Fundamentals of Microservices

Microservices Architecture

Yan Liu
yanliu.sse@tongji.edu.cn
givemeareason@gmail.com
School of Computer Science, Tongji University

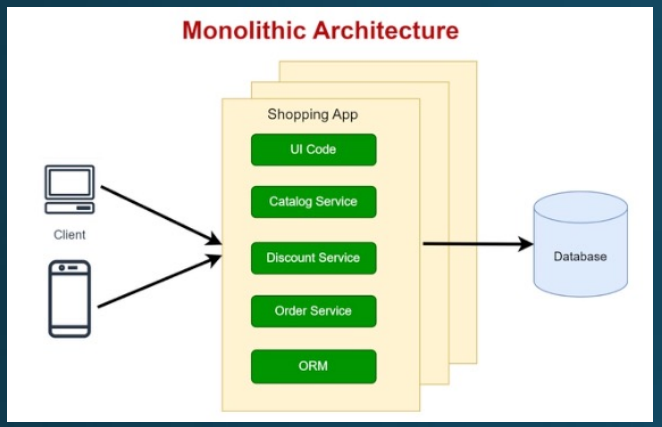
Index

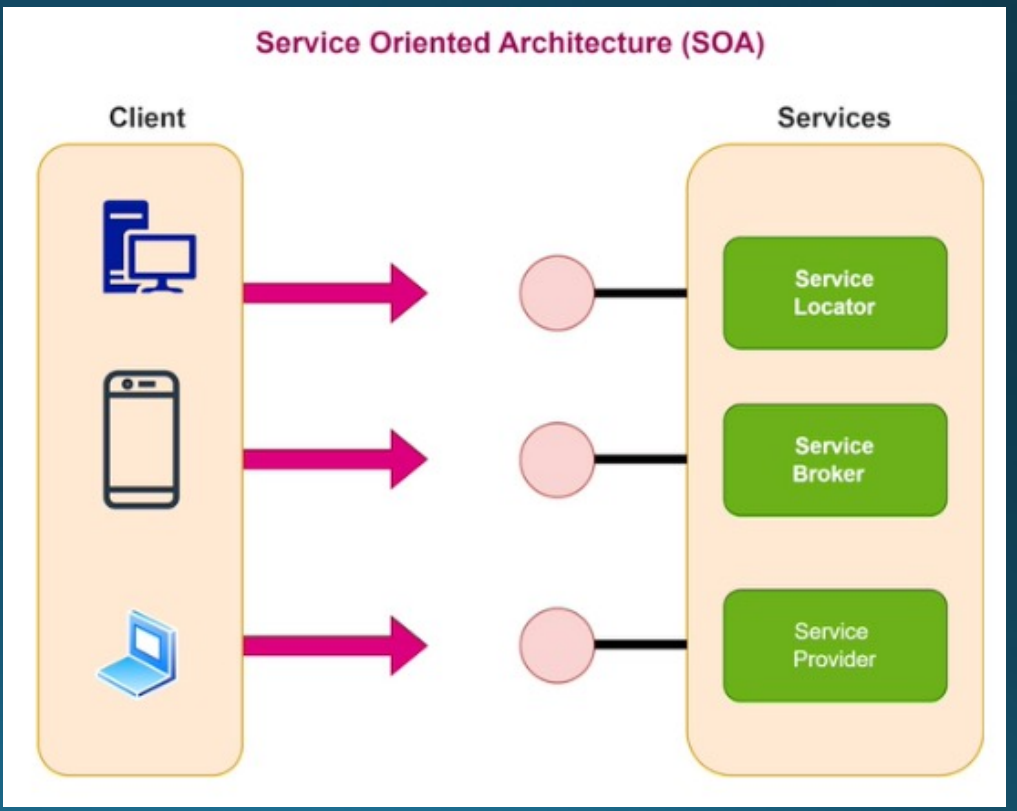
- Evolution of Microservices
- What are Microservices: practical perspectives
- More on Related Architectural Styles
- Core Concepts Involved
- Deciding Boundaries of Microservice
- Understanding Services Composition and API Integration
- Be Aware of the Complexity in Reality

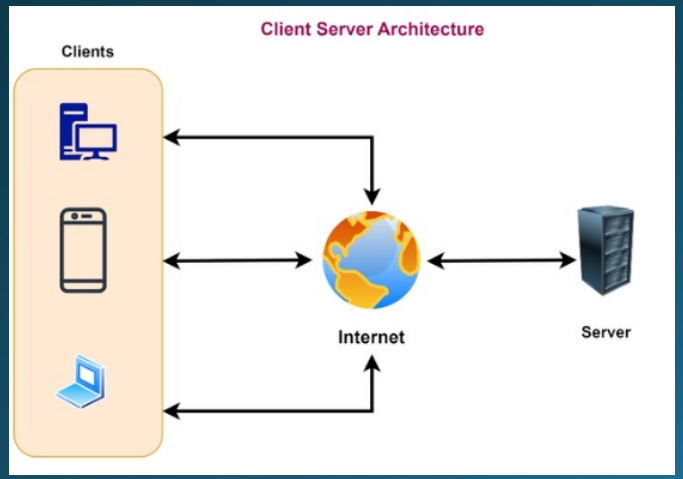
Evolution of Microservices

- **Microservices** is an **architecture style** and an approach for software development to satisfy modern business demands.
- Microservices are not invented, it is more of an **evolution** from the previous architecture styles.
- Microservices is one of the increasingly popular architecture patterns next to **SOA,** complemented by **DevOps** and **Cloud**.
- Its evolution has been greatly influenced by the disruptive digital innovation trends in **modern business** and the **evolution of technology** in the last few years.

