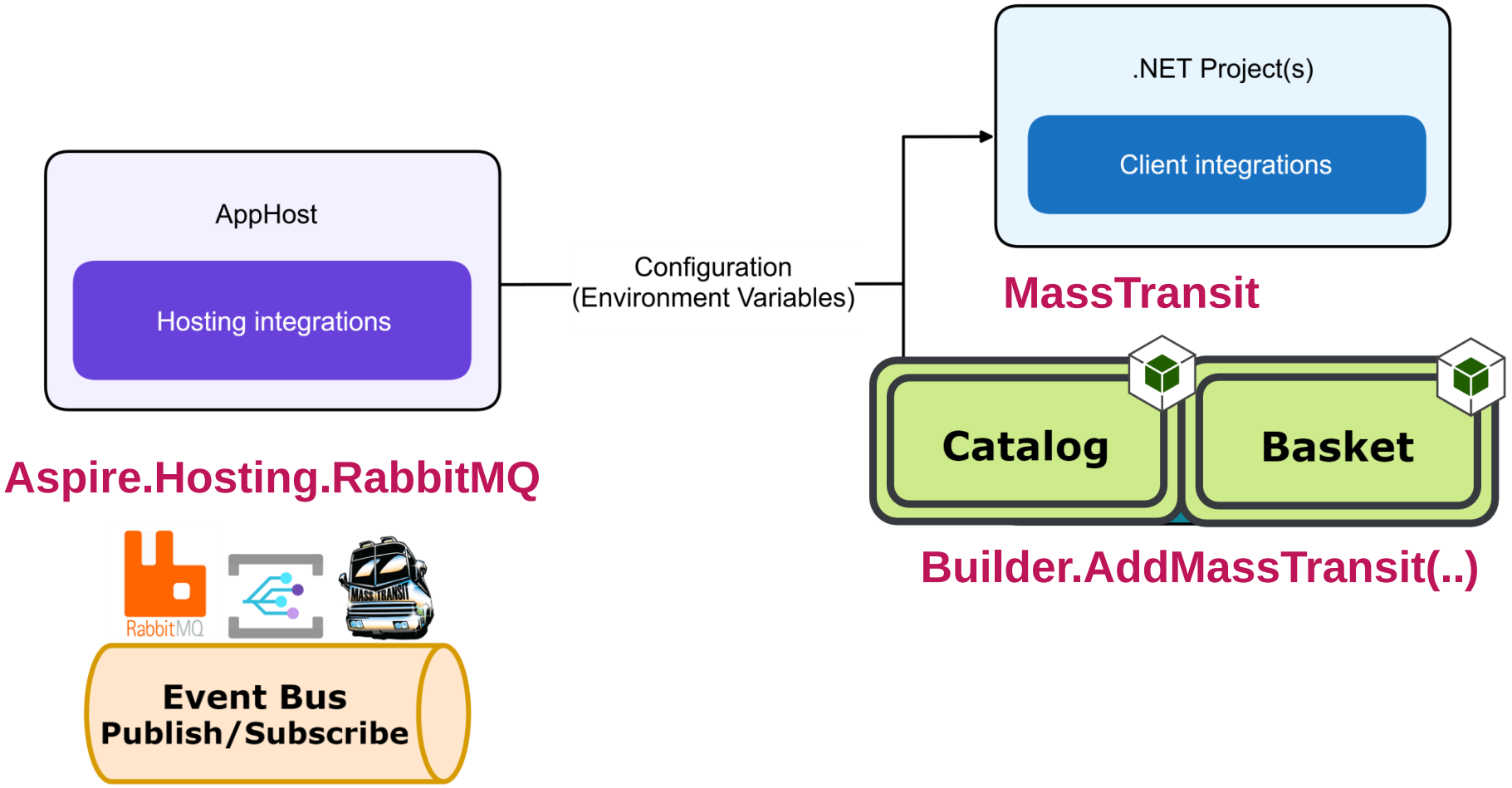
1. **RabbitMQ Distributed Message Broker**
2. **Client Integrations w/ MassTransit**

.NET Aspire

Async Communications w/ RabbitMQ & MassTransit

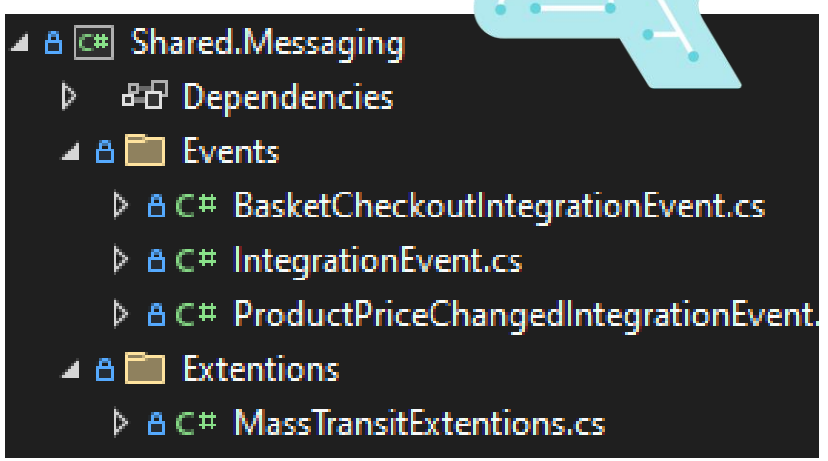


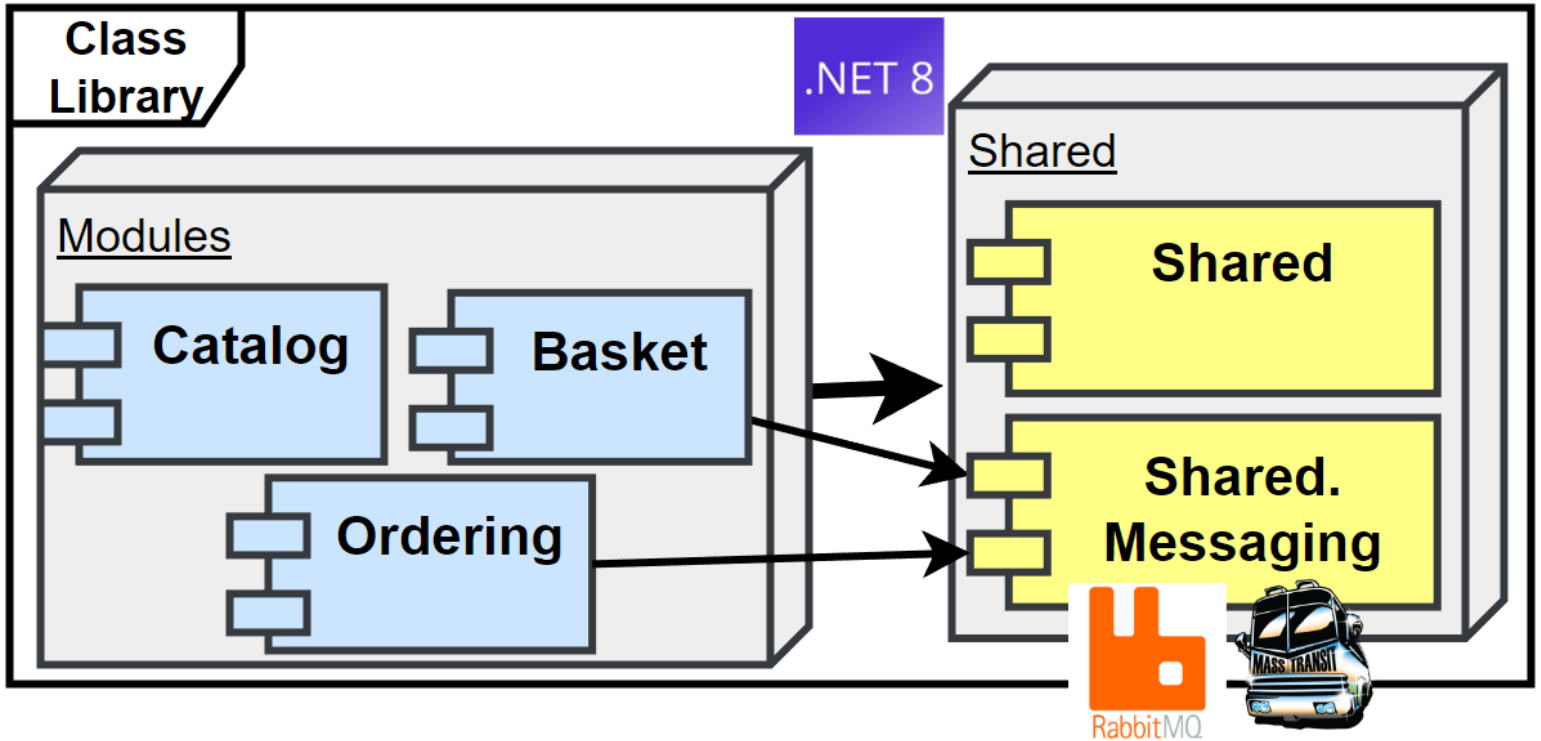
.NET Aspire



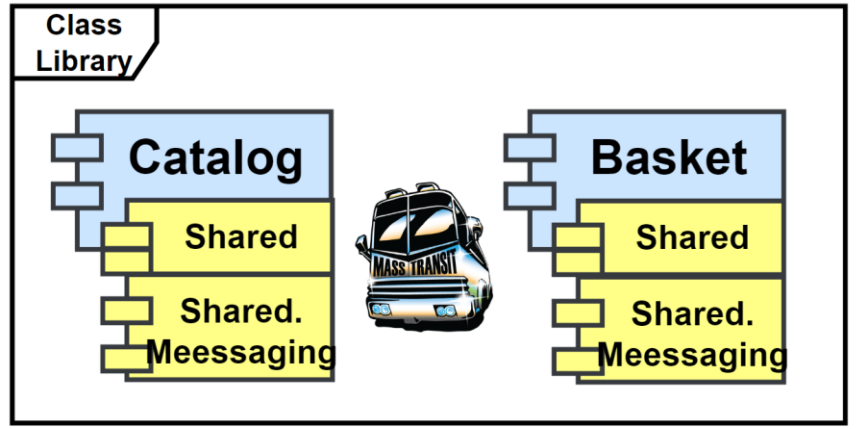
Class Library Architecture of Async Communicatio

- **Catalog** -- referenced **Shared** and **Shared.Messaging**
- **Basket** -- referenced **Shared** and **Shared.Messaging**
- **Identity**
- **Ordering**-- referenced **Shared** and **Shared.Messag**





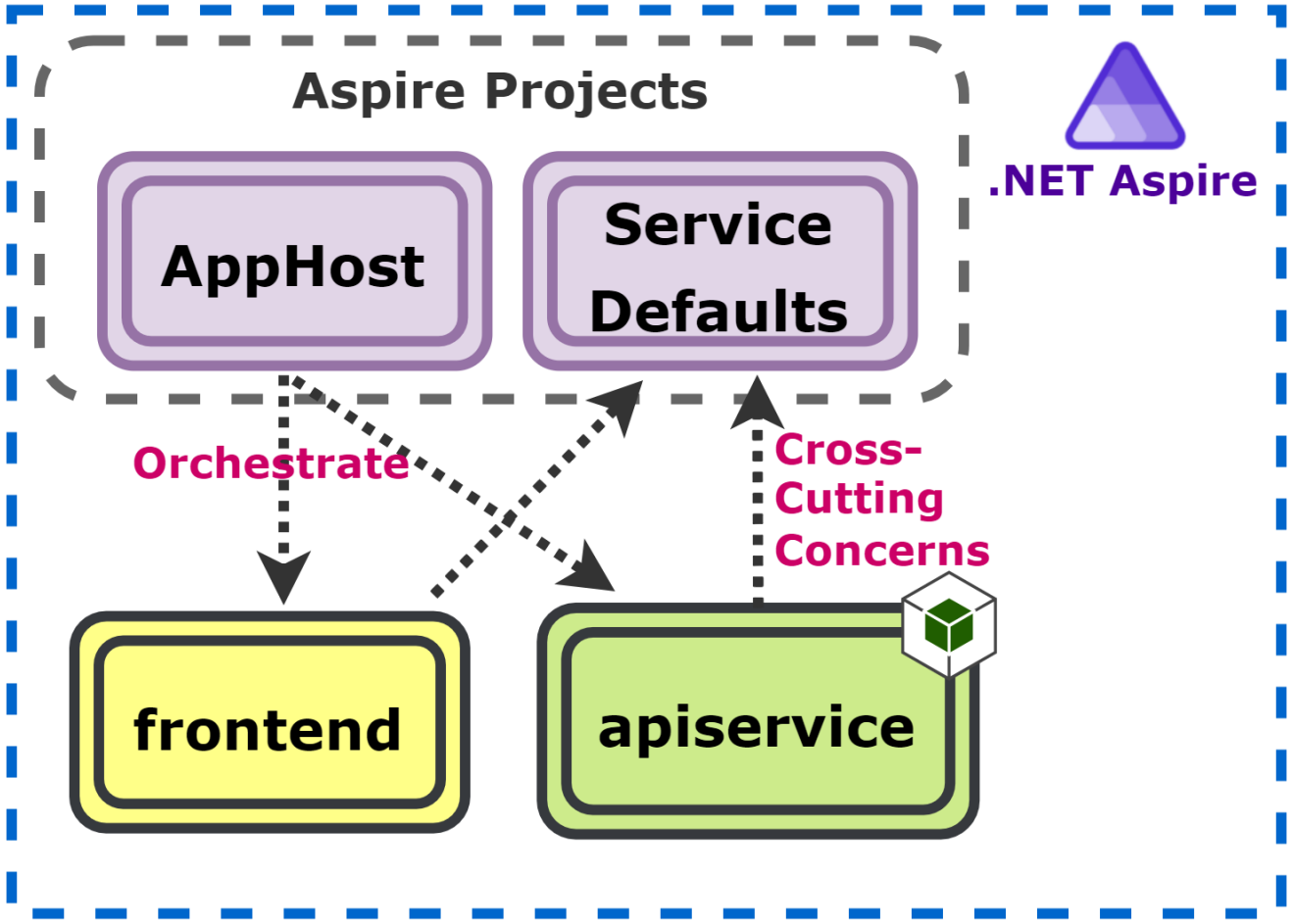
Module Project References for Async Call



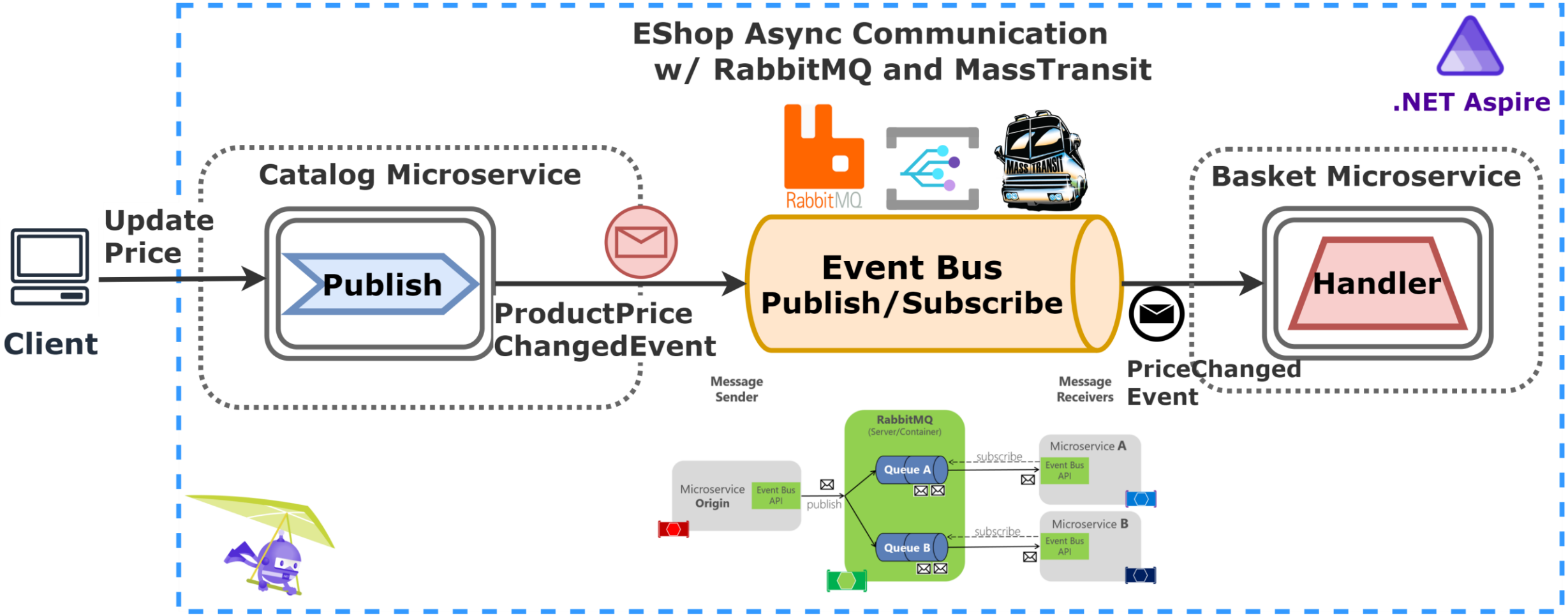
Project Dependencies

AppHost → references ApiService & Web for orchestration

ApiService/Web → reference ServiceDefaults for cross-cutting concerns



Async Communications w/ RabbitMQ & MassTransit



Secure Basket with Keycloak Authentication orchestrate .NET Aspire

Keycloak Identity and Access Management

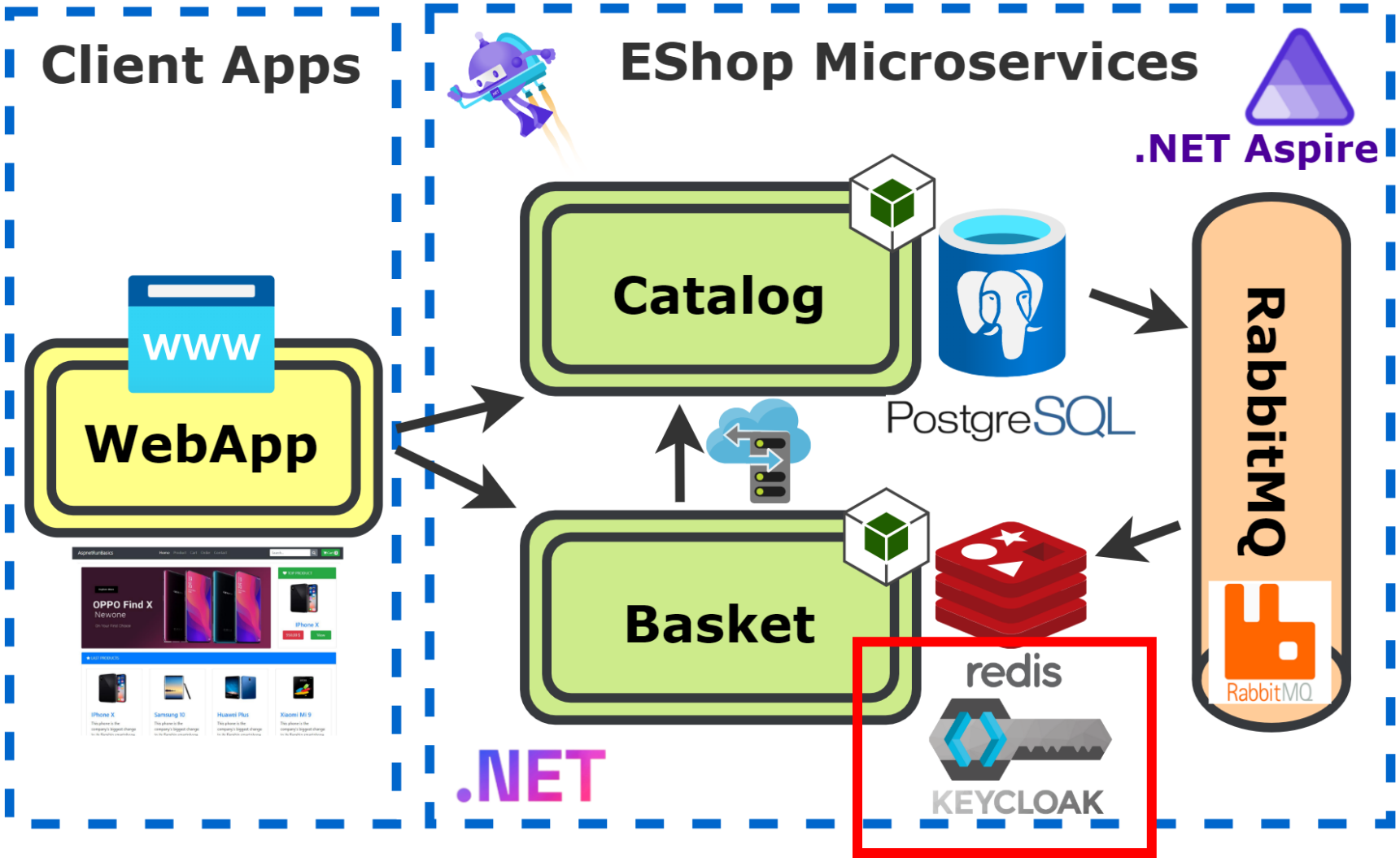
Keycloak Hosting Integration .NET Aspire

Create Realm, User and Client for OpenID Connect with Keycloak Identity Provider

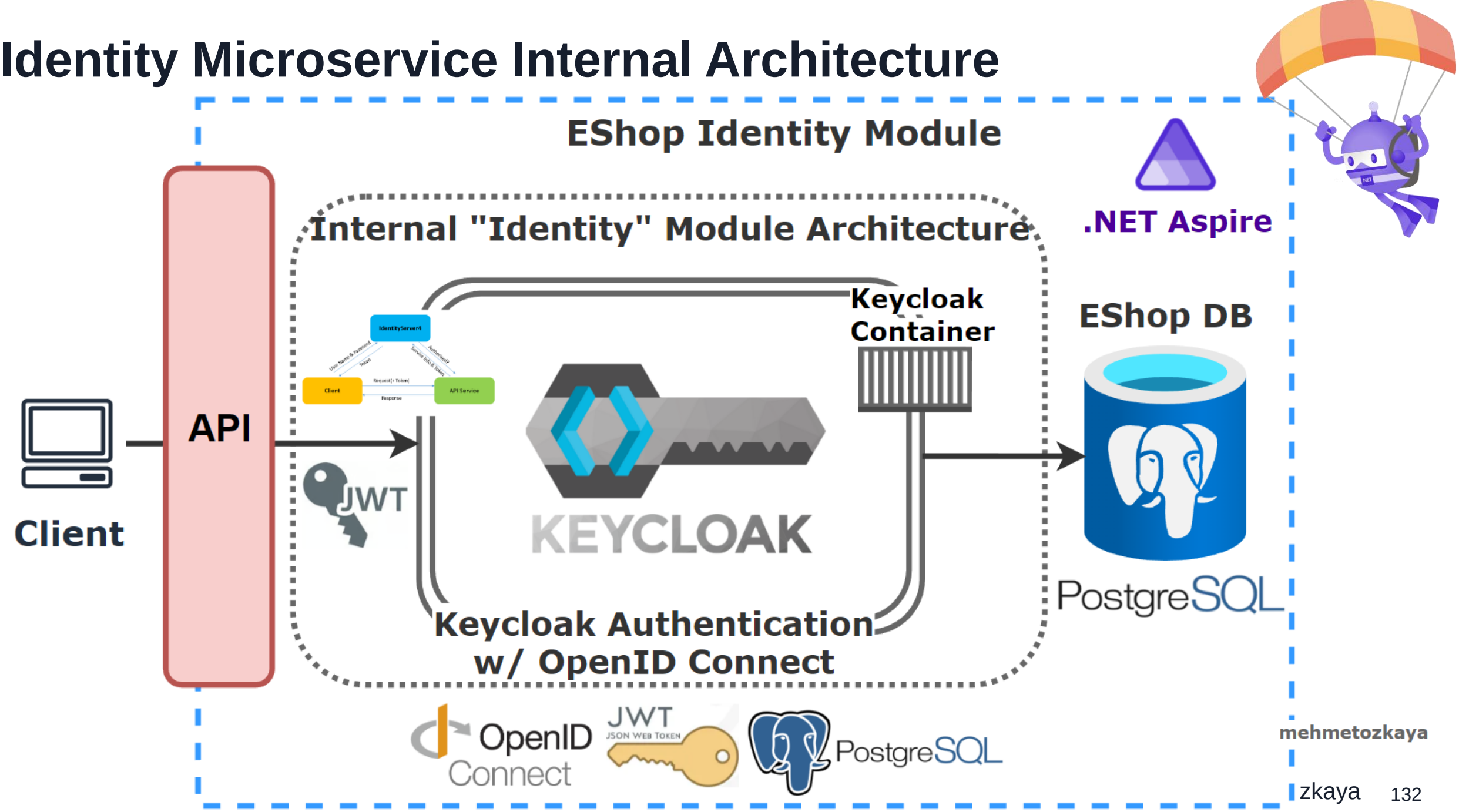
Keycloak Client Integration - Secure Basket ms endpoints w/ .Net Aspire

