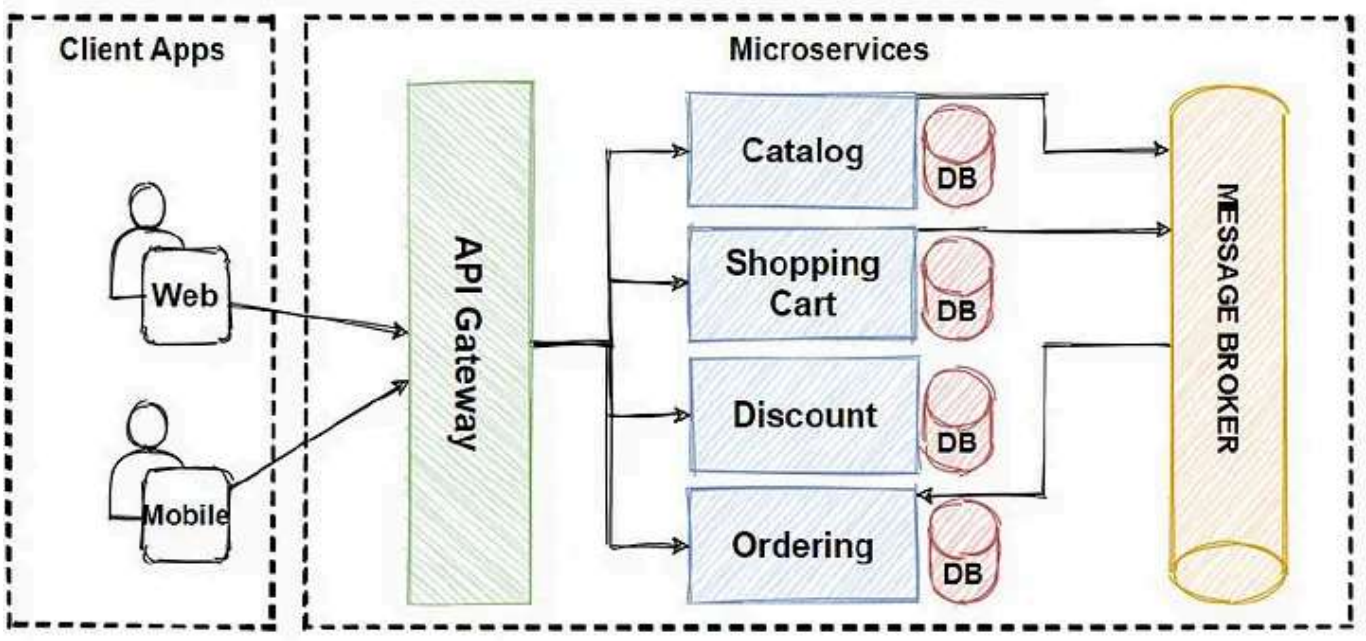
Microservices Architecture

Mehmet Ozkaya · Follow

Published in Design Microservices Architecture with Patterns & Principles ·
11 min read · Sep 6, 2021

In this article, we're going to learn **Microservices Architecture.** Of course we will start with **definitions, benefits** and **challenges.** After that we will **design Microservices architecture** on e-commerce domain.



By the end of the article, you will Learn where and when to **apply Microservices Architecture** with designing system with **high availability,**

high scalability, and low latency with microservices architectures.

Step by Step Design Architectures w/ Course

with Patterns & Principles.

In this course, we're going to learn **how to Design Microservices Architecture** with using **Design Patterns, Principles** and the **Best Practices.** We will start with designing **Monolithic** to **Event-Driven Microservices** step by step and together using the right architecture design patterns and techniques.

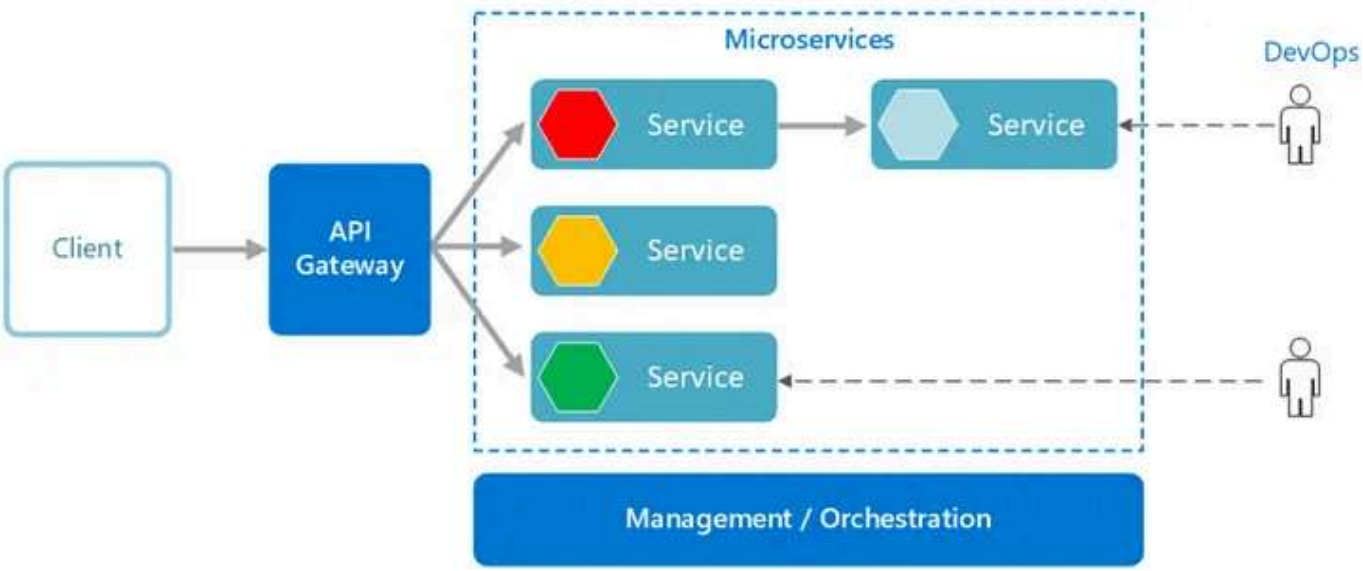
What are Microservices ?