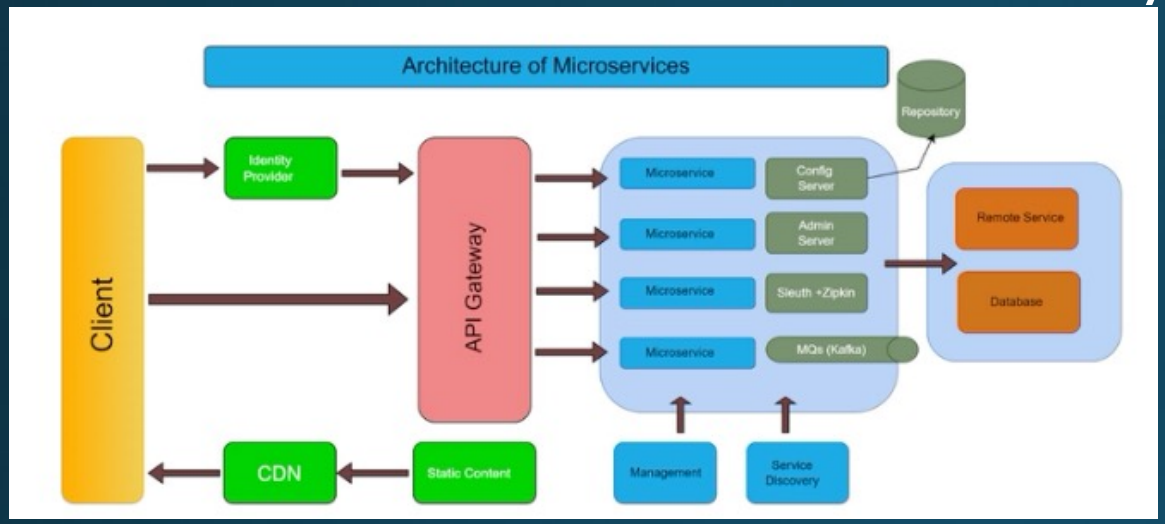
Common Architectural Styles

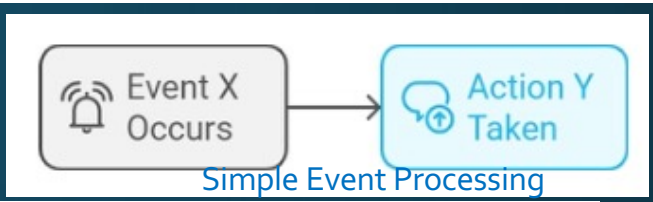




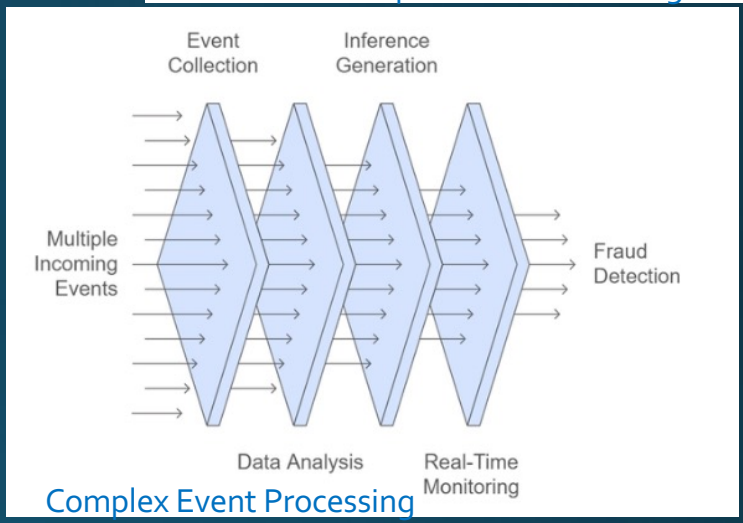


More on Modern Architectural Styles

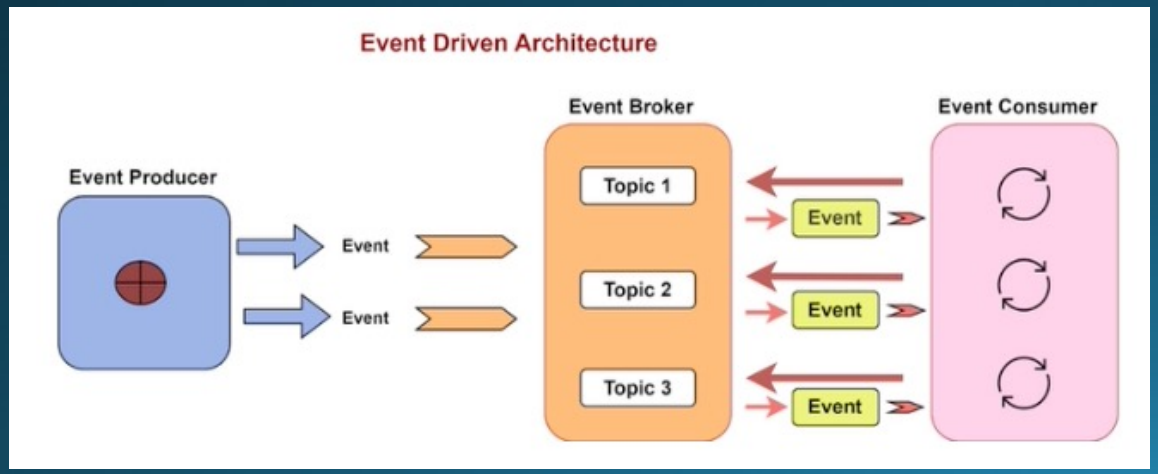


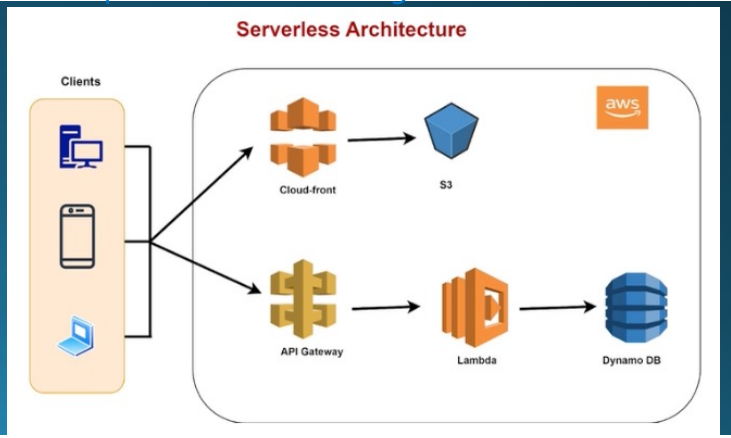


Simple Event Processing



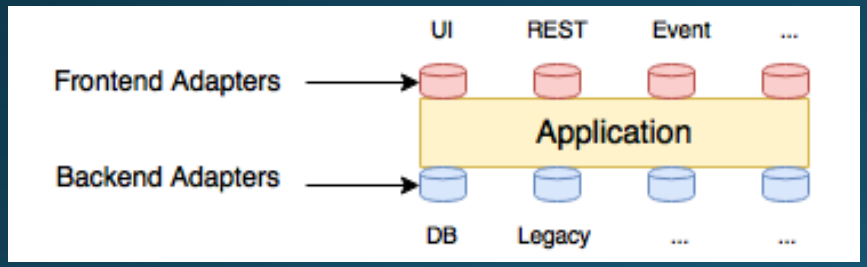
Complex Event Processing

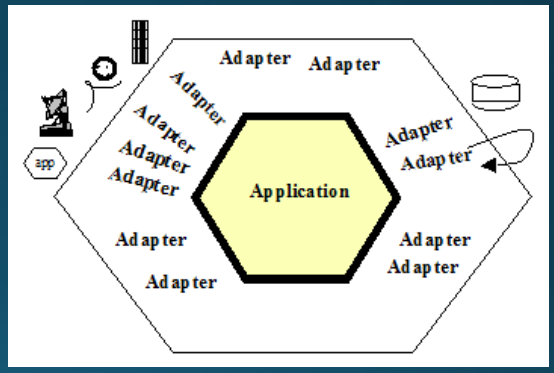




Revisit the Definition: What are Microservices? (1)

- Microservices originated from the idea of **Hexagonal Architecture**, which was coined by Alister Cockburn back in 2005.
  - Hexagonal Architecture, or Hexagonal pattern, is also known as the **Ports and Adapters** pattern.
  - Hexagonal architecture advocates to encapsulate business functions from the rest of the world.
  - http://alistair.cockburn.us/Hexagonal+architecture



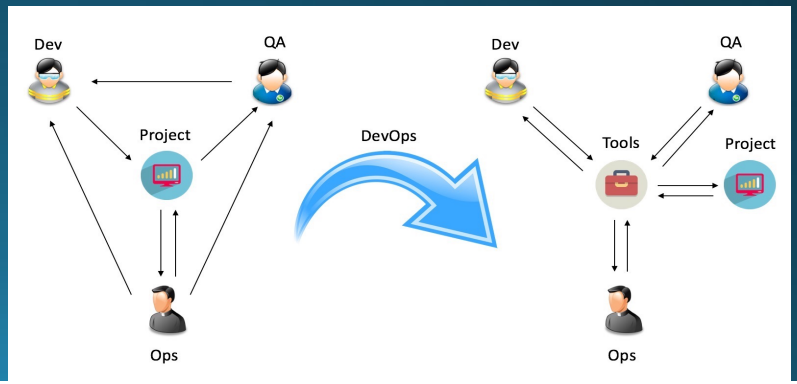


- *Microservices is an architectural style or an approach for building "IT" (digital) systems as a set of **business capabilities** that are autonomous, self contained, and loosely coupled.*
- Microservices became popular as an architectural style after James Lewis and Martin Fowler published their seminal paper on the subject, "Microservices", in 2014.

