Course Introduction

Micorservices Architecture

Yan Liu
givemeareason@gmail.com
School of Software Engineering, Tongji University

Index

- Course Overview
- Resources and Materials
  - Prerequisites
  - References
  - Main Tools, Frameworks and Environments
- Assessment
- Introduction to the Lecture Topics
- Microservices at a Glance
- The Evolution towards Microservices Architecture
- What is Microservices Architecture?
- Characteristics of Microservices
- Benefits and **Pain Points** of Microservices Architecture
- Microservices and ...
  - Microservices and APIs
  - Cloud Native and Microservices
- Microservices are more than Writing the Code
  - Examples of Design Patterns and Strategies for Microservices
  - Enabling Technology
  - Examples of Technology Solutions
- *A Simple Demo*

Course Overview

- Microservice architectures are the 'new normal'. Building small, self-contained, ready to run applications can bring great flexibility and added resilience to your code.
  - Services are distributed components that provide well-defined interfaces that process and deliver XML (*or in other data interchange format*) messages.
  - **Microservices** are independently releasable services that are modeled around a business domain.
- In this course , you will learn the basic concepts, design and implementation technologies that support microservices architecture.
- How to build and enable Microservices will be explored with Spring Cloud, Spring Boot and Docker. How to mediate and orchestrate services will be explained with examples.
- Furthermore, this course will introduce design guidelines and patterns for microservices. The strategies available to realize each design guideline are also covered.

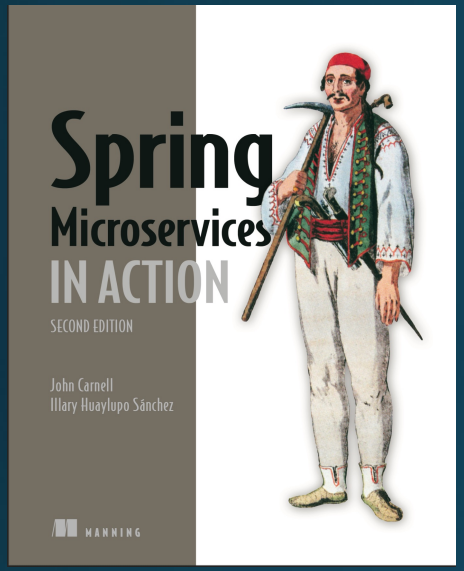
Prerequisites

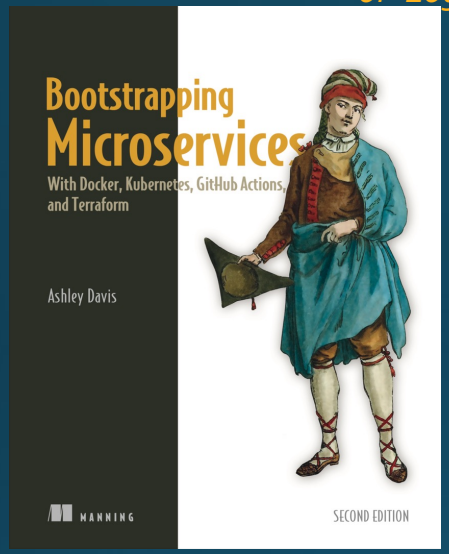
- Preferably programming skills in Java
- Developed distributed applications with Java or Python
- Knowledge on JavaEE architecture and programming experiences will be helpful but not mandotory.

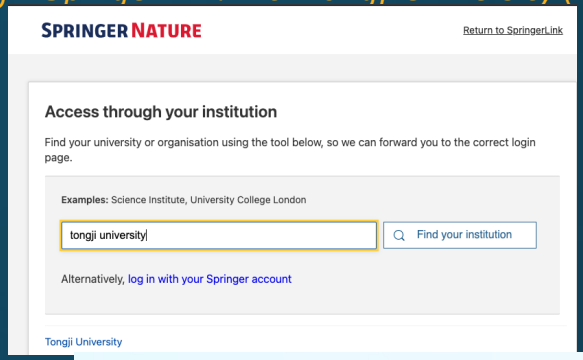
References

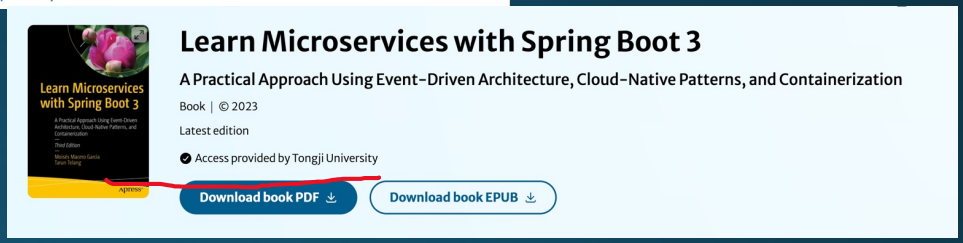
- *Some E-Books Available Through Tongji Campus Network*

or Login SpingerLink with Tongji University (Access via Institution)

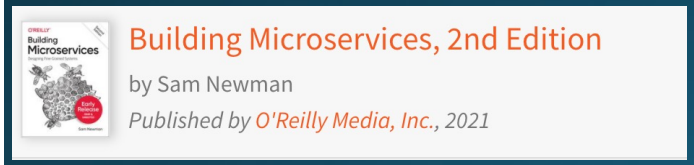


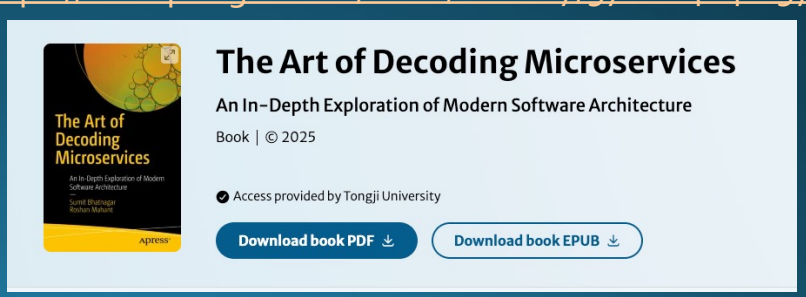






https://link.springer.com/book/10.1007/978-1-4842-9757-5



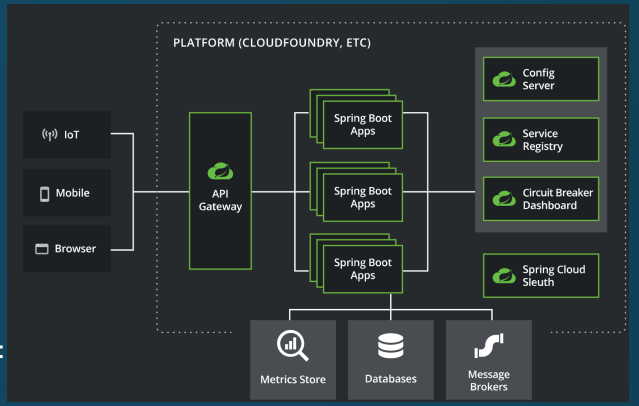


https://link.springer.com/book/10.1007/979-8-8688-1267-5

Many online resources…

Tools, Frameworks & Environments

- IDEs
  - Backend: IntelliJ IDEA, NetBeans, or VS Code…
  - Frontend: Webstorm or Visual Studio Code…
- API Platform and tools
  - API Documentation and Design Tools Swagger, Apifox, or YApi…
  - API Tools: PostMan, Advanced Rest Client, Insomonia API Client, or Paw (https://paw.cloud/)
- Frameworks and Platforms for Building and Deploying Microservices:
  - Spring Boot & Spring Cloud
  - Flask, FastAPI (Python Web APIs)
- Language, Tools, and Platforms for Integration
  - Anypoint Platform / Studio & Mule
  - GraphQL
- DevOps Tools
  - Github Actions, Jenkins…
  - Docker, Kubernetes…
- Many Platforms, frameworks, and more middlewares….
  - AXON, https://github.com/AxonFramework/AxonFramework
  - Dubbo, https://github.com/apache/dubbo
  - Kong, https://github.com/Kong/kong
- Cloud Platforms
  - Alibaba MSE, https://www.alibabacloud.com/en/product/microservices-engine
  - AHAS, https://www.alibabacloud.com/en/product/ahas



https://spring.io/microservices

Polyglot programming:
Microservices are free to
Choose the technical stack

Spring Cloud Alibaba

Components

Sentinel: Sentinel takes "traffic flow" as the breakthrough point, and provides solutions in areas such as flow control, concurrency, circuit breaking, and load protection to protect service stability.

Nacos: An easy-to-use dynamic service discovery, configuration and service management platform for building cloud native applications.

RocketMQ: A distributed messaging and streaming platform with low latency, high performance and reliability, trillion-level capacity and flexible scalability.

Seata: A distributed transaction solution with high performance and ease of use for microservices architecture.

Alibaba Cloud OSS: An encrypted and secure cloud storage service which stores, processes and accesses massive amounts of data from anywhere in the world.

Alibaba Cloud SMS: A messaging service that covers the globe, Alibaba SMS provides convenient, efficient, and intelligent communication capabilities that help businesses quickly contact their customers.

Alibaba Cloud SchedulerX: Accurate, highly reliable, and highly available scheduled job scheduling services with response time within seconds.

https://github.com/alibaba/spring-cloud-alibaba

Assessment

- Attendance and Class Performance, *10%*
- Individual Project, *30%*
  - Detailed requirements will be released on Week 3.
  - Programming with APIs & Web APIs;
  - Working with data interexchange formats and protocols;
  - Building a Mashup or integrated Application/App.
- Team Project (no more than 5 members), *60%*
  - Project requirements will be released on Week 3.
  - Project proposal, planning and presentation; 15%
  - Middle-Term presentation and project progress; 15%
  - Final presentation and qualities of final deliveries; 30%

Lecture Topics (1)

- Course introduction and Microservices Overview
- Microservices Fundamentals & Simple Case Study
- Domain-Driven Design for Microservices
- Microservices Communication Styles
- Microservices Enabling Technologies
- *Individual Assignment & Team project proposal*
- Exploring Microservices with Spring Cloud and Spring Boot
- Designing Microservices: Splitting and Integration

Lecture Topics (2)

- Design Patterns for Microservices and API Styles
- *Middle-term presentation*
- Building Event-Driven Microservices
- Microservices Data Decomposition
- Healthy Microservices
- Emerging trends & reviews
- *Final presentation*

Office Hours

- WeChat Group
- TA: To be announced…

Share Your Experiences with Us…

Think – Pair –Share (5 minutes)

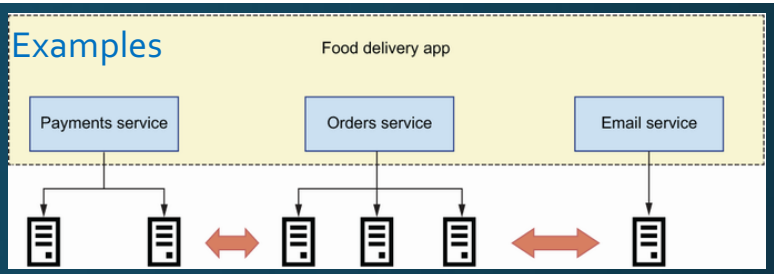
- Any Java or Python Programming Experiences?
- Any previous programming experiences on networking, remote invocations, or REST APIs?
- Any previous experiences on building solutions for complex business scenarios?
- What is your current understanding of Microservices Architecture?

Introducing Microservices

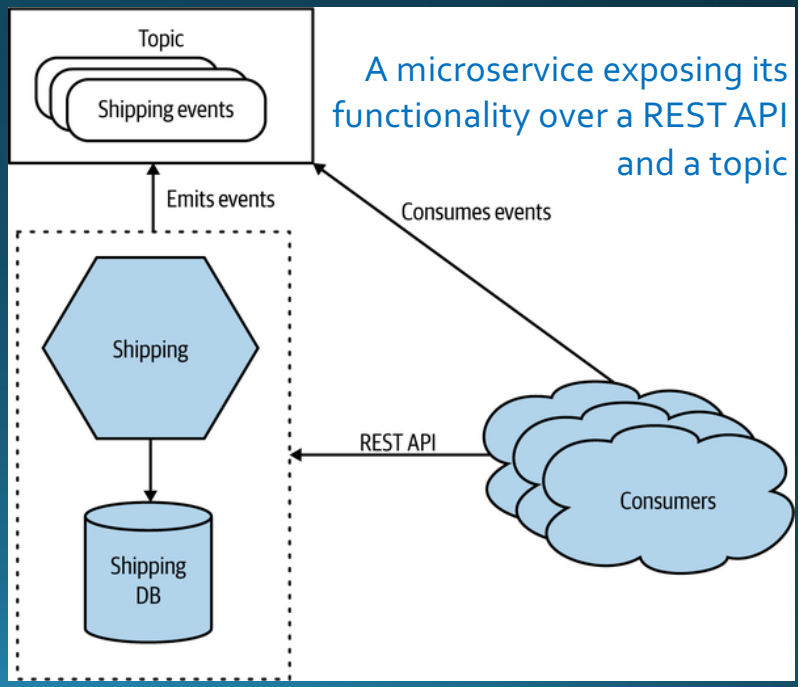
In short, the microservice architectural style is an approach to developing a single application as a **suite of small services**, each **running in its own process** and communicating with lightweight mechanisms, often an HTTP resource API. These services are **built around business capabilities** and **independently deployable** by fully automated deployment machinery. There is a **bare minimum of centralized management** of these services, which may be written in different programming languages and use different data storage technologies.