Mehmet Ozkaya

Secure Basket with Keycloak Authentication orchestrate .NET Aspire



Identity Microservice Internal Architecture



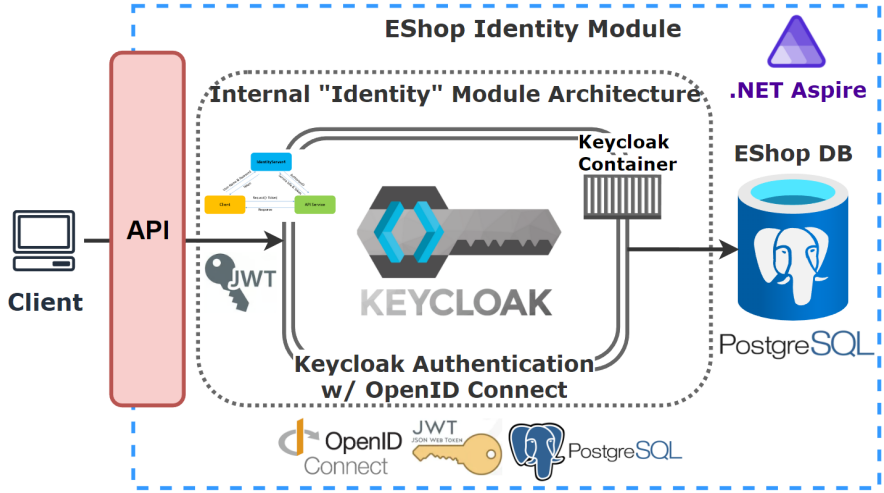
Secure Basket with Keycloak Authentication

Keycloak Identity and **Access Management**

OAuth2 + OpenID Connect Flows with **Keycloak**

Keycloak as a **Backing Services** into **.NET Aspire**

Setup Keycloak into **.NET Aspire** for **Identity Provider**



Definitions:

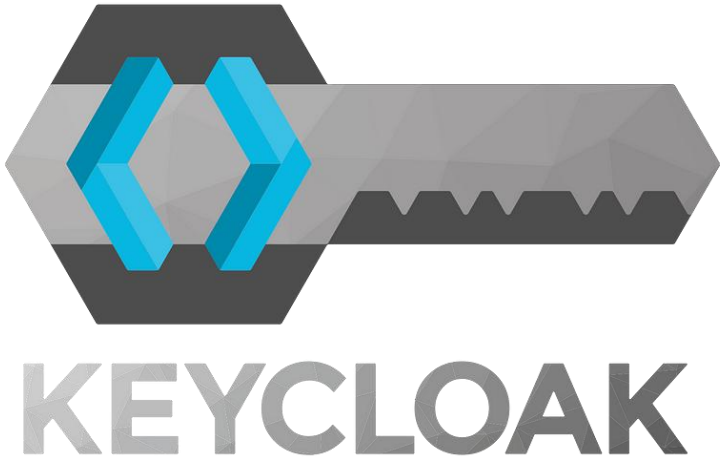
Create Realm, User and Client for OpenID Connect with Keycloak Identity

JwtBearer token for OpenID Connect with Keycloak Identity

Get Current User from Token with ClaimsPrincipal in Aspnet Authentication

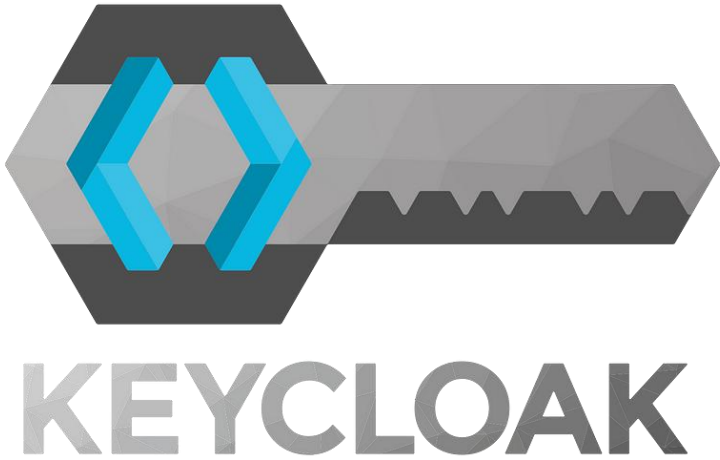
Keycloak Identity and Access Management

- **Open-source Identity** and **Access Management** solution aimed at modern applications and services.
- Provides **comprehensive support** for **authentication** and **authorization**, offering features such as **Single Sign-On (SSO), Identity Brokering**, and **Social Login**.
- **Simplifies** the **process** of **securing apps** and **services** with **minimal coding** effort.
- **Single Sign-On (SSO):** Users can log in once and access multiple app.
- **Identity Brokering and Social Login:** Integrate with external identity providers like Google, Facebook, etc.
- **User Federation:** Connect to existing user databases.
- **Centralized Management:** Manage all user identities and permissions in one place.
- **Standard Protocols:** Supports OAuth 2.0, OpenID Connect, SAML 2.0.



Why Use Keycloak ?

- Significantly **reduce the complexity** of **managing user identities** and **access controls** in your apps.
- **Security:** Robust security features that adhere to industry standards.
- **Scalability:** Capable of handling millions of users and requests.
- **Flexibility:** Easy to integrate with various applications and services.
- **User Experience:** Enhances user experience with SSO and Social Login.



OAuth2 + OpenID Connect

OAuth2

- Authorization framework that allows apps to obtain limited access to user accounts on an HTTP service.
- Delegating user authentication to the service that hosts the user account and authorizing third-party apps to access the user account.
- **Key Concepts of OAuth2:** Resource Owner, Client, Authorization Server, Resource Server.

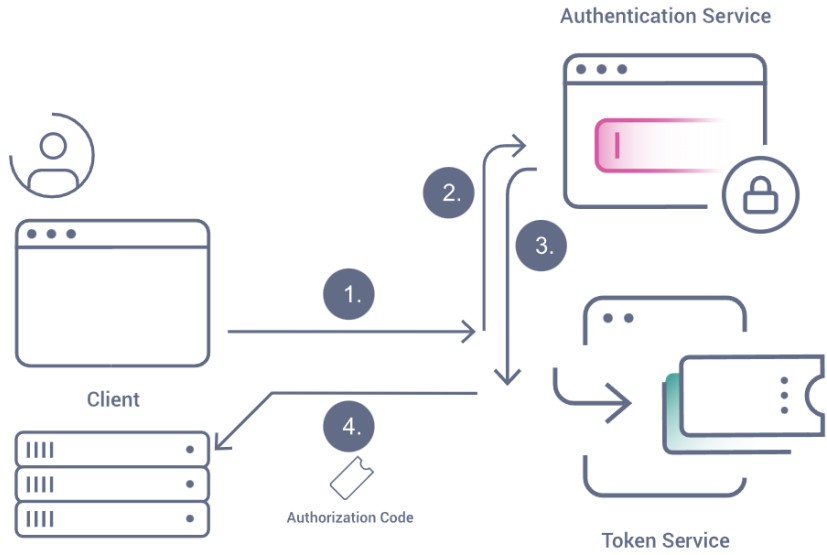
OpenID Connect (OIDC)

- Identity layer built on top of OAuth2.
- Clients to verify the identity of the end-user based on the authentication performed by an authorization server and to obtain basic profile information about the end-user.
- **ID Token:** A JSON Web Token (JWT) that contains user profile info.
- **UserInfo Endpoint:** An endpoint that returns additional profile information about the user.

OAuth2 + OpenID Connect Flows

Authorization Code Flow

- 1. The client directs the user to the Keycloak login page.
- 2. The user authenticates and consents.
- 3. Keycloak redirects the user back to the client with an authorization code.
- 4. The client exchanges the authorization code for an access token and an ID token.



Implicit Flow

- SPAs where tokens are returned directly to the client. However, it's less secure and generally not recommended for new applications.

Client Credentials Flow

- Used for machine-to-machine (M2M) communication. The client authenticates directly with Keycloak using its client ID and secret to obtain an access token.

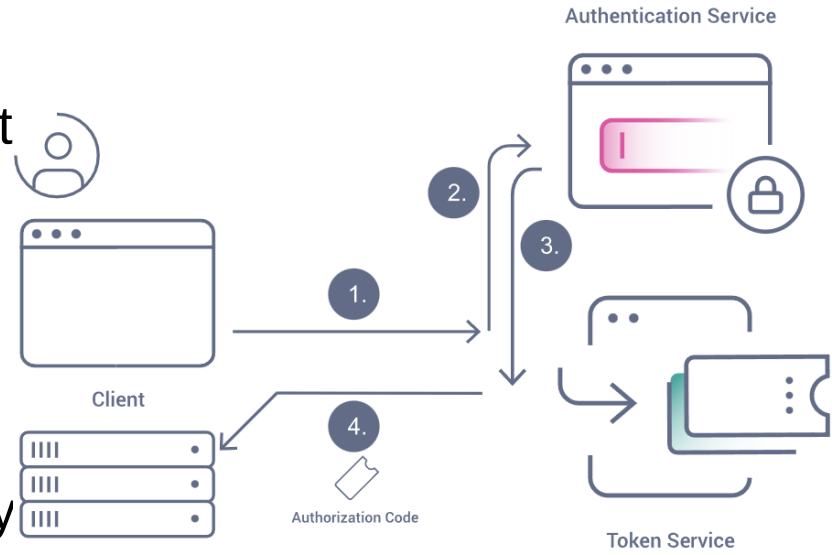
OAuth2 + OpenID Connect Flows - 2

Resource Owner Password Credentials Flow

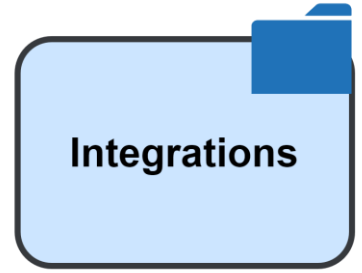
- The client to collect the user's credentials and exchange them for an access token.
- This flow is suitable only for highly trusted clients and is generally not recommended.

OpenID Connect Flow

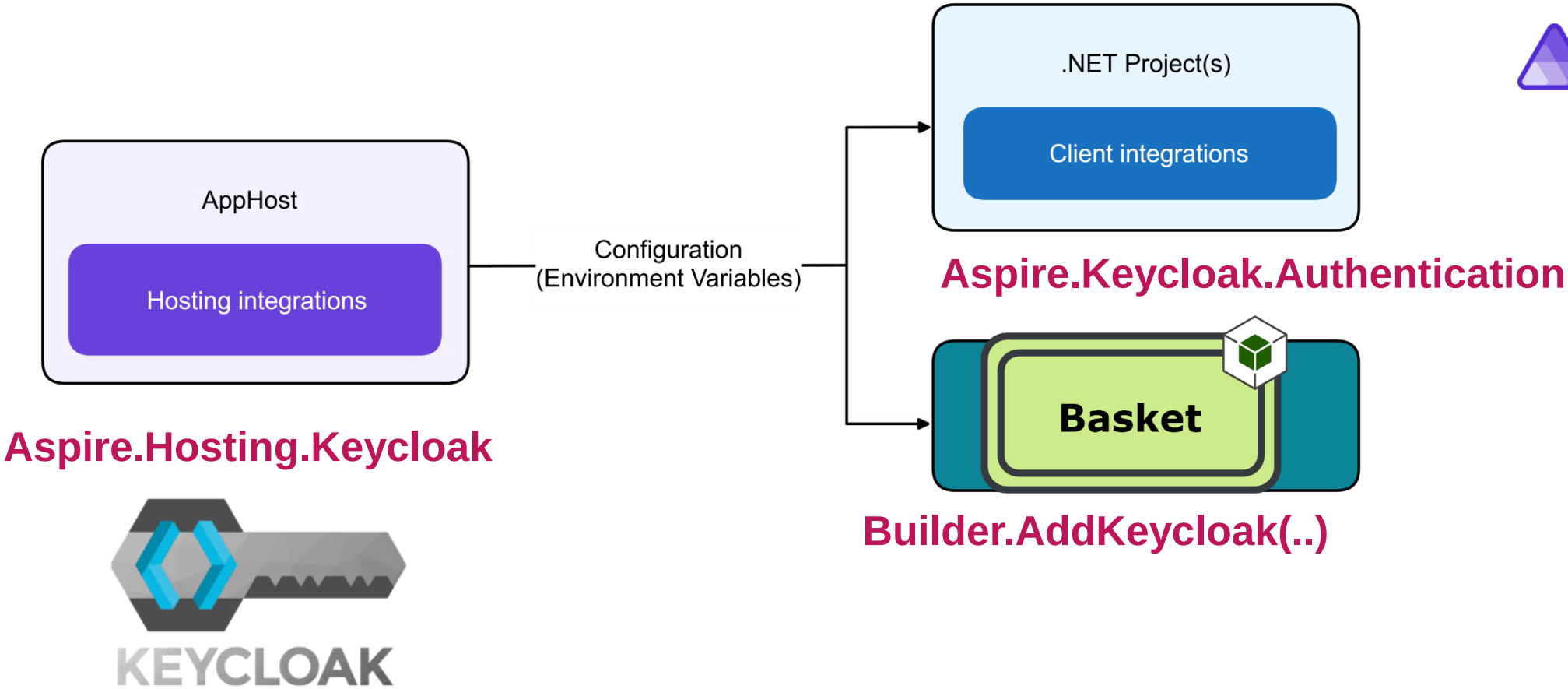
- Extends the OAuth2 Authorization Code Flow to include an ID token, which contains user profile information.
- This flow is ideal for applications that need to verify the user's identity and obtain basic profile information.



Secure Basket with Keycloak Authentication



.NET Aspire



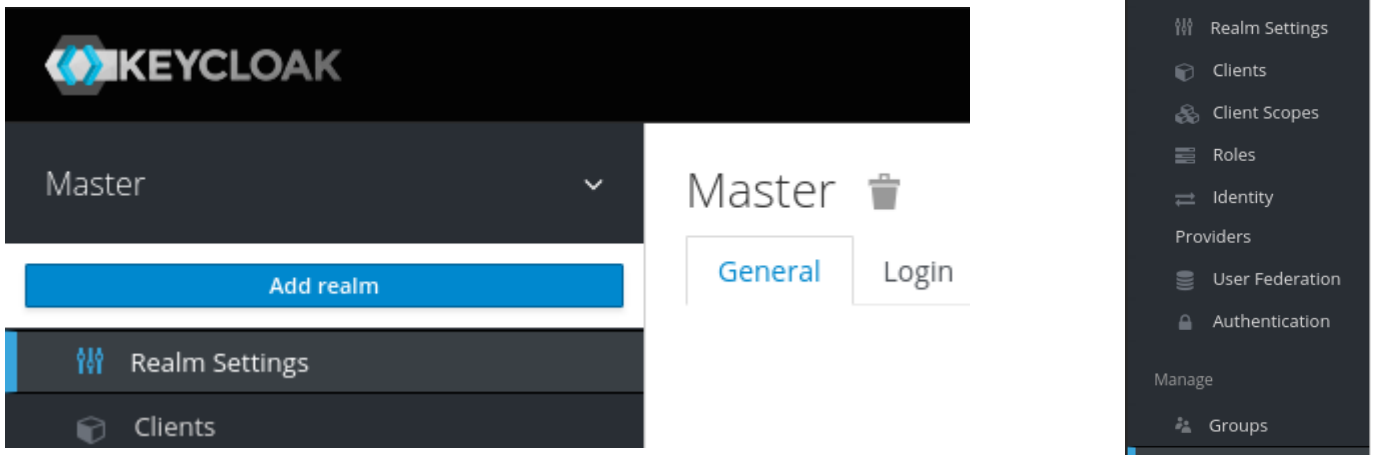
Aspire.Hosting.Keycloak

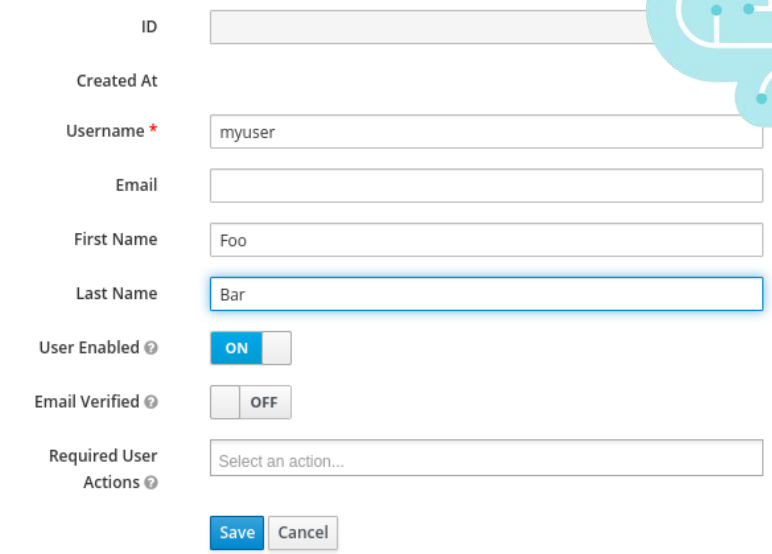
KEYCLOAK

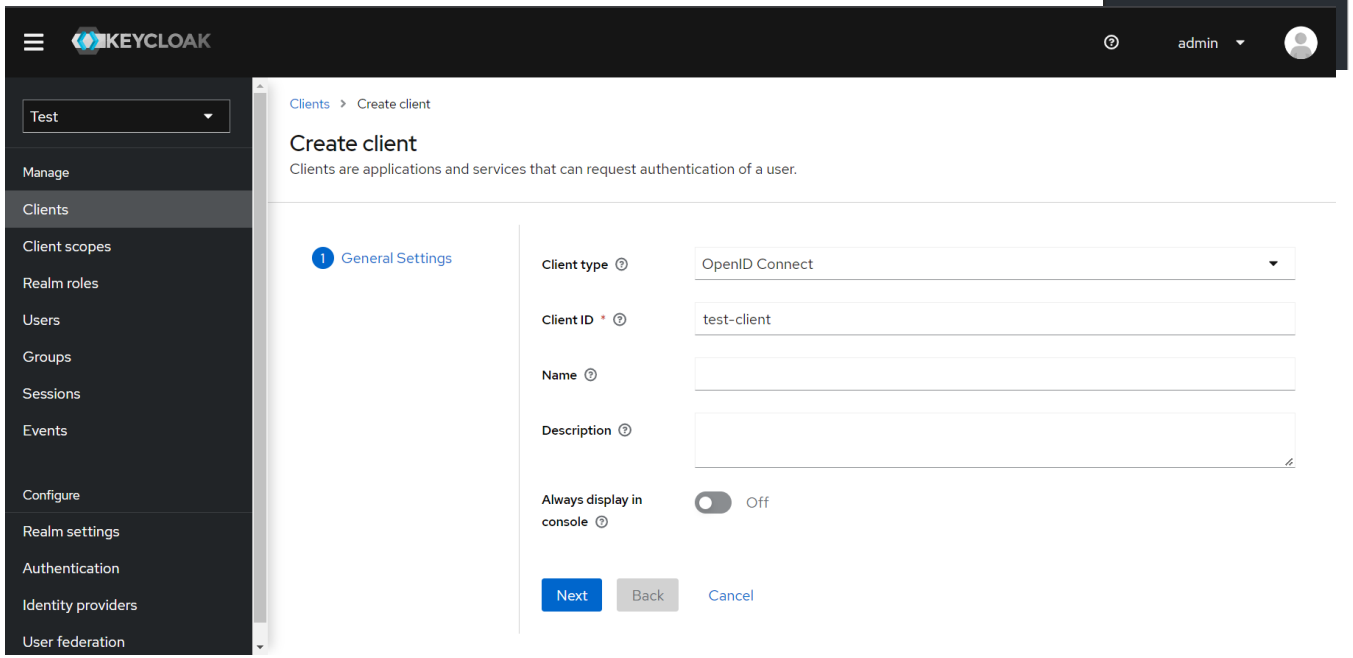
Aspire.Keycloak.Authentication

Builder.AddKeycloak(..)