Microservice are **small business services** that can work together and can be deployed **autonomously / independently.** These services communicate with each other by talking over the **network** and bring many advantages with them. One of the biggest advantages is that they can be d**eployed independently.** However, it offers the opportunity to work with many different technologies.

From **Martin Fowlers** Microservices article;

> *The microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP or gRPC API.*

These services are built around **business capabilities** and **independently deployable** by fully automated deployment process. There is a minimum of centralized management of these services, which may be written in different programming languages and use **different data storage** technologies.



So we can say that, Microservices architecture is a **cloud native architectural approach** in which applications composed of many loosely coupled and independently deployable smaller components.

Microservices;

- have their own **technology stack**, included the database and data management model;

- communicate to each other over a **combination of REST APIs**, event streaming, and message brokers;
- are **organized by business capability**, with the line separating services often referred to as a bounded context.

Microservices Characteristics

Microservices are **small**, **independent**, and **loosely coupled**. A single small team of developers can write and maintain a service. Each service is a **separate codebase**, which can be managed by a small development team.

Services can be **deployed independently**. A team can update an existing service without **rebuilding** and **redeploying** the entire application. Services are responsible for persisting their **own data** or external state. This differs from the traditional model, where a separate data layer handles data persistence.

Services communicate with each other by using **well-defined APIs**. Internal implementation details of each service are hidden from other services. Services don't need to share the same technology stack, libraries, or frameworks.

During the article, we are going to refer to these **microservices characteristics** when we develop our reference application.

Again if we referring the Martin Fowler article, There are some common characteristics for microservices architectures that fit the label.
Martin Fowler explain these characteristics at below captions;

Componentization via Services

Component is a unit of software that is independently replaceable and upgradeable.

Organized around Business Capabilities

The microservice approach to division is splitting up into services organized by business capability.

Products not Projects

This is Amazon's notion of "you build, you run it" where a development team takes full responsibility for the software in production.

Smart endpoints and dumb pipes

Microservices aim to be as decoupled and as cohesive as possible, so they own their own domain logic and receiving a request, applying logic and producing a response with using restful apis.

Decentralized Governance

Netflix is a good example of an organization that follows this philosophy. Sharing useful and all tested code as libraries encourages other developers to solve similar problems in similar ways.

Decentralized Data Management

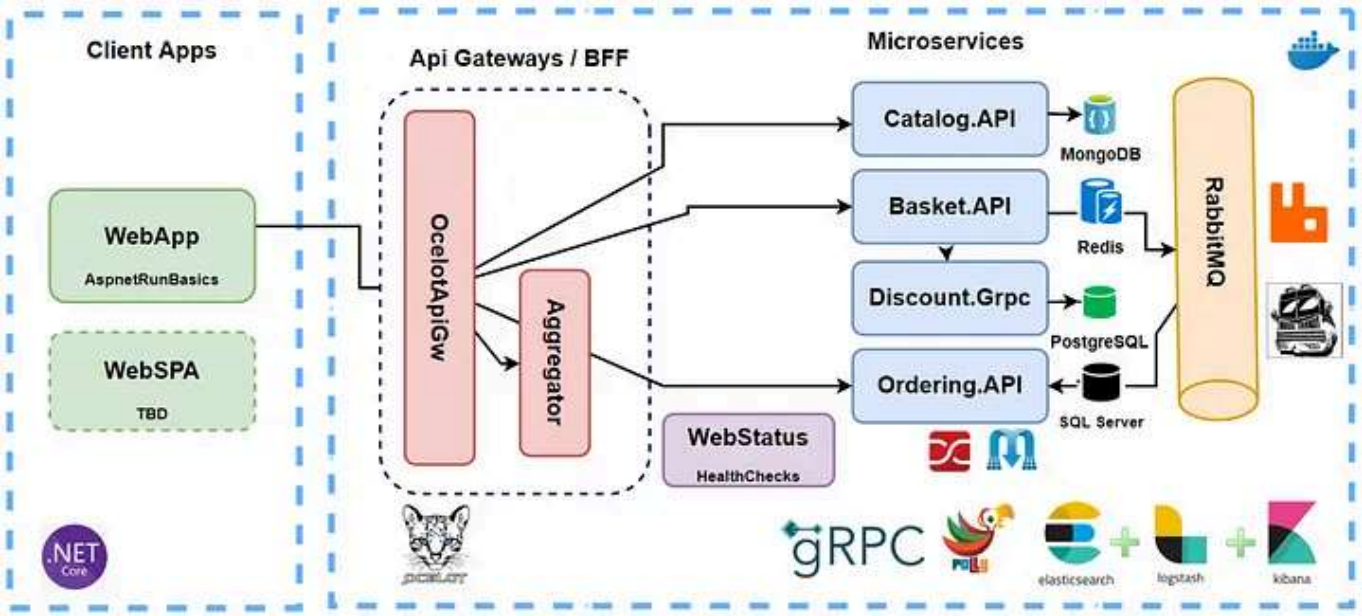
Microservices also decentralize data storage decisions. We can say this approach as a Polyglot Persistence or Polyglot Databases. That means Microservices prefer letting each service manage its own database, either different instances of the same database technology, or entirely different database systems.

Infrastructure Automation

That means automate deployment to each new environment and for every microservices with separately.

Design for failure

Microservices design by dealing failures and try to manage failures with managing errors with proper actions.



So these Characteristic brings us some key benefits;

- Code can be updated more easily — **new features or functionality** can be added without touching the entire application
- Teams can use **different tech stacks** and **different programming languages** for different components.
- Services can be **scaled independently**, it is **reducing the waste** and **cost associated** scale to entire applications because of a single feature might be facing too much load.

So we can say that; Each service is:

- **Highly maintainable** and **testable** — enables rapid and frequent development and deployment
- **Loosely coupled** with **other services** — enables a team to work independently the majority of time on their service(s) without being impacted by changing to other services and without affecting other services
- **Independently deployable** — enables a team to deploy their service without having to coordinate with other teams
- Capable of being **developed by a small team** — essential for high productivity by avoiding the high communication head of large teams

As as summarized;

Services communicate using either synchronous protocols such as **HTTP/REST** or **asynchronous protocols** such as **AMQP**. Services can be developed and **deployed independently** of one another. Each service has its own database in order to be **decoupled** from other services.