-- James Lewis and Martin Fowler (2014)

https://www.martinfowler.com/microservices/

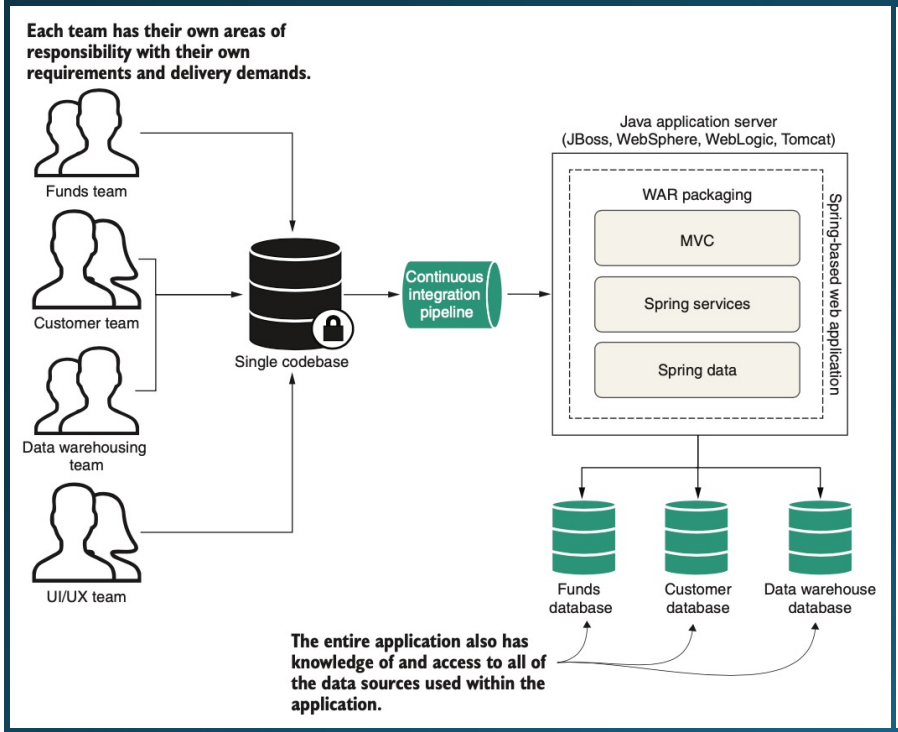


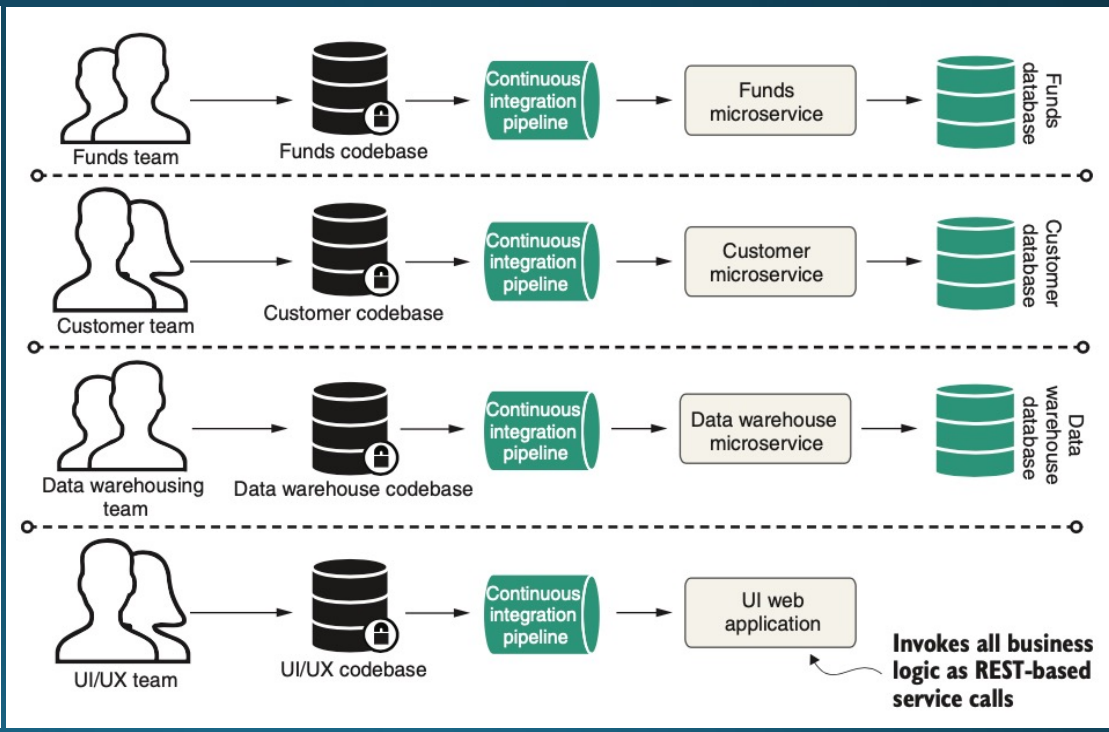
In microservices architecture, every service implements a specific business subdomain and is deployed as an independent component that runs in its own process.



A microservice exposing its functionality over a REST API and a topic

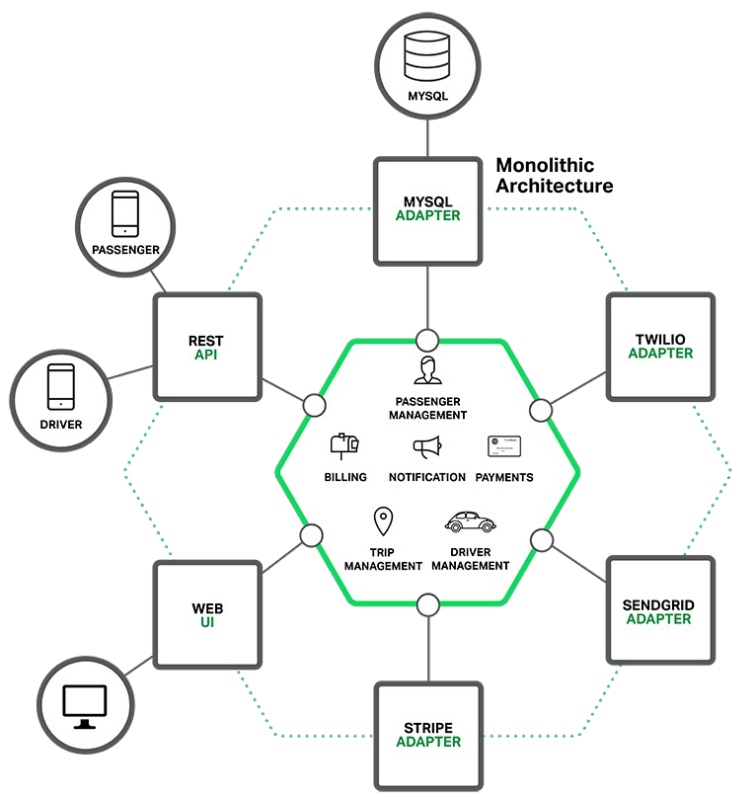
The evolution towards a microservice architecture

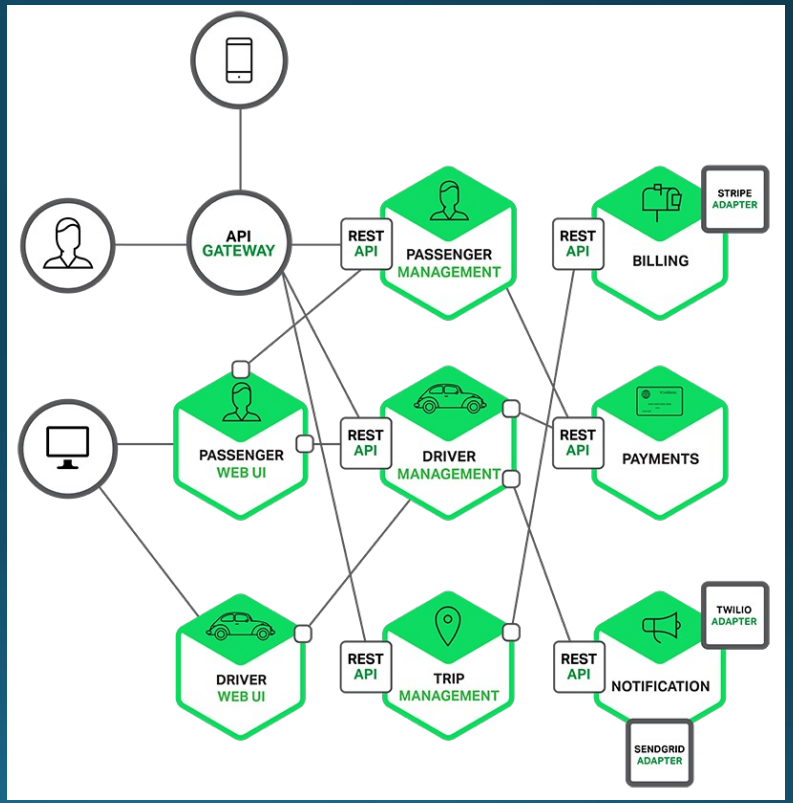




A *microservice* is a small, loosely coupled, distributed service.

Monolithic vs. Microservices





https://www.nginx.com/blog/introduction-to-microservices/

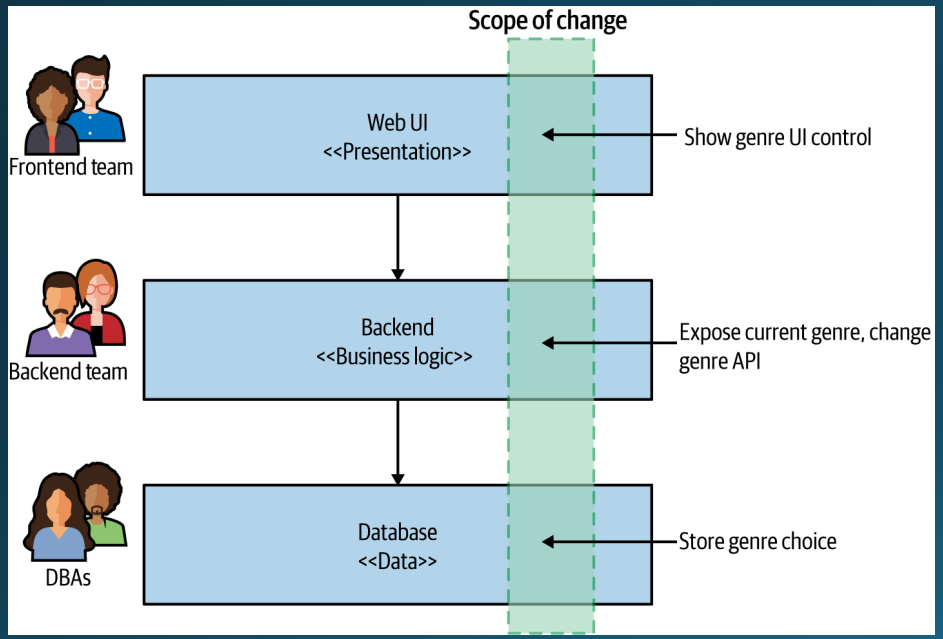
Imagine that you were starting to build a brand new taxi-hailing application

Sample Scenarios: Monolithic to Microservices

Online music store with a 3-tier architecture

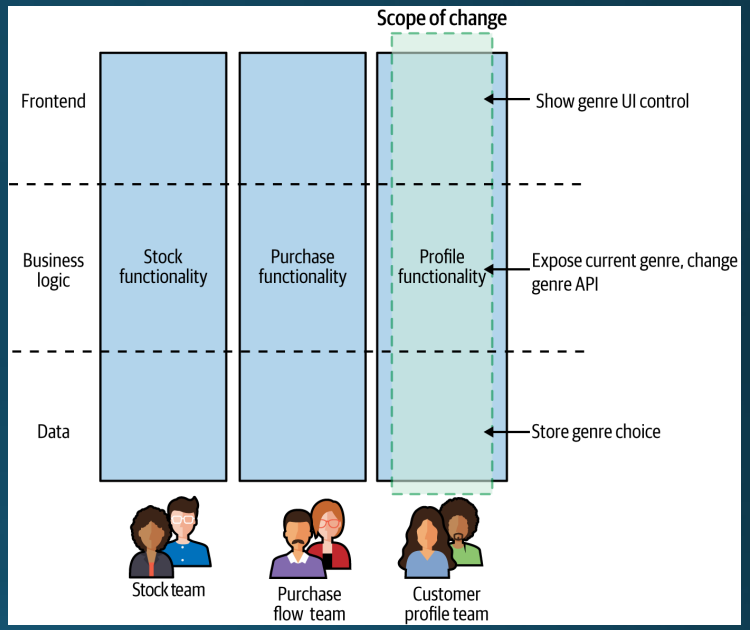
To add a new functionality:

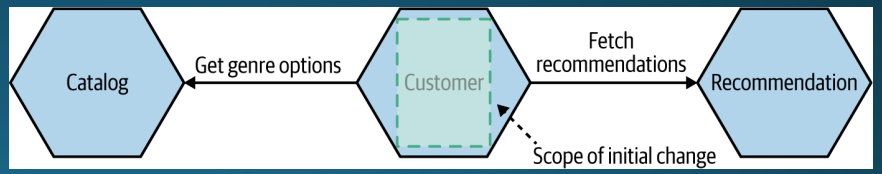
allow customers to specify their favorite genre of music.



A traditional three-tiered architecture

Making a change across all three tiers is more involved



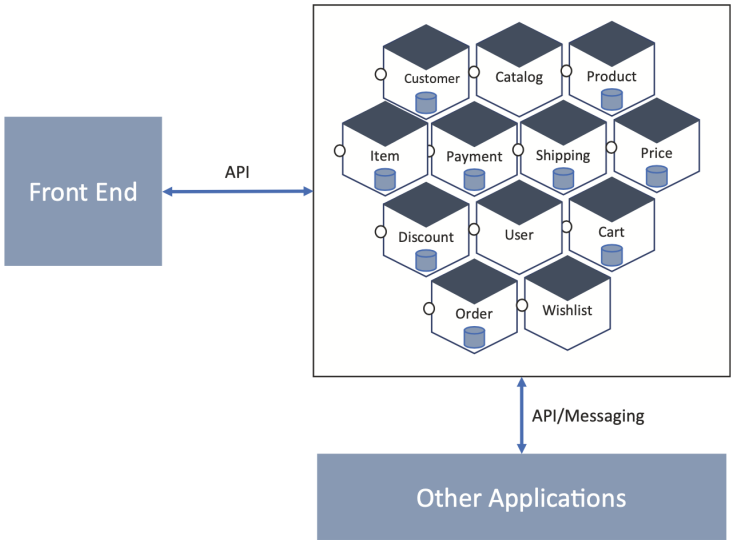


- The UI is broken apart and is owned by a team that also manages the server side functionality that supports the UI.
- A dedicated Customer microservice can make it much easier to record the favorite musical genre for a customer.

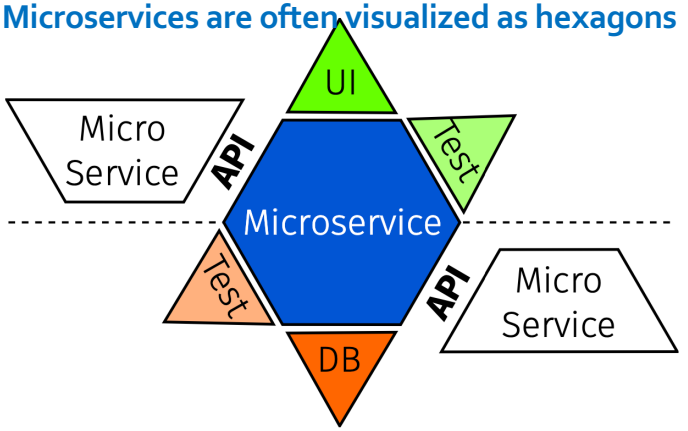
What is a Microservices Architecture?

A microservices architecture is an approach to developing one business application as a suite of domain services, with **each domain service running on its container and communicating with a lightweight mechanism in a synchronous or asynchronous manner** by using HTTP, GRPC, or messaging.

- These domain services are developed around business capabilities using **independently deployable but fully automated** deployment tools.
- Each domain service is **decentralized in nature** and developed with the polylithic and polyglot principles in mind.