Create Realm, User and Client for OpenID Connect w/ Keycloak Identity Provider

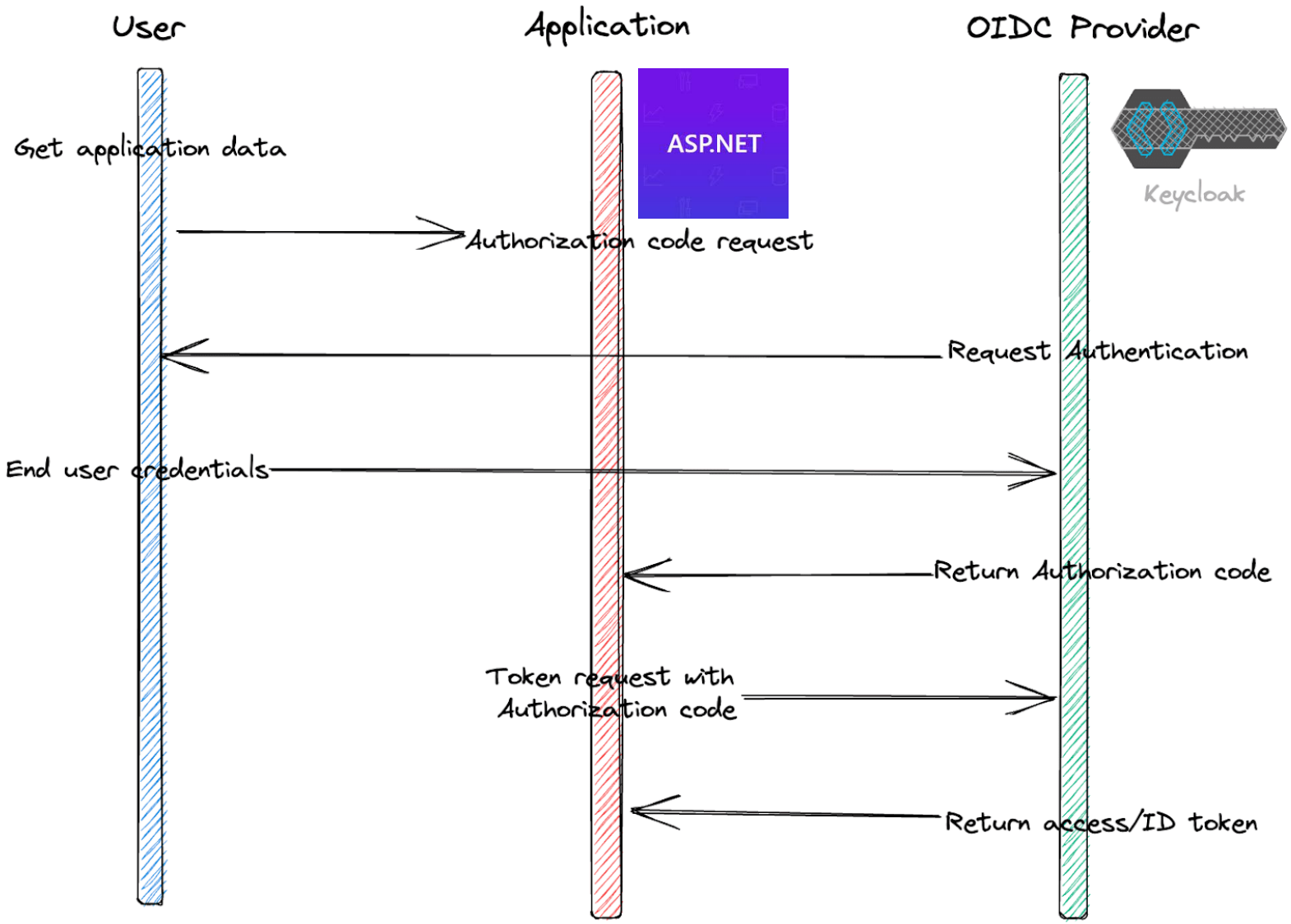


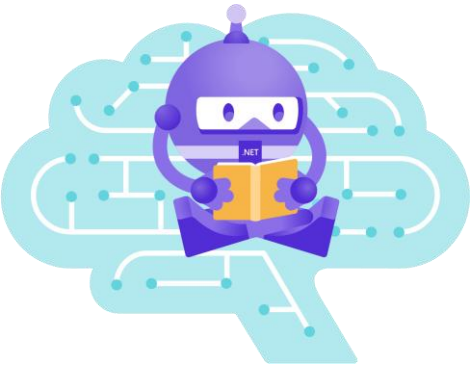




https://learn.microsoft.com/en-us/dotnet/architecture/microservices/

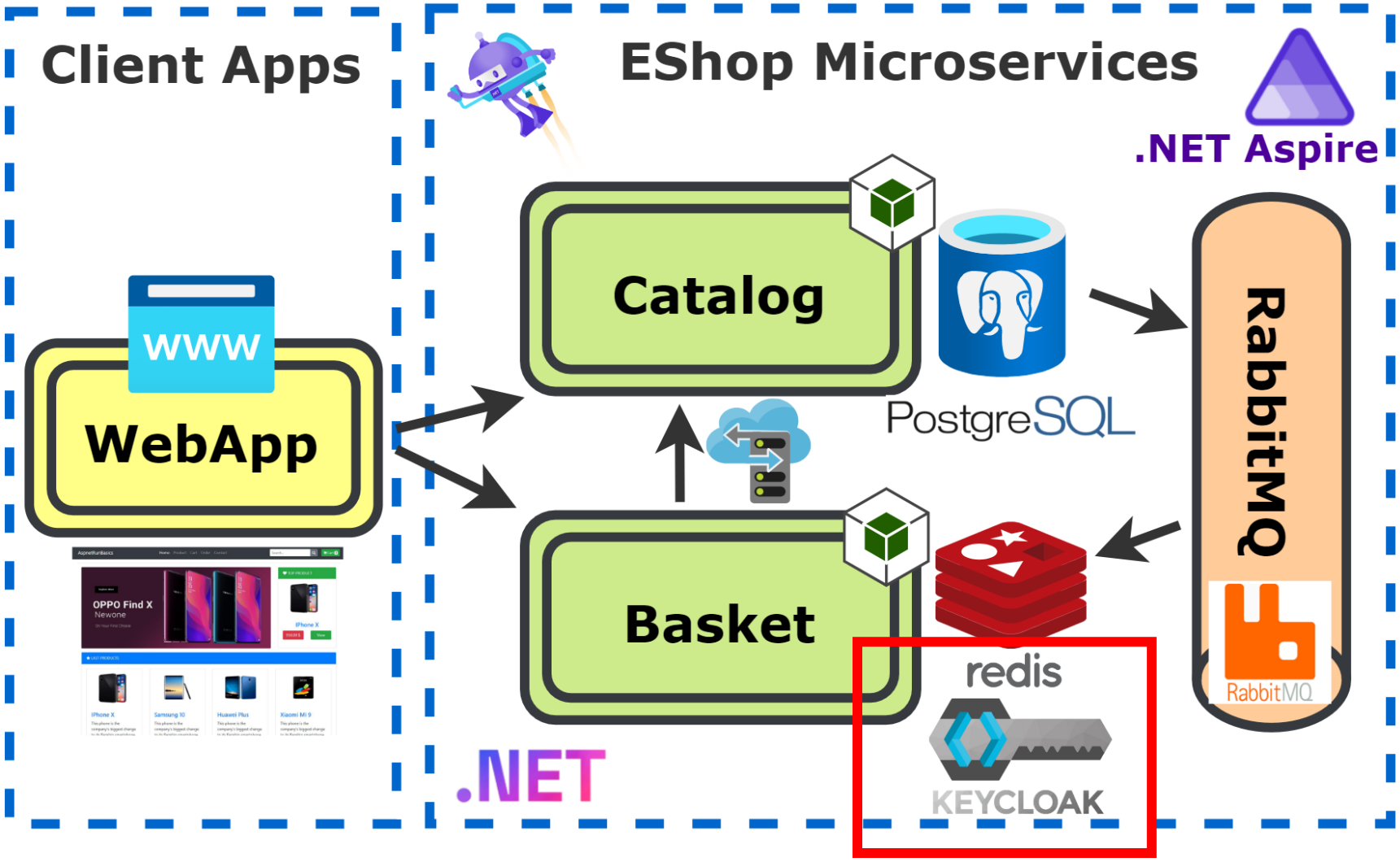
Secure EShop Modules with Keycloak OpenId Connect



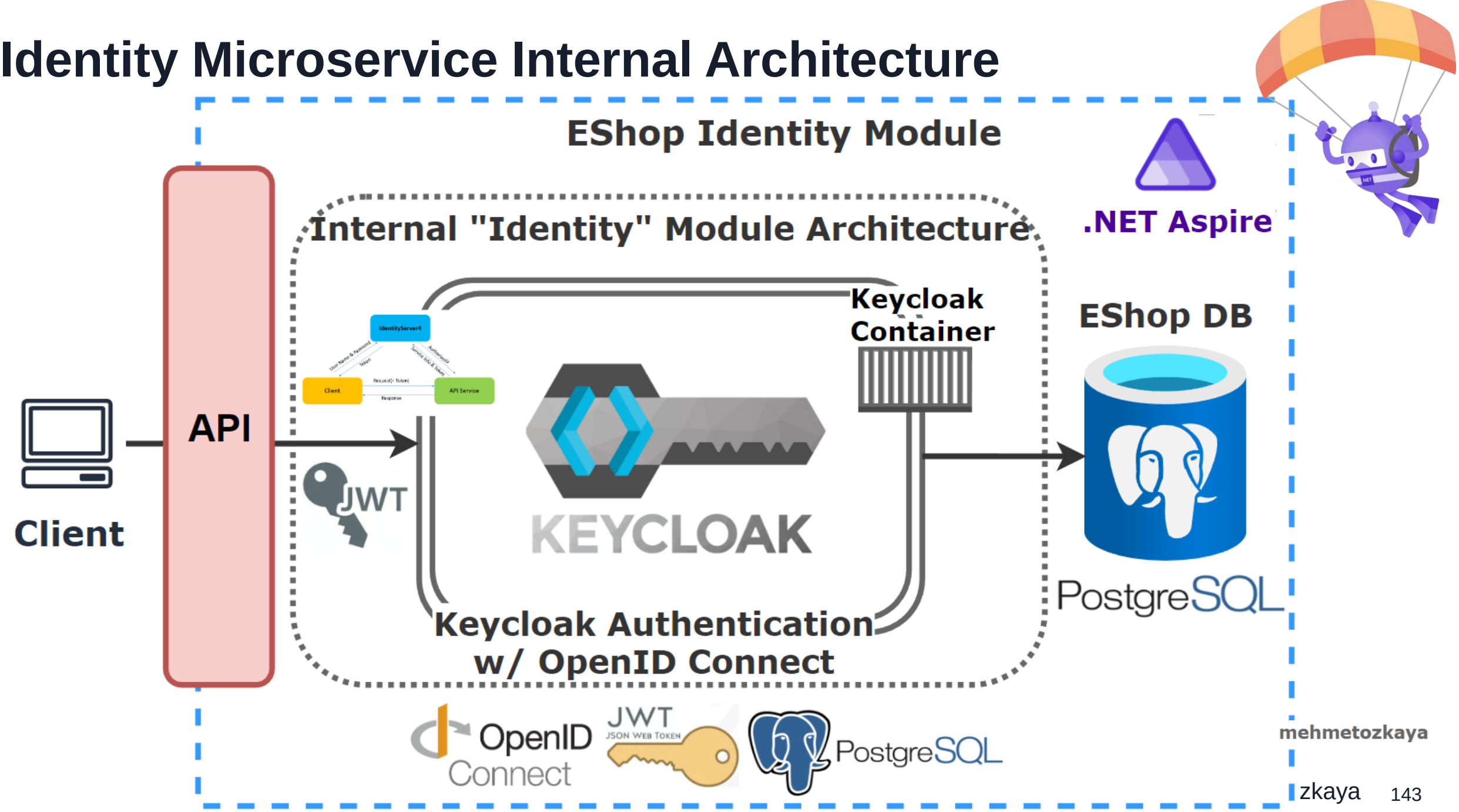


https://www.cncf.io/blog/2023/05/17/securing-cloud-native-microservices-with-role-based-access-control-using-keycloak/

Secure Basket with Keycloak Authentication orchestrate .NET Aspire



Identity Microservice Internal Architecture



Develop Client Blazor Web Application

Create Blazor Web Application orchestrate .NET Aspire

Client WebApp integrate to Catalog w/ CatalogApiClient.cs

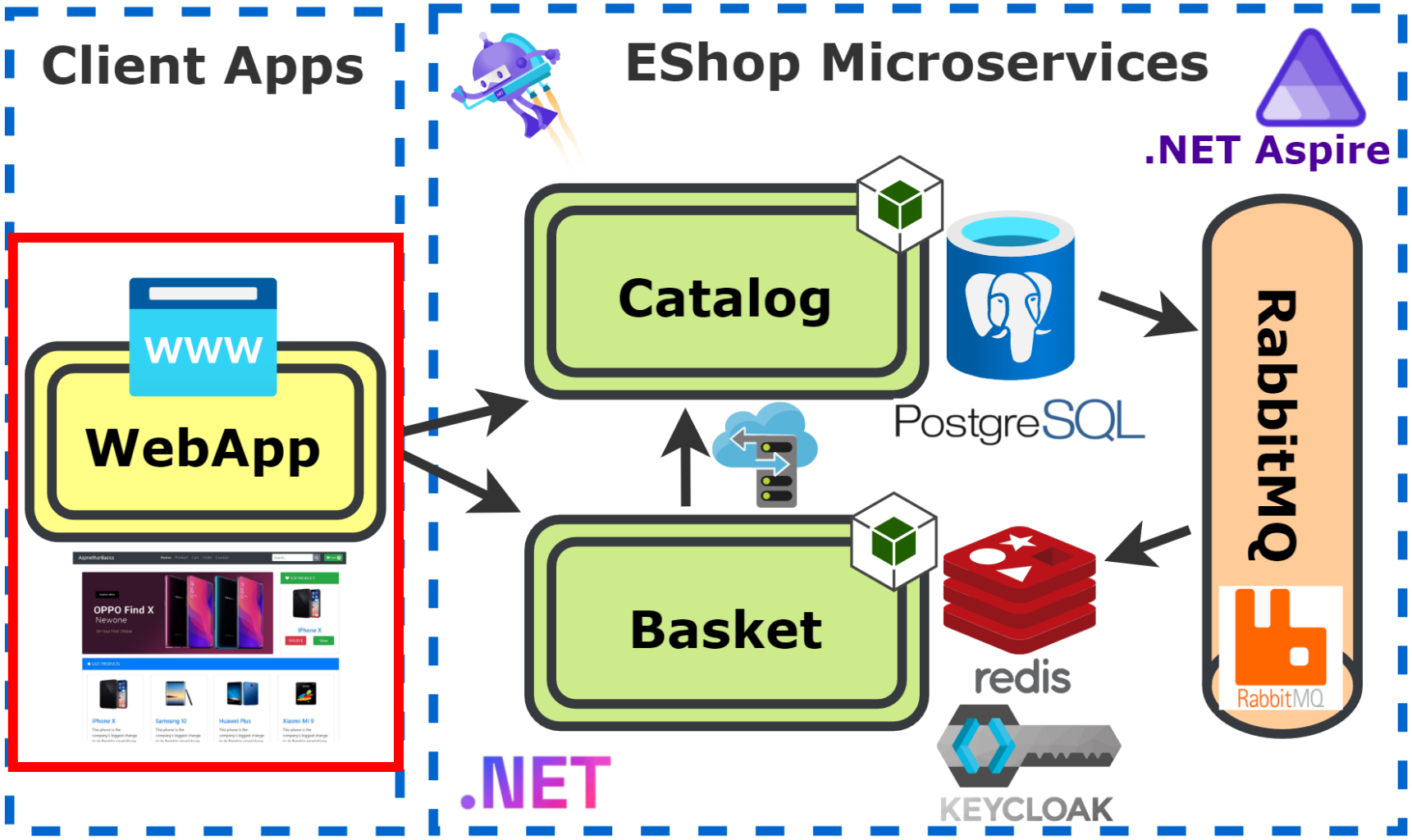
Register CatalogApiClient with Aspire integrations for Service Discovery

WebApp Product Page Development

Output Caching for Products Page

Mehmet Ozkaya

Develop Client Blazor Web Application

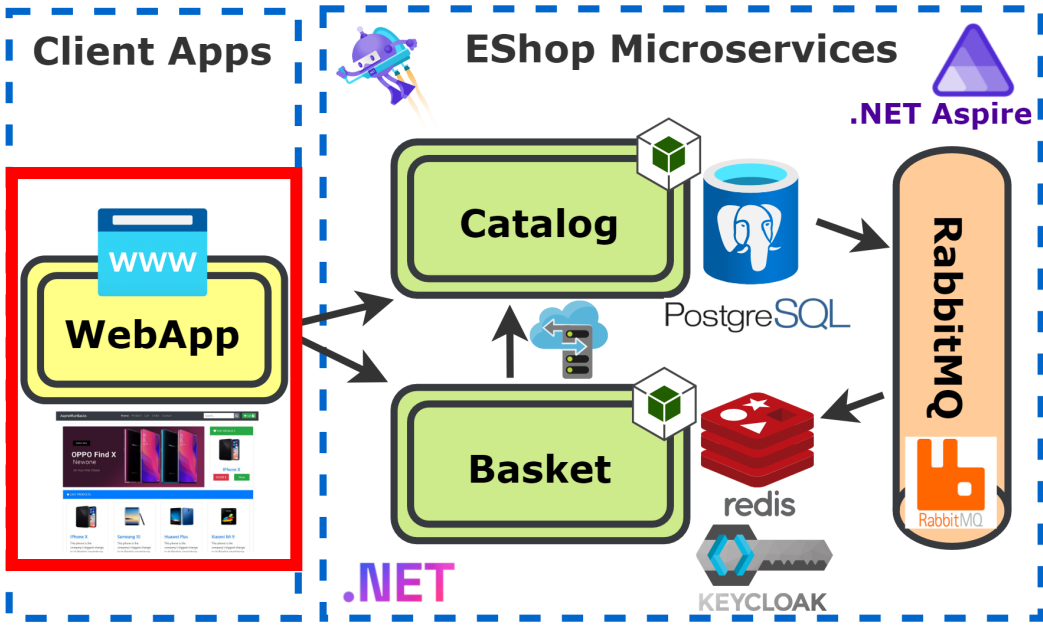


Develop Client Blazor Web Application

Blazor Web App and include .NET Aspire

Products razor page and populate Products from Catalog microservice

Output Caching



Later:

Deploy .NET Aspire distributed apps into ACA

Develop Semantic Search and AI-related pages

Develop Client Blazor Web Application

Products

Here are some of our amazing outdoor products that you can purchase.

Image	Name	Description
	Hiking Poles	Ideal for camping and hiking trips
	Outdoor Rain Jacket	This product will keep you warm and dry in all weathers
	Survival Kit	A must-have for any outdoor adventurer
	Outdoor Backpack	This backpack is perfect for carrying all your outdoor essentials

Remember: Service Discovery

.NET Aspire

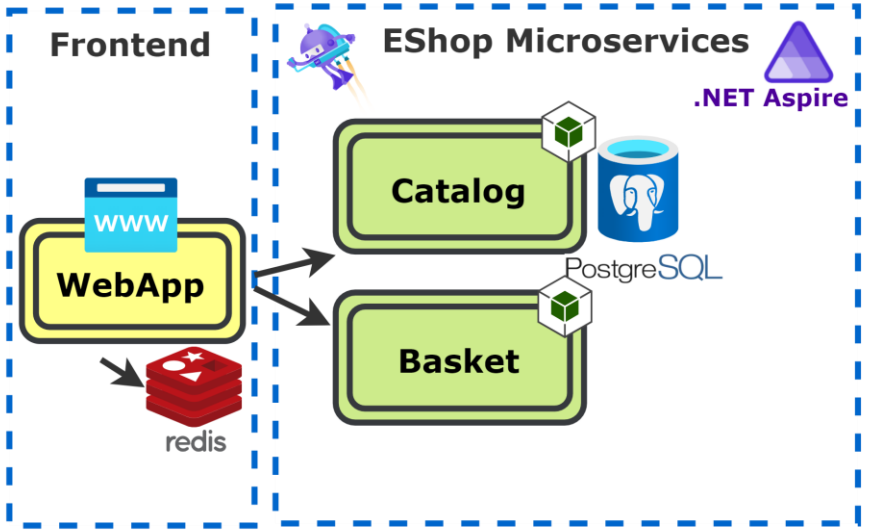
Service Discovery

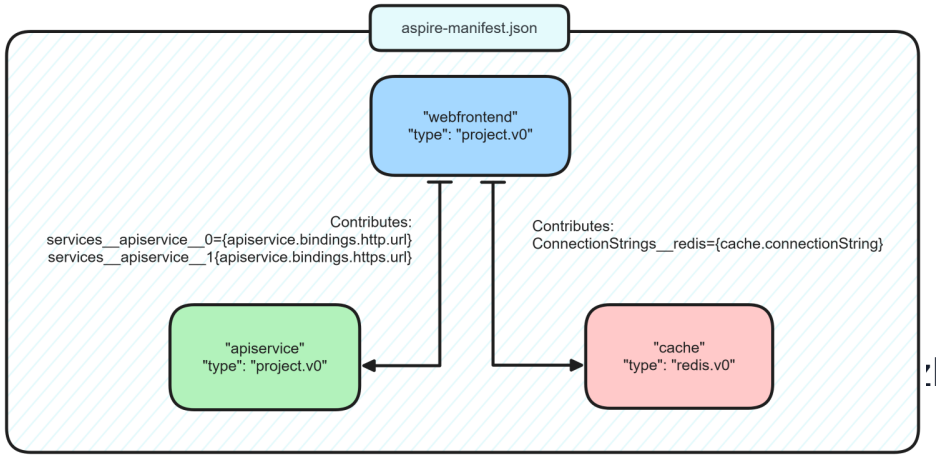
Containers→injects→connection string

Projects→injects→microservice endpoints

```
var builder = DistributedApplication.CreateBuilder(args);

var cache = builder.AddRedis("cache");

var catalog = builder.AddProject<Projects.CatalogService>
("catalog");
var basket = builder.AddProject<Projects.BasketService>("basket");

var frontend = builder.AddProject<Projects.MyFrontend>("frontend")
  .WithReference(cache)
  .WithReference(catalog)
  .WithReference(basket);
```

```
WithReference(cache)
  ConnectionStrings__cache="localhost:62354"

WithReference(apiservice)
  services__apiservice__http__0="http://localhost:5455"

 services__apiservice__https__0="https://localhost:7356"
```





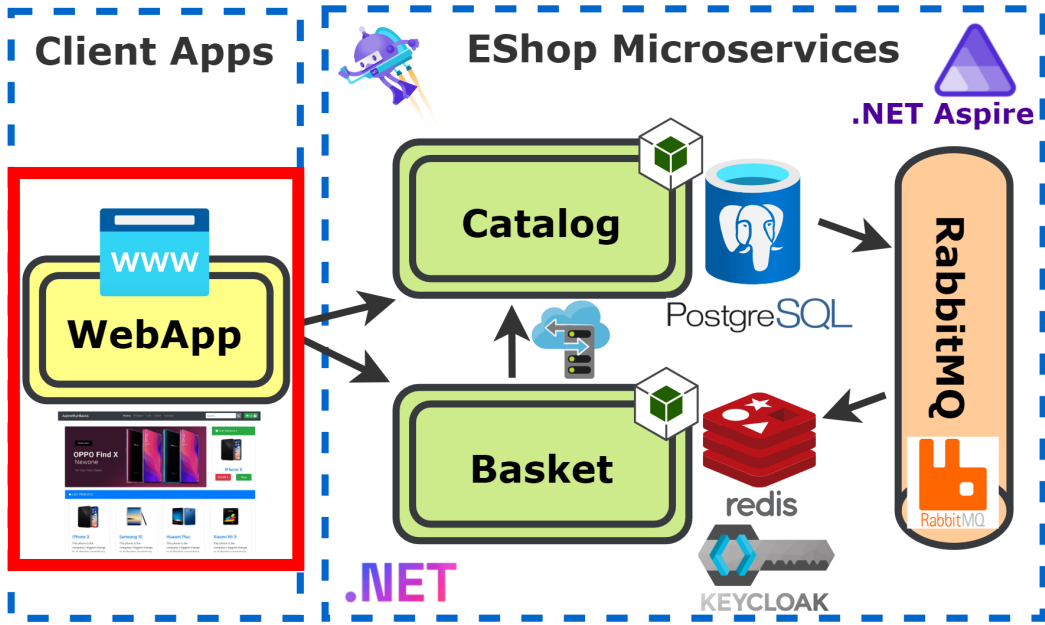
:kaya

What Is Output Caching?

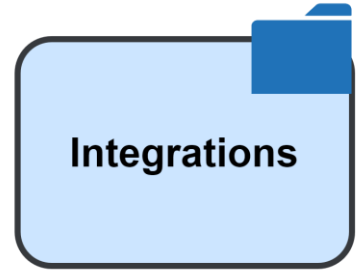
HTML output is cached, so repeated requests return the same content without re-running logic

If data changes frequently, cache duration or invalidation must be considered

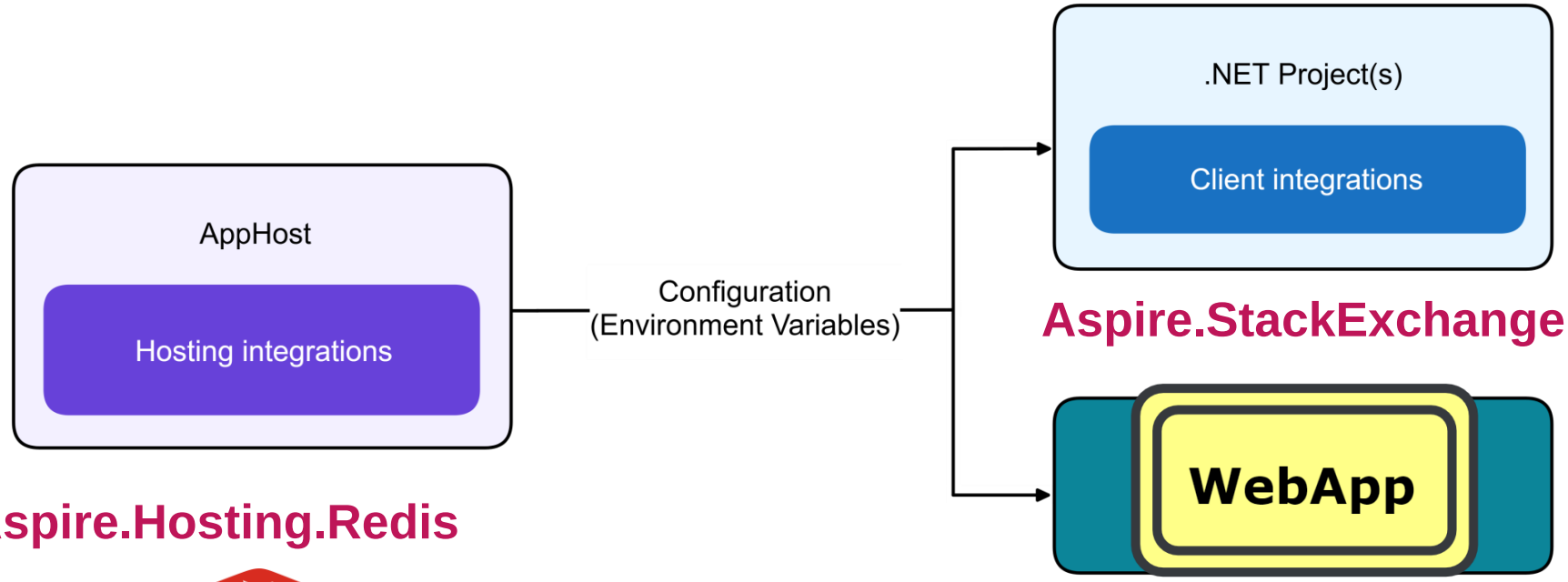
.NET Aspire integrates with Redis for distributed output caching



Secure Basket with Keycloak Authentication



.NET Aspire



Aspire.StackExchange.Redis.OutputCaching

Aspire.Hosting.Redis

redis

```
builder.AddRedisOutputCache("cache");
app.UseOutputCache();
```

Deploy EShop Aspire Project to Azure Container Apps

What is ACA - Azure Container Apps ? How to Deploy ?