Benefits of Microservices Architecture

We are going to see Benefits-pros-advantages of microservices architecture. Lets elaborate these benefits one by one.

Agility

One of the most important characteristic of microservices is that because the services are smaller and independently deployable, it's easier to manage bug fixes and feature releases. You can update a service without re-deploying

the entire application, and **roll-back** an **update** if something goes wrong. In monolithic applications, if a bug is found in one part of the application, it can block the entire release process. So new features was waiting for a bug fix to be integrated, tested, and published.

Small, focused teams

A microservice should be small enough that a single feature team can build, test, and deploy it. The microservices model enables an organization to create small, cross-functional teams around one service or a collection of services and have them operate in an **agile fashion.** Small team sizes increase the agility. Large teams tend be less productive, due to communication is slower and management overhead goes up So that occurs the agility diminishes.

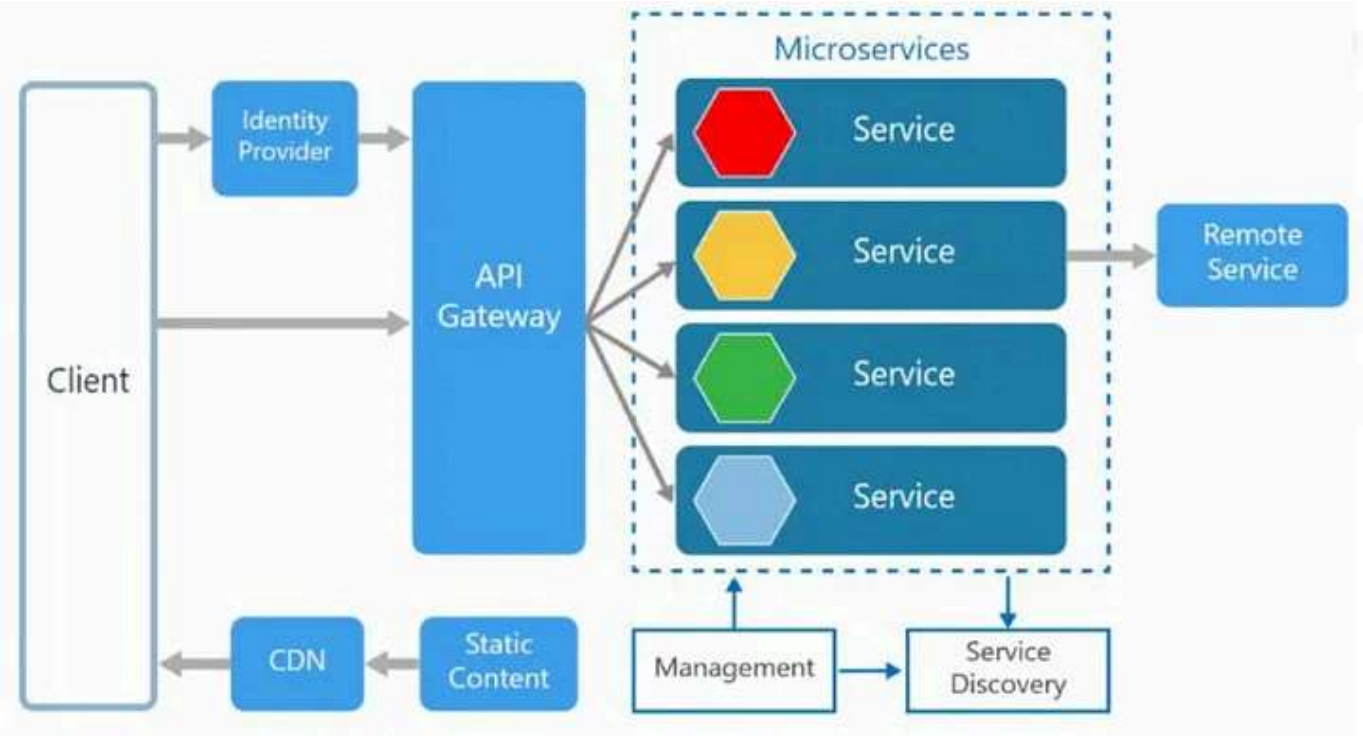
Small and Separated Code Base

In a monolithic application, The code base is going to be so bigger over time for code dependencies to become tangled. Try to **Adding a new feature** requires lots of refactoring on existing code base. Since microservices are not sharing code or data stores with other services, it minimizes dependencies, and that makes easier to adding new features.

Right tool for the job

In traditional layered architectures, an application typically shares a common stack, with a large relational database supporting the entire application. This approach has several challenges for example every component of an application must share a **common stack**, data model and database even if there is a clear, better tool for the job for certain modules. It's really frustrating for developers who are aware that a better, more efficient way to build these components is available. Also developers is

frustrating when the application stack is too old and can't apply new best practices on their projects.



https://www.futurefundamentals.com/what

But in microservices architecture, small teams can **pick the technology** that best fits their microservice and using a mix of technology stacks on their services. Because of microservices are deployed independently and communicate over **some combination of REST, event streaming** and message brokers. It's possible for the stack of every individual service to be optimized for that service.

Technology changes all the time, development libraries and tools also evolving very fast so since an application composed of multiple, smaller services, it is much easier and less expensive to evolve with more desirable technology into microservices.

Fault Isolation

Microservices loose coupling also builds with fault isolation and better resilience into applications. If one of your microservice becomes unavailable, it won't affect the entire application. Of course you should design your microservices are **fault tolerance** and **handle faults** correctly for example by implementing retry and circuit breaking patterns. Even failures happened, if you fix that failures without any business affect, you customer will always happy.

Scalability

Microservices can be scaled independently, so you scale out sub-services that require less resources, without scaling out the entire application. So we can say that, microservices require **less infrastructure** than monolithic applications because they enable precise scaling of only the required services,
instead of scaling the entire application.