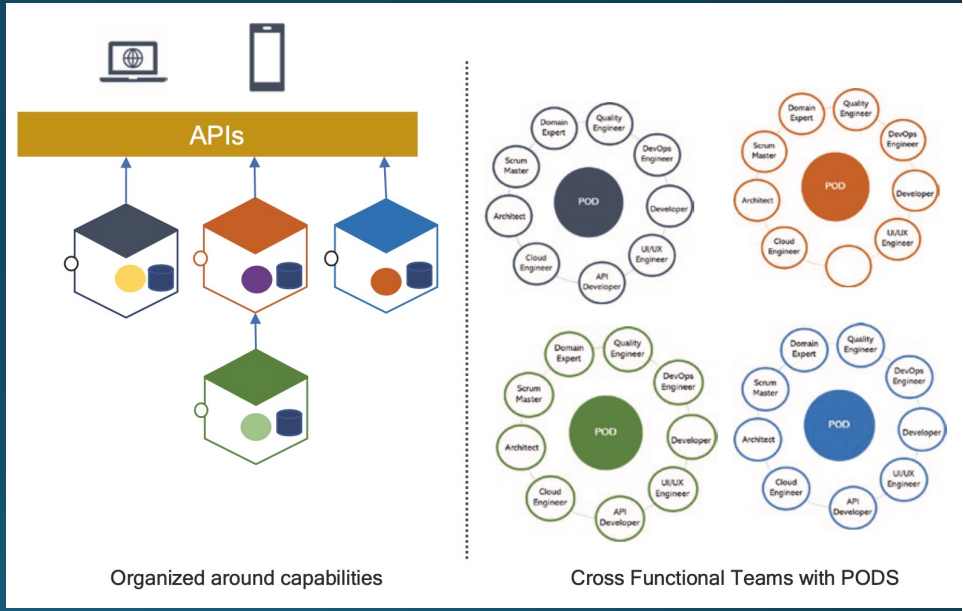


Microservices are often visualized as hexagons

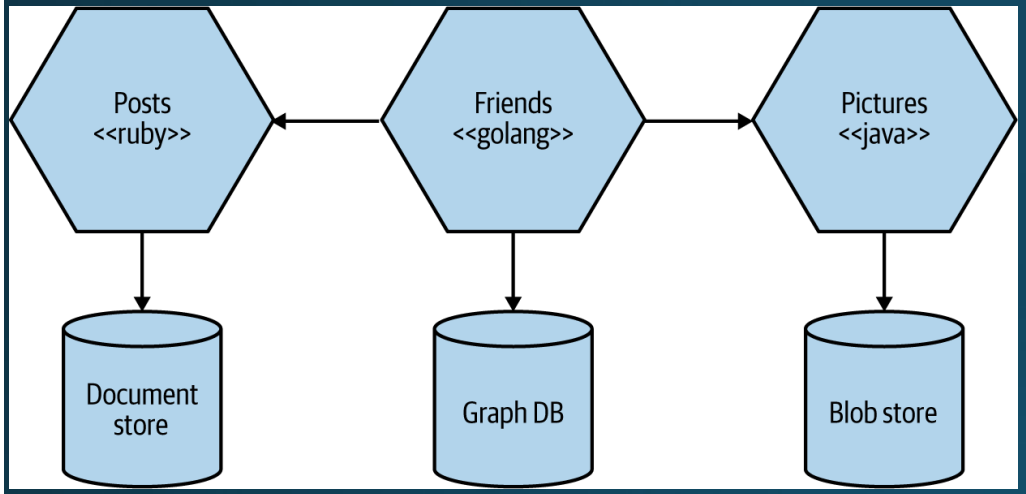


Characteristics of Microservices

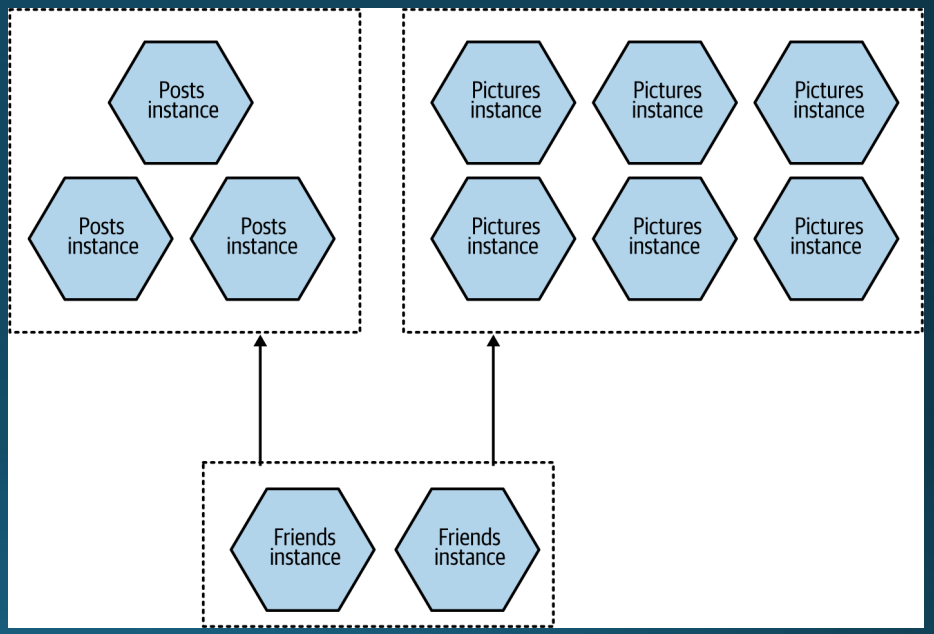
- Componentization via services
- Organized around business capabilities
- Products not projects
- Smart endpoints and dumb pipes
- Decentralized governance
- Decentralized data management
- Infrastructure automation
- Design for failure
- Evolutionary design



Some Benefits of Microservices



Microservices can allow you to more easily embrace different technologies



You can target scaling at just the microservices that need it

More benefits and challenges brought to developers will be discussed later on …

Microservice Pain Points

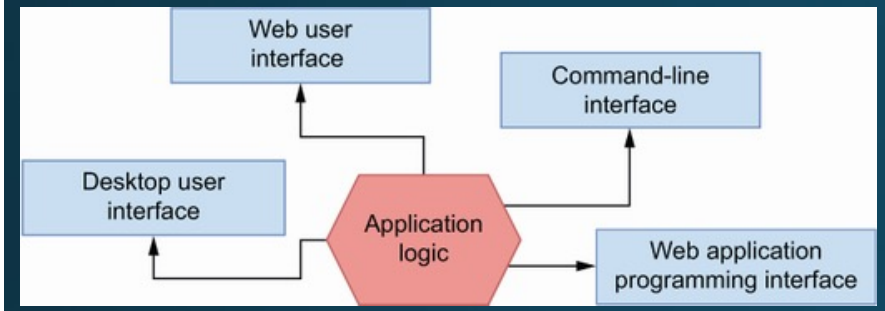
- Developer Experience
- Technology Overload
- Cost
- Reporting
- Monitoring and Troubleshooting
- Security
- Testing
- Latency
- Data Consistency

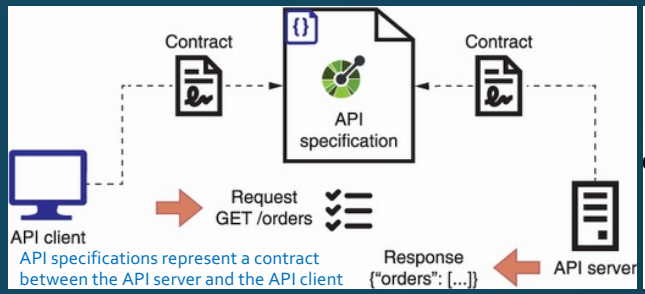
- Agile Practices
- Business Modeling
- Architecture Evolution
- Teams & Communication
- System Thinking & Design Thinking
- Data Intensive or Process Centric
- Patterns
- Enablers
- XaaC
- Platforms & Tools
- API-Led
- Cloud-Native
- DevOps
- Automation
- Develop on Cadence, Release on Demand
- ....

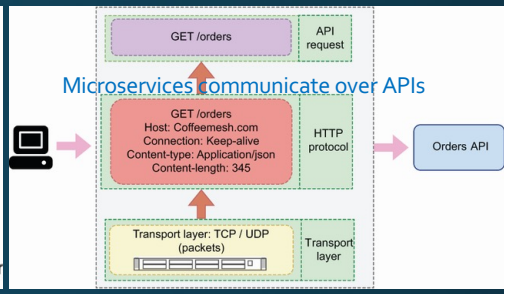
Relationship between Microservices and APIs

An **API** is an interface that allows us to programmatically interact with an application.

- Application can have one or more of these interfaces.

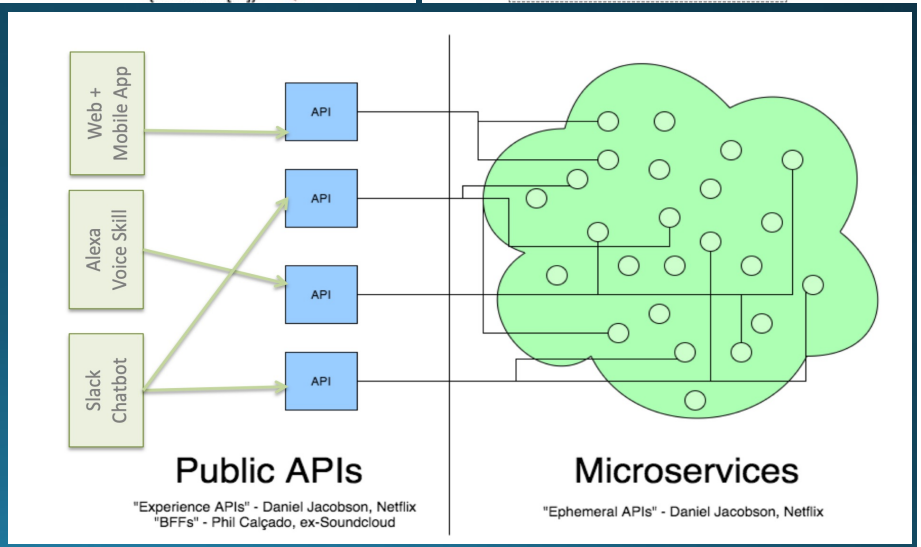




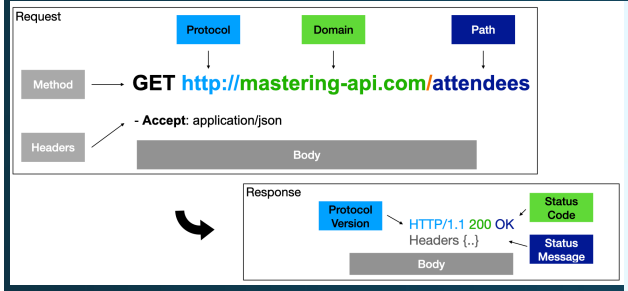


A **web API** is an API that uses the Hypertext Transfer Protocol (HTTP) protocol to transport data.

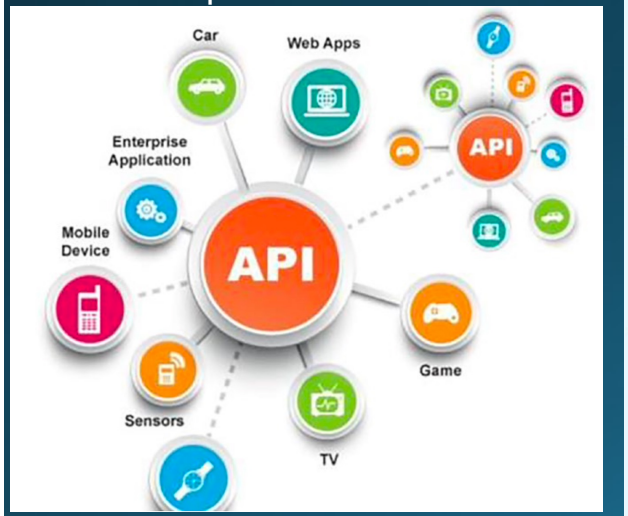
- HTTP is the communication protocol that underpins the internet, and it allows us to exchange different kinds of media types, such as text, images, video, and JSON, over a network.
- HTTP uses the concept of a Uniform Resource Locator (i.e., URL) to locate resources on the internet, and it has features that can be leveraged by API technologies to enhance the interaction with the server, such as request methods (e.g., GET, POST, PUT) and HTTP headers.
- Web APIs are implemented using technologies such as SOAP, REST, GraphQL, gRPC, and so on.