Revisit the Definition: What are Microservices? (2)

- There is no standard for communication or transport mechanisms for microservices.
  - In general, microservices communicate with each other using widely adopted lightweight protocols, such as HTTP and REST, or messaging protocols, such as **JMS** or **AMQP**.
  - In specific cases, one might choose more optimized communication protocols, such as **Thrift**, **ZeroMQ**, **Protocol Buffers**, or **Avro**.
- DevOps and cloud are two facets of microservices.
  - DevOps is an IT realignment to narrow the gap between traditional IT development and operations for better efficiency.

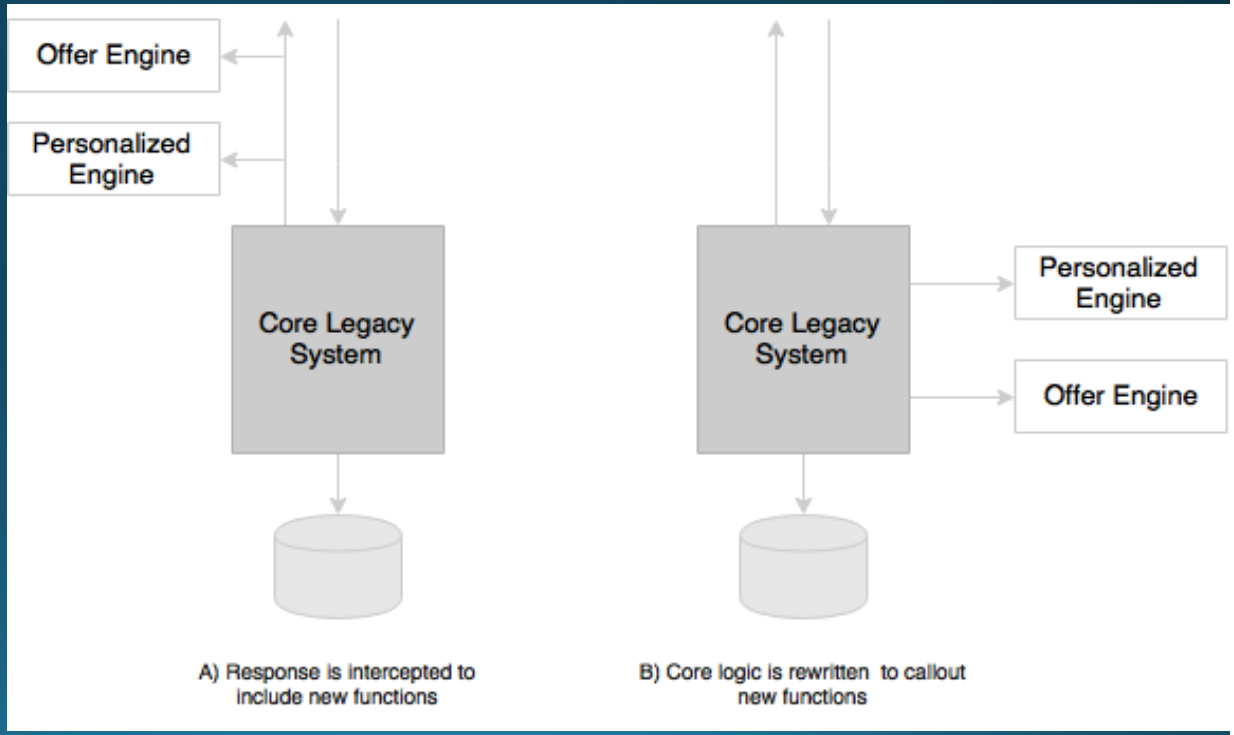
  http://dev2ops.org/2010/02/what-is-devops/



Business demand as a catalyst for Microservices evolution

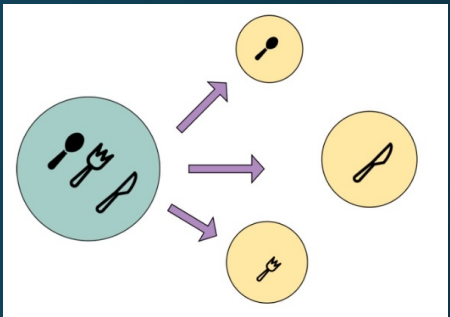
- In this era of digital transformation, Enterprises are primarily using social media, mobile, cloud, big data, and Internet of Things (IoT) as vehicles for achieving the disruptive innovations.
- ***Example***: Innovate sales by:
  - ✓Offering products personalized to a customer based on past shopping preferences;
  - ✓Enlightening customers by offering products based on propensity to buy a product

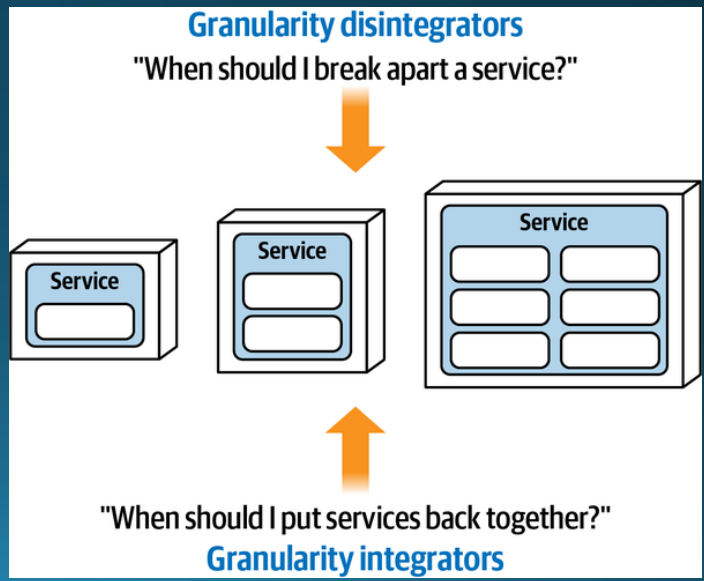
These functions are typically written as microservices, rather than rebuiding the core legacy system



SRP in Microservices

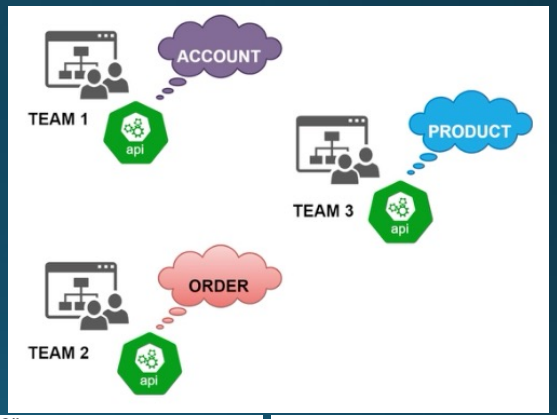
- Single Responsibility Principle
  - each service should be designed to perform only a single business function
- Implementing SRP in Microservices
  - Identifying business capabilities
  - Designing self-contained services (logic, data, and dependencies)
  - Ensuring loose coupling (API, messaging or event-driven systems)
  - Practicing continuous refactoring
  - Leveraging automation and CI/CD
- Challenges and Considerations
  - Balancing granularity
  - Managing data consistency
  - Taming the wild: handling cross-cutting concerns