Also scaling is very easy with using an container orchestrator tool like Kubernetes, you can pack a higher volume of services onto a single host, which allows for more efficient utilize of hardware resources.

Data isolation

Since microservices following the **database-per-service patterns**, databases are separated with each other according to microservices design. So it gets easier to perform schema updates, because only a single database is affected. In a monolithic application, schema updates can become very challenging, and risky.

As you can see that we have seen lots of Benefits of microservices, so now its time to see challenges of microservices architecture.

Challenges of Microservices Architecture

Microservices has significant benefits but also has significant challenges. Moving from monolith to microservices means a lot more management complexity. Here are some of the challenges we need to consider before applying to microservices architecture.

Complexity

Microservices application has lots of services need to work together and should create a value. Since there are lots of services, that means there is more moving parts than the monolithic application. Each service is simpler, but the **entire system is more complex**. Even deployments can be complicated for hundreds of services deploy different times hard to manage versions. Think about the communication. Its really easy to communicate components in monolithic application because it is inter-process communication, can be the same machine and same process. But microservices **communication** is **hard topic** and need to have a strategy to manage inter-service communications between server even different geo-locations.

Network problems and latency

So since microservice are small and communicate with inter-service communication, we should **manage network problems**. If we call chain of services for particular request, this will increate latency problems and need correct design to APIs for proper communication. In order to avoid chatty API calls. You need to consider asynchronous communication patterns like message broker systems.

Development and Testing

Think about the E2E process one of the long business requirement. If the requirement touch on several microservices that need to act as a 1

application, its hard to develop and testing these **E2E processes** in microservices architectures if we compare to monolithic ones. Existing tools are not always designed to work with service dependencies. Refactoring across service boundaries can be difficult.

Data Integrity

Microservice has its own data persistence. So data consistency can be a challenge. Mostly we should follow **eventual consistency** where possible. But transactional operations are always will be challenging.

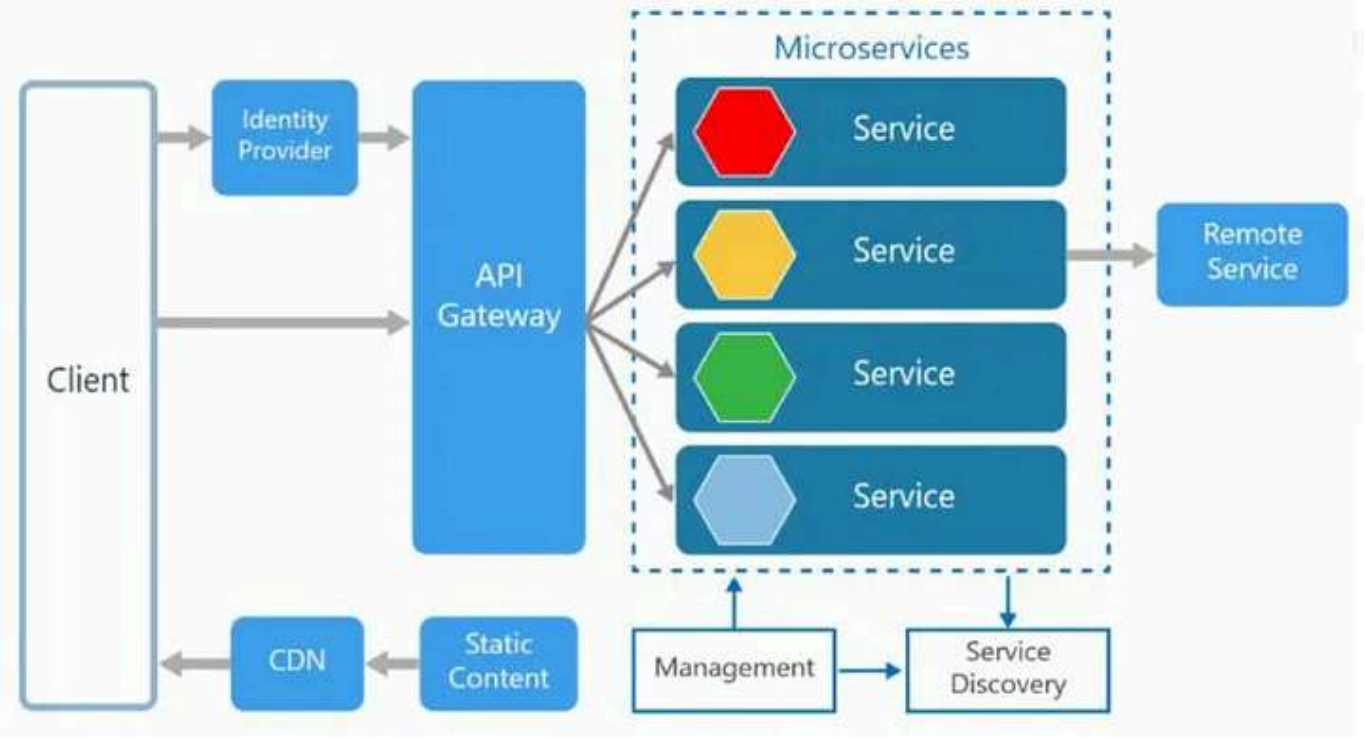
Nevertheless, these challenges aren't stopping to adopting microservices. Most of the organizations accept these challenges and adapting their technologies to microservices architecture in order to get benefits of this architecture.

Reference Architectures of Microservices Architecture

We are going to see example of Reference architectures of Microservices Architecture. So we will follow these reference architectures and create our own e-commerce architecture. Let's take an action.

Reference Microservice Architectures 1

First of all, we start with the simple ones.