Example: Web API and API-Connected Enterprise

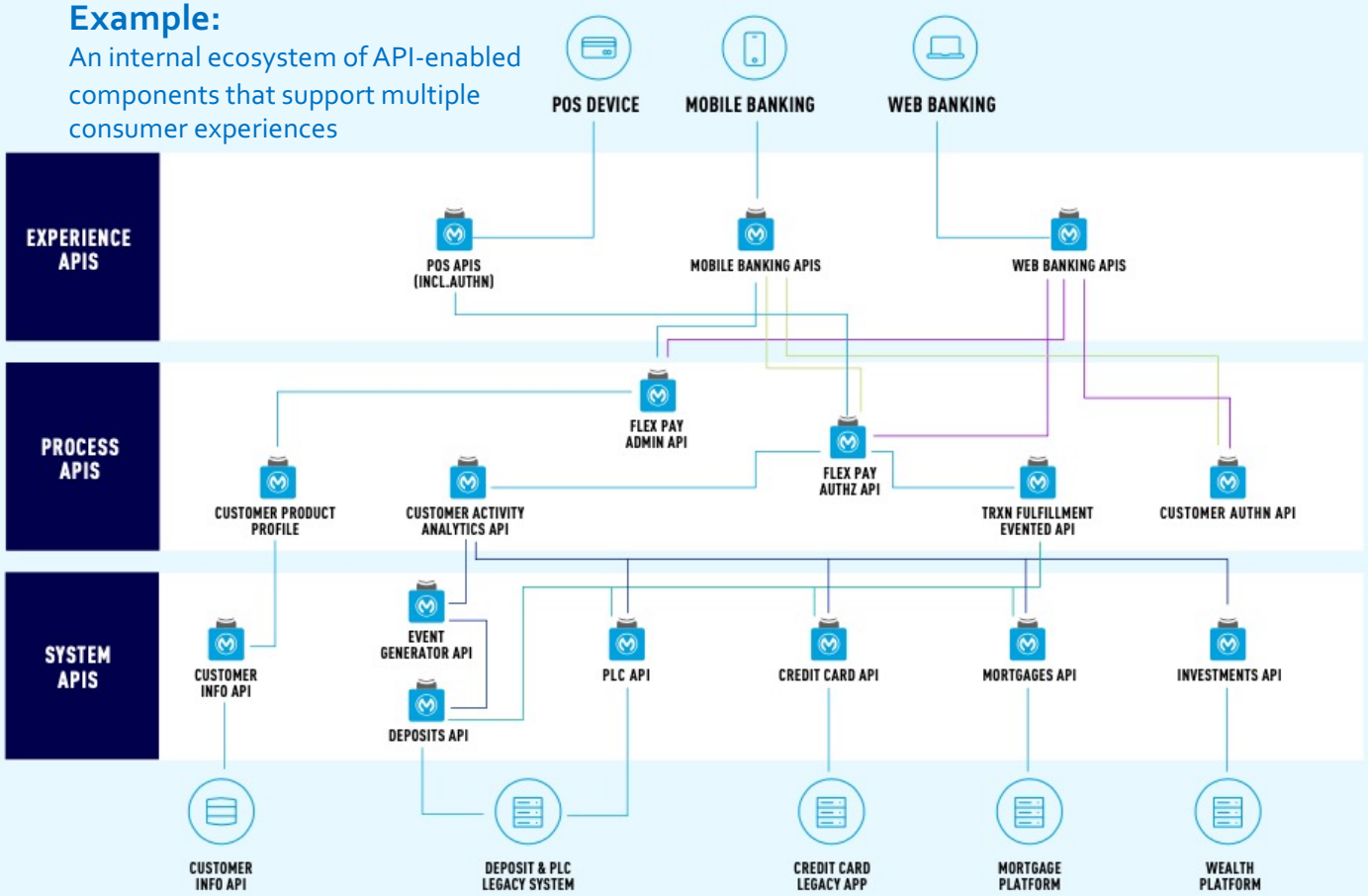


Anatomy of a RESTful Request and Response over HTTP

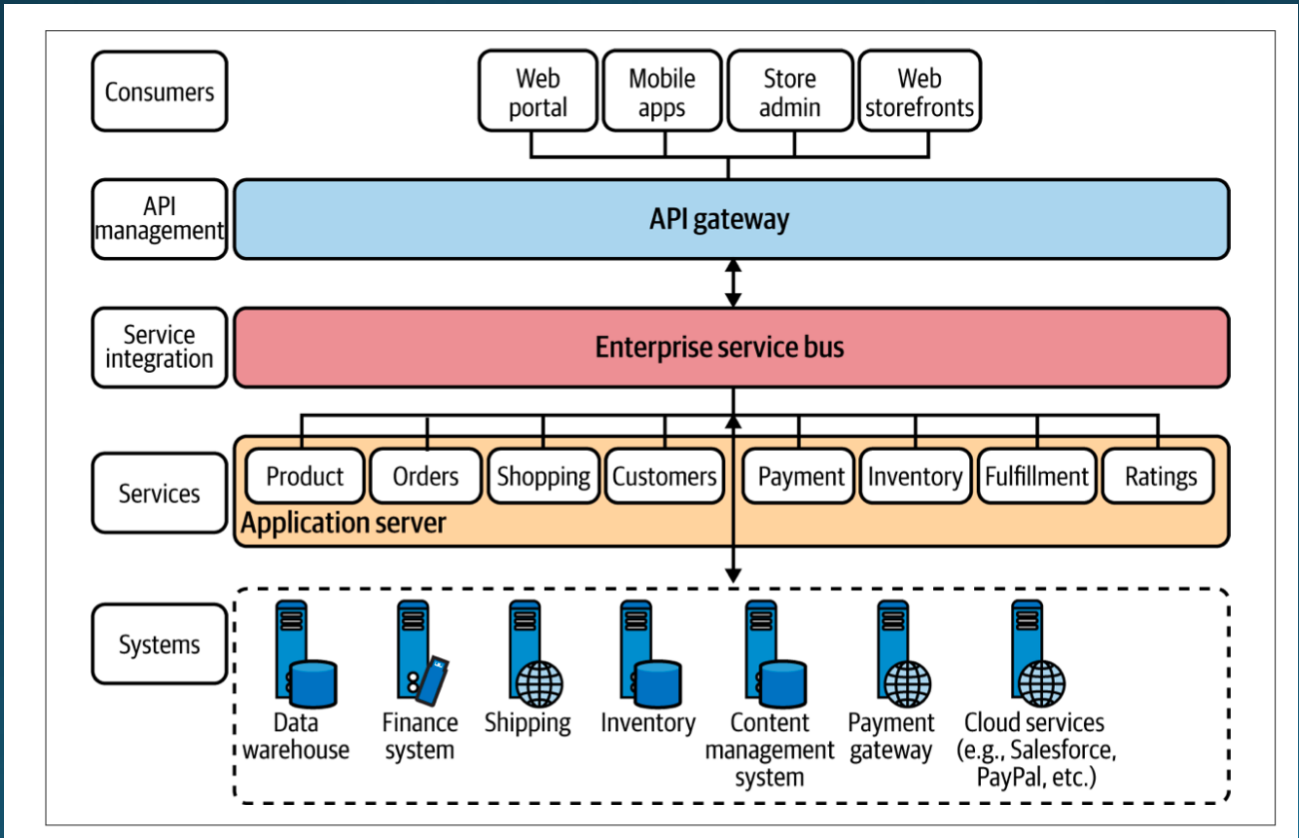


Example:

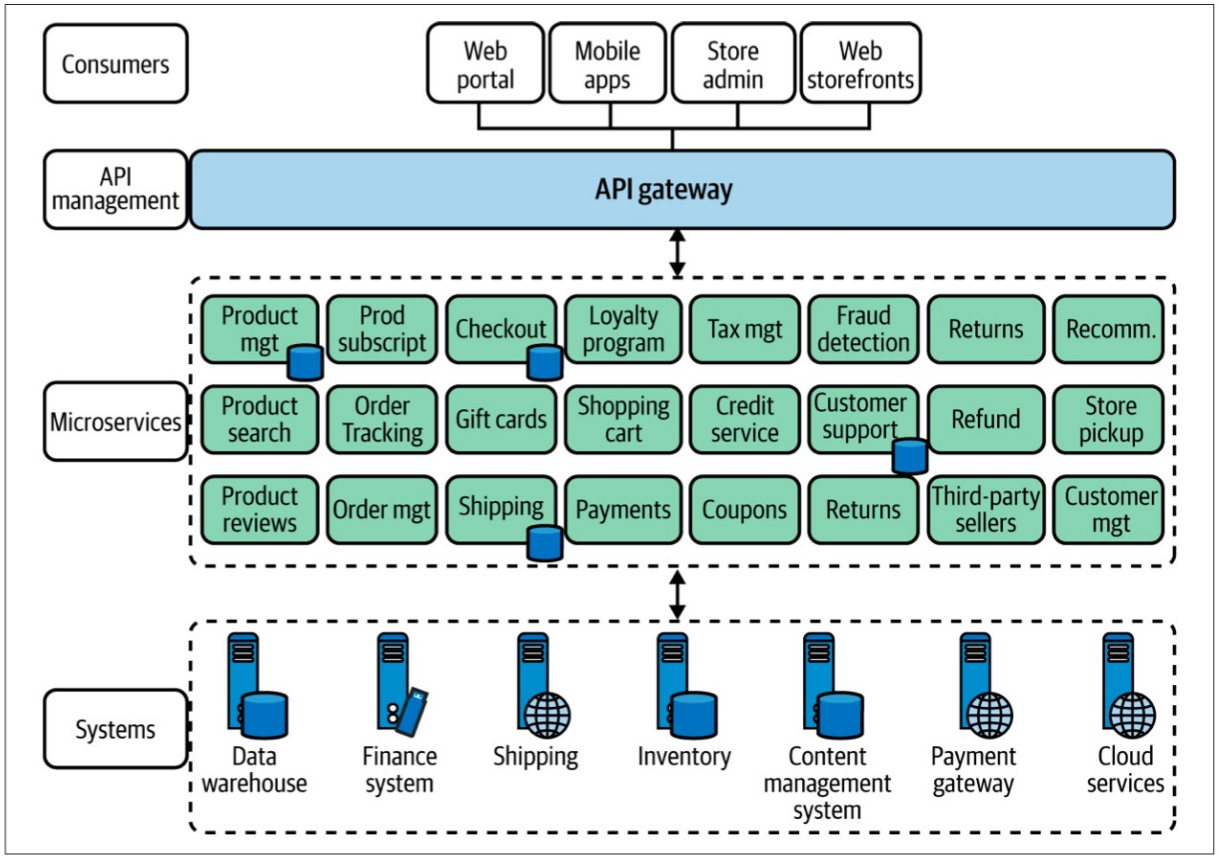
An internal ecosystem of API-enabled components that support multiple consumer experiences



Example: An online retail application scenario built using an SOA/ESB with API management



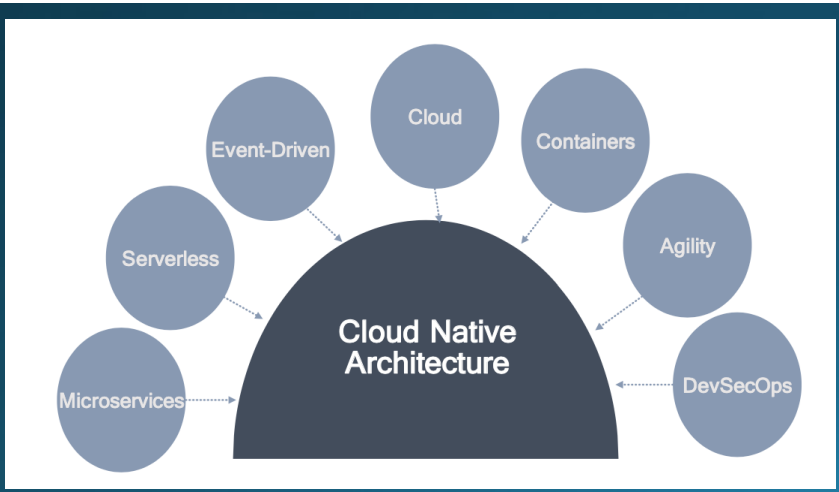
Example: An online retail application built using microservices architecture



Cloud Native and Microservices

Cloud Native Definition from CNCF

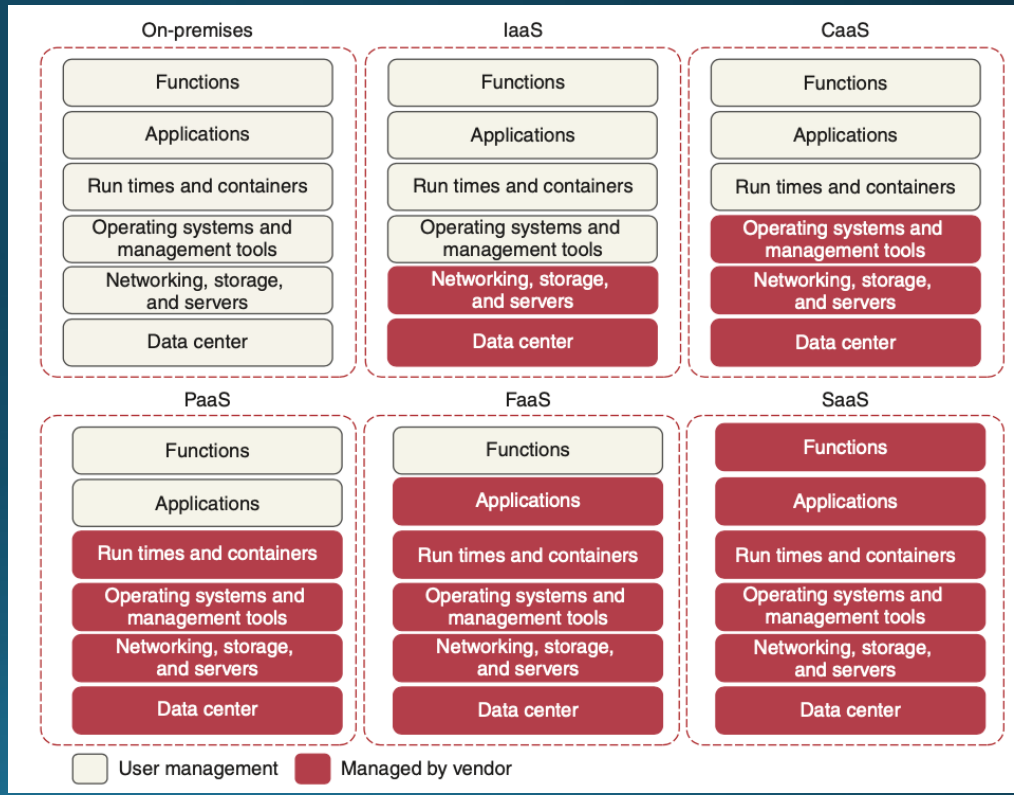
Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach. These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil.



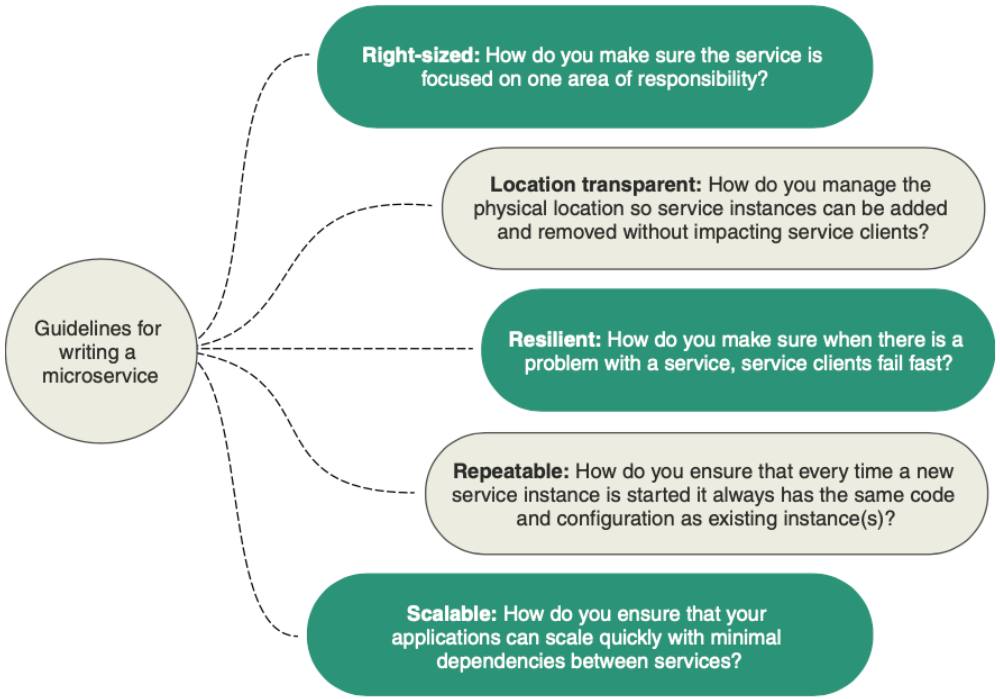
- A cloud native application is designed as **a collection of loosely coupled and independent services** that are serving a well-defined business capability.
  - These are known as *microservices*. Microservices are the foundational architectural principle that is essential to building cloud native applications.
  - It's virtually impossible to build a proper cloud native application without knowing the basics of microservices architecture.
- *Microservices architecture* is a style of building software applications.
  - Before the advent of microservices architecture, we used to build software applications as monolithic applications catering to various complex business scenarios.
  - These monolithic applications are inherently complex, hard to scale, expensive to maintain, and hinder the agility of development teams.
  - Monolithic applications communicate with one another by using proprietary communication protocols and often share a single database.