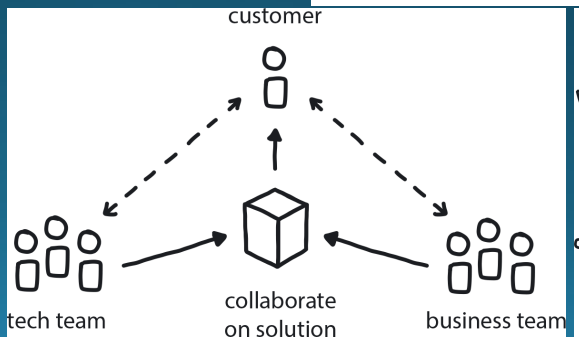


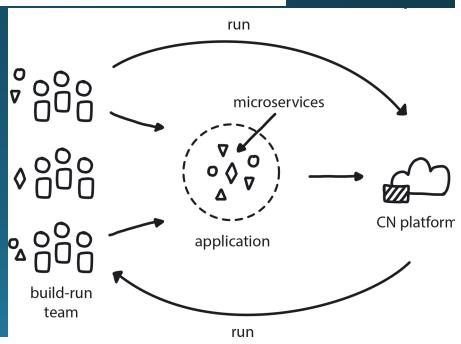


Indenpendency and Autonomy

- Each microservice can function well without having to reply on other parts of the system; Each microservice controls its own data, logic, and makes it's own decision.
- Implementing independence and autonomy in microservices
  - Designing self-contained services
  - Decentralized data management
  - Independent deployment pipelines
  - Embracing asynchronous communication
  - Implementing API gateways and service meshs
  - Adopting DevOps practics
  - Autonomous team
- Challenges and Considerations
  - Balancing independence and consistency
  - Managing cross-cutting concerns
  - Avoiding overhead and complexity
  - Ensuring team collaboration

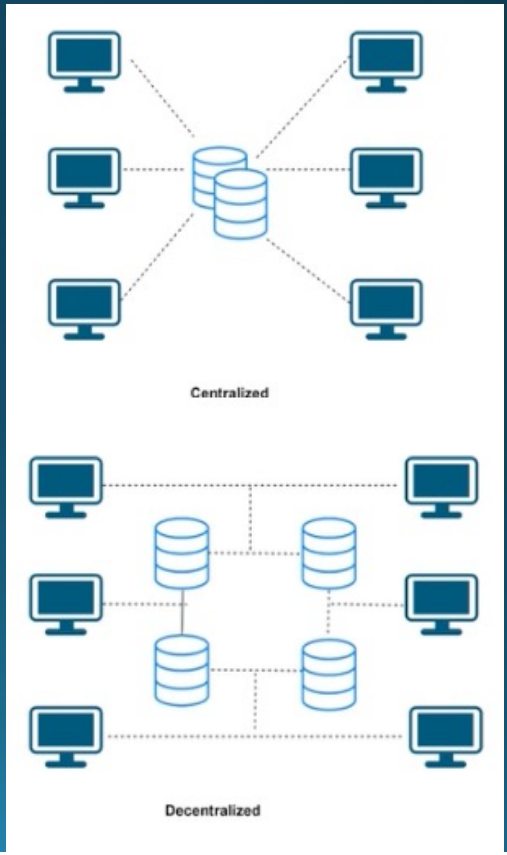






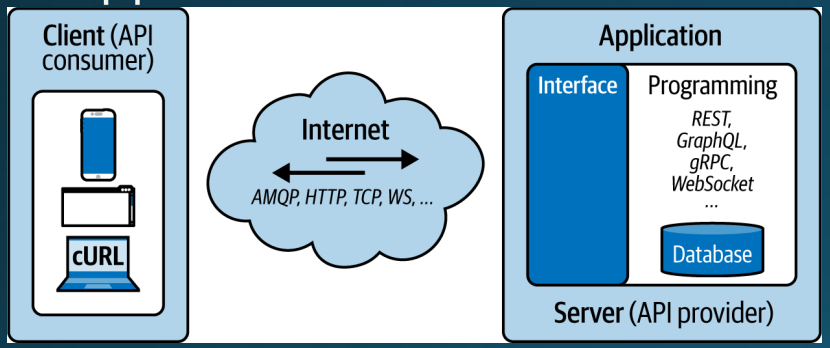
Decentralized Data Management

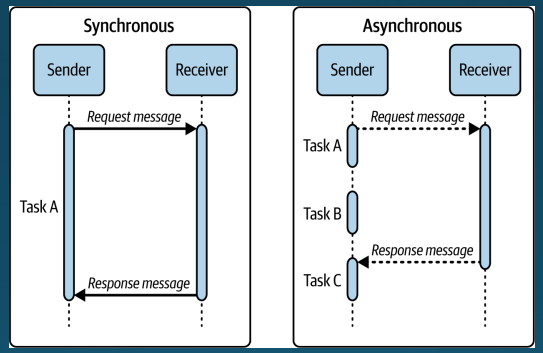
- Decentralized data management isn't just a technical choice; it's a **mindset** shift that unlocks the full potential of microservices.
- Implementing decentralized data management
  - Decoupling data access
  - Managing data consistency
  - Ensuring data integrity
- Best Practices
  - Use APIs for data access
  - Implementing data backups and recovery
  - Monitor and audit data access
  - Embrace polyglot data persistence
  - Regular review and refactor data models
  - Handle cross-service queries
- Challenges and considerations
  - Managing distributed data
  - Addressing data security
  - Balancing consistency and availability
  - Ensuring interoperability