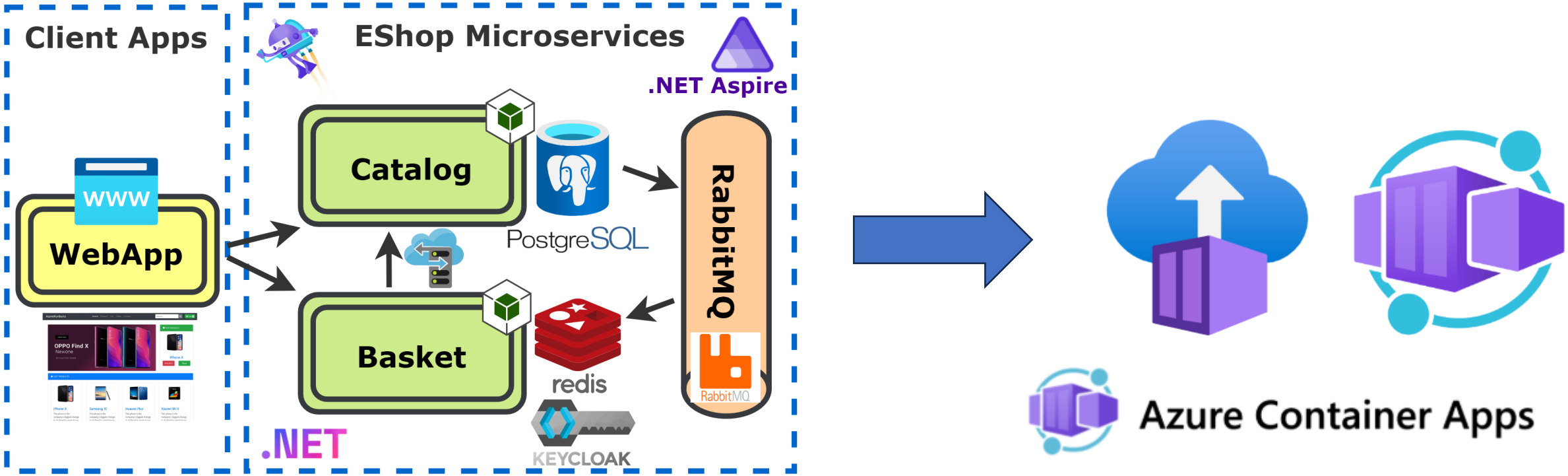
Data volumes don't work on ACA

Deploy .NET Aspire App with azd commands to ACA

Test Successful Deployment to ACA

Mehmet Ozkaya

Deploy EShop Aspire Project to Azure Container Apps



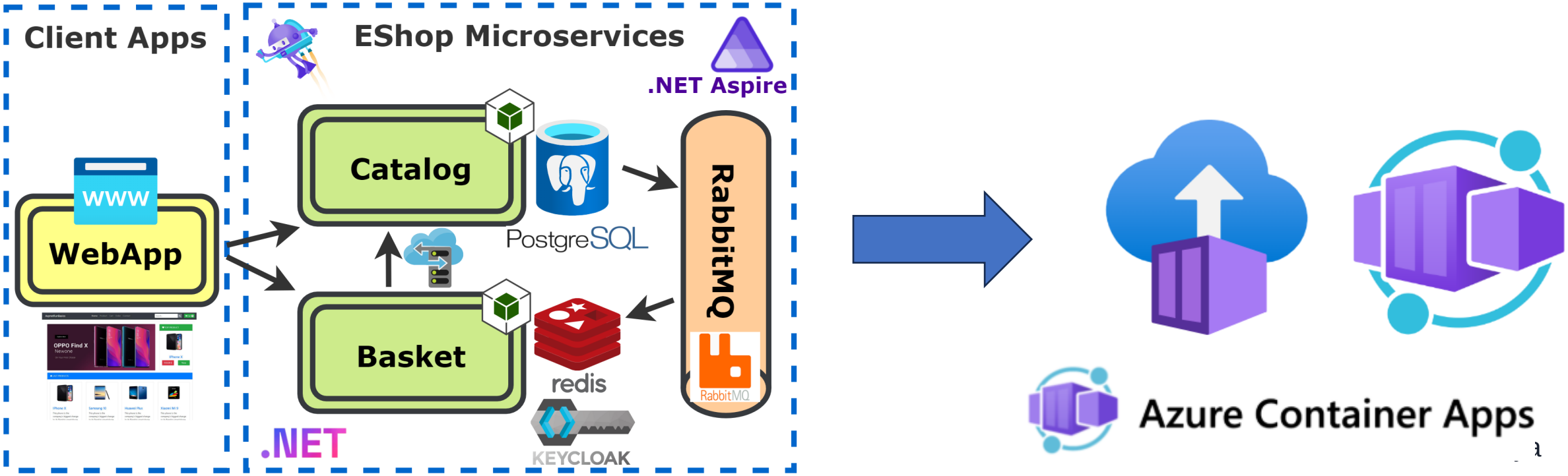
Deploy EShop Aspire Project to Azure

Azure Container Apps

Deploy a .NET Aspire project to Azure Container Apps

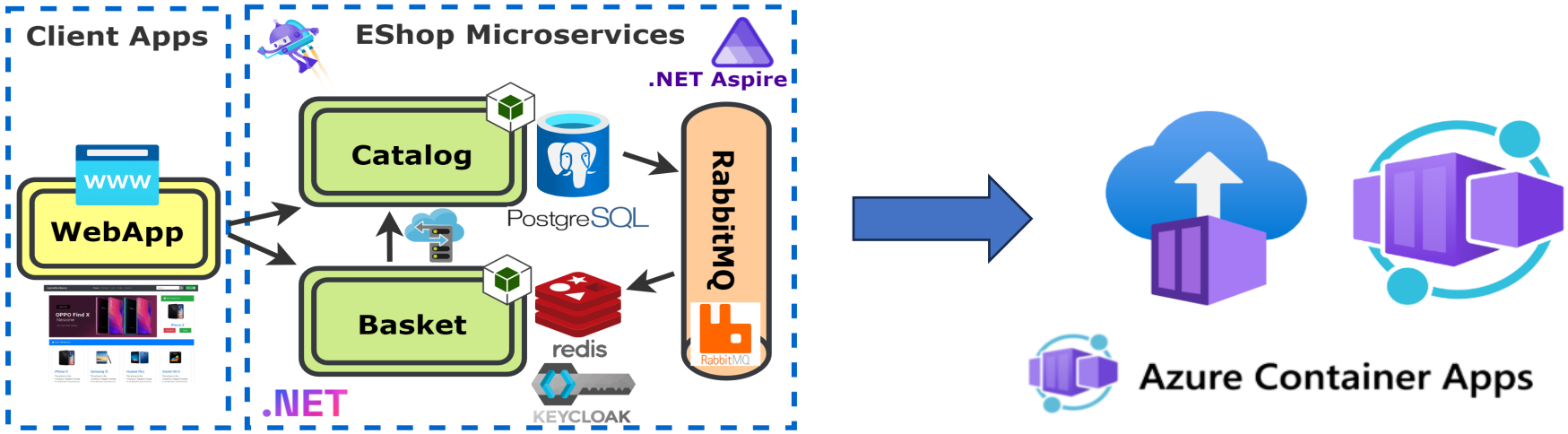
Deploy .NET Aspire projects with azd (Azure Developer CLI)

Azure Dashboard



Steps to Deploy EShop Aspire Project to Azure

1. Provision an Azure resource group and Container Registry
2. Publish the .NET Aspire projects as container images in Azure Container Registry
3. Provision a backing containers (postgres, redis) in Azure
4. Deploy the apps to an Azure Container Apps environment
5. View application console logs to troubleshoot application issues

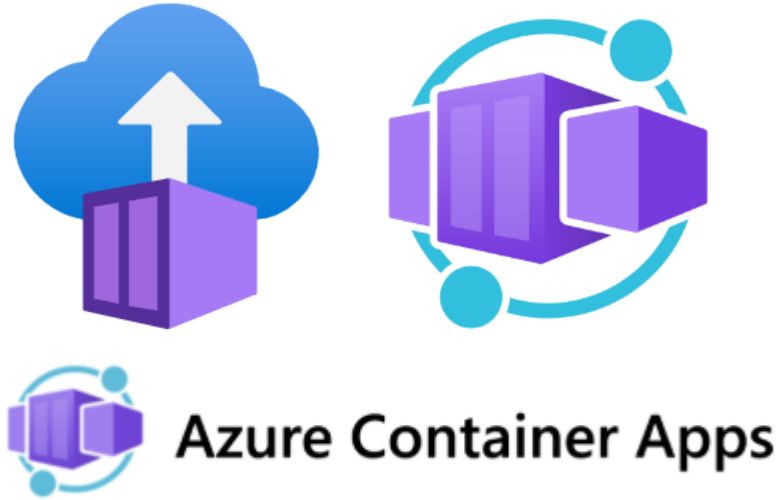


Azure Container Apps (ACA)

Microsoft's **managed container hosting** for **microservices**

Simplifies running containers in a **serverless** fashion

Perfect for **.NET Aspire solutions** needing **easy scale** & **integrated logs**



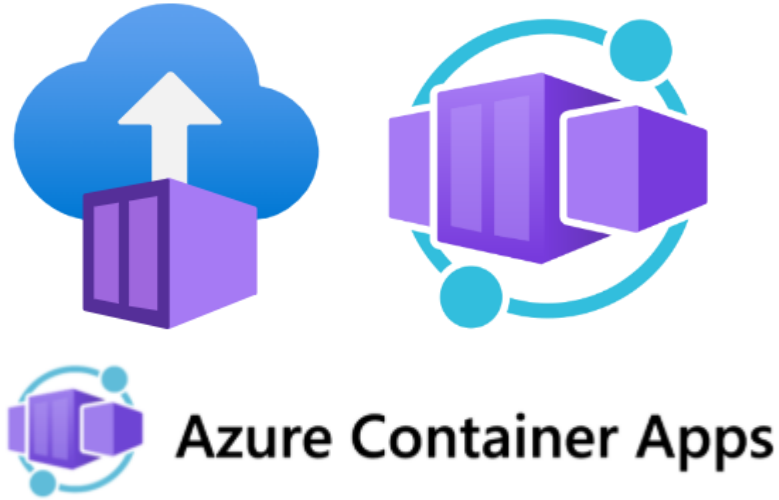
Key Benefits of Azure Container Apps

Automatic scaling and **revision management**

Dapr integration (optional) for sidecar patterns

KEDA for **event-based scaling**

Simple **container-based deployment** from the command line



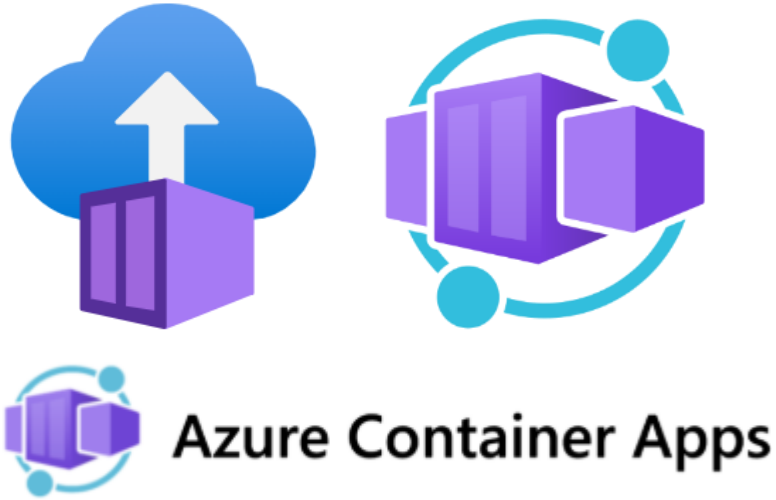
Deploying a .NET Aspire Project to ACA

Containerize each microservice automatically with .NET Aspire

azd init to set up environment files

azd up to provision Azure resources

azd down to tear them down



Detailed azd commands

azd init

Creates .azure folder or config files for naming & region choices

Optionally picks an existing subscription

Typically done once per solution

Azure Container Apps

azd up

Builds containers, pushes to Azure Container Registry

Provisions Container Apps environment, secrets, and logs

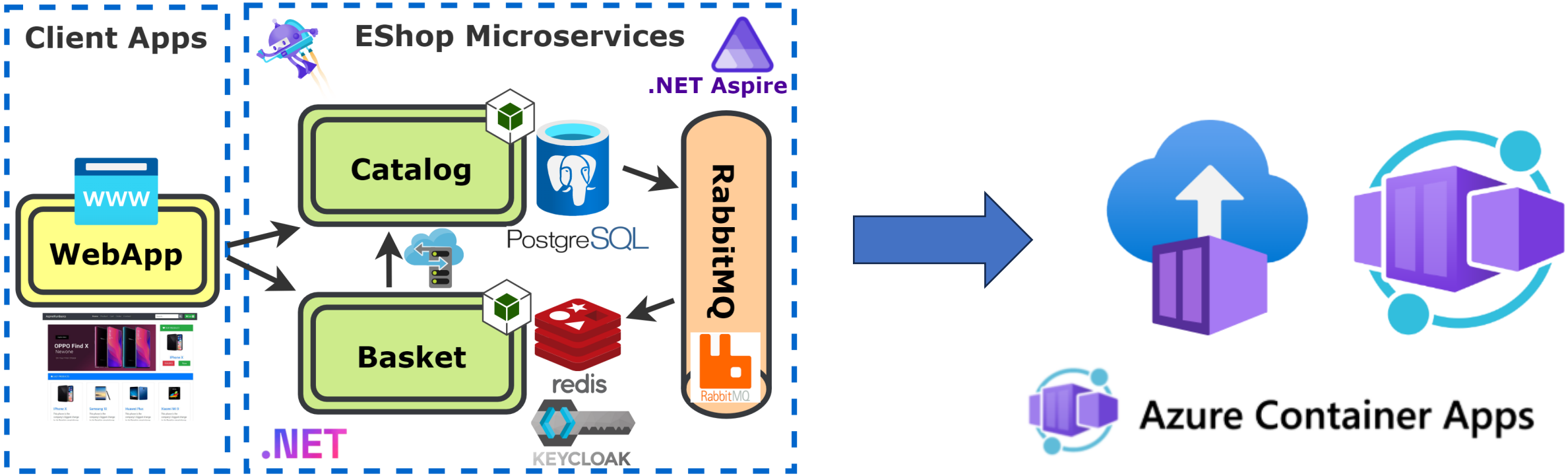
Deploys each microservice in your .NET Aspire solution

azd down

Frees up resource usage in your subscription

Removes container apps, registry, logs, secrets

Deploy EShop Aspire Project to Azure Container Apps

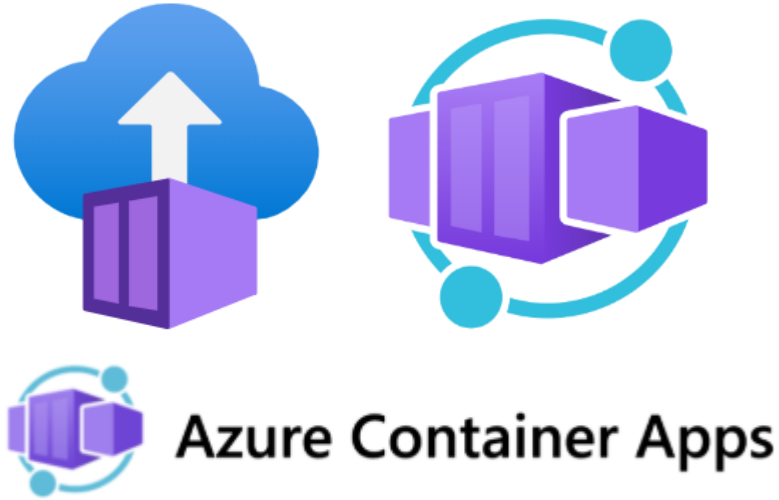


Data volumes don't work on ACA

ACA has Azure Storage limitations for container volumes

Some Linux containers (Postgres, etc.) can’t mount volumes properly

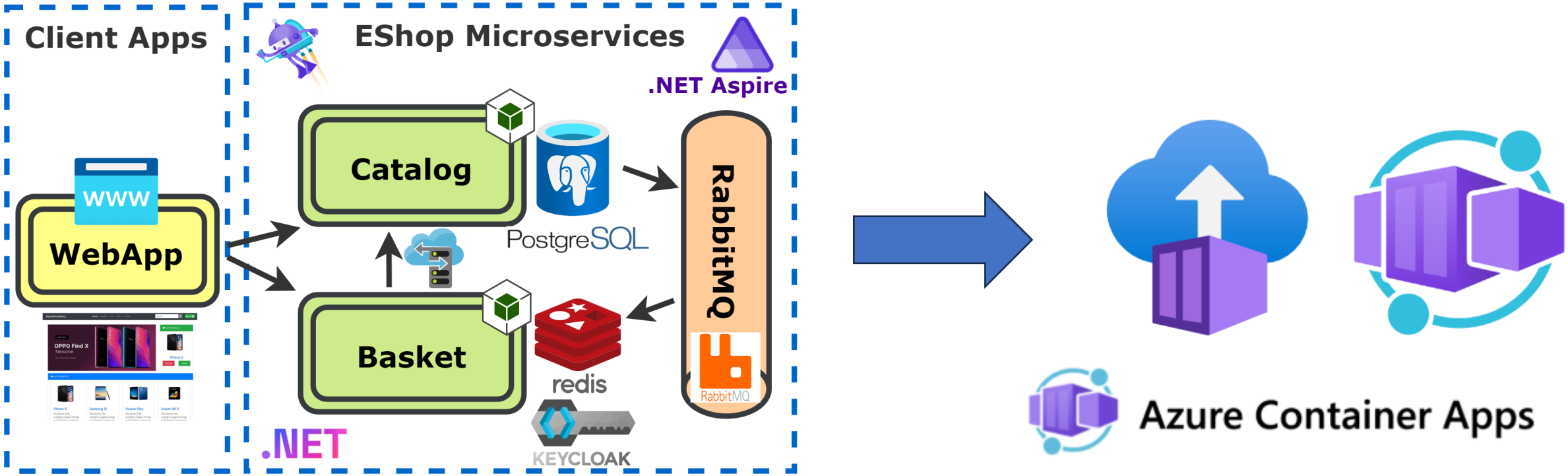
.NET Aspire approach: conditionally add volumes or switch to cloud services



[dotnet/aspire #6671 GitHub Issue]

[.NET Aspire Docs – Azure Postgres Resource]

Deploy EShop Aspire Project to Azure Container Apps

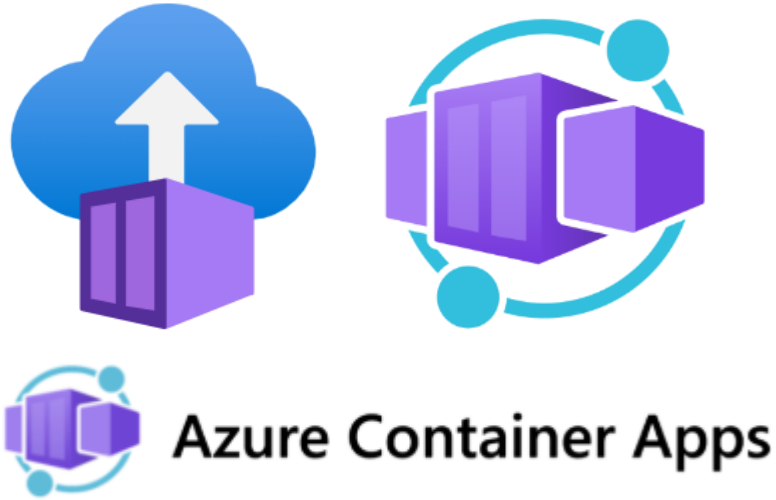


Prerequisites

An Azure subscription (e.g., Free Trial)

azd CLI installed on your machine

A .NET Aspire solution with container-based microservices



azd version

azd auth login

azd init

azd up

azd down

.NET GenAI with Microsoft.Extensions.AI for Chat AI and Semantic Search

What are Large Language Models (LLMs) ?

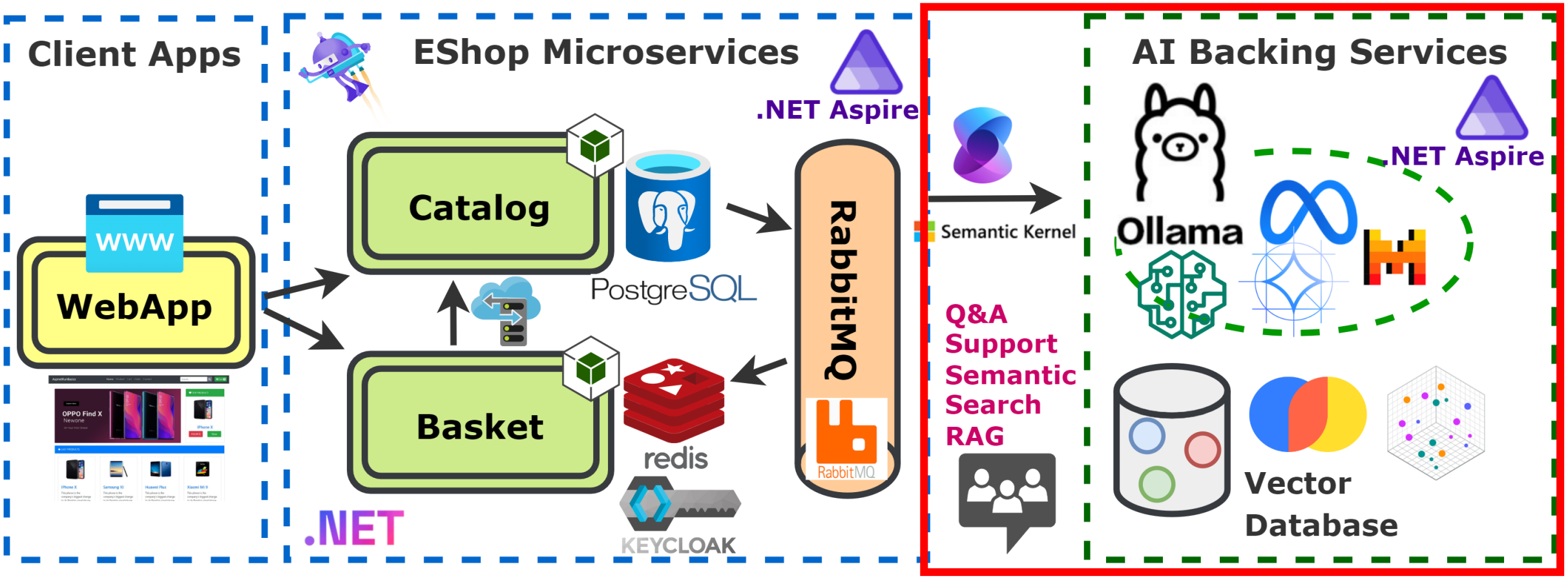
Ollama – Run LLMs Locally

Using LLMs and VectorDBs as Cloud-Native Backing Services

Download Llama Model LLM inside Ollama with .NET Aspire

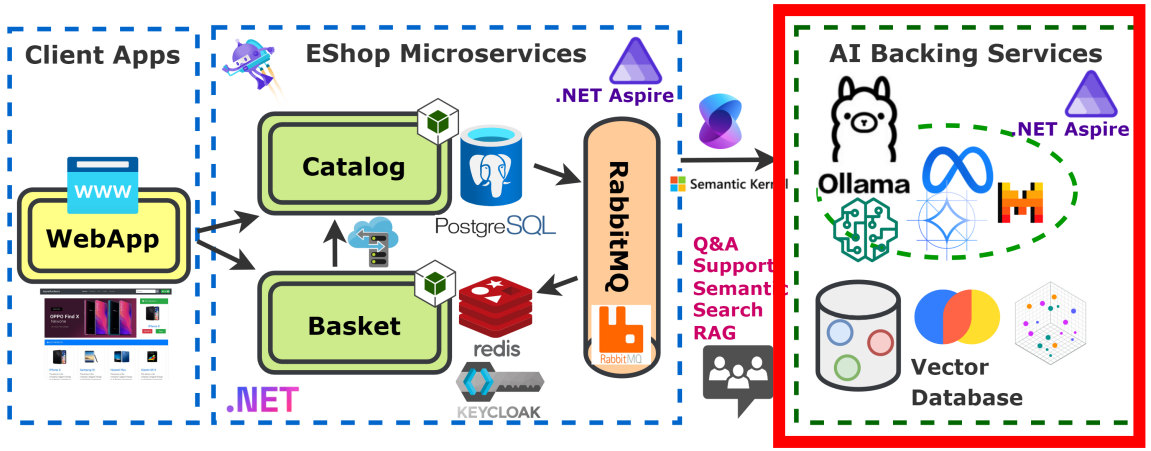
Mehmet Ozkaya

.NET GenAI with Microsoft.Extensions.AI for Chat AI and Semantic Search



.NET GenAI with Microsoft.Extensions.AI for Chat AI

- Ollama for Local LLM Inference, Integrate models like Llama3 or Phi3
- Develop Customer Support Q&A chatbot
- Vector Database (in-memory), Store embeddings for semantic search
- Blazor Frontend for AI: Build pages that let users interact with the Q&A or advanced semantic product searches



What are Large Language Models (LLMs) ?