Cloud and Microservices

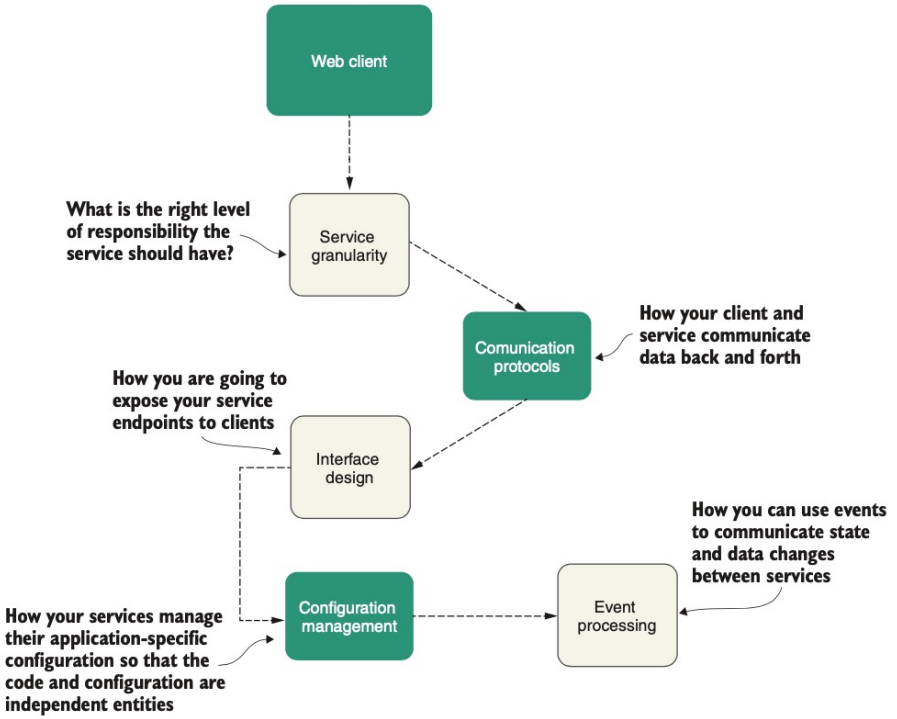
- Cloud computing is the delivery of computing and virtualized IT services—databases, networking, software, servers, analytics, and more—through the internet to provide a flexible, secure, and easy-to-use environment.
- The cloud computing models let the user choose the level of control over the information and services that these provide.
  - These models are known by their acronyms, and are generically referred to as *XaaS*—an acronym that means *anything as a service*.
- When writing a microservice, sooner or later you're going to have to decide whether your service is going to be deployed to one of the following:
  - Physical Server, Virtual machine images, or Virtual Container…
  - Example: deploy your microservices to CaaS-based cloud provider



Microservices are more than writing the code

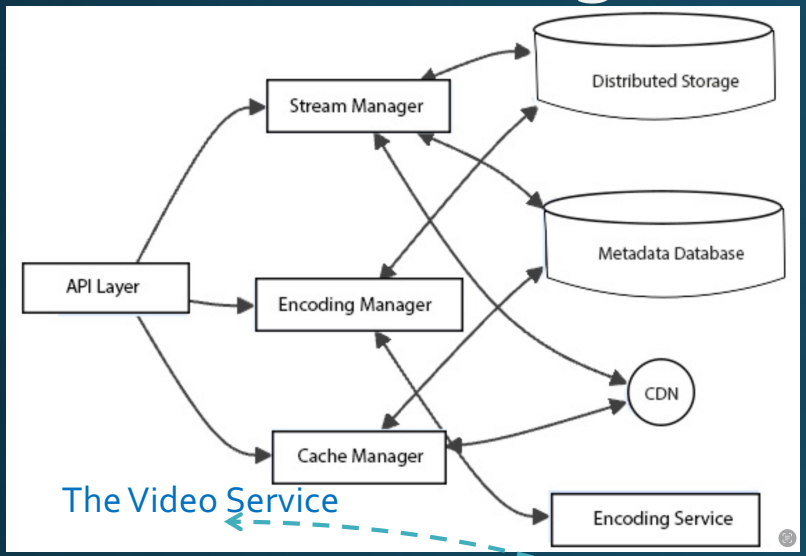


Microservices are more than the business logic

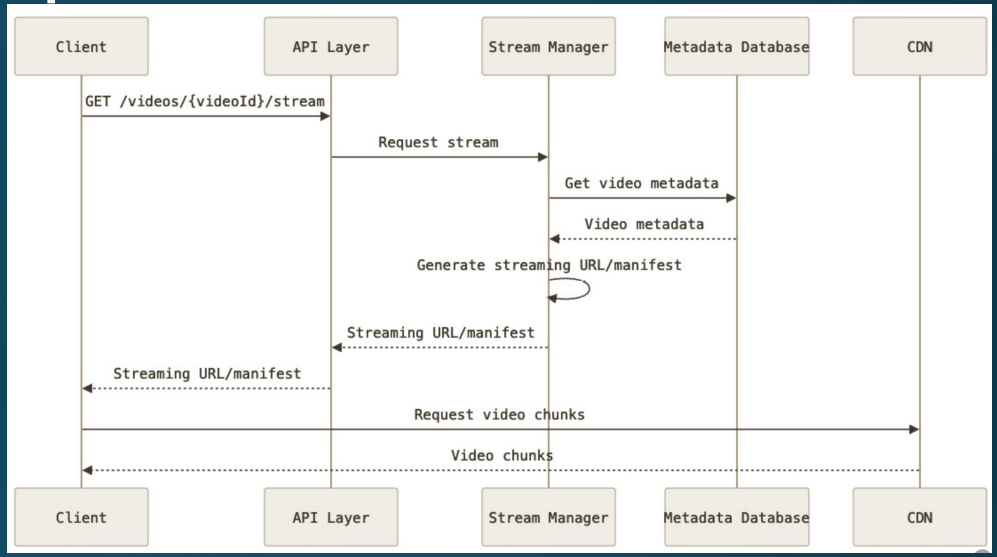


When designing the microservices, you need to think about how the services will be consumed and communicated with

Service Design Example: A Video Service



The Video Service

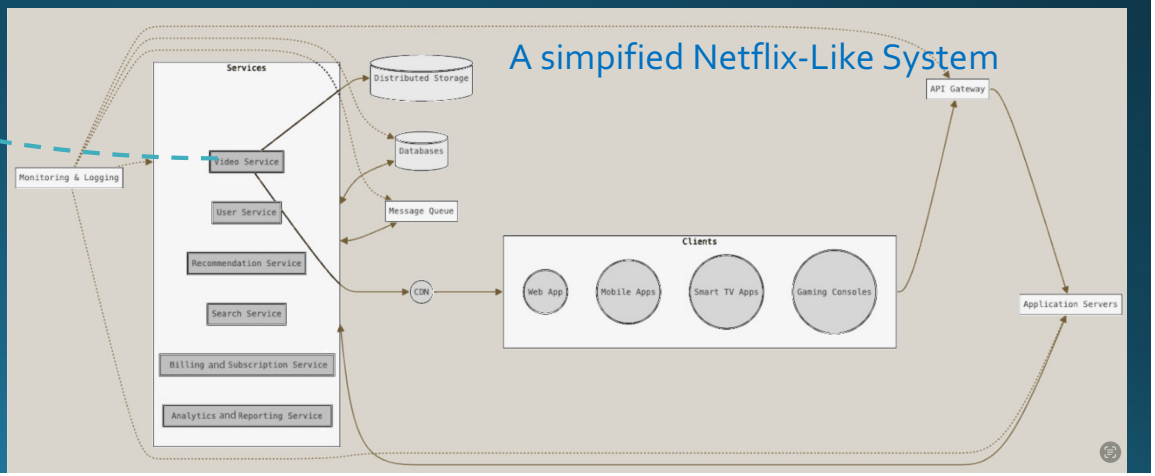


POST /videos: Upload a new video file
Request body: The video file and metadata (title, description, duration, etc.).
Response: The video ID and status.

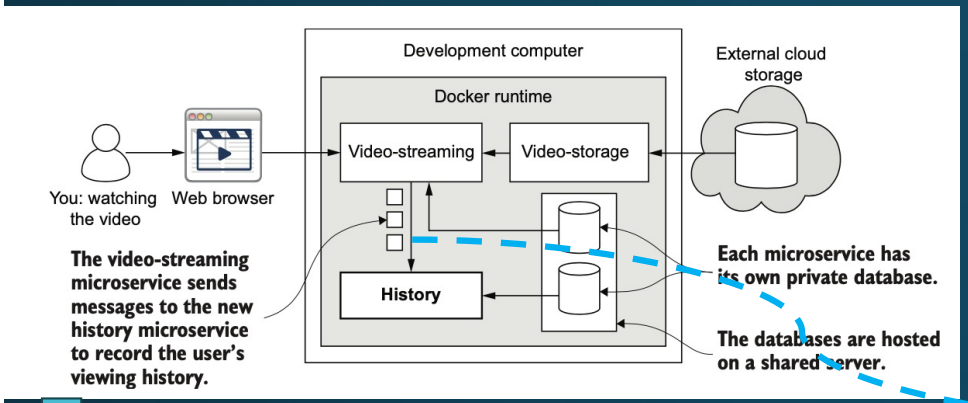
GET /videos/{videoId}: Retrieve video metadata
Response: Video metadata (title, description, duration, etc.).

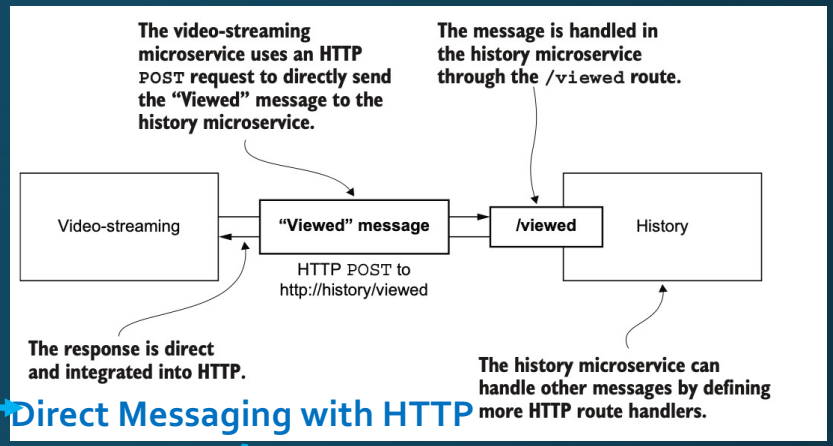
GET /videos/{videoId}/stream: Stream the video content
Request parameters: Video ID, quality/bitrate, offset.
Response: Video chunk or segment."

A simpified Netflix-Like System



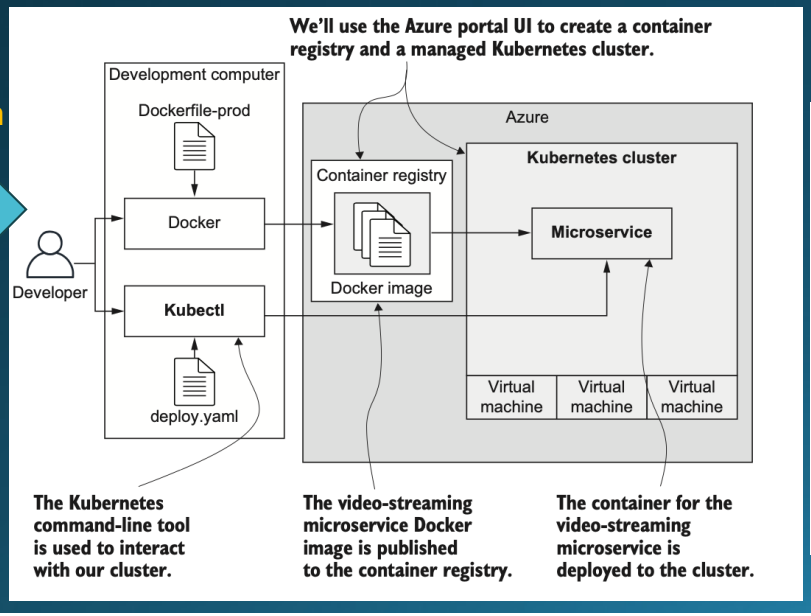
Example: Microservices Running in Production



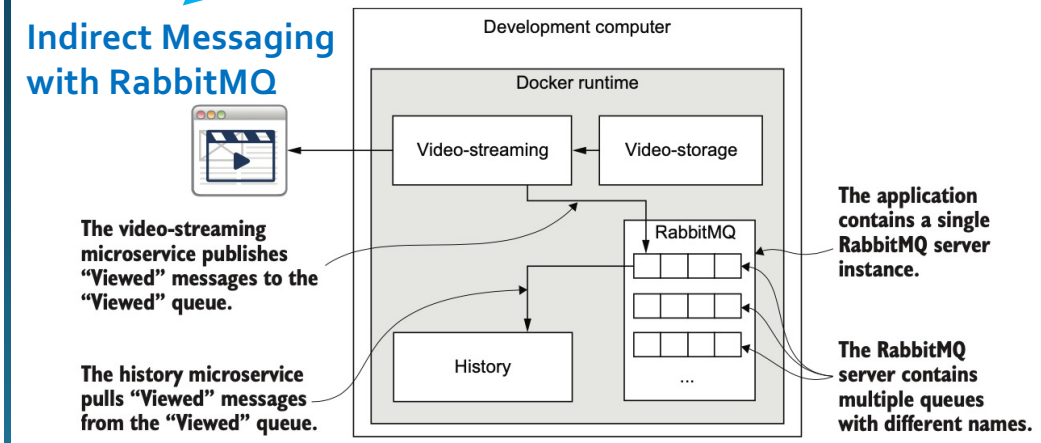


Direct Messaging with HTTP

To Production



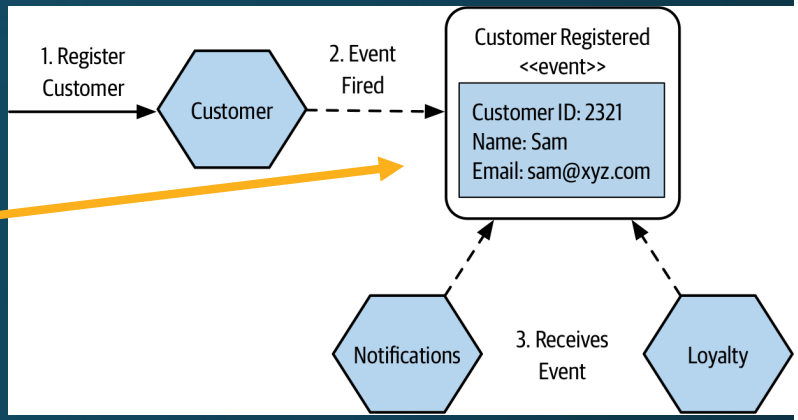
Indirect Messaging with RabbitMQ



Communication Styles

An event with more information in it can allow receiving microservices to act without requiring further calls to the source of the events