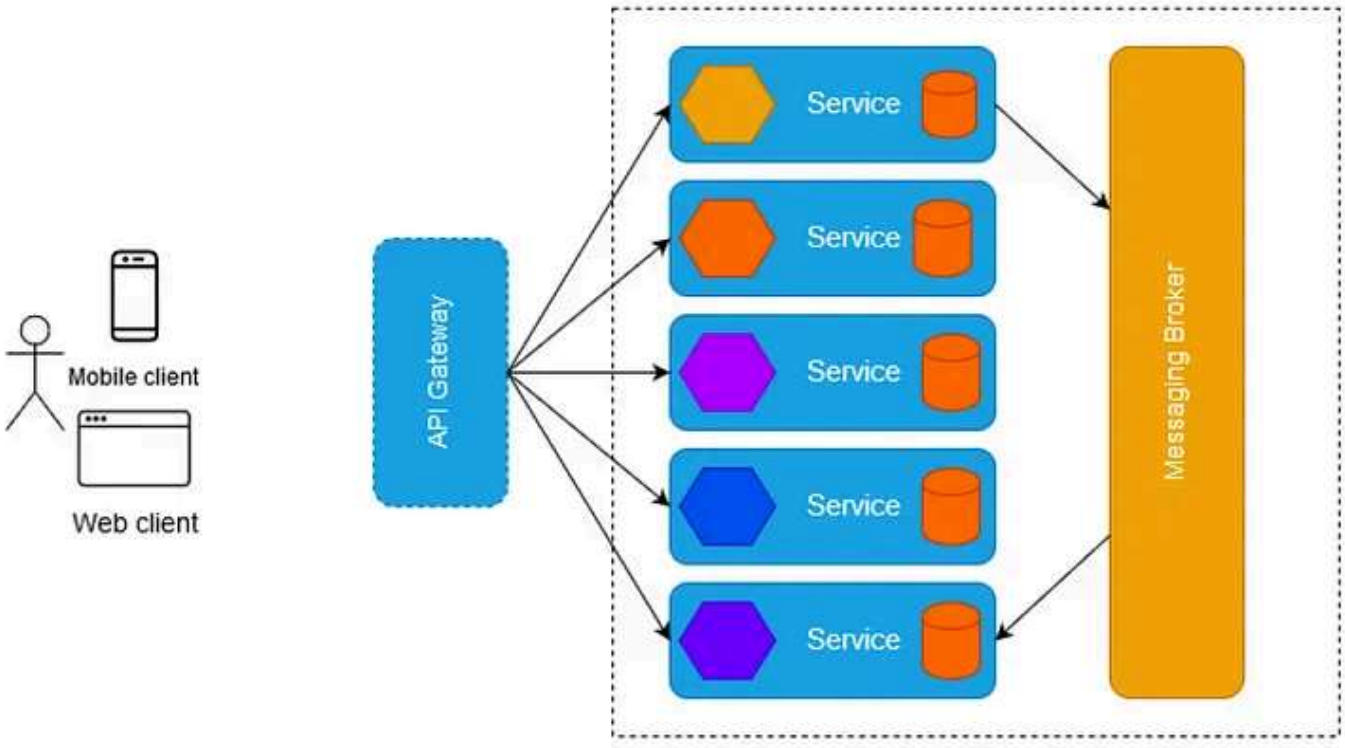


https://www.futurefundamentals.com/what

As you can see the image has several microservices and one API GW which provide to communicate microservices to client applications.

Reference Microservice Architectures 2

See the different illustrations of microservices architecture.

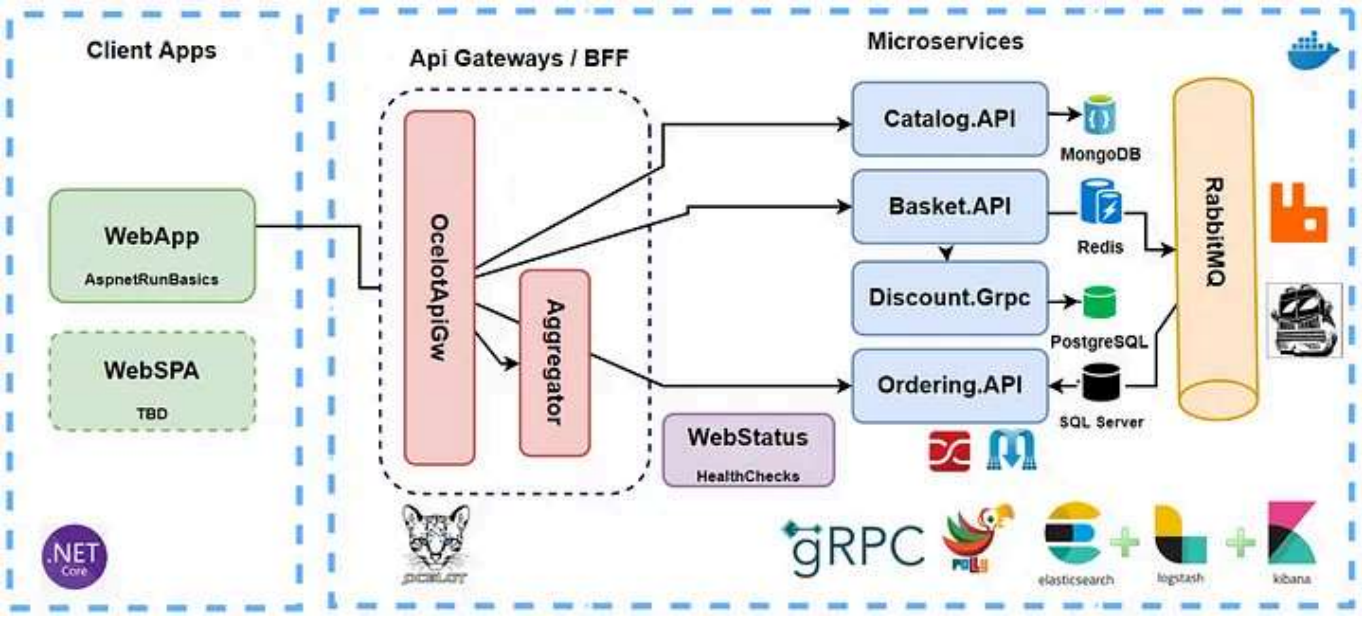


https://feras.blog/microservices

This is so common view of microservices architecture. It has several microservices, API GW and the message broker for provide communication among microservices.