LLMs are **advanced AI models** trained on **large datasets -** massive amounts of **text data from books, websites, research papers**

They can **understand**, **generate**, and **process natural language** that mimics natural human communication

Heart of **AI chatbots**, content **creation tools**, **translation services**

Large Language Models (LLMs)

LLMs are **machine learning models** specialized in **natural language**

They operate on a **vast number of parameters—billions** or **even trillions**

Trained to perform **tasks** such as **text generation**, **summarization**, **Q&A**, **translation**, and more

Building **intelligent language applications**

Why Are LLMs Considered Large ?

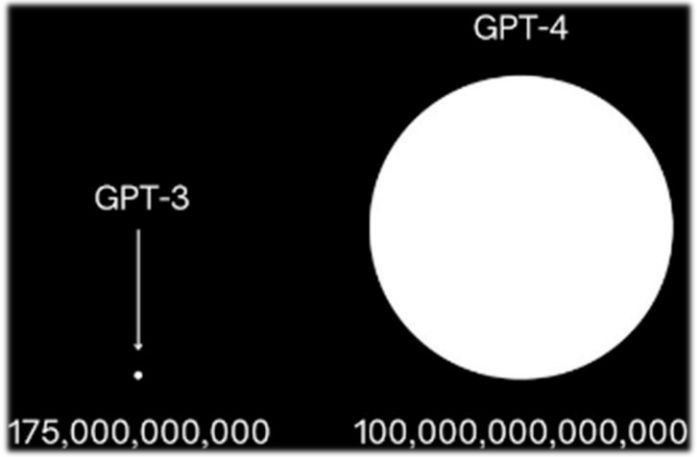
'Large' in LLM refers to both the size of the model: **the number of parameters** and **the training data**

LLMs contain **billions of parameters** (GPT-3 has 175 billion)

Training data: Trained on diverse text datasets (books, articles, web pages)

Larger models provide **better accuracy** and **context understanding**

More parameters = better performance but **higher computational costs**



Ollama – Run LLMs Locally

Ollama is a platform to **run large language models** (LLMs) or **small language models** (SLMs) **locally**

Offers **private, secure** AI solutions **without requiring the cloud**

Ideal for **developers** and **businesses** seeking **offline AI** capabilities w/ **privacy**, **low latency**, and **control** over AI models

Excellent choice for those looking to **integrate AI without depending** on constant **internet access**

Key Features of Ollama

Local Execution
Run LLMs and Small Language Models (SLMs) directly on your device

Pre-built Models
Includes optimized models for coding, chat, creative tasks, and more

Privacy-First
All data remains on your machine, protecting sensitive information

Customization
Allows model fine-tuning and adaptation for specific needs

Low Latency
Responses are quick without network dependency

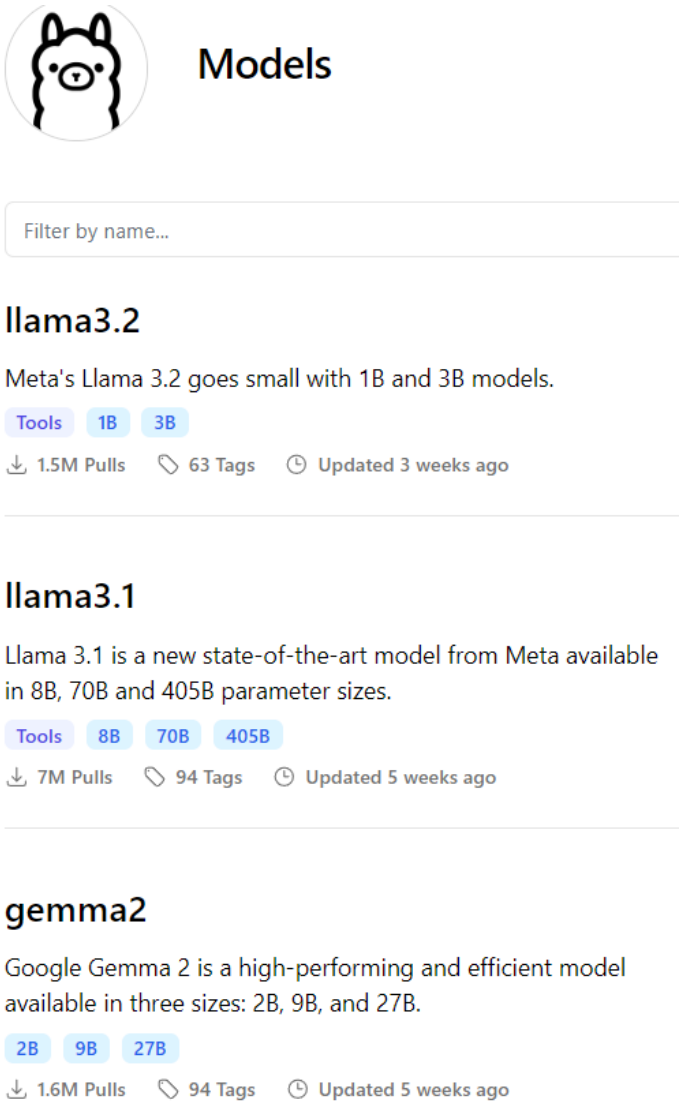
Available Models on Ollama

Variety of Models: **llama, gemma, qwen, phi, mistral**

Code Generation Models-codegemma,codellama
AI assistants specialized in code generation

Creative Models-llava
Text-to-image, story generation, and poetry models

Domain-Specific Models-medllama
Finance, healthcare, and other industry-specific LLMs



Benefits and Use Cases for Ollama

Benefits

Offline Capability, Data Privacy, Cost Savings, Low Latency

Use Cases

Software Development, Customer Support, Education and Creative work

Models

Filter by name...

llama3.2

Meta's Llama 3.2 goes small with 1B and 3B models.

Tools 1B 3B

1.5M Pulls 63 Tags Updated 3 weeks ago

llama3.1

Llama 3.1 is a new state-of-the-art model from Meta available in 8B, 70B and 405B parameter sizes.

Tools 8B 70B 405B

7M Pulls 94 Tags Updated 5 weeks ago

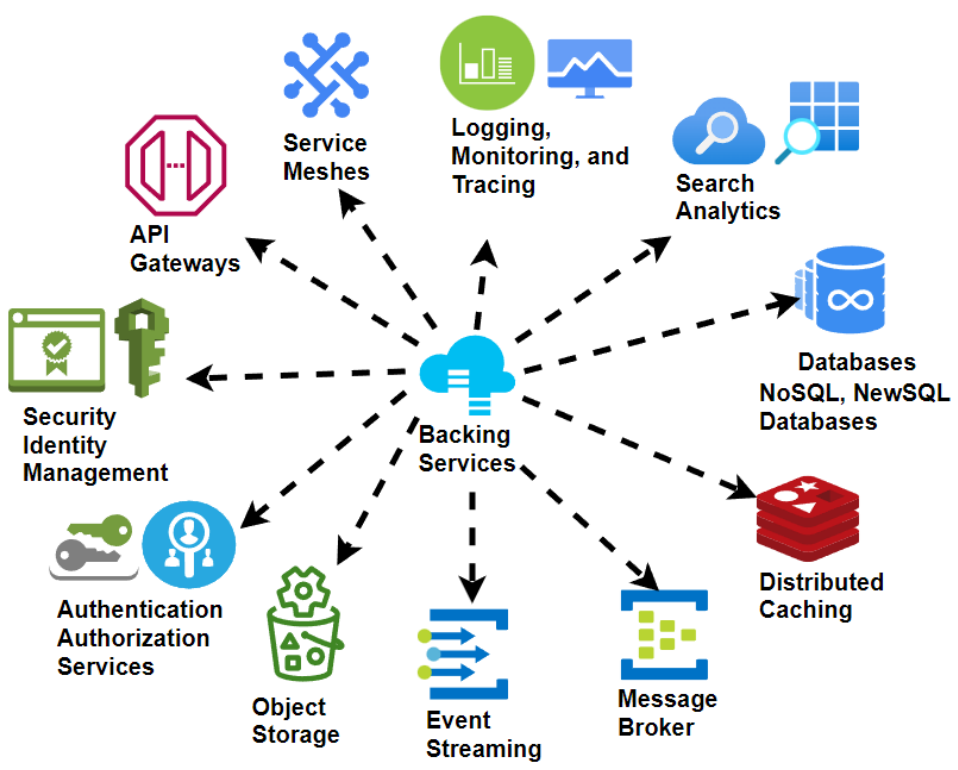
gemma2

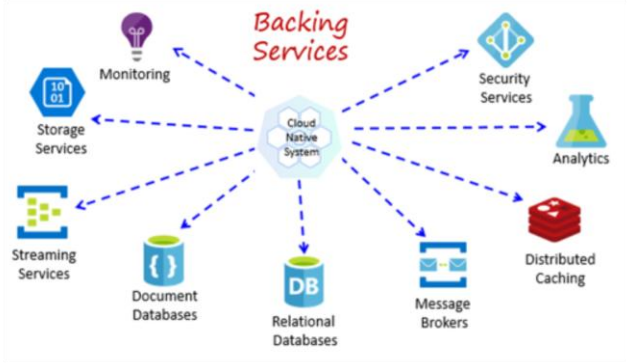
Google Gemma 2 is a high-performing and efficient model available in three sizes: 2B, 9B, and 27B.

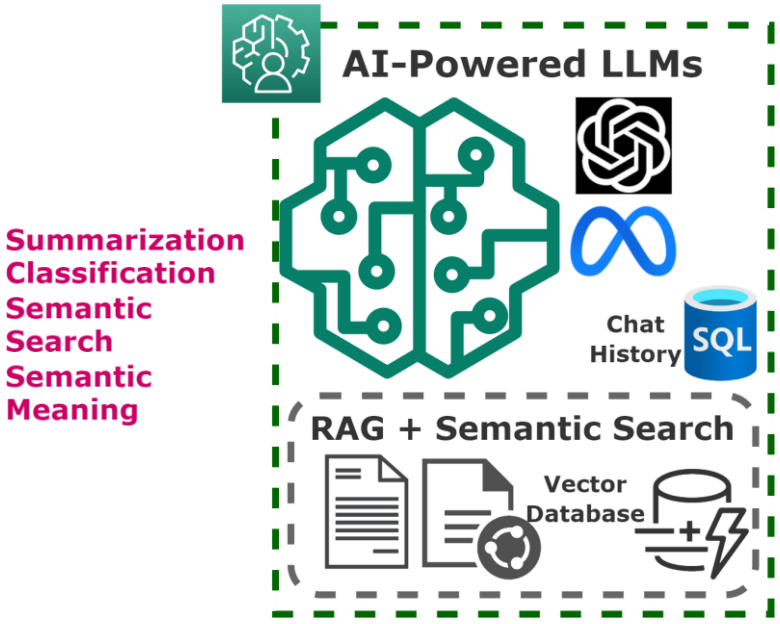
2B 9B 27B

1.6M Pulls 94 Tags Updated 5 weeks ago

The New Era of Cloud-Native Backing Services







Why Use LLMs and VectorDBs as Backing Services ?

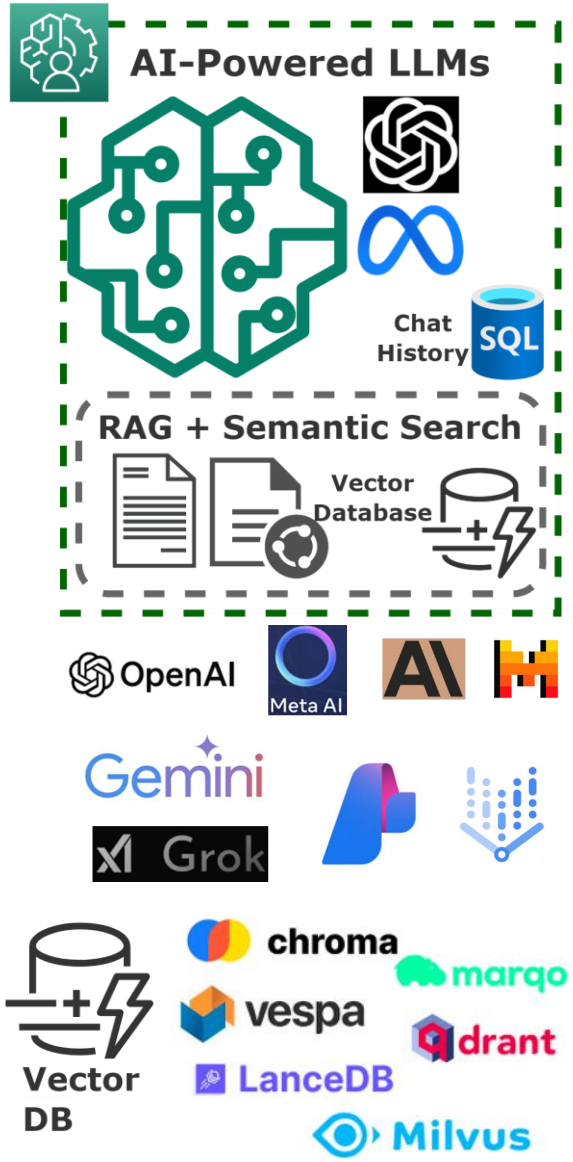
LLMs provide features like **summarization**, **classification**, **sentiment analysis** and chat-based **Q&A**

VectorDBs enable **similarity-based search** and **knowledge retrieval** to support **complex queries**

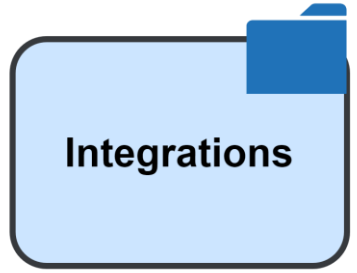
Leverage **pre-trained models** and **semantic search** engines to **integrate** powerful **AI capabilities** with minimal effort

Automate **customer support**, real-time **recommendations**, **semantic search** for enterprise knowledge management

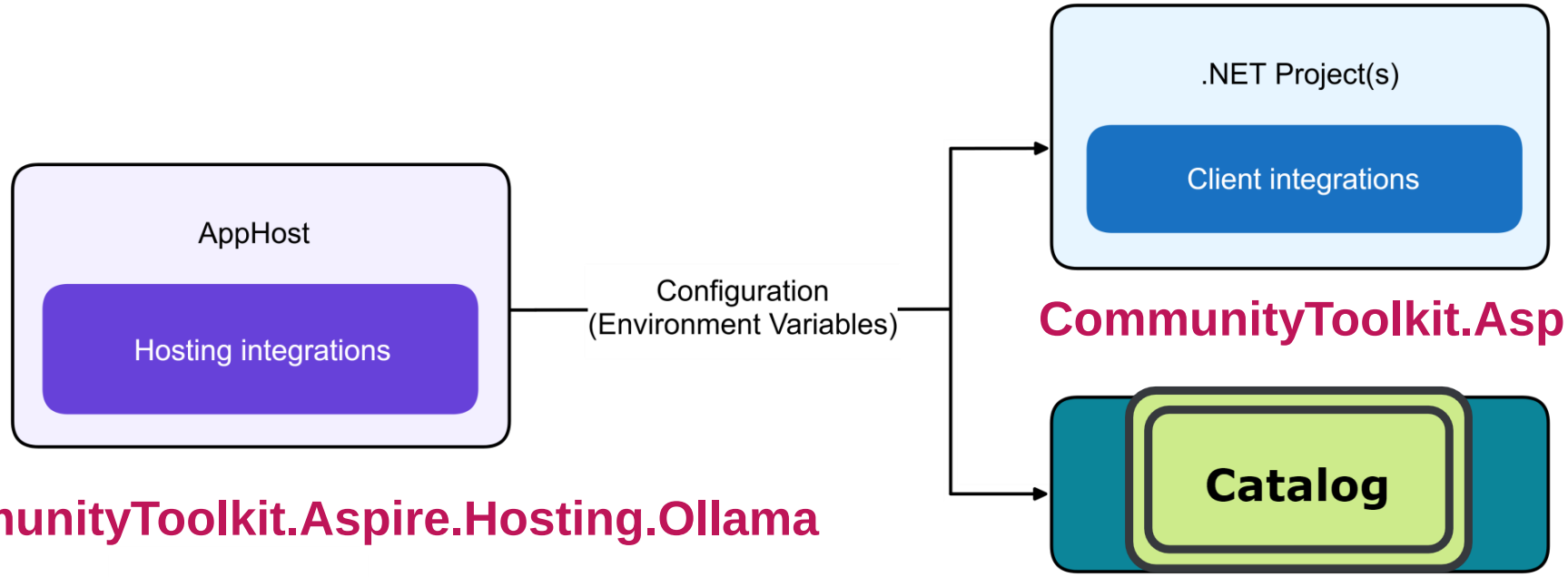
LLMs and **VectorDBs** as the **brain** and **memory** of your architecture



Ollama and Llama Model Integrations



.NET Aspire

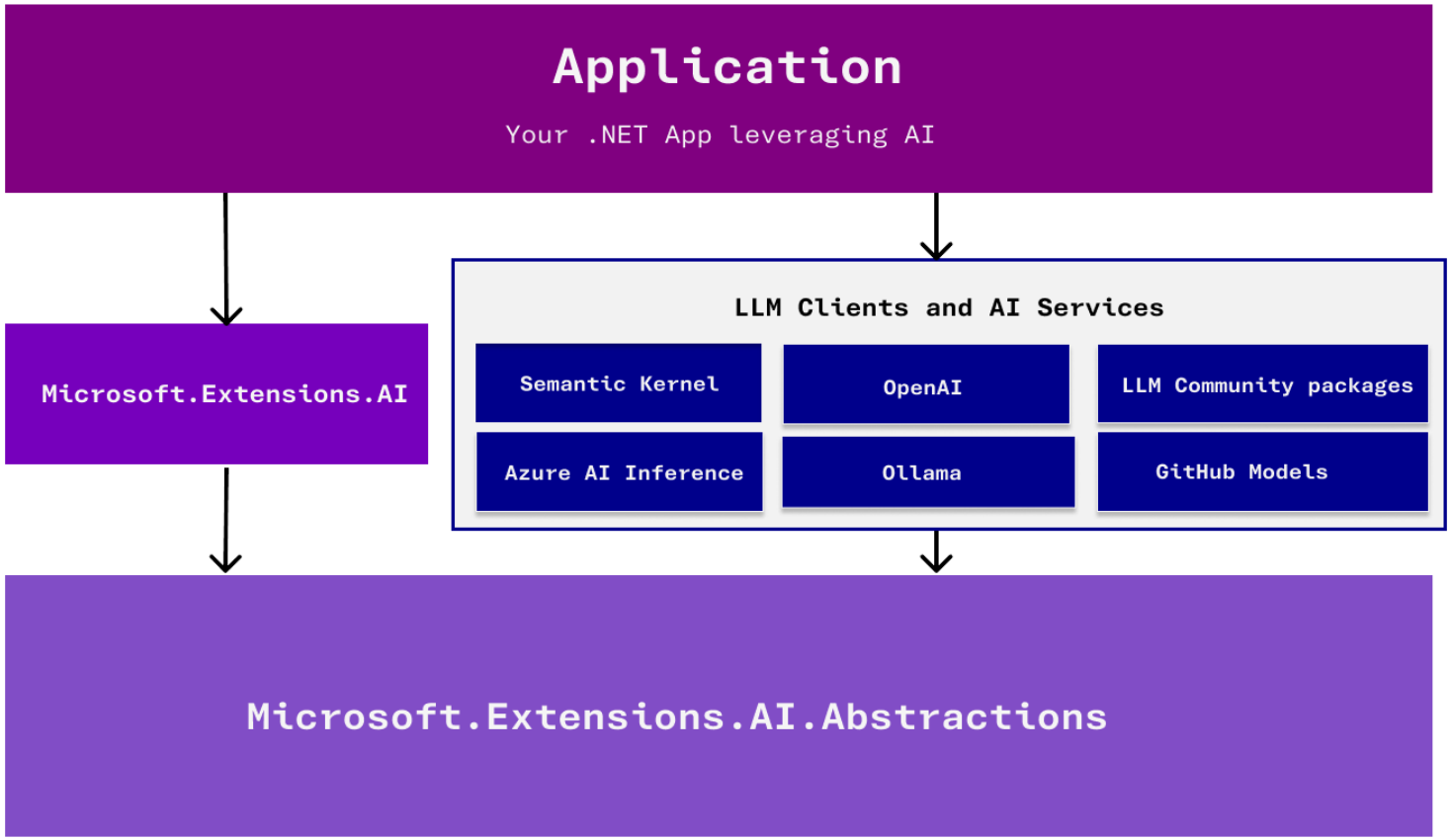


CommunityToolkit.Aspire.OllamaSharp

CommunityToolkit.Aspire.Hosting.Ollama

builder.AddOllamaSharpChatClient("llama");

Microsoft.Extensions.AI Structure



Support for DI, Pipelines and Middleware (Caching, Tool Calling, Telemetry)

Provides an IChatClient that connects to LLM providers

Core Types
IChatClient
ChatMessage
Embeddings
Content Types: TextContent, ImageContent, AudioContent etc.