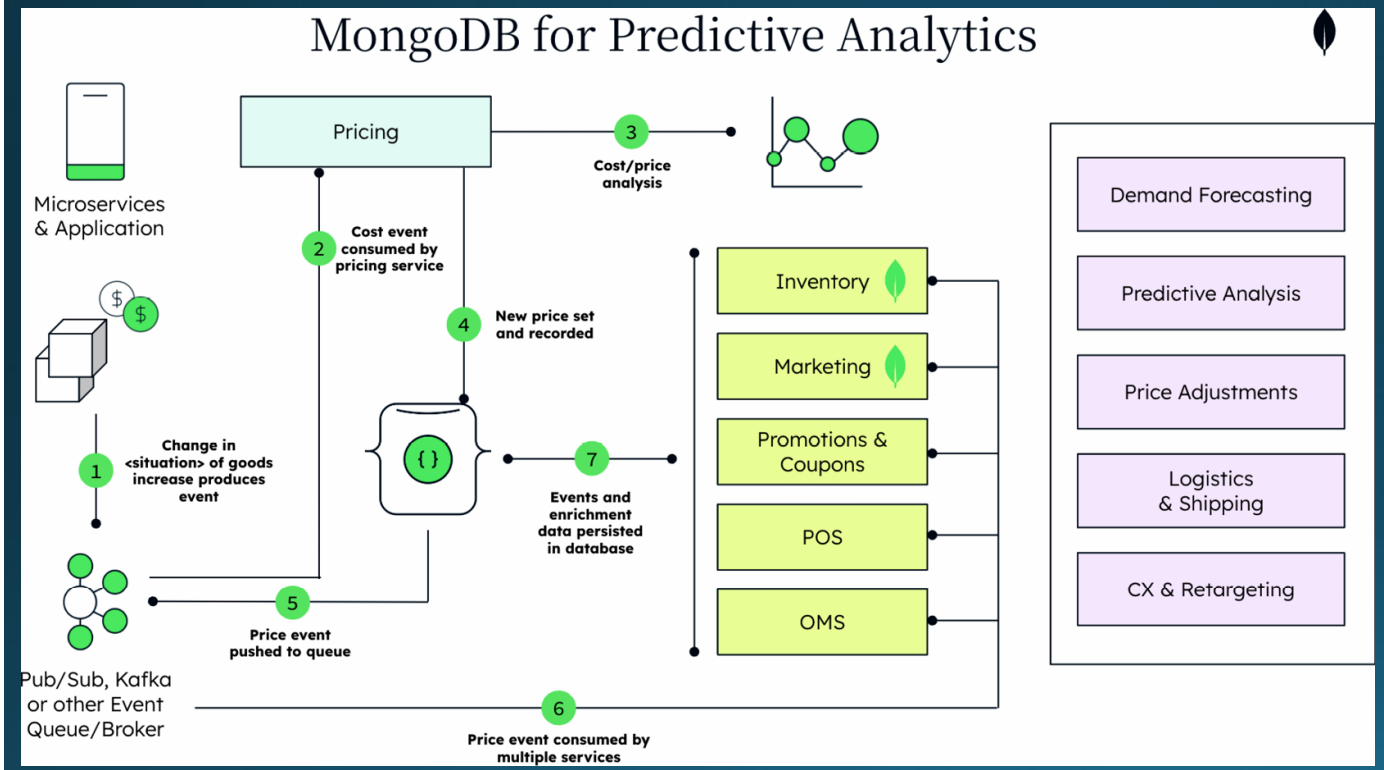
Different styles of inter-microservice communication along with example implementing technologies



	Synchronous Blocking		Asynchronous Non-Blocking	
Collaboration Style	Request-Response		Event-Driven	Common Data
Common Implementation Choices	REST over HTTP RPC	Queue-based brokers RPC	Topic-based brokers REST over HTTP (Atom)	Database File system

Asynchronous events aid in decoupling services from each other and increase overall system availability, but they also encourage service authors to think in terms of **eventual consistency**

Example:
Microservices Architecture for Predictive Analytics



An illustration of a price-change scenario where fuel costs have risen, which leads to a rise in shipping costs and, in turn, pricing

[1] This produces events about the cost increase and places them in the message stream, where the event queue makes them available. All microservices are listening for such messages.

[2–4] The pricing microservice consumes the event, analyzes it against existing data, and further conveys the new pricing into the message stream.

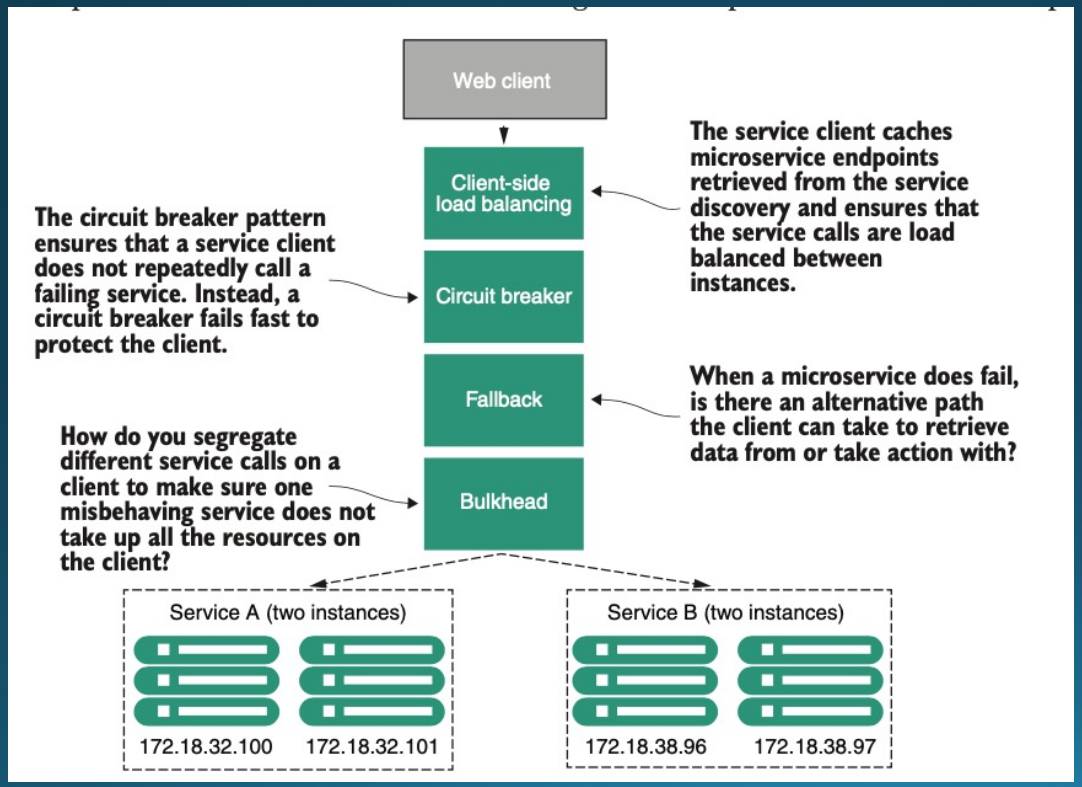
[5–6] The database pushes those messages to the event queue, which makes them available to all consumers listening for messages. Microservices directly impacted by pricing changes, such as those that manage inventory, marketing, promotions, coupons, **point of sale** (**POS**), and the e-commerce provider's **order management system** (**OMS**), consume the price change events and update their individual databases accordingly.

[7] The centralized database aggregates and persists events, enriches event streams with data from other sources, including historical data, and provides a central repository for multiple event streams.

Examples of Patterns for Microservices (1)

Microservice client resiliency:

Cleint-side load balancing, Circuit breaker pattern, Fallback pattern, Bulkhead pattern…

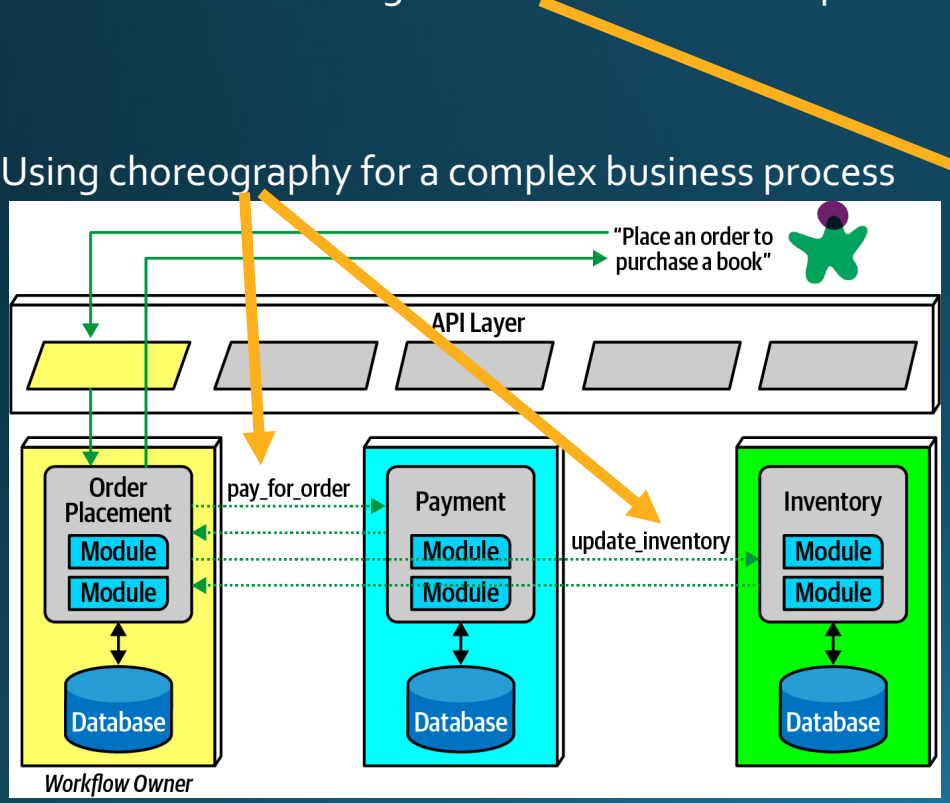


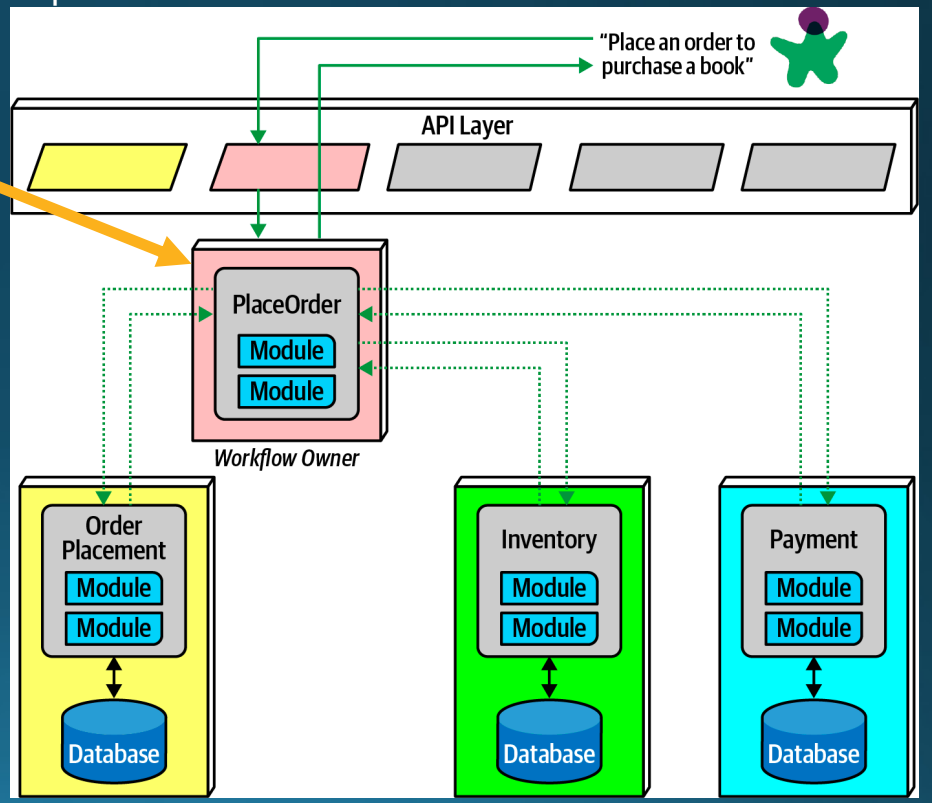
Examples of Patterns for Microservices (2)

Service Composition: Choreography and Orchestration

Using orchestration for a complex business process

Using choreography for a complex business process

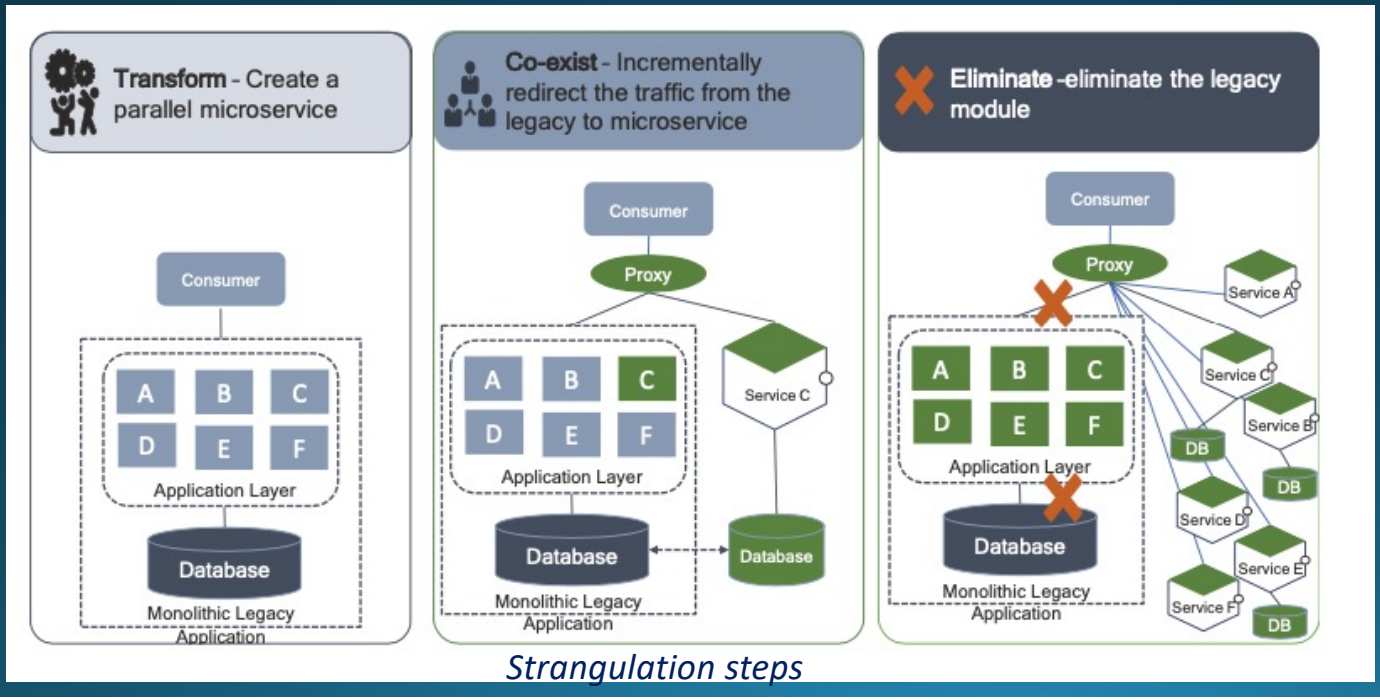




Examples of Patterns for Microservices (3)