Reference Microservice Architectures 3

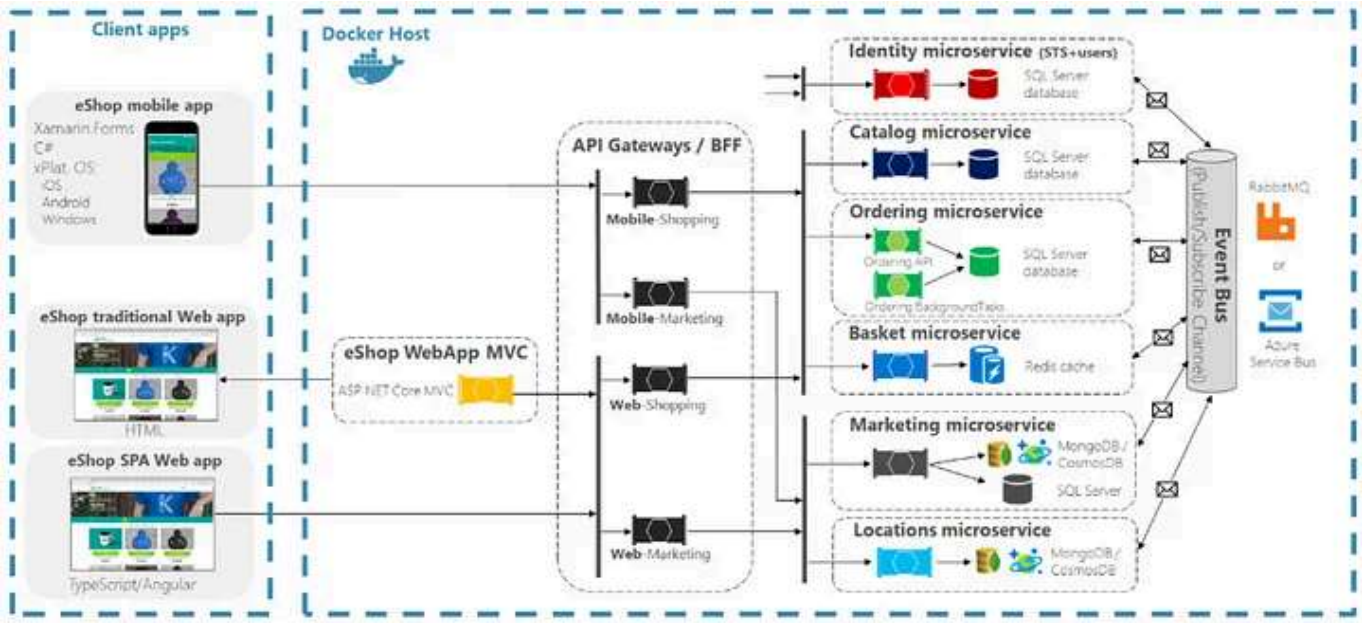
See the different illustrations of microservices architecture.



And you can see another reference architecture. This is also one of the e-commerce reference architecture that we will evolve step by step this architecture.

Reference Microservice Architectures 4

See the different illustrations of microservices architecture.



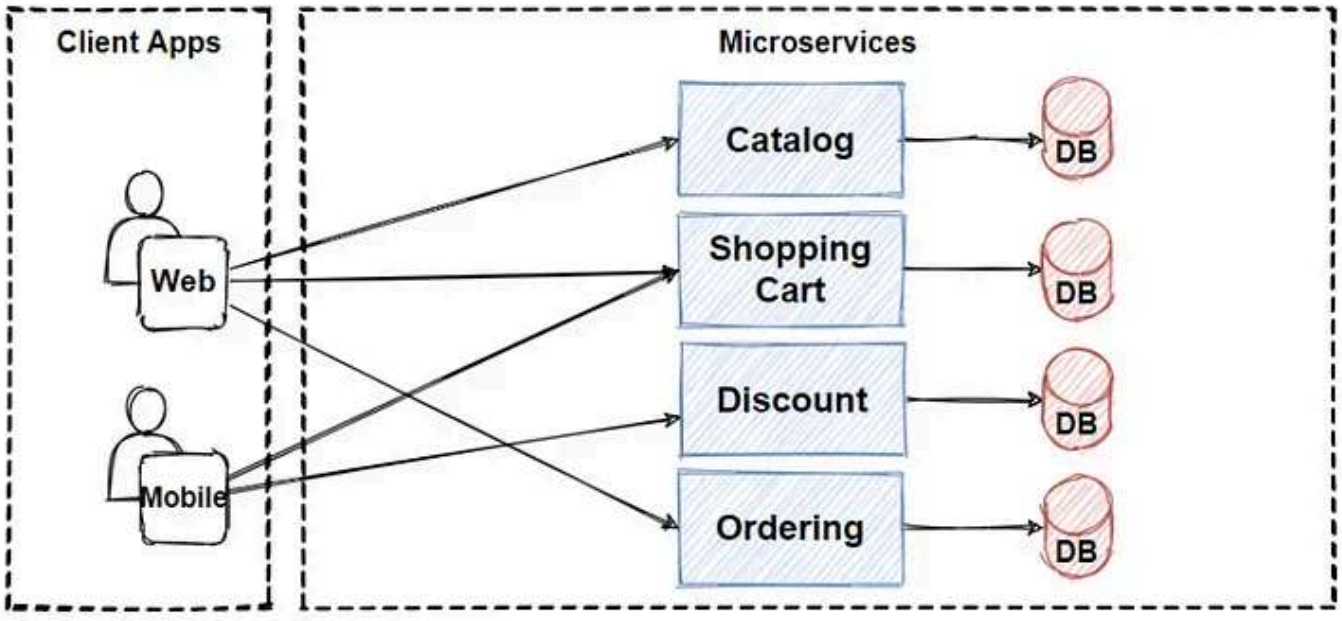
https://docs.microsoft.com/en

And lastly we can see some advance microservices architecture. You can see several **API GWs** for different client applications. And several microservices communication via Event Bus systems.

We will see why we use these components when architecting microservice architecture step by step in the next caption.

Design Microservice Architecture — E-Commerce App

We are going to design the Microservice architecture step by step. Iterate the arch design one by one as per requirements. We had a requirement that save the orders see the baskets so on. So we should need database design in this architecture.