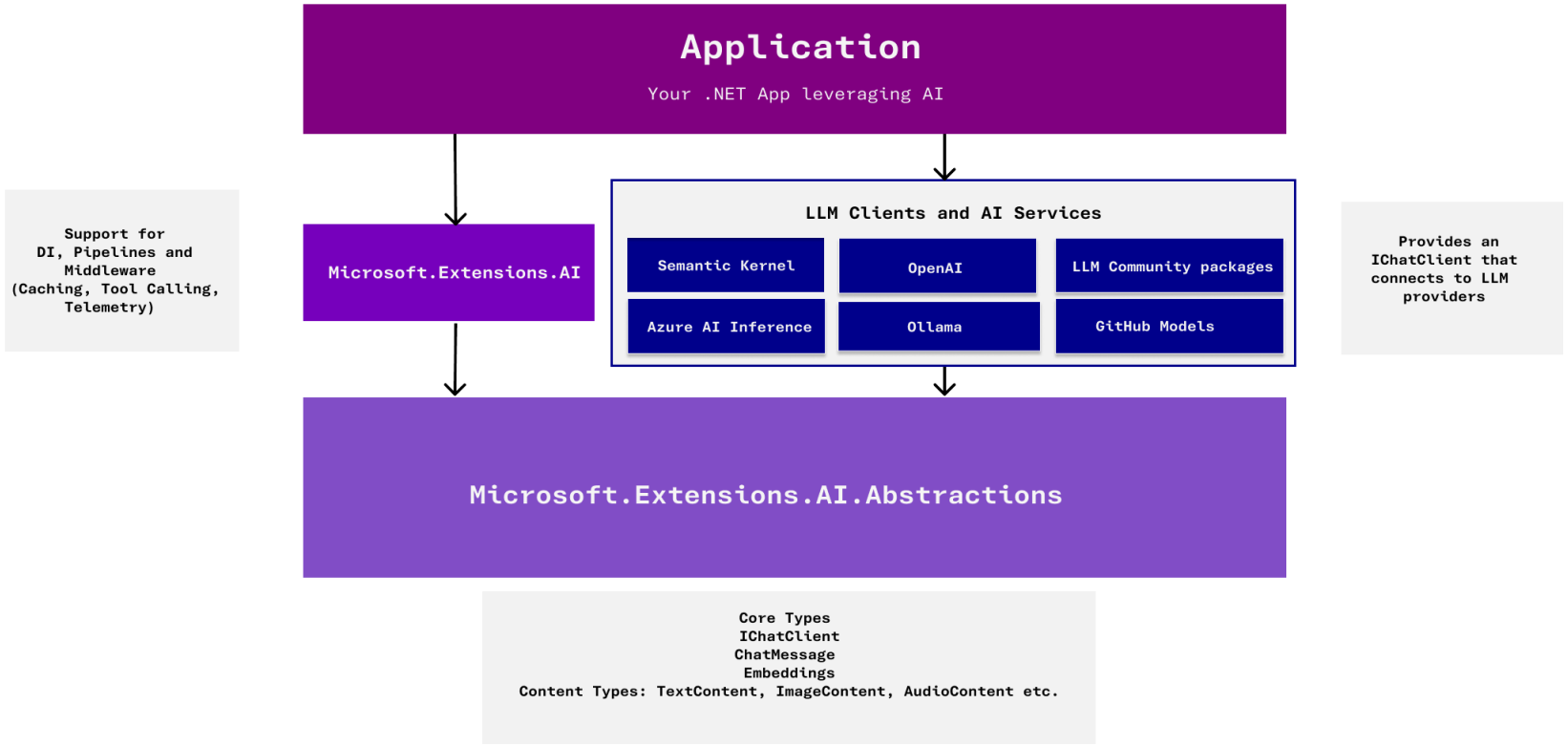
https://learn.microsoft.com/en-us/dotnet/ai/ai-extensions

Work with Abstractions for Common AI Services

Chat: Conversational AI or ChatGPT-like interactions

Embeddings: Vector search & semantic understanding

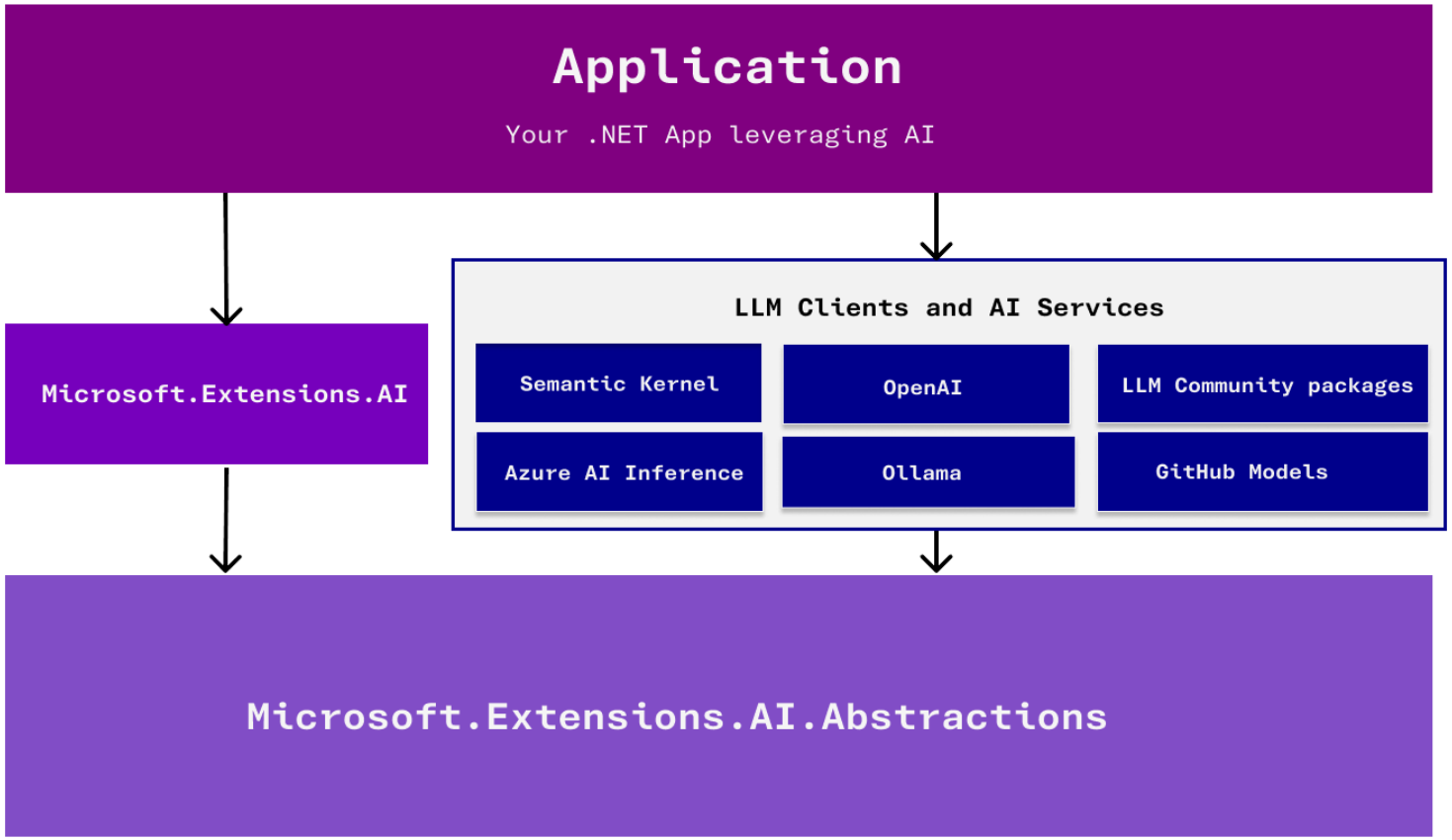
Tool calling: Orchestrate AI requests with external services



Example: IChatClient

```
IChatClient client =
    environment.IsDevelopment
        ? new OllamaChatClient(...)
        : new AzureAIInferenceChatClient(...);

var response = await client.GetResponseAsync(messages, options, cancellationToken);
```

Microsoft.Extensions.AI Structure



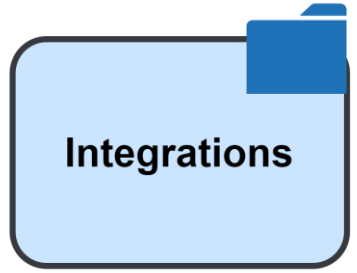
Support for DI, Pipelines and Middleware (Caching, Tool Calling, Telemetry)

Provides an IChatClient that connects to LLM providers

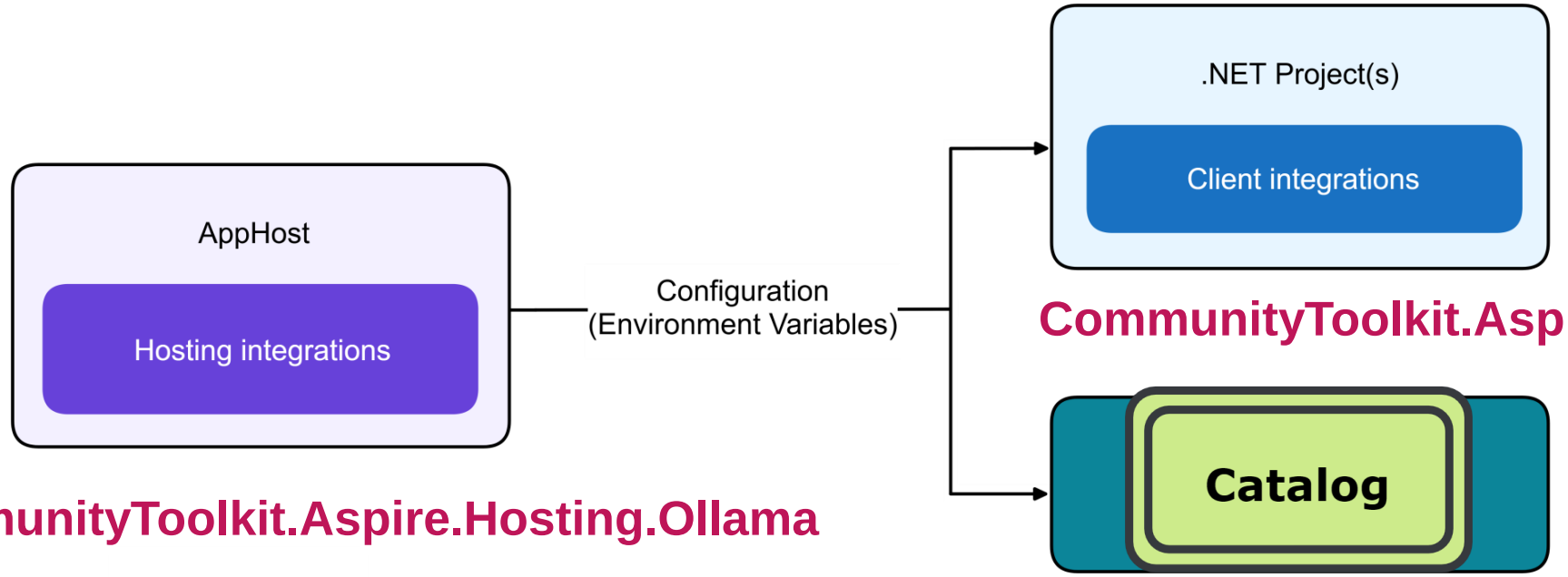
Core Types
IChatClient
ChatMessage
Embeddings
Content Types: TextContent, ImageContent, AudioContent etc.

https://learn.microsoft.com/en-us/dotnet/ai/ai-extensions

Ollama and Llama Model Integrations



.NET Aspire

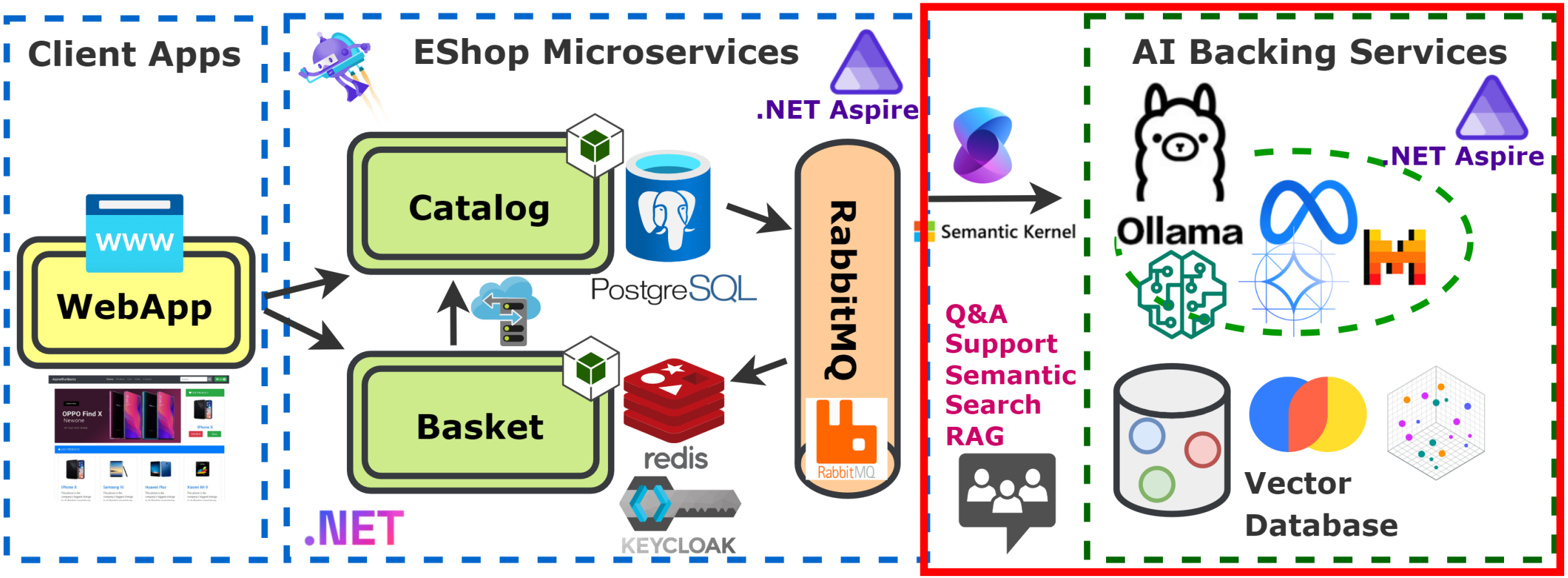


CommunityToolkit.Aspire.OllamaSharp

CommunityToolkit.Aspire.Hosting.Ollama

builder.AddOllamaSharpChatClient("llama");

.NET GenAI with Microsoft.Extensions.AI for Chat AI and Semantic Search



Develop Blazor Web Application

Products

Here are some of our amazing outdoor products that you can purchase.

Image	Name	Description
	Hiking Poles	Ideal for camping and hiking trips
	Outdoor Rain Jacket	This product will keep you warm and dry in all weathers
	Survival Kit	A must-have for any outdoor adventurer
	Outdoor Backpack	This backpack is perfect for carrying all your outdoor essentials

Semantic Product Search with Vector Embeddings and Vector DB

What is a Vector Database?

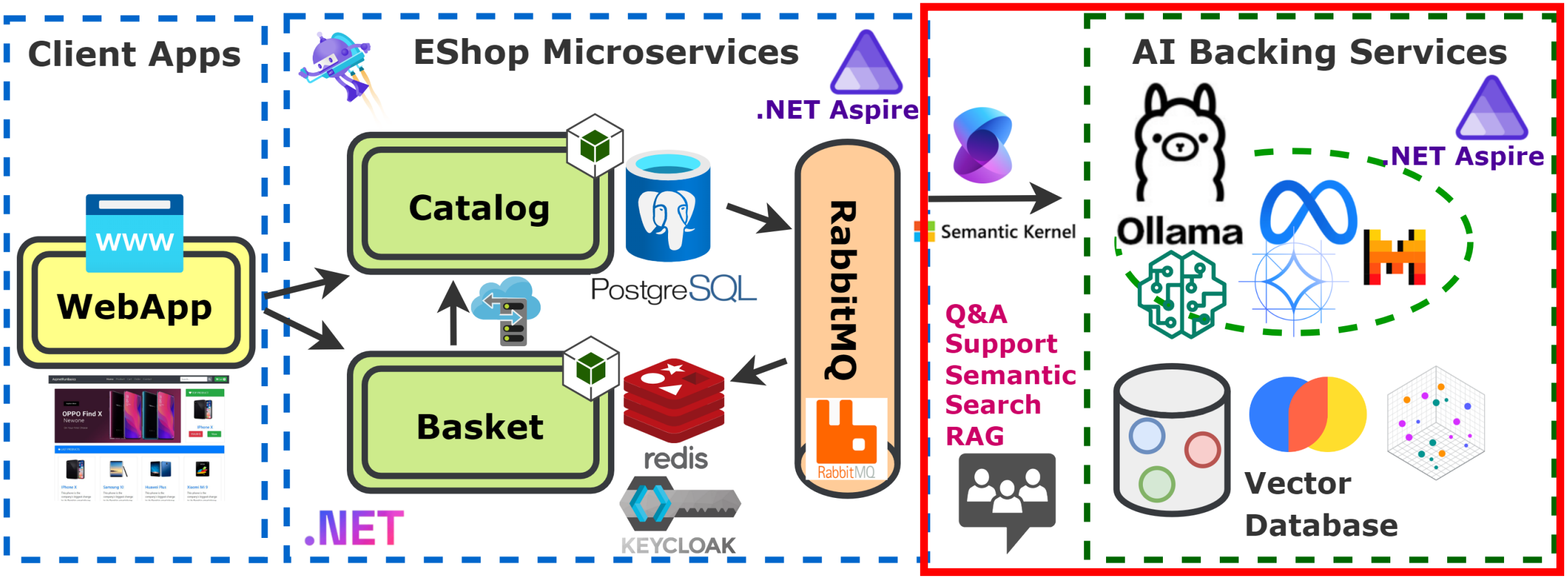
What are Vectors and Vector Embeddings ?

Hosting Integration for Ollama all-minilm Embeddings Model

Client Integration Packages with SemanticKernel and Extensions.VectorData

Mehmet Ozkaya

Semantic Product Search with Vector Embeddings and Vector DB



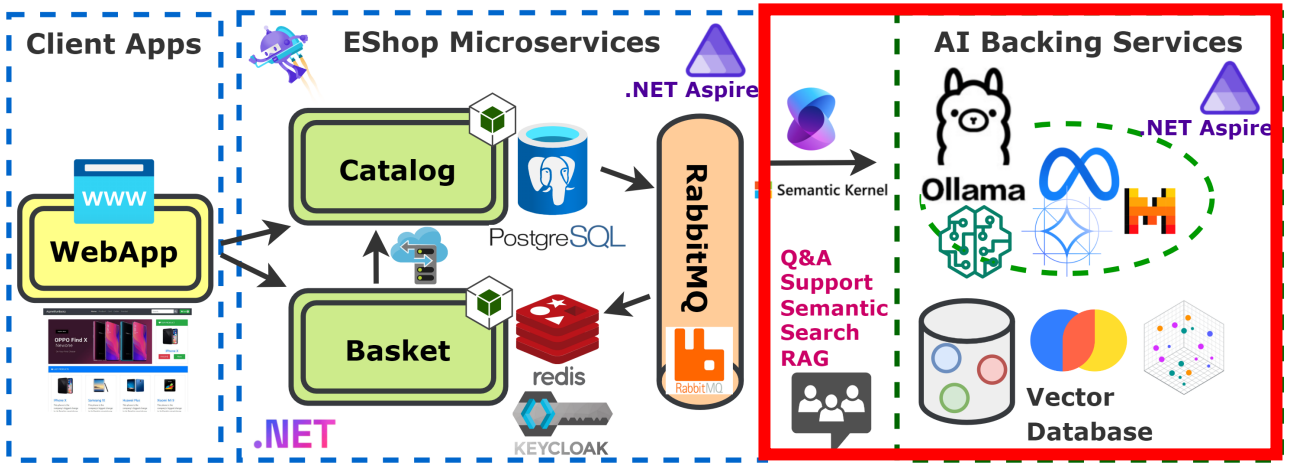
Semantic Product Search with Vector Embeddings and Vector DB

Hosting: Add a model (all-minilm) in Ollama for generating embeddings

Registration: Configure IEmbeddingGenerator to produce product embeddings

Vector Database: Store & Retrieve embeddings using a collection from Microsoft.Extensions.VectorData.Abstractions

Endpoints: We'll provide an API or method to transform a user query into an embedding



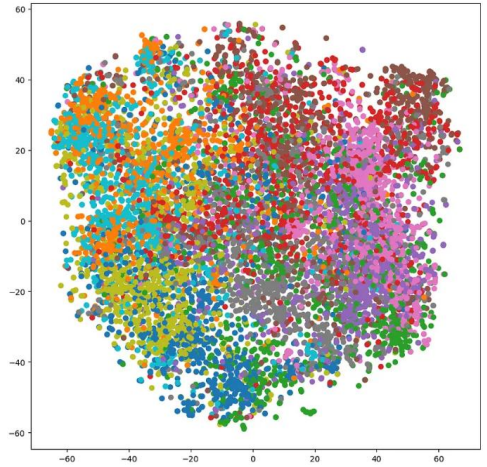
What is a Vector Database?

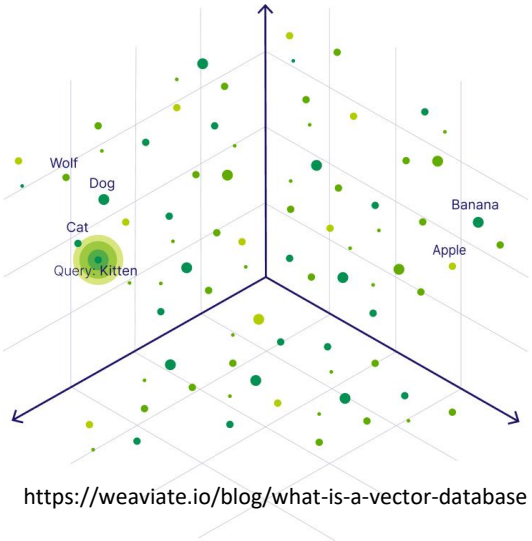
Vector Database is a **specialized database** designed to store, manage, and query **high-dimensional vectors**

Vectors are **numerical representations** that capture the **semantic meaning** of data

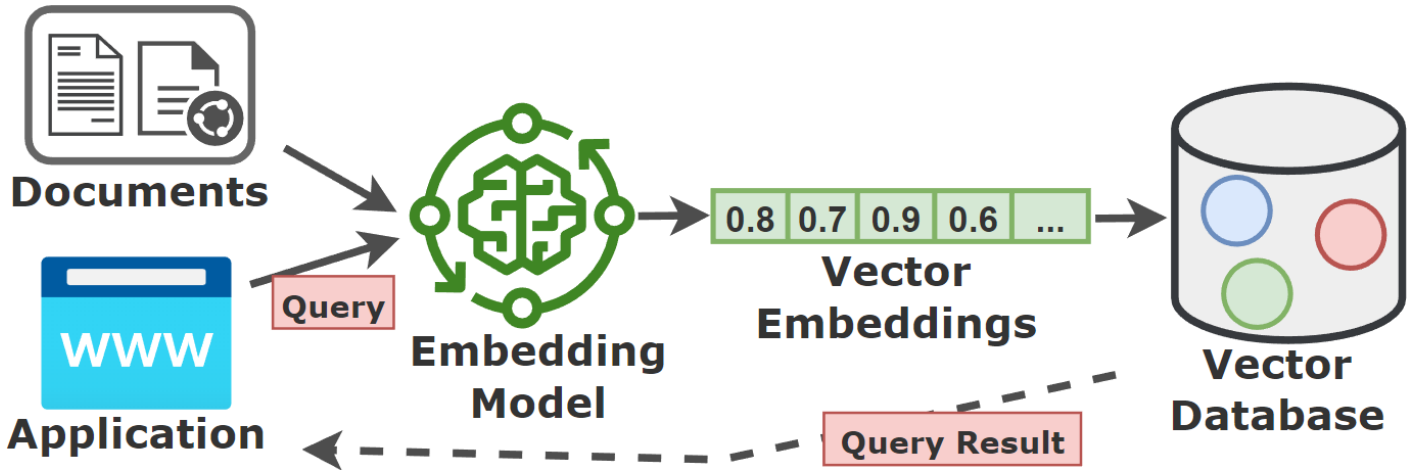
Indexes and **stores vector embeddings** for **fast retrieval** and **similarity search**

Enable **contextual searches based on** the **meaning** behind the data





https://weaviate.io/blog/what-is-a-vector-database



The Evolution from Traditional to Vector Databases

Traditional databases store data in **structured formats** like **rows** and **columns**

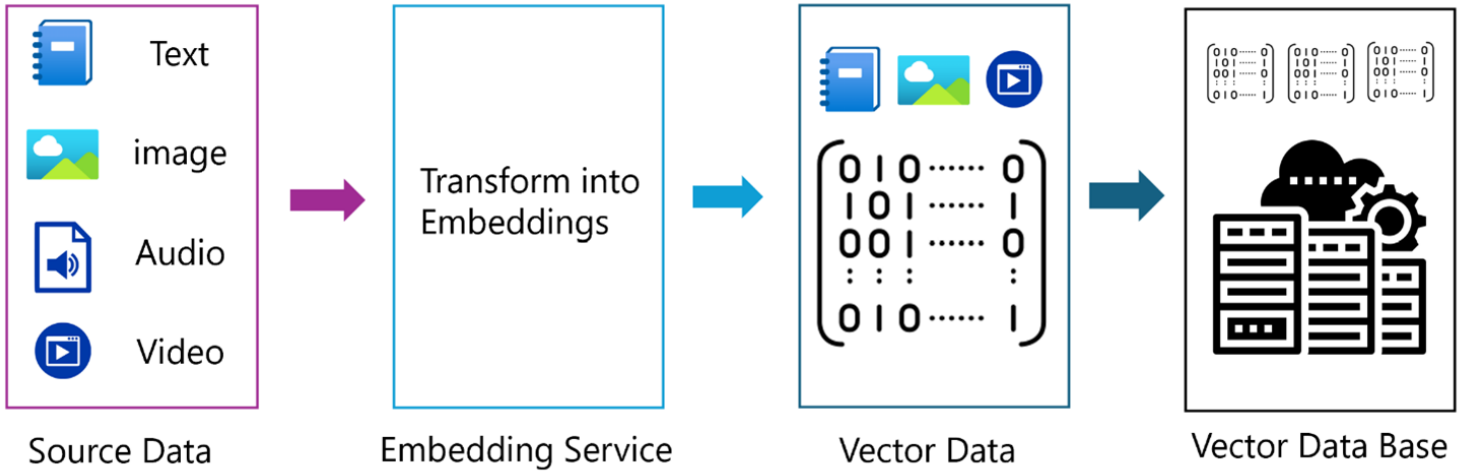
Fall short when we **need** to **perform semantic searches**—searches **based on** the **meaning** of **data** rather exact words

Search for "**laptop**": **Traditional Db** retrieve **exact word**, **Vector Db** understand that "**notebook computer**"

Vector Db store **unstructured data** like documents, images, audio, video, social media posts

Customers

CustomerId	FirstName	LastName	DateCreated
1	Homer	Simpson	13/06/2014 3:33:37 PM
2	Peter	Griffin	13/06/2014 9:09:56 PM
3	Stewie	Griffin	13/06/2014 9:16:07 PM
4	Brian	Griffin	13/06/2014 9:16:36 PM
5	Cosmo	Kramer	13/06/2014 9:16:41 PM
6	Philip	Fry	13/06/2014 9:17:02 PM
7	Amy	Wong	13/06/2014 9:22:05 PM
8	Hubert J.	Farnsworth	13/06/2014 9:22:19 PM
9	Marge	Simpson	13/06/2014 9:22:37 PM
10	Bender	Rodríguez	13/06/2014 9:22:52 PM
11	Turanga	Leela	13/06/2014 9:23:37 PM
(New)			15/06/2014 9:00:01 PM



https://learn.microsoft.com/en-us/data-engineering/playbook/solutions/vector-database/

Why Are Vectors Important?