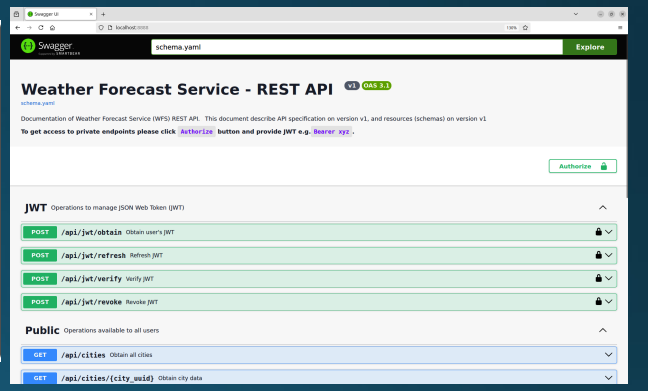


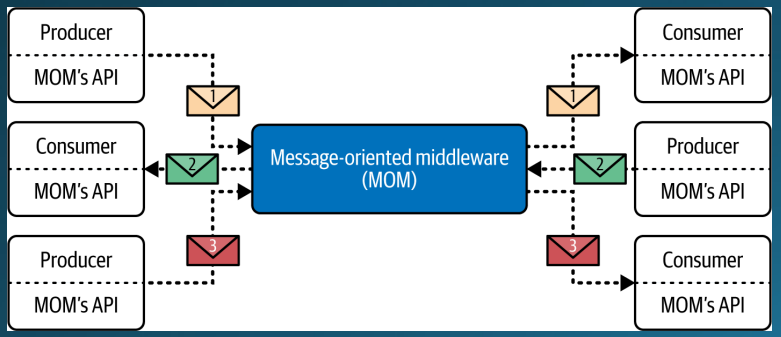
APIs and Communication

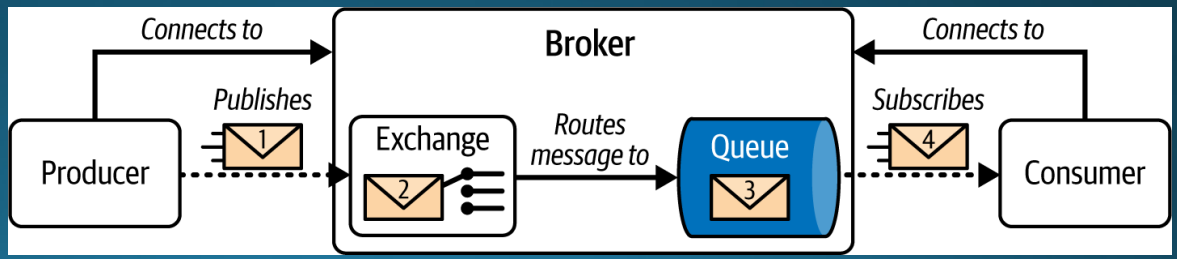
- In the world of microservices, APIs are the "official handshake" through which services expose their functionality, while communication strategies dictate how these services share data and cooperate to form a fully functioning application.