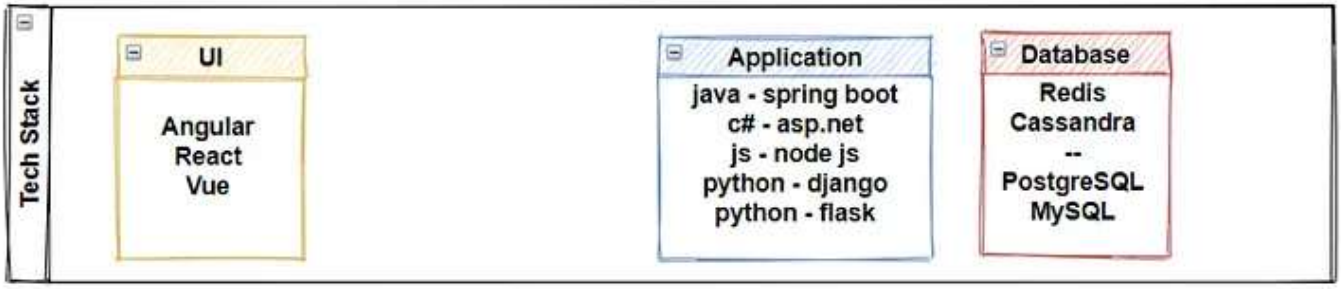
As you can see that we have applied "**Database per Microservices**" and put database for every microservices on our e-commerce application. So now these database can be polyglot persistence.
That means Product microservice can use NoSQL document database Shopping Cart microservice can use NoSQL key-value pair database and

Order microservice can use Relational database as per ms data storage requirements.

As you can see that we have design our **e -commerce** application as a **Microservices architecture** and handle to FR and NFR requirements like **scalability, reliability** and so on.

Technology choices — Adapting Technology Stack

We are going to **Adapting Technology Stack**, implement possible Technology choices.



Microservices are polyglot environments. You can pick any tech stack, as per microservices. **Communication** will be handle via **REST APIs.**

- java — spring boot
- c# — asp.net
- js — node js
- python — django
- python — flask

These are **backend web api** language and frameworks that can be suited. Since ms are polyglot environments you can pick one of them for every particular microservices.

Databases
Now we can choose nosql or relational DBs as per ms persistence requirements.

- NoSQL
- MongoDb
- Redis
- Cassandra

For Relational

- PostgreSQL
- MySQL
- Oracle
- Sql Server

So these are the options that we can use on our e-commerce microservice architecture. We will decide this tools as per our requirements and company it strategy.

So what's next ?

See that UI and MS communication are direct, and it seems hard to manage communications. We now we should focus on microservices communications with applying **API GW pattern** and evolving these architecture step by step.

Step by Step Design Architectures w/ Course

I have just published a new course — Design Microservices Architecture with Patterns & Principles.

In this course, we're going to learn **how to Design Microservices Architecture** with using **Design Patterns, Principles** and the **Best Practices.** We will start with designing **Monolithic** to **Event-Driven Microservices** step by step and together using the right architecture design patterns and techniques.

Microservices | Microservice Architecture | Software Architecture | Api Gateway | Microservices Pattern

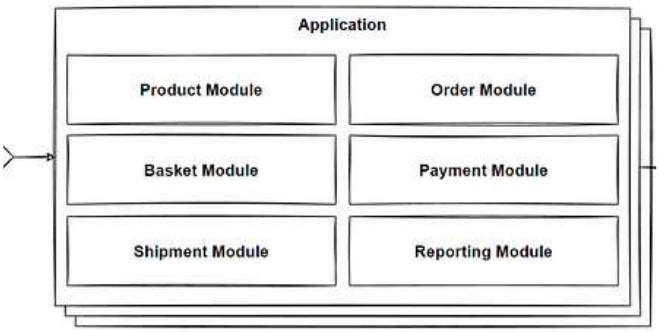
Written by Mehmet Ozkaya

Follow

7.4K Followers · Editor for Design Microservices Architecture with Patterns & Principles

Software Architect | Udemy Instructor | AWS Community Builder | Cloud-Native and Serverless Event-driven Microservices https://github.com/mehmetozkaya

More from Mehmet Ozkaya and Design Microservices Architecture with Patterns & Principles



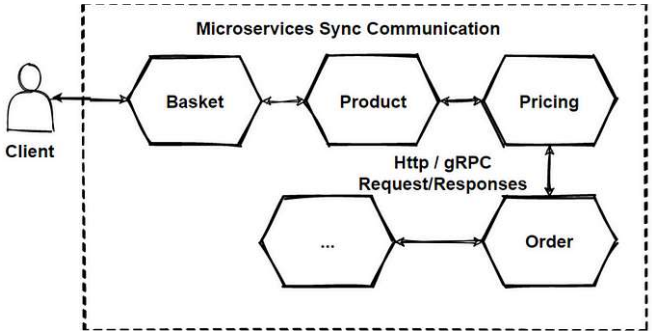
Mehmet... in Design Microservices Architecture w...

Microservices Killer: Modular Monolithic Architecture

In this article, we are going to learn Modular Monolithic Architecture and Best Practices...

8 min read · Feb 16, 2023

284 7



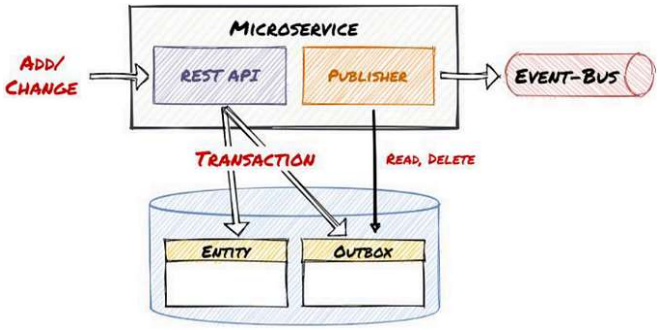
Mehmet... in Design Microservices Architecture w...

Microservices Communications

In this article, we're going to learn Microservices Communications. We will lear...