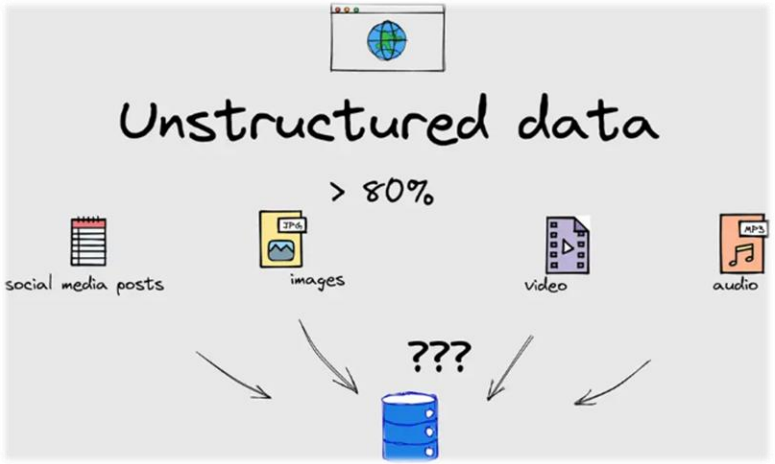
%80 of data on the internet is **Unstructured Data** (like documents, images, audio, video, social media posts)

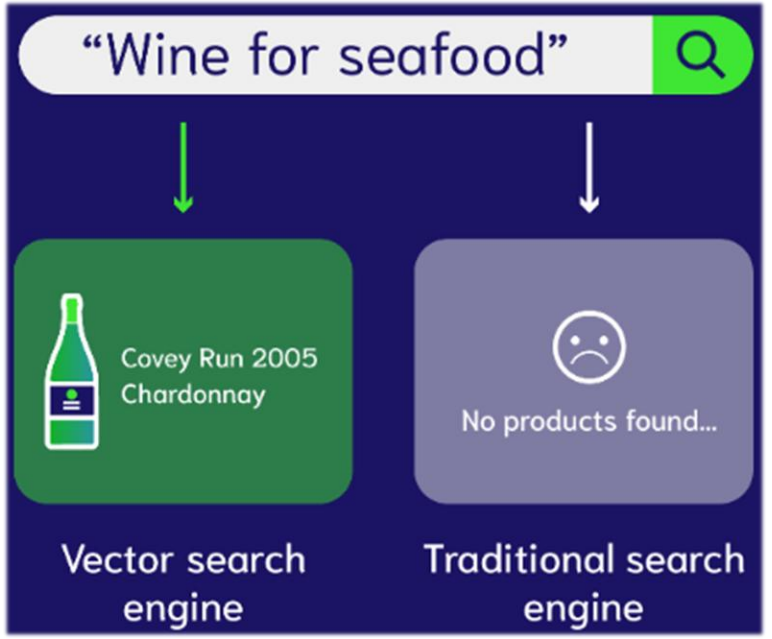
Vectors **capture** the **meaning behind data**, not just its surface

Understand synonyms, **paraphrases**, and even **nuanced relationships** between **data points**

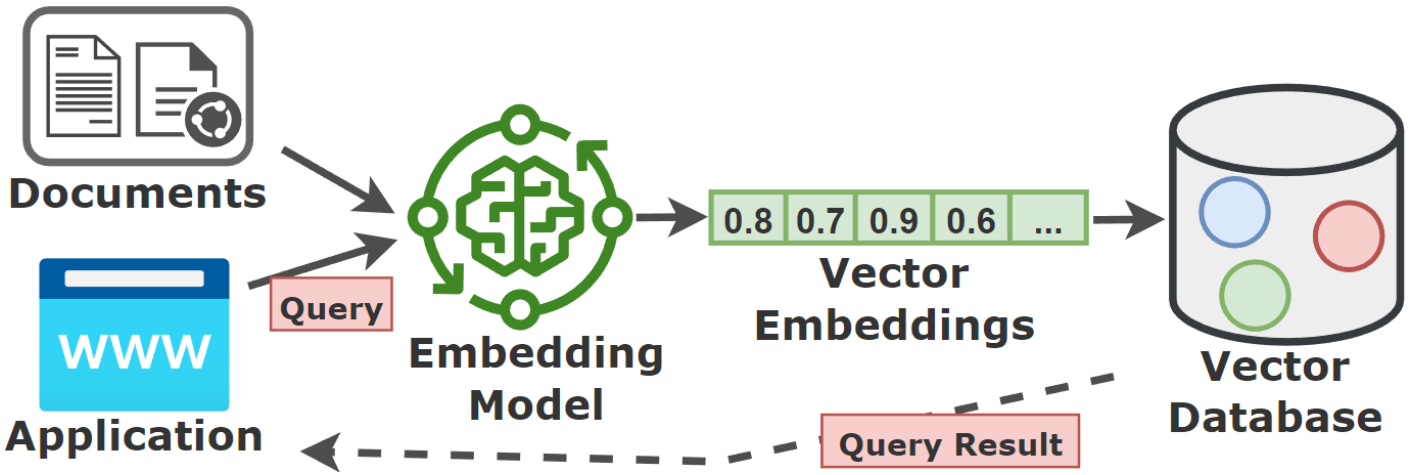
Search for “**mobile phone**” retrieve results for “**smartphone**”

Semantic search, personalized **recommendations**





https://weaviate.io/blog/vector-embeddings-explained



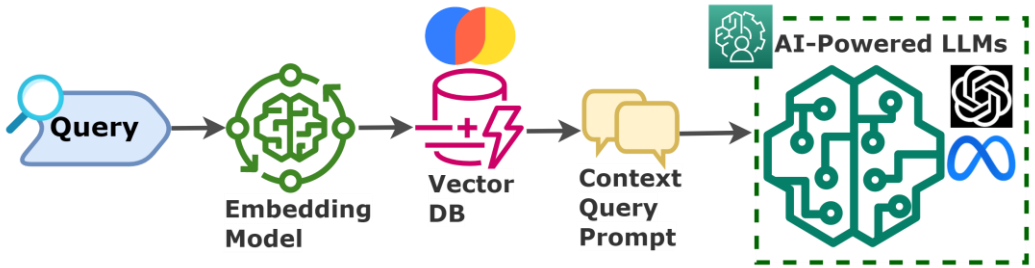
Core Features of Vector Databases

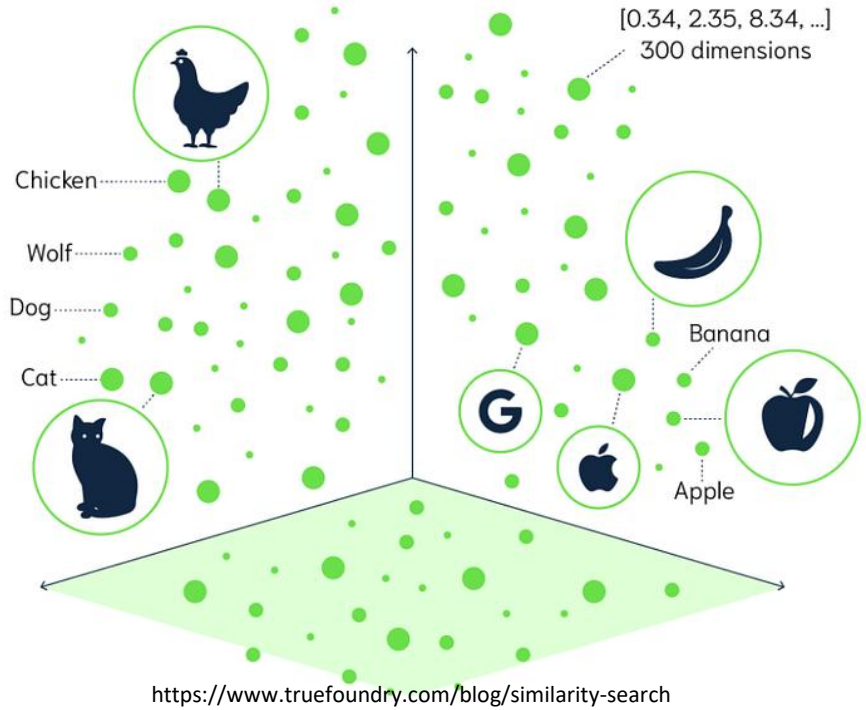
High-Dimensional Data Handling
Efficiently stores and queries data represented as vectors with hundreds or thousands of dimensions

Fast Similarity Search
Approximate Nearest Neighbor (ANN) to quickly retrieve the most relevant vectors, even from large datasets

Scalability
Supports millions or even billions of vectors, suitable for enterprise-scale applications

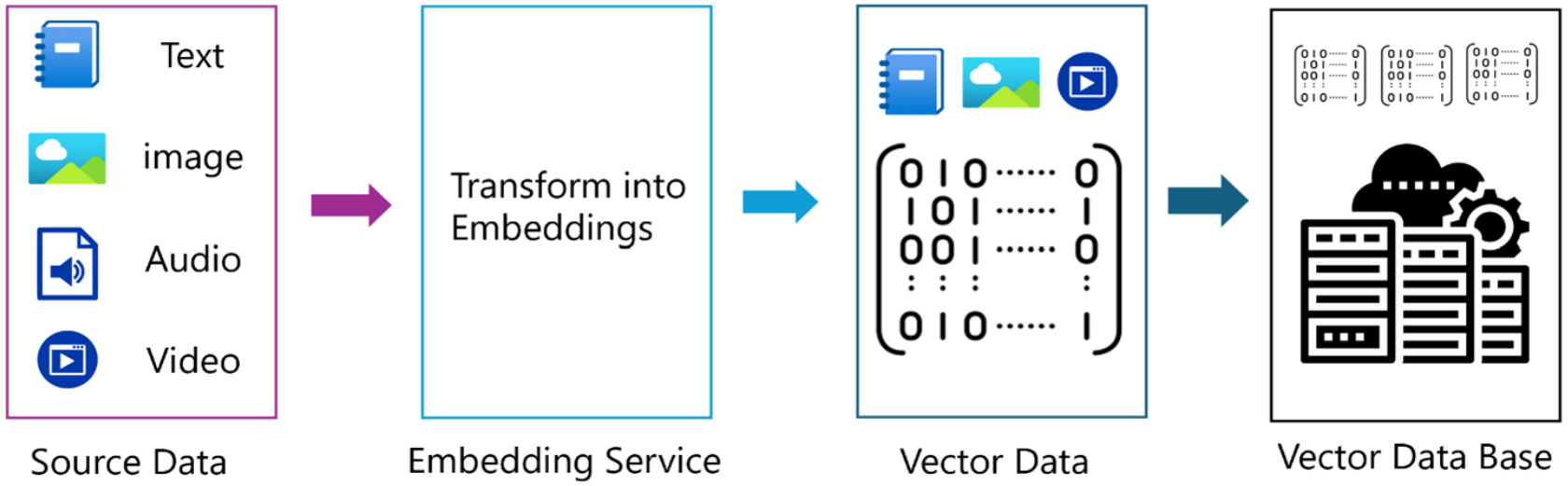
Integration with AI Models
Seamlessly works with embedding models from platforms like OpenAI, Hugging Face, or custom-trained models



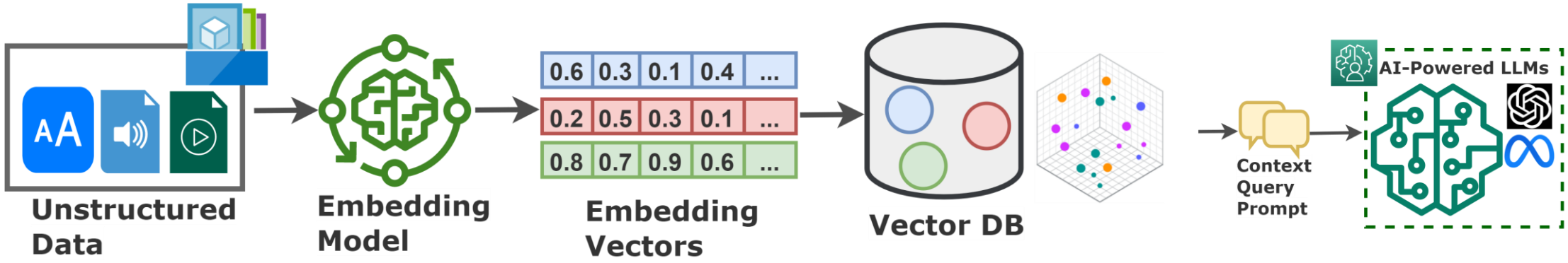


https://www.truefoundry.com/blog/similarity-search

What is a Vector Database? - Summary



https://learn.microsoft.com/en-us/data-engineering/playbook/solutions/vector-database/



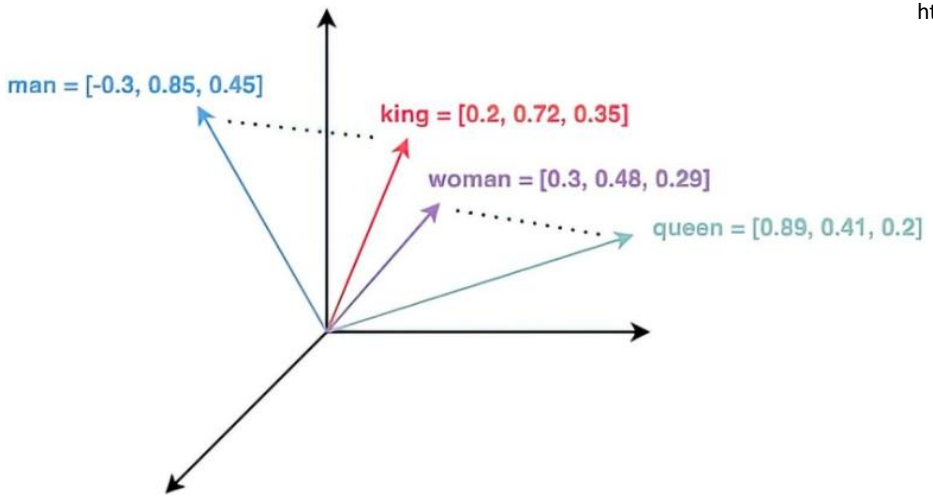
What is a Vector?

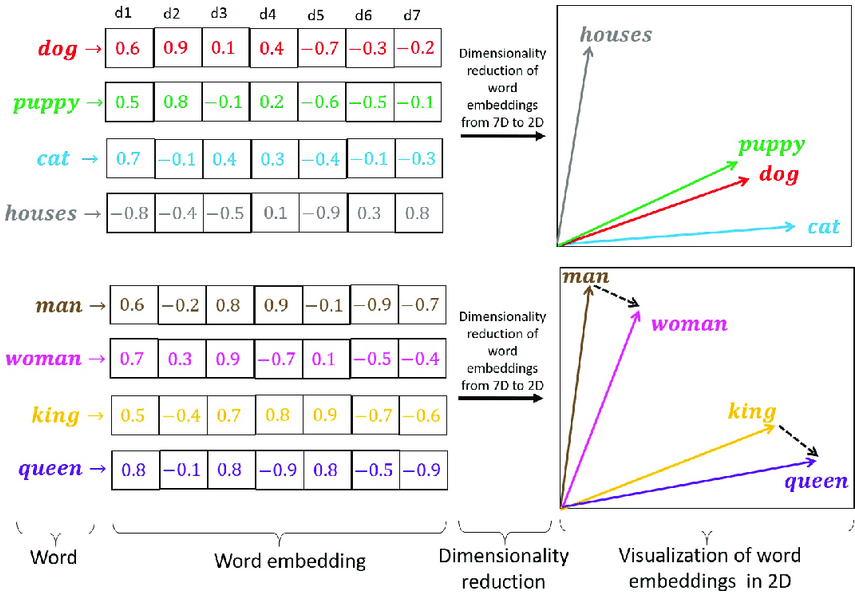
A **vector** is a **mathematical object** that has **both magnitude** and **direction**, represented as a list of numbers: **[1.2, 3.4, -0.8]**

Vectors are used to **represent complex data** in a way that **AI models** can **process**, **numerical summaries** of information

Vectors exist in **high-dimensional space**, which is **harder** to **visualize** but incredibly **powerful** for **computations**

Instead of saying "**it's round, orange, and sweet**", encode these features **numerically** in a **vector**: **[roundness: 0.9, color: 0.8, sweetness: 0.7]**





Rozado, David (2020). Word embeddings map words in a corpus of text to vector space.. PLOS ONE. Figure. https://doi.org/10.1371/journal.pone.0231189.g008

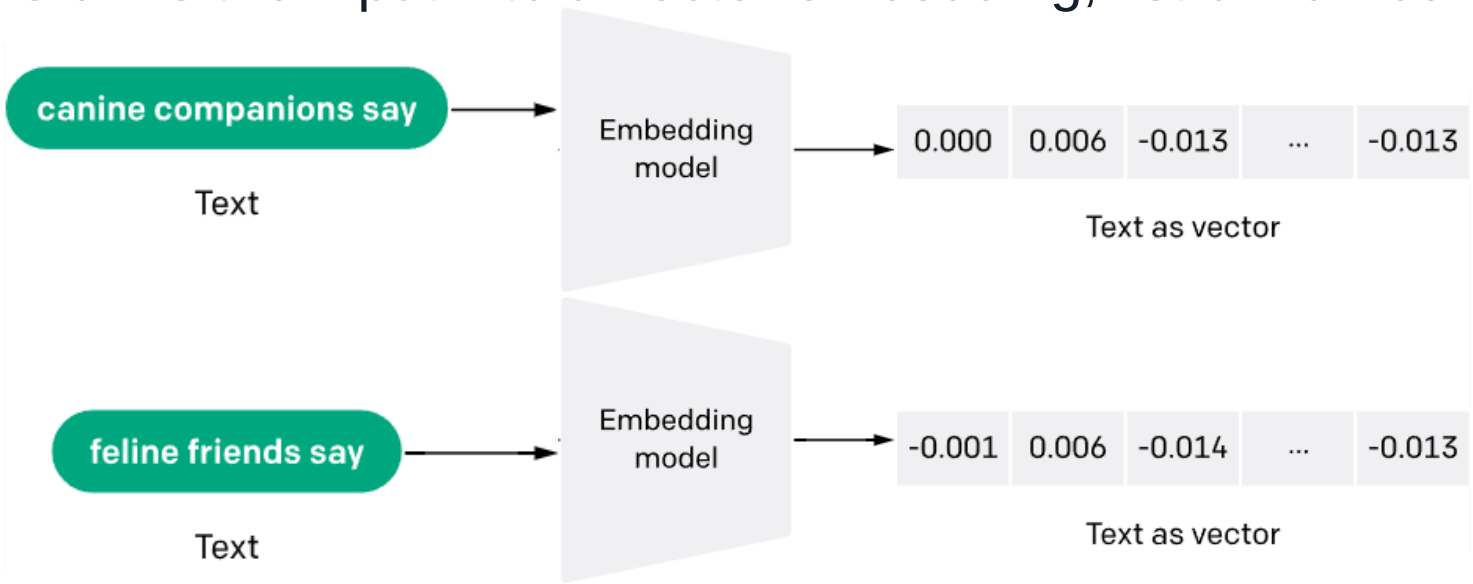
What are Vector Embeddings?

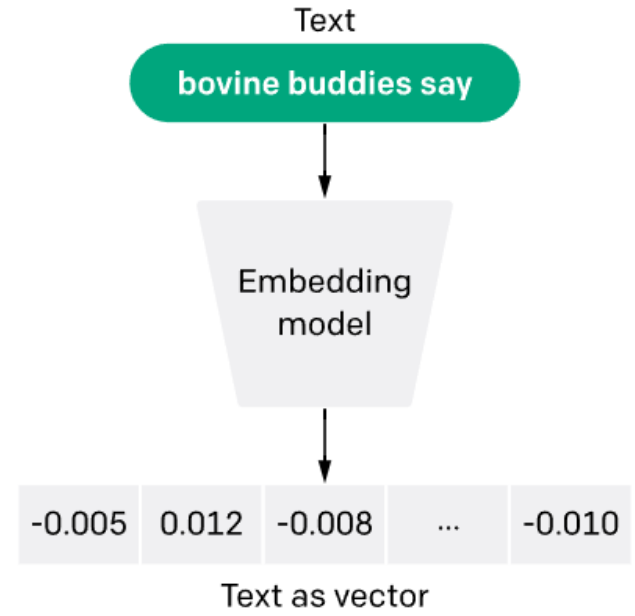
Dense numerical representations of **data**, **capture** the **semantic meaning** of text, images, audio or other data types

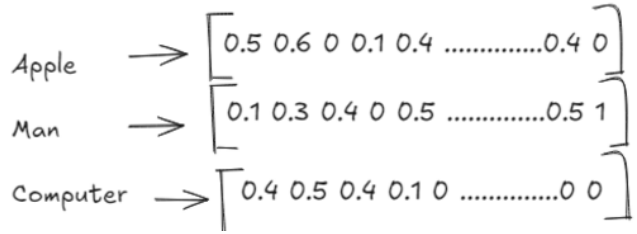
Step 1: Input Data
Start with raw data, like a sentence, an image, or a sound file

Step 2: Use an AI Embedding Model
Pass the data through an AI model, like a Transformer model

Step 3: Output Embedding
Transforms the input into a vector embedding, list of numbers







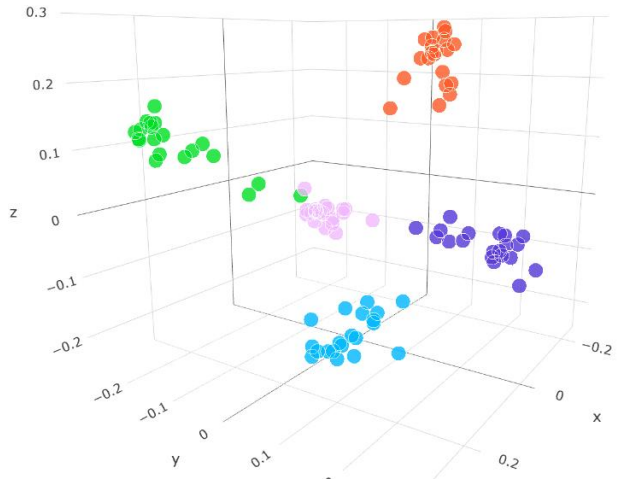
Why Are Embeddings Important?