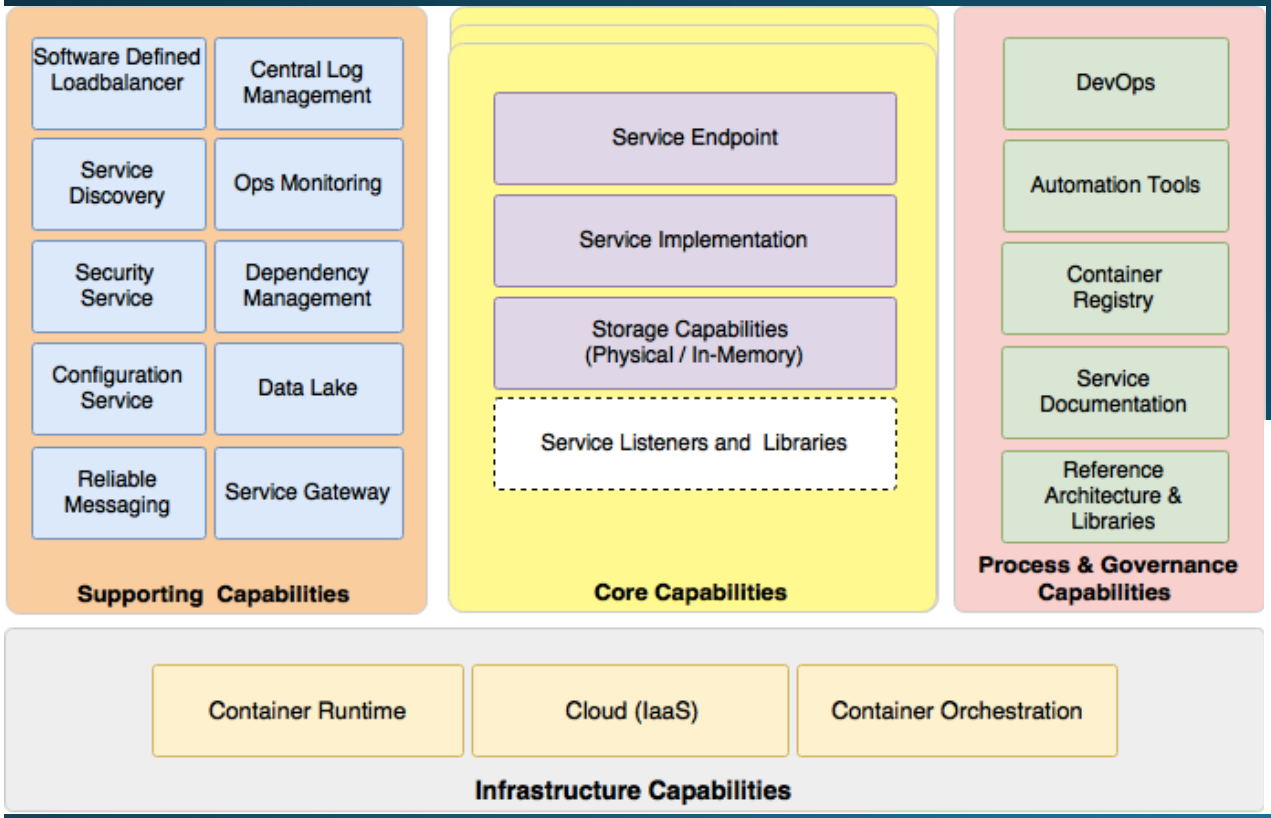
Strangler Pattern

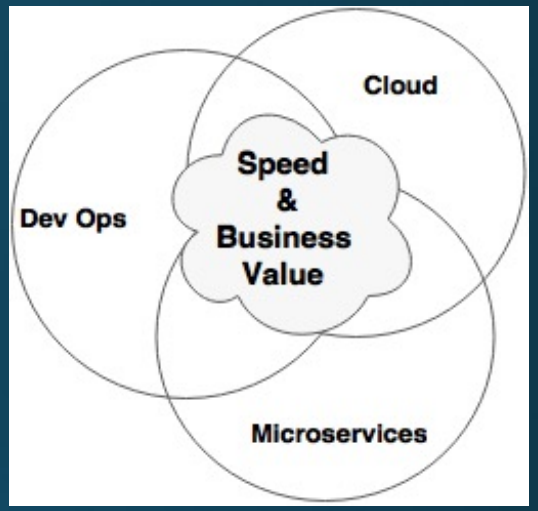
- You cannot apply the big-bang approach when decoupling legacy monolithic applications to microservices; it must be done incrementally.

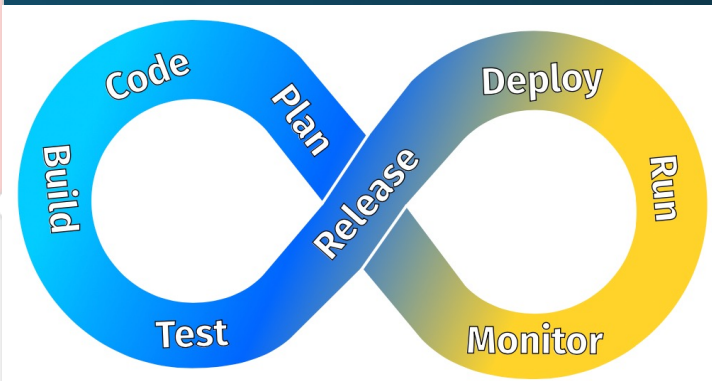


Strangulation steps

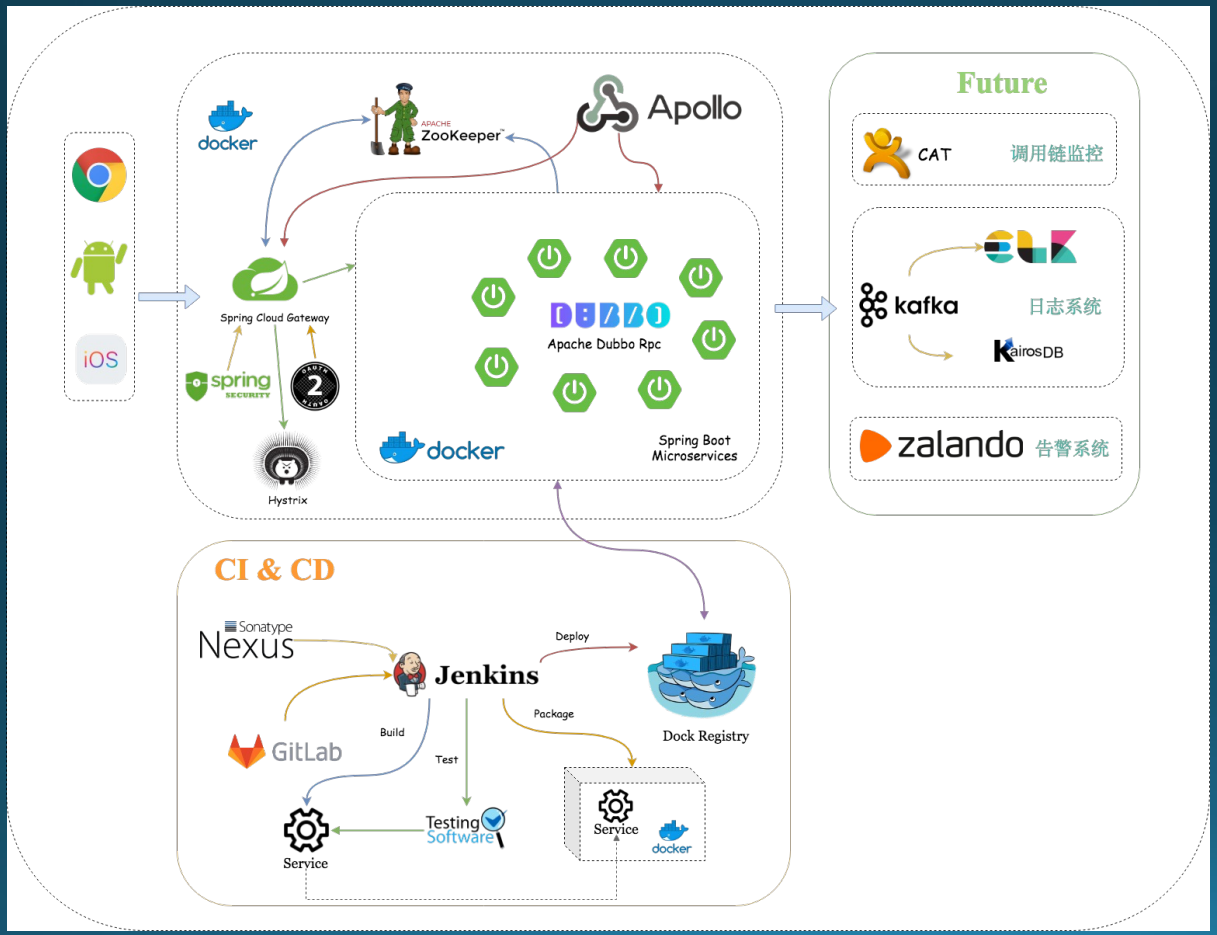
Enabling Technology



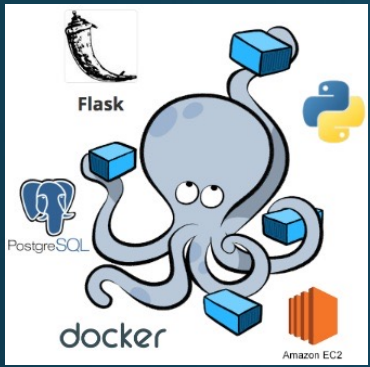




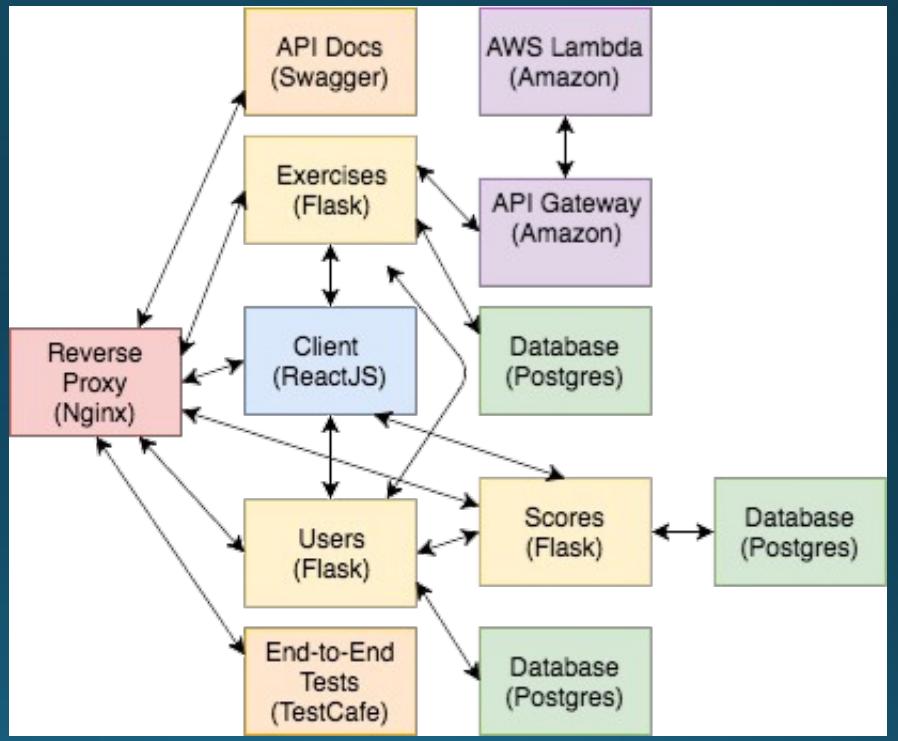
Technology Stack: Example 1



Technology Stack: Example 2

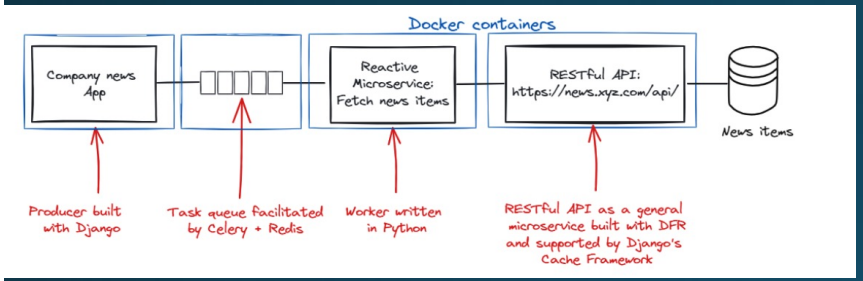


Python v3.6.4
Flask v0.12.2
Docker v17.12.0-ce
Docker Compose v1.18.0
Docker Machine v0.13.0
Docker Compose file v3.4
Flask-SQLAlchemy v2.3.2
psycopg2 v2.7.3.2
Flask-Testing v0.6.2
Gunicorn v19.7.1
Nginx v1.13.8
Bootstrap 4.0.0

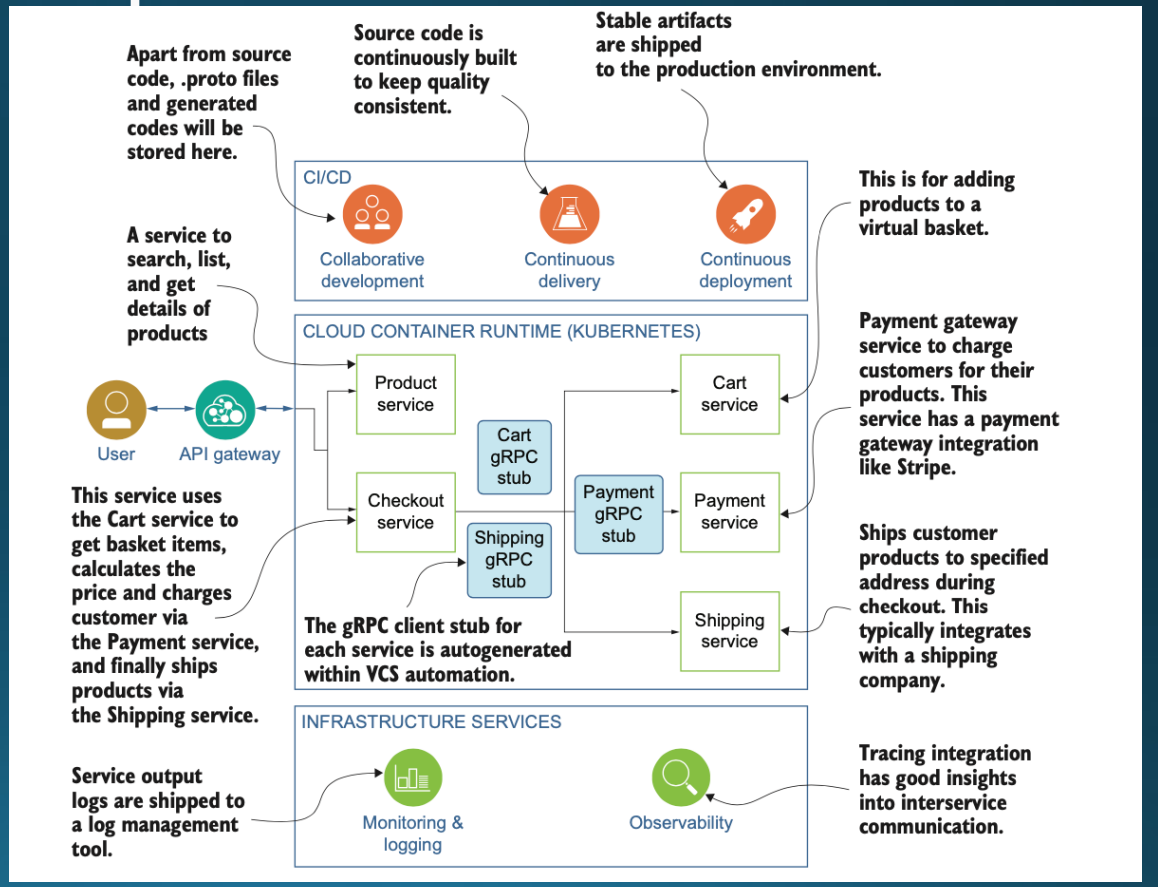


Microservices solution of a code evaluation tool for grading code exercises

More Technology Options…



Simple Microservices with Django Framework



Architecture diagram of an e-commerce product built with Go microservices on top of Kubernetes, including CI/CD flow and observability