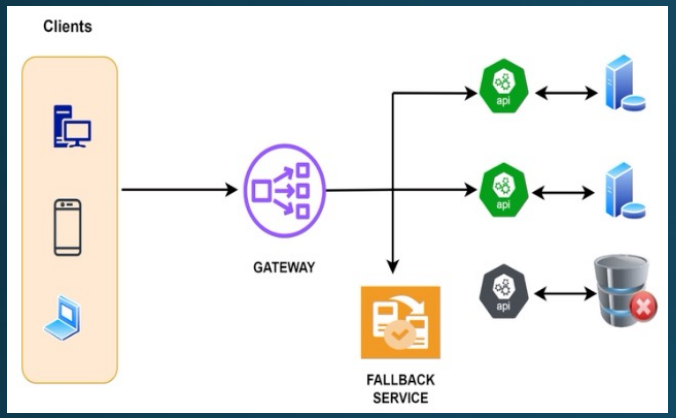




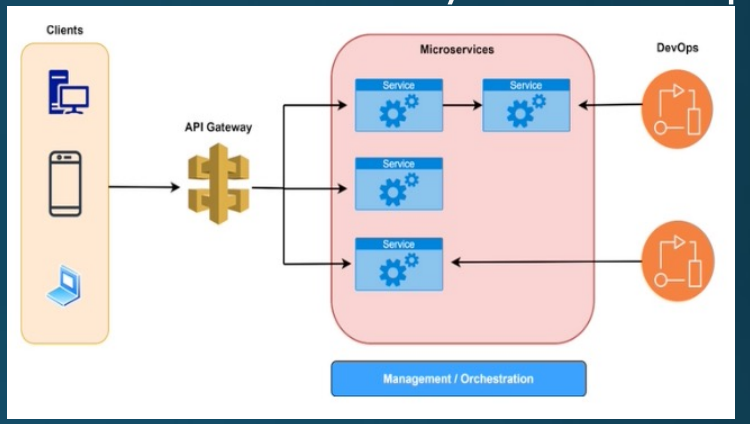


More on the Natures of Microservices

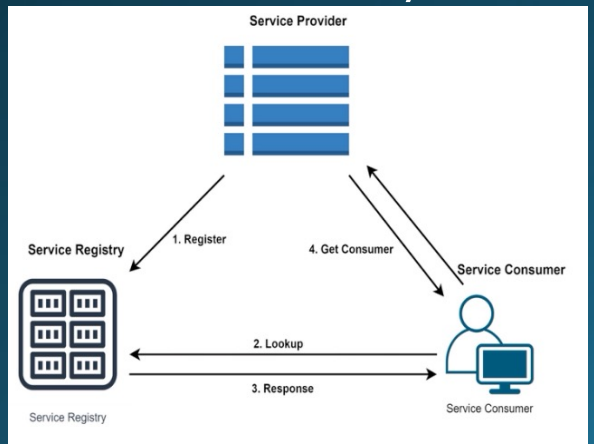
- Resilience and fault isolation



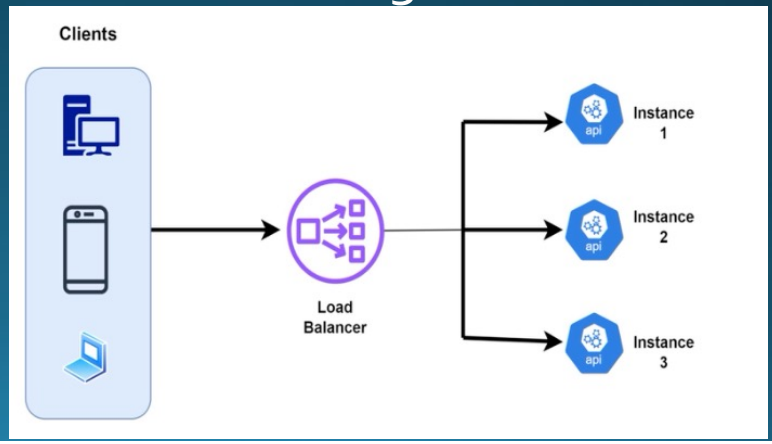
- Continous Delivery and DevOps



- Service Discovery



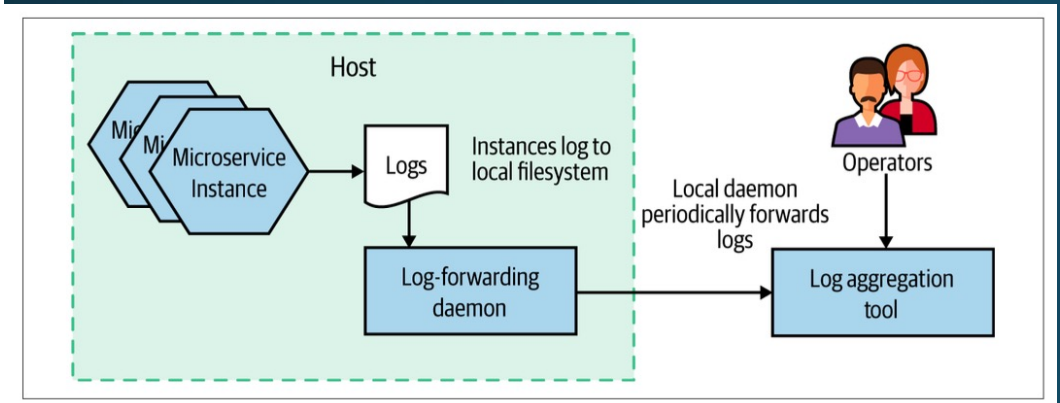
- Load Balancing



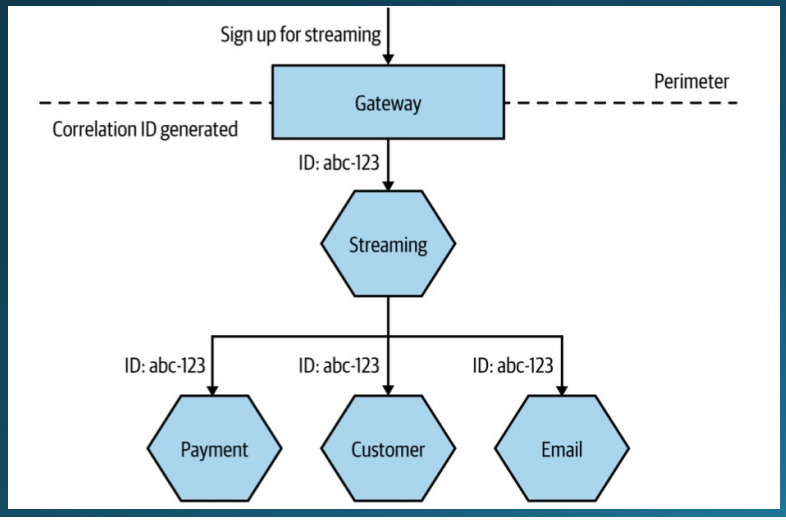
Microservices Enabling Technology

- **Log Aggregation and Distributed Tracing**
  - With the increasing number of processes you are managing, it can be difficult to understand how your system is behaving in a production setting. This can in turn make troubleshooting much more difficult.
  - These systems allow you to collect and aggregate logs from across all your services, providing you a central place from which logs can be analyzed, and even made part of an active alerting mechanism.
- **Containers and Kubernetes**
  - *Containers* provide a lightweight way to provision isolated execution for service instances, resulting in faster spin-up times for new container instances, along with being much more cost effective for many architectures.
  - Container orchestration platforms like Kubernetes allowing you to distribute container instances in such a way as to provide the robustness and throughput your service needs, while allowing you to make efficient use of the underlying machines.
- **Streaming**
  - We need to find ways to share data between microservices.
  - Products that allow for the easy streaming and processing of what can often be large volumes of data have become popular with people using microservice architectures.
- **Public Cloud and Serverless**
  - Public cloud providers offer a host of managed services, from managed database instances or Kubernetes clusters to message brokers or distributed filesystems.
  - By making use of these managed services, you are offloading a large amount of this work to a third party that is arguably better able to deal with these tasks.

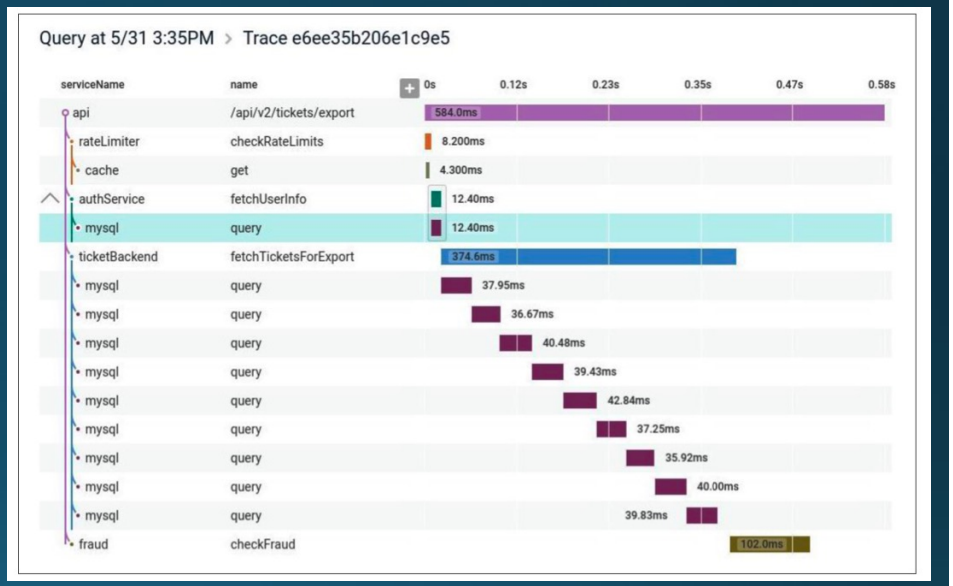
Log Aggregation and Distributed Tracing



logs are collected as part of log aggregation



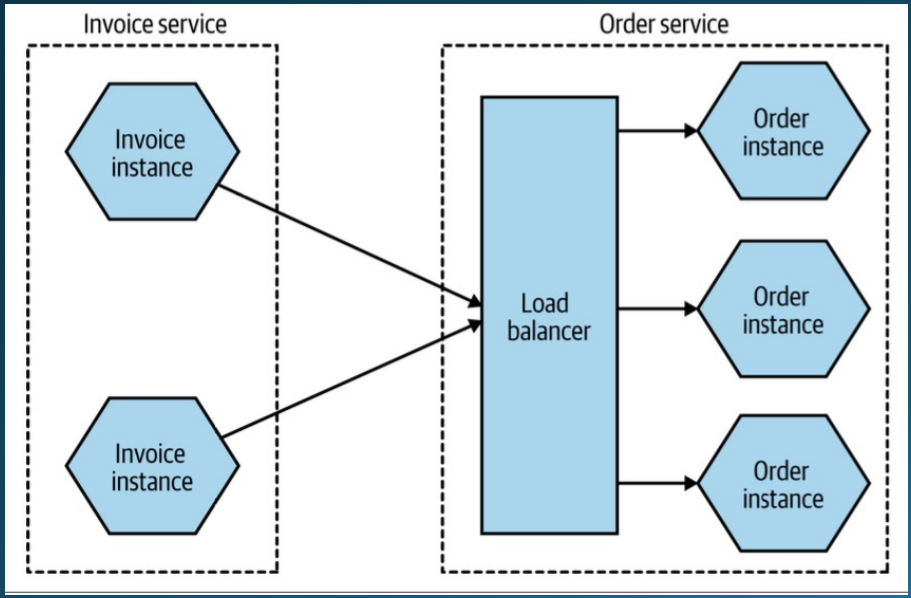
Generating a correlation ID for a set of calls



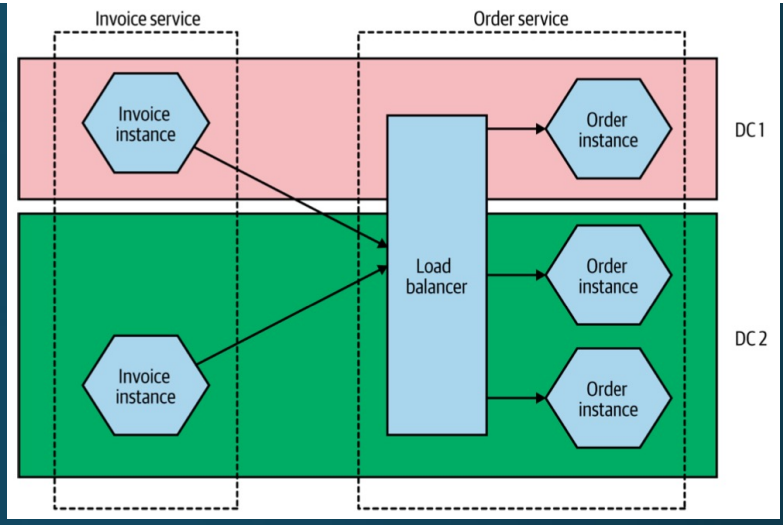
A distributed trace shown in Honeycomb, allowing you to identify where time is being spent for operations that can span multiple microservices