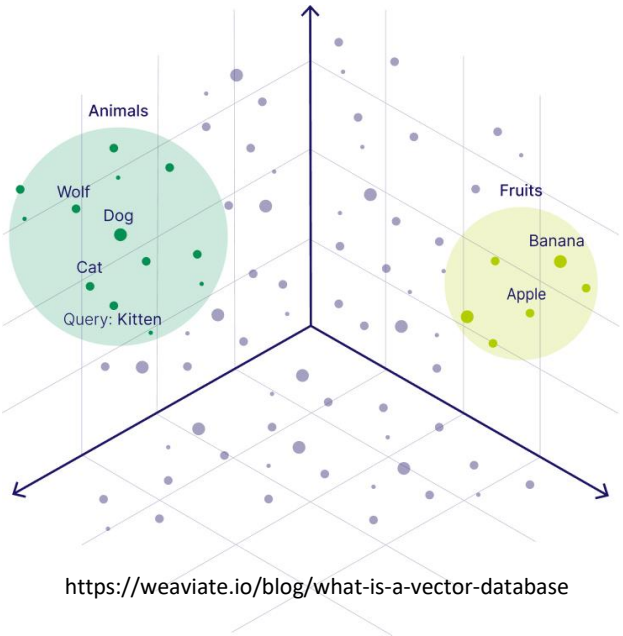
Compare and **analyze data** based on **semantic meaning** rather than surface-level features

If **two sentences** have **similar meanings**, their **embeddings** will **be close** in **high-dimensional space**

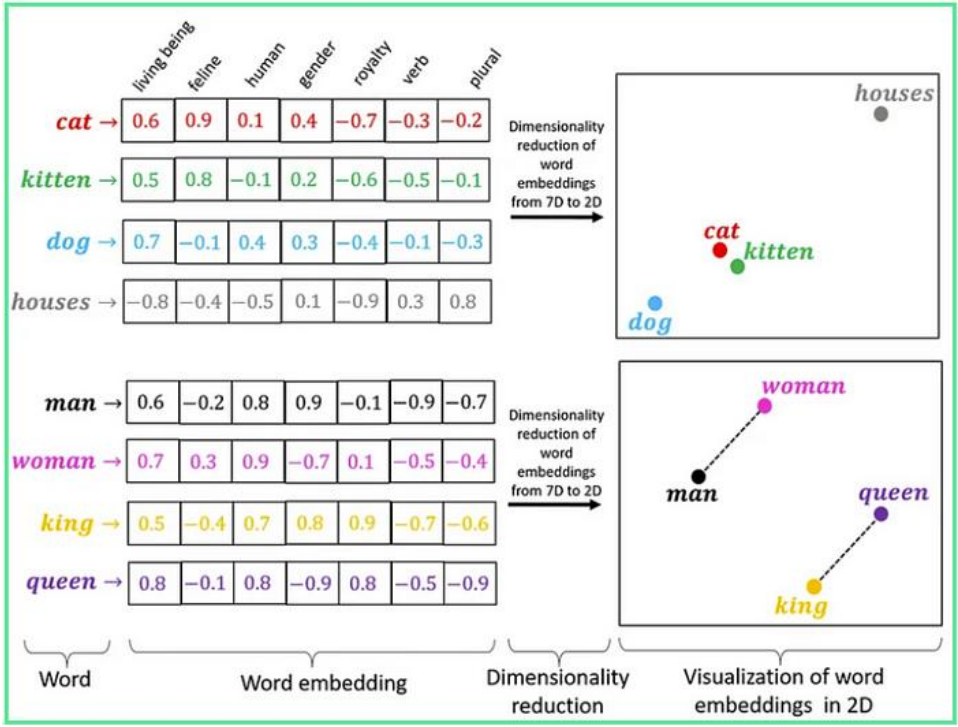
Enable systems to **retrieve relevant information quickly**

Same embedding techniques can be applied to text, images, and audio

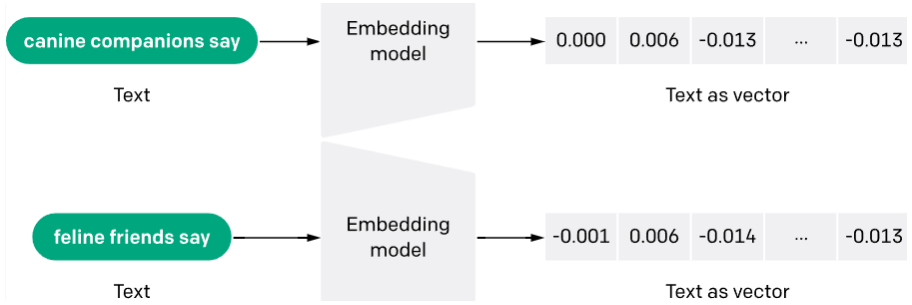




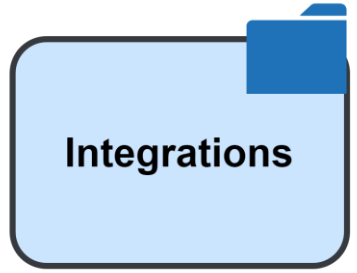
https://weaviate.io/blog/what-is-a-vector-database



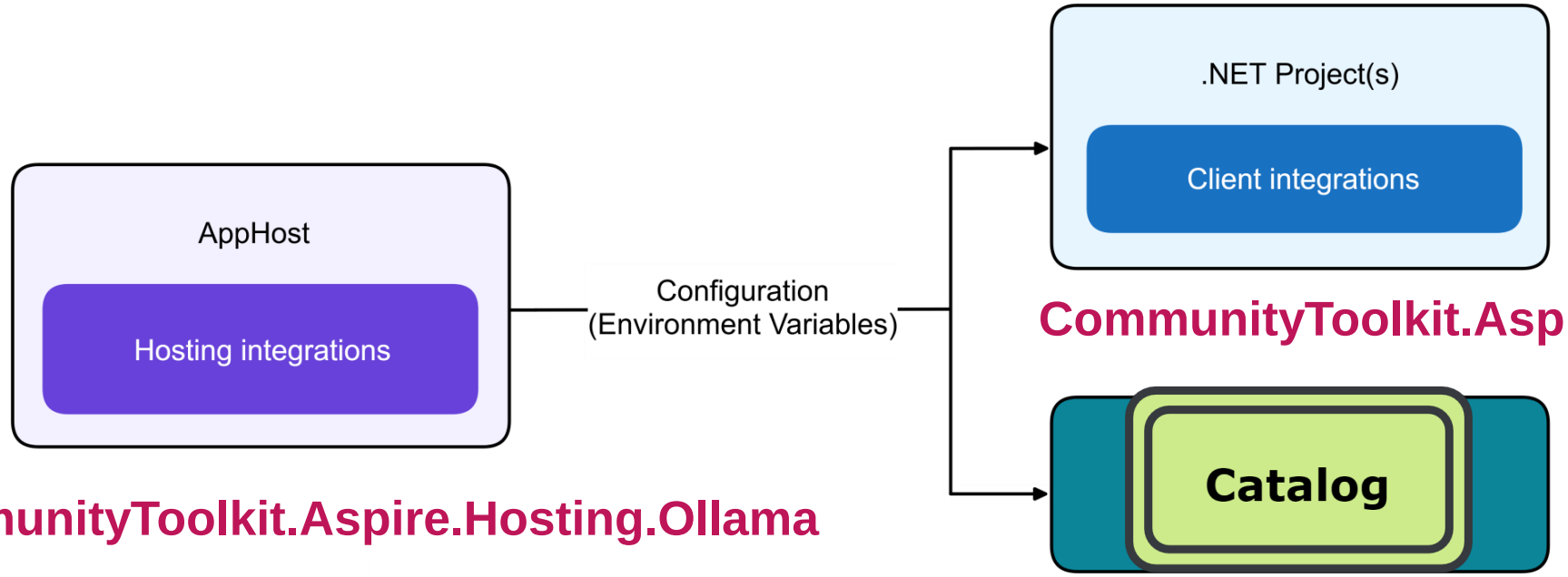
Rozado, David (2020). Word embeddings map words in a corpus of text to vector space.. PLOS ONE. Figure. https://doi.org/10.1371/journal.pone.0231189.g008



Ollama and all-minilm Model Integrations



.NET Aspire



CommunityToolkit.Aspire.OllamaSharp

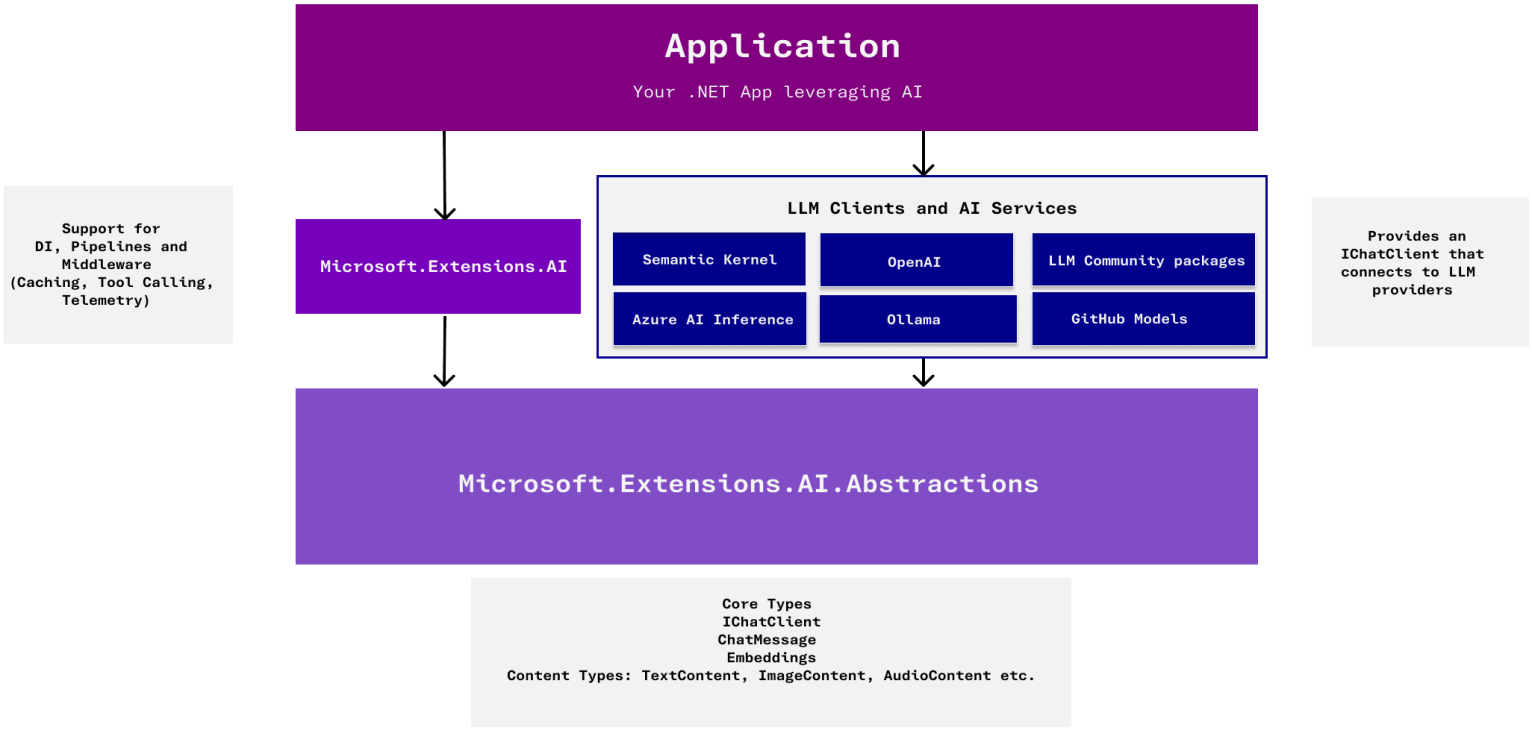
CommunityToolkit.Aspire.Hosting.Ollama

builder. AddOllamaSharpEmbeddingGenerator(..);
builder.Services.AddInMemoryVectorStoreRecord Collection(..)

Microsoft.Extensions.VectorData

Abstraction layer manipulate vector data without coding directly against a specific database

Like Entity Framework for embeddings—create, read, update, delete on vector-based records in a consistent manner



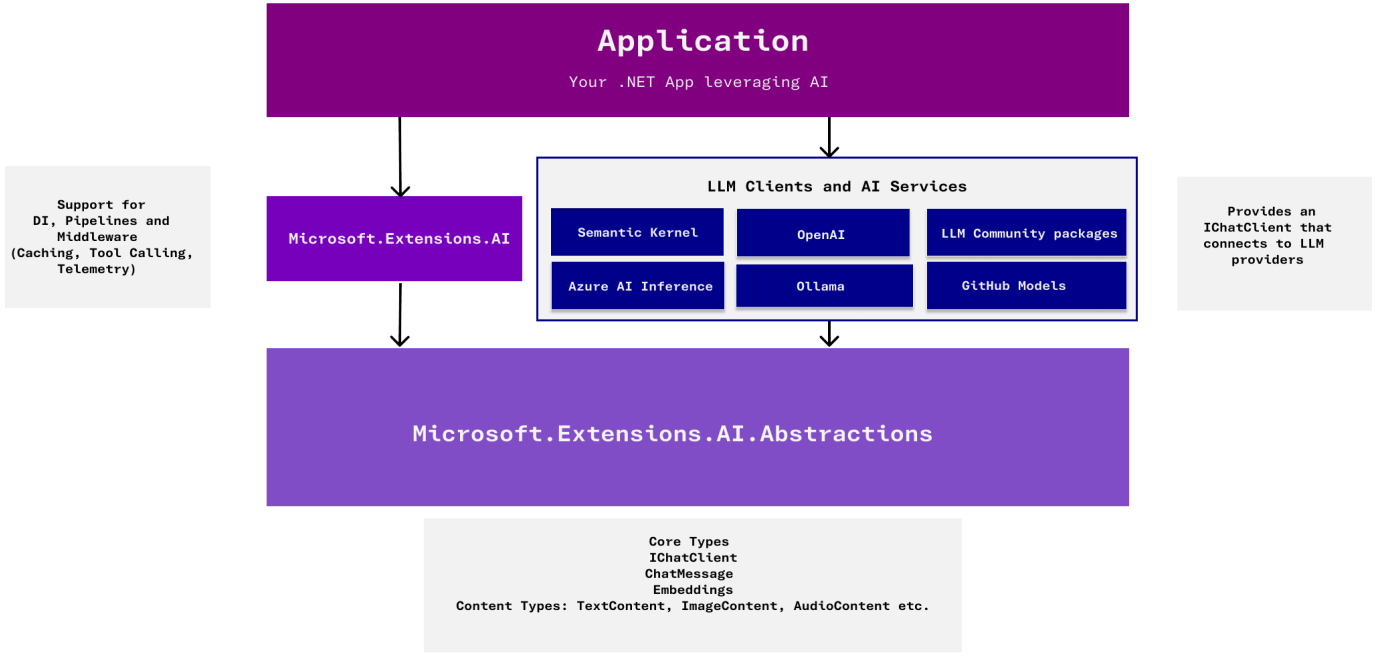
Getting Started with SemanticKernel Connectors

Use Semantic Kernel vector store connectors for Extensions.VectorData

Use the in-memory vector store implementation

Ollama reference implementation in Microsoft.Extensions.AI

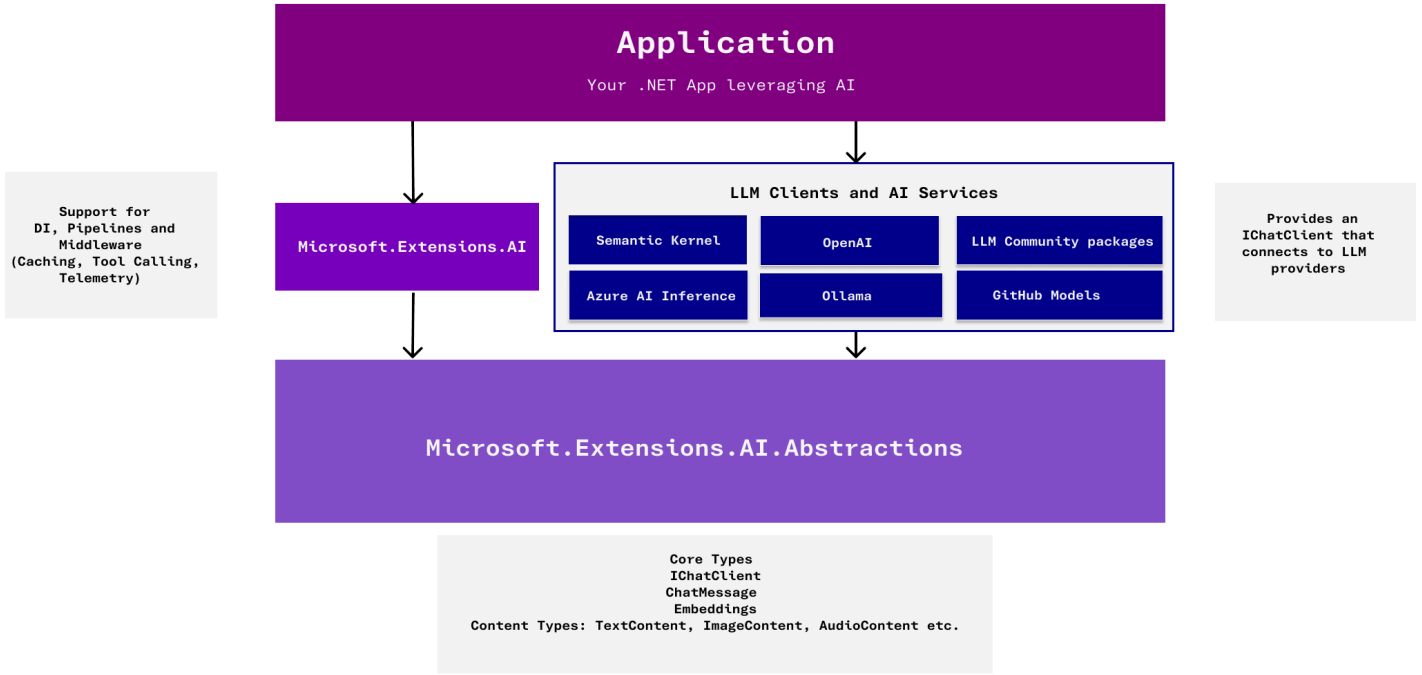
Reference IRecordCollection<TRecord> stays consistent across all connectors



Additional Packages & Setup

Microsoft.Extensions.VectorData.Abstractions

Microsoft.SemanticKernel.Connectors.InMemory



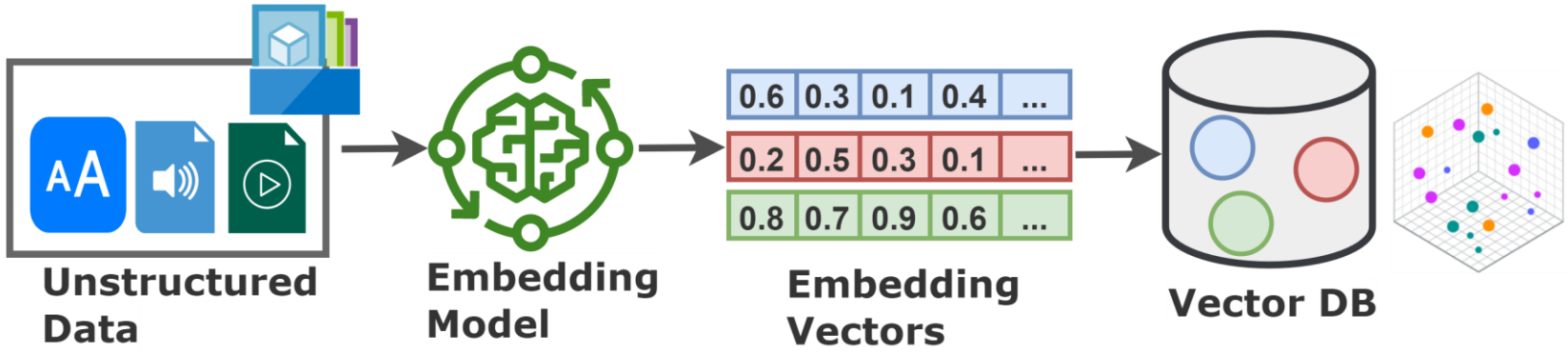
Product Vector Search

Models layer: The newly created ProductVector to store vector embeddings

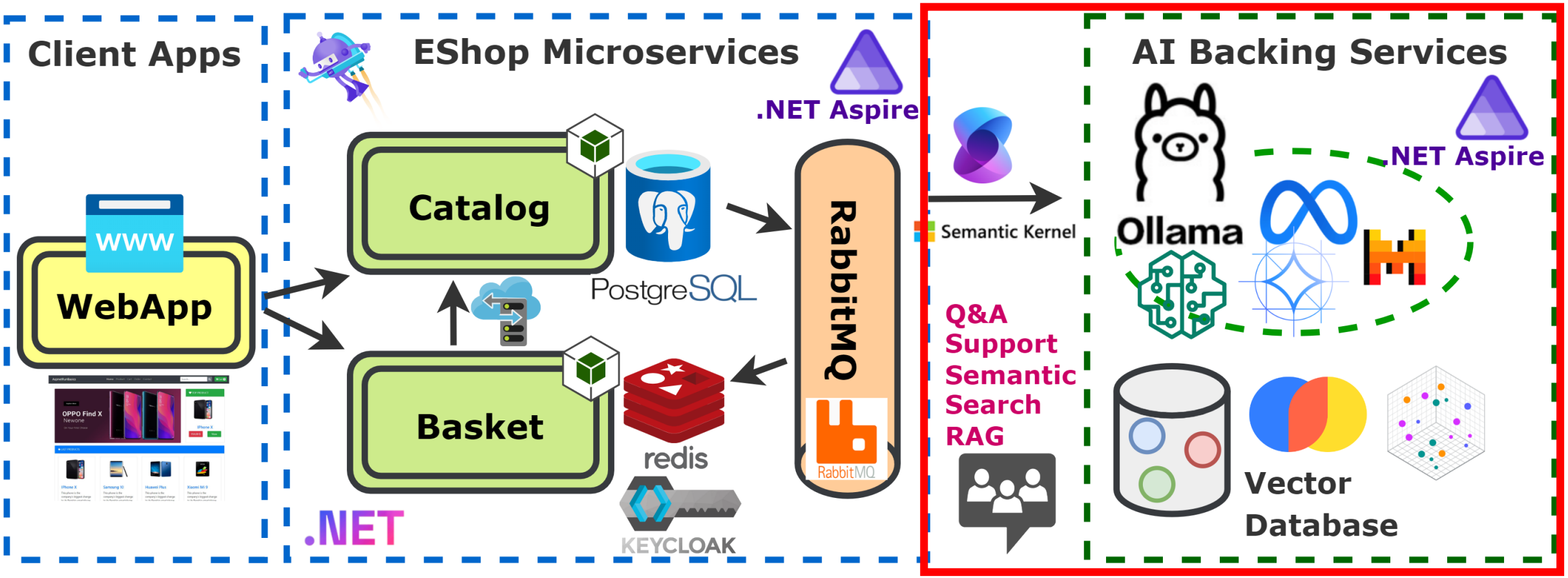
Services layer: Will handle the actual searching—keyword or semantic

Endpoints layer: Minimal API routes in the Catalog microservice

Frontend: A Blazor page with a checkbox toggling AI or normal search



.NET GenAI with Microsoft.Extensions.AI for Chat AI and Semantic Search



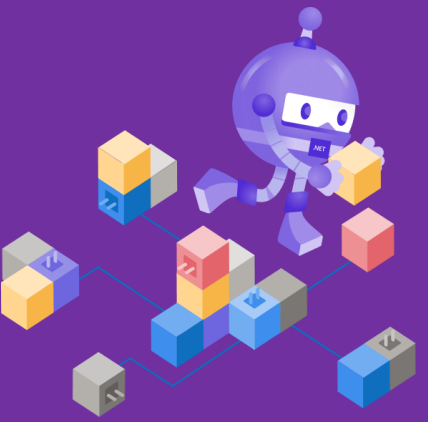
Develop Blazor Web Application

Products

Here are some of our amazing outdoor products that you can purchase.

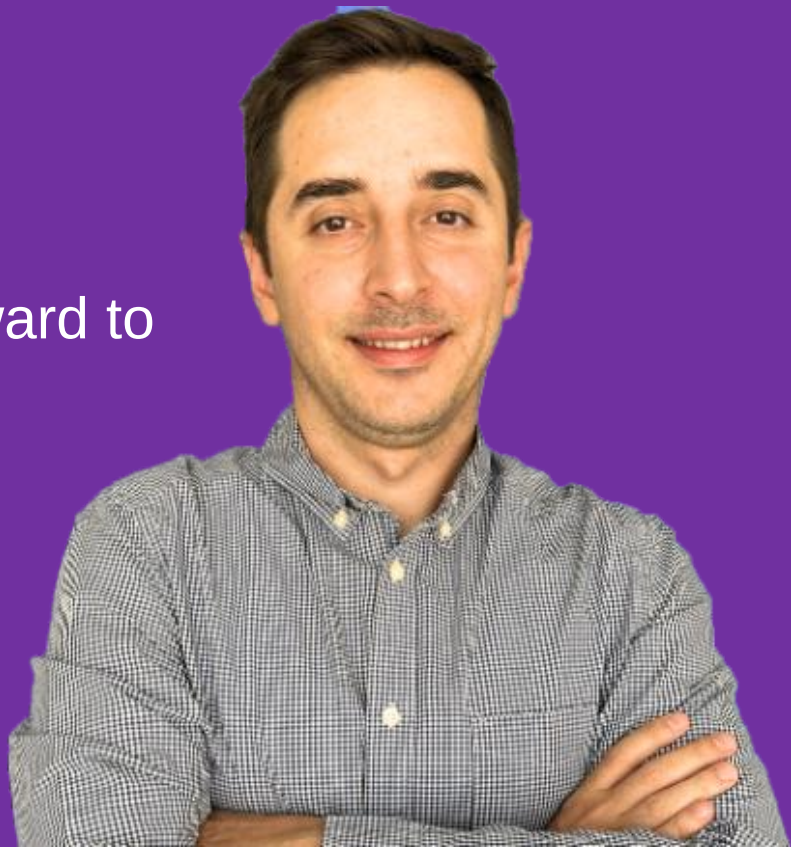
Image	Name	Description
	Hiking Poles	Ideal for camping and hiking trips
	Outdoor Rain Jacket	This product will keep you warm and dry in all weathers
	Survival Kit	A must-have for any outdoor adventurer
	Outdoor Backpack	This backpack is perfect for carrying all your outdoor essentials

Thanks



Thank you so much for being with me on this journey.

Reviews and feedback is really encourage to me for pushing forward to create new courses like this.



Mehmet Ozkaya