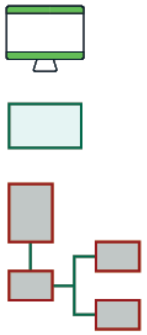
What is happening now: towards the Agentic System

LEGACY

UI

App

Data



Monolithic relational

Slow and expensive
(Rigid schema, Slow to adopt and limited scalability)

MICROSERVICES

Multi-channel Applications

APIs & Events

Domain Datastores

Domain Datastores

Faster development cycles
(JSON (in-motion & at-rest), APIs & Event, Domain schemas)

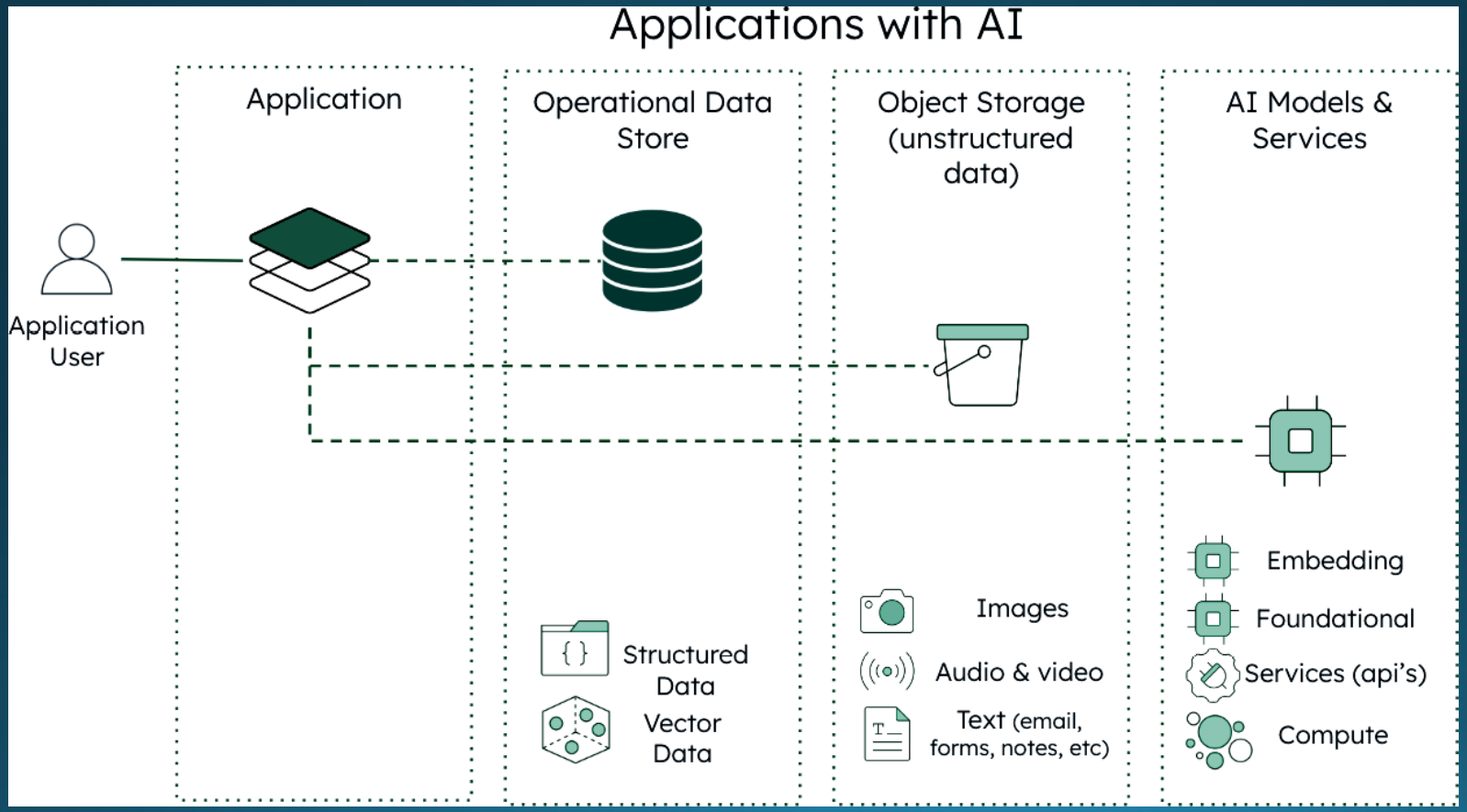
AI

Smart Business Apps

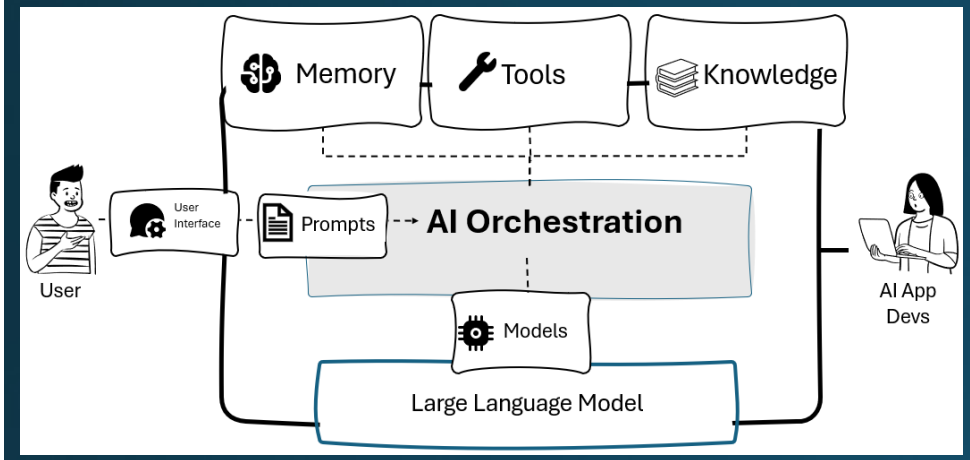
Agent Workers

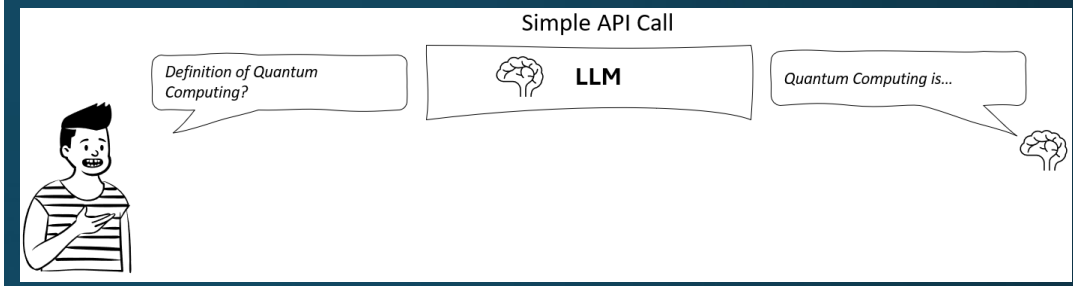
How do we architect data for agentic AI?

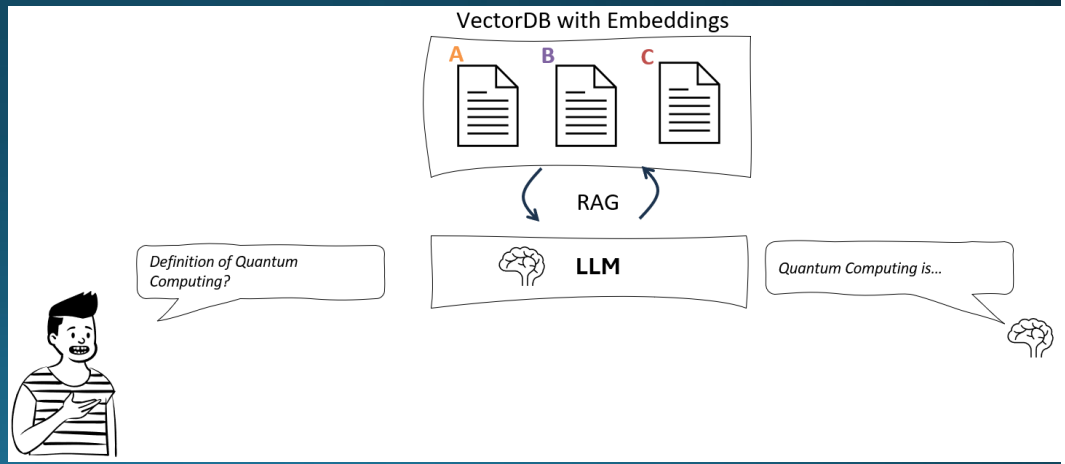
Example: AI-Enabled Application Architecture with Integrated Data and Model Services



From Services Orchestration to AI Orchestration





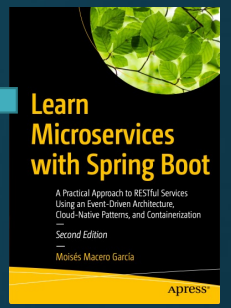


More technical needs for an open, agentic ecosytem:

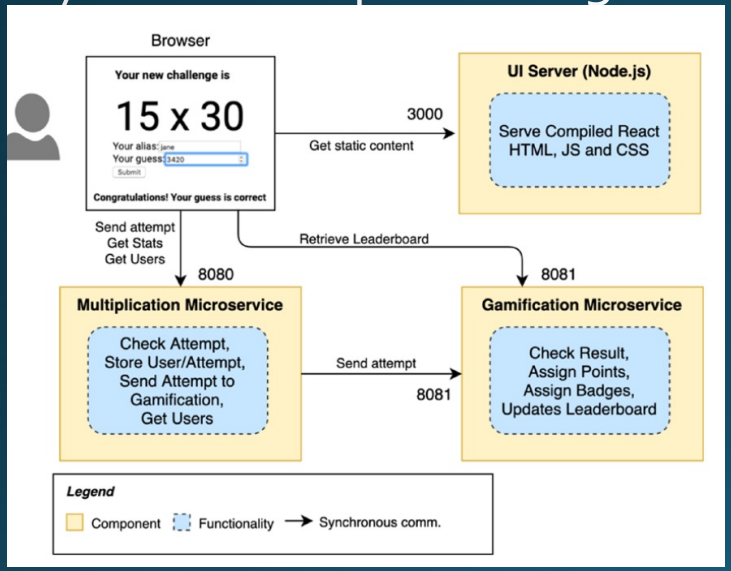
- Memory and Context Mangement
- Tools and External Integration
- Next-Gen Agent Protocols:
  - Model Context Protocol
  - Agent2Agent (A2A)
  - ...

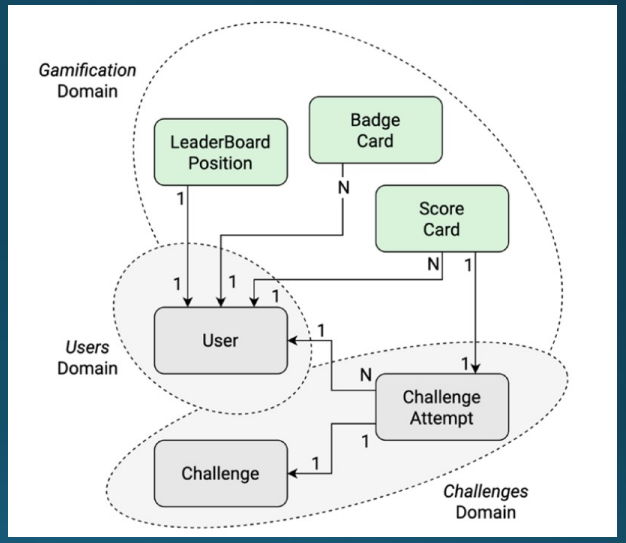
A Simple Demo

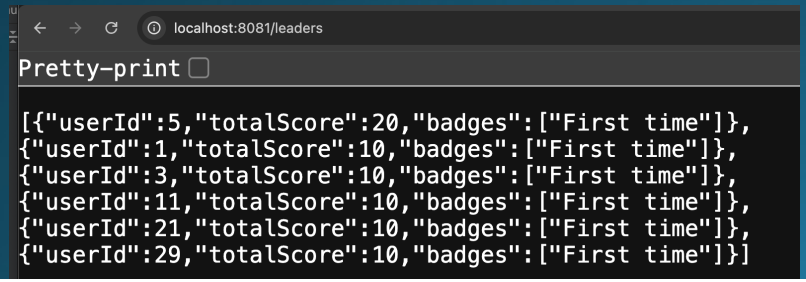
Chapter 6

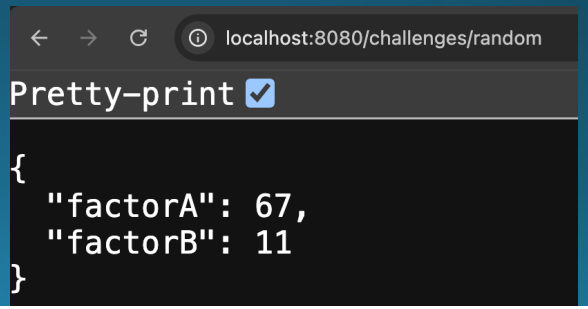


- Building Microservices with a simple approach that uses synchronous processing and *orchestration.*



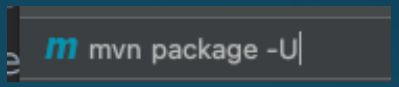






Exercise: explore the demo by yourself…

- Get the example from https://github.com/Book-Microservices-v2/chapter06
- Open the project with IntelliJ IDEA or another popular IDE, fix the environment settings if there is any issues.
- Oberserve the usage of Maven and the setting in pom.xml, modify the versions JDK and spring-boo if needed.
- Update local maven reporsitory if necessary.



- Run the microservices and access the REST API with your browser or Postman/ RestClient
- Use the command line to find out which port is open and learn how to solve port conflict issue.
- Run the front-end application and observe the services integration.

```
lsof -i :8081
```

- Observer the response styles in your page.
- Examine the application.properties file, and try to change the default port of your microservies.
- Learn the typical service implementation structure in Spring Boot.

Further Reading

https://www.martinfowler.com/microservices/