From Logical to Physical

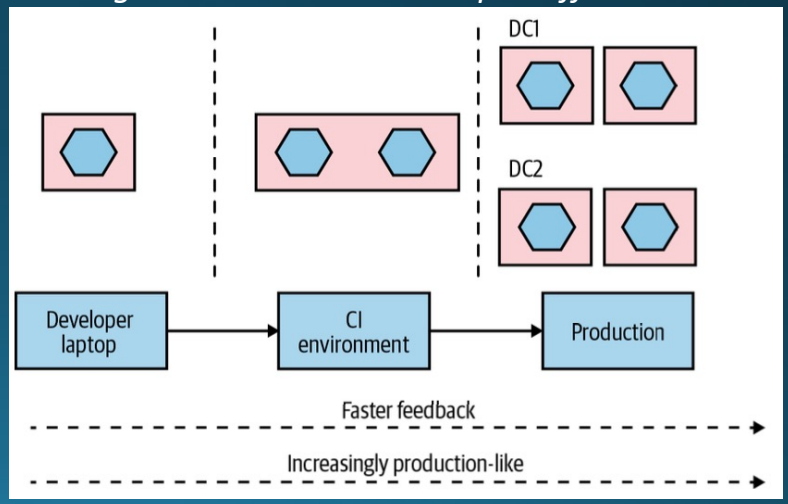
logical view of two microservices



Using a load balancer to map requests to specific instances of the Order microservice

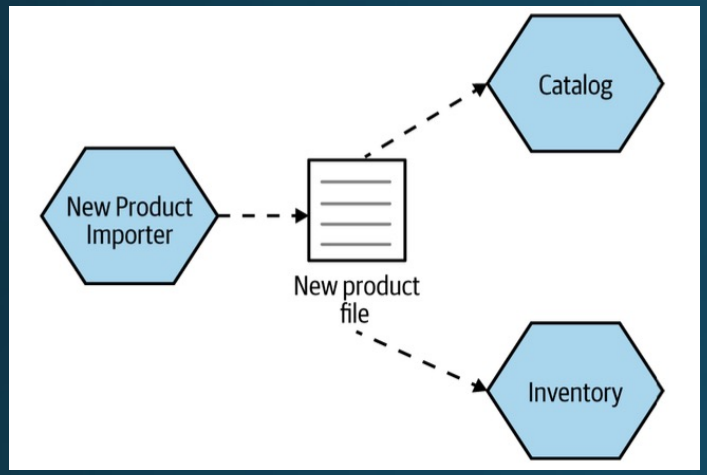


Distributing instances across multiple different data centers

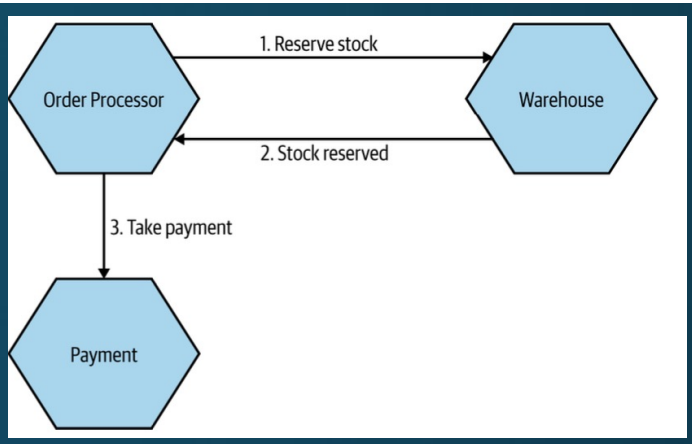


A microservice can vary in how it is deployed from one environment to the next

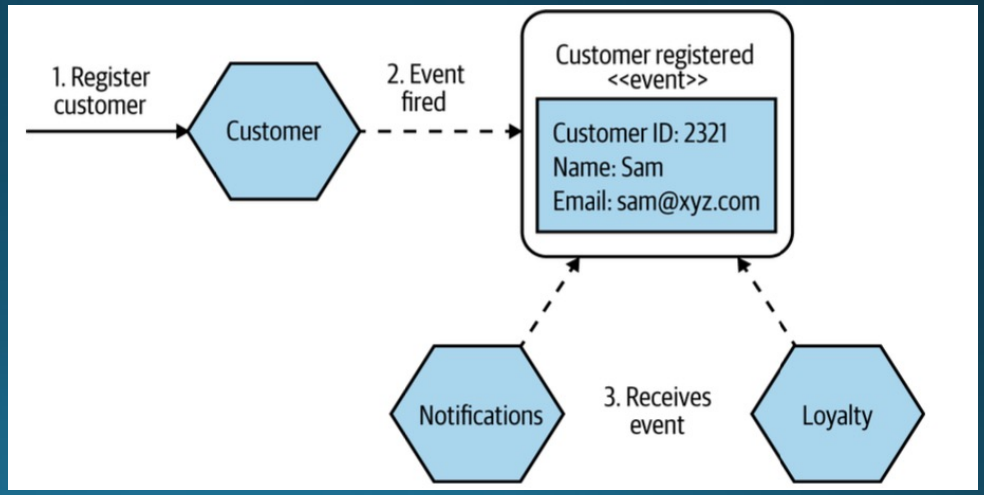
Streaming



One microservice writes out a file that other microservices make use of

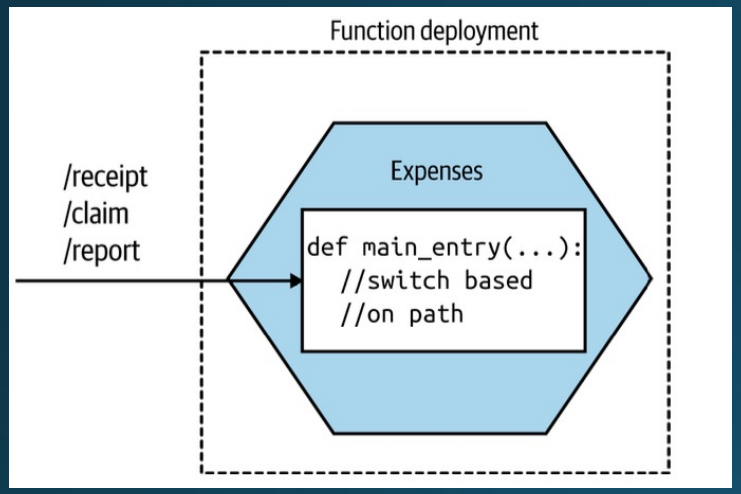


Request-Response Communication:
Order Processor needs to ensure stock can be reserved before payment can be taken

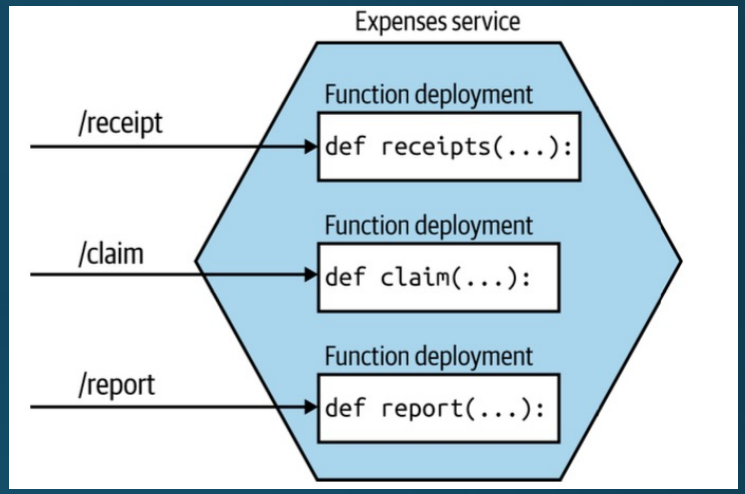


An event with more information in it can allow receiving microservices to act without requiring further calls to the source of the event