17 min read · Sep 7, 2021

1.1K 11



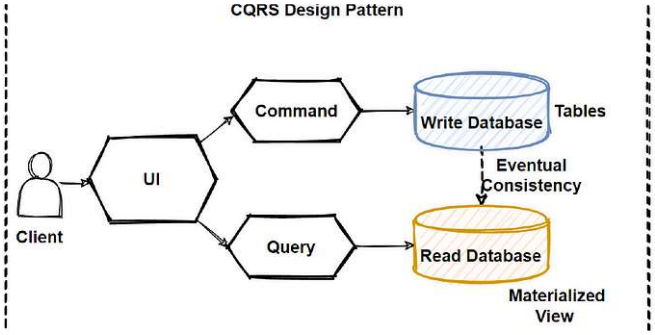
Mehmet... in Design Microservices Architecture w...

Outbox Pattern for Microservices Architectures

In this article, we are going to talk about Design Patterns of Microservices architectur...

3 min read · Sep 8, 2021

336 4



Mehmet... in Design Microservices Architecture w...

CQRS Design Pattern in Microservices Architectures

In this article, we are going to talk about Design Patterns of Microservices architectur...

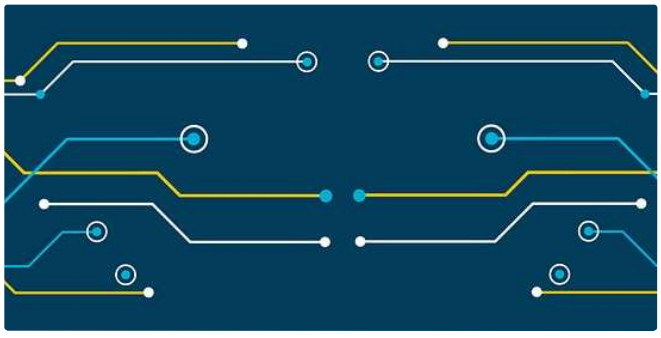
6 min read · Sep 8, 2021

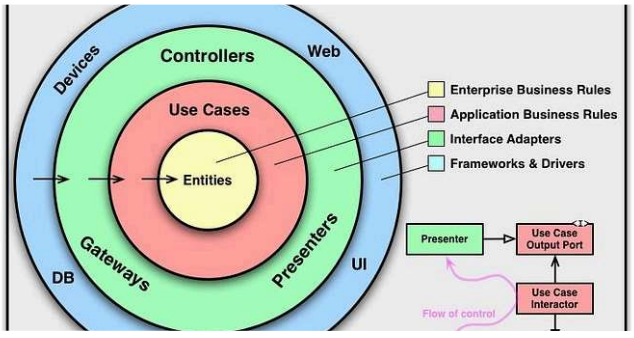
697 11

See all from Mehmet Ozkaya

See all from Design Microservices Architecture with Patterns & Principles

Recommended from Medium





Capital One Tech in Capital One Tech

10 microservices design patterns for better architecture

Consider using these popular design patterns in your next microservices app and make...

11 min read · Jan 10, 2024

2.5K 16

Jeslur Rahman

Clean Architecture Fundamentals | .NET Core Web API Microservice...

Building scalable, maintainable, and testable applications is paramount in the ever-...

6 min read · Jan 3, 2024

57

Lists

Stories to Help You Grow as a Software Developer

19 stories · 1051 saves

Staff Picks

639 stories · 970 saves

General Coding Knowledge

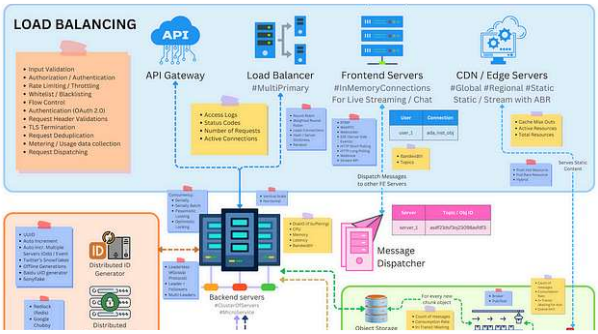
20 stories · 1205 saves

Pramitha Jayasooriya

Monolithic vs. SOA vs. Microservices Architecture: A Jav...

In the evolving landscape of software development, the architecture of how...

5 min read · Feb 8, 2024



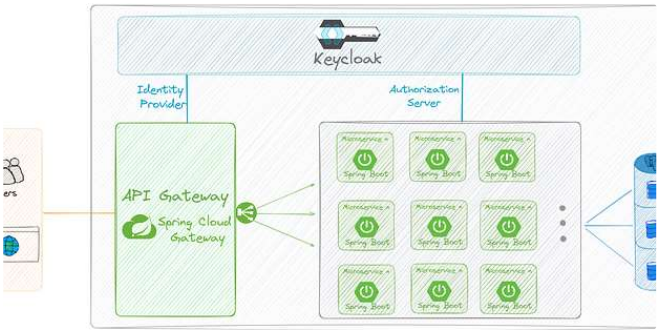
Love Shar... in ByteByteGo System Design Allian...

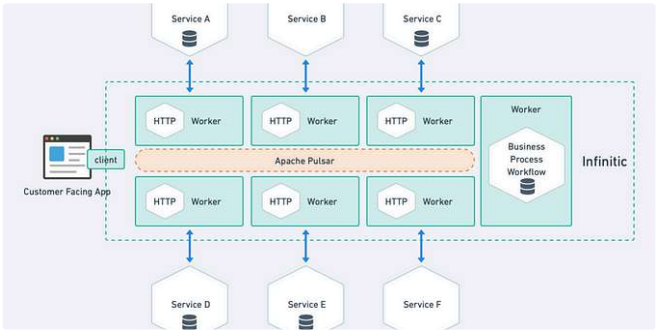
System Design Blueprint: The Ultimate Guide

Developing a robust, scalable, and efficient system can be daunting. However,...

9 min read · Sep 17, 2023

1 8.4K 58





Andrea Zagarella

Microservice Architecture | a real business-world example with API…

A real business-world example of a Microservice Architecture with a particular…

8 min read · Feb 26, 2024

60

Gilles Barbier

An Easy Path From API-Based Microservices to An Event-Driven…

Building reliable business processes using API-driven microservices is a common goal…

9 min read · Apr 30, 2024

484 3

See more recommendations