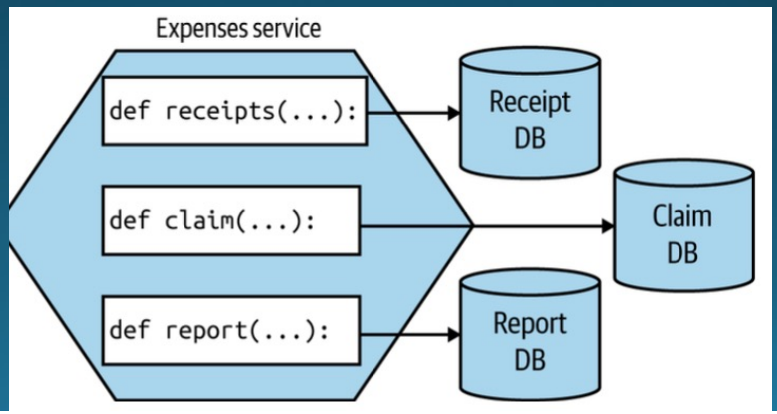
Leveraing Public Cloud: PaaS, FaaS



Function per microservice



Function per Aggregate



Each function using its own database

What Makes a Good Microservice Boundary?

- Information Hiding
- Cohesion

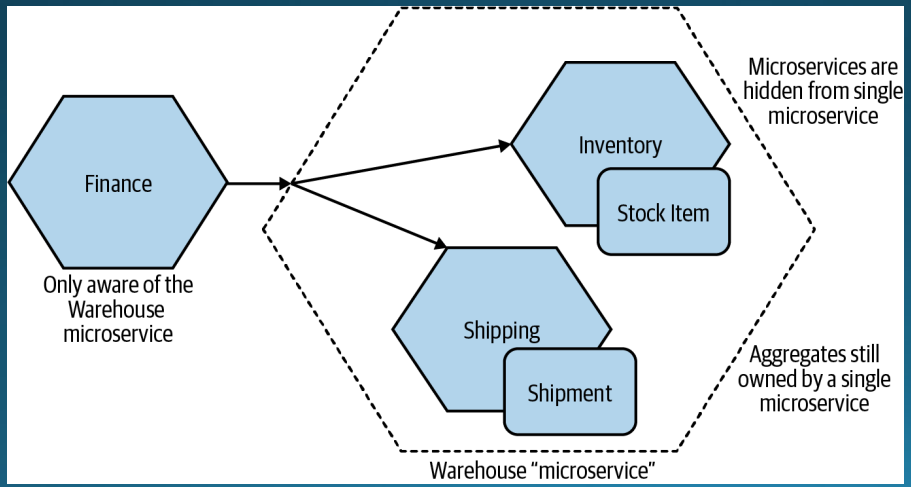
  We want the functionality grouped in such a way that we can make changes in as few places as possible

- Coupling

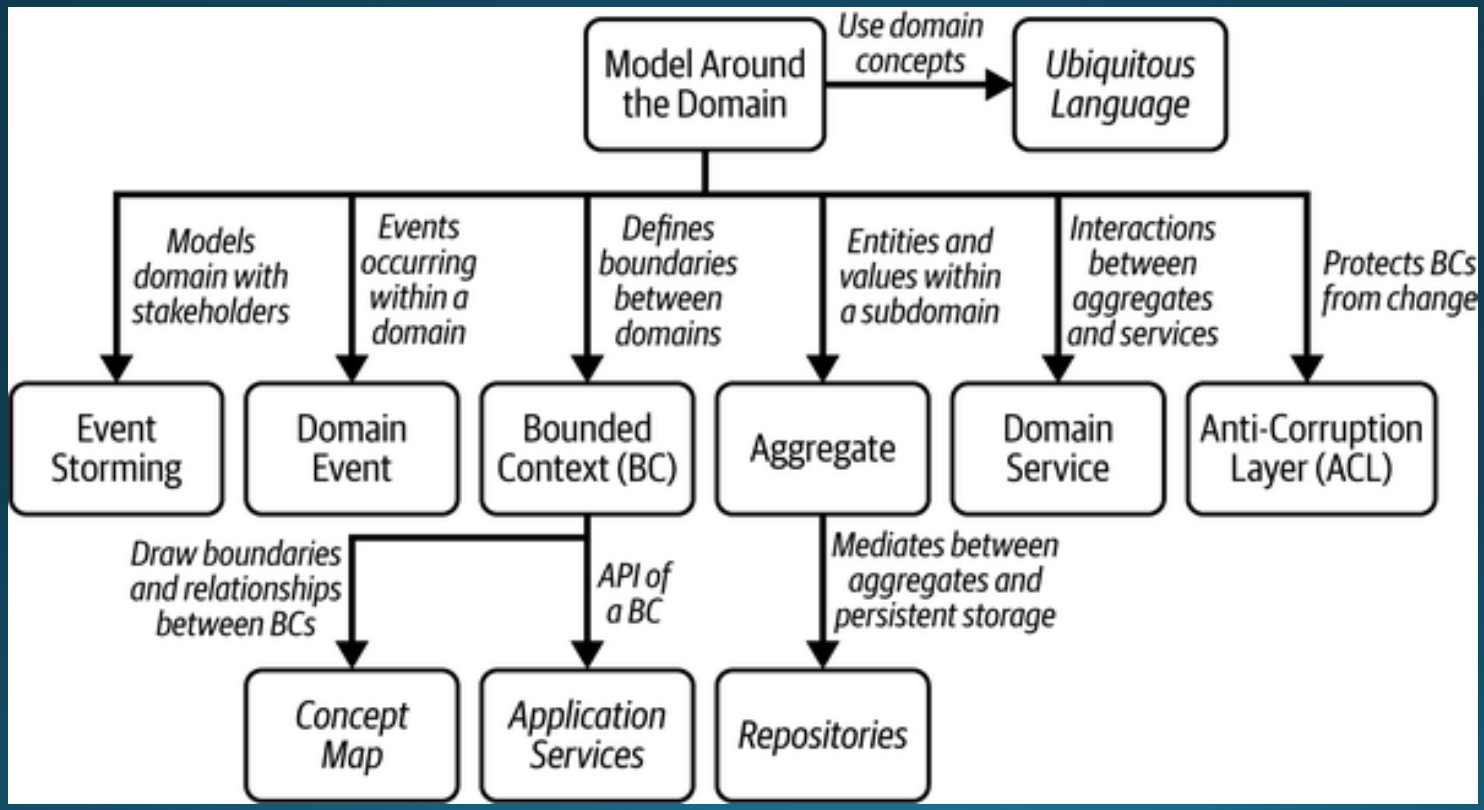
  A loosely coupled service knows as little as it needs to about the services with which it collaborates.

- The Interplay of Coupling and Cohesion

  A structure is stable if cohesion is strong and coupling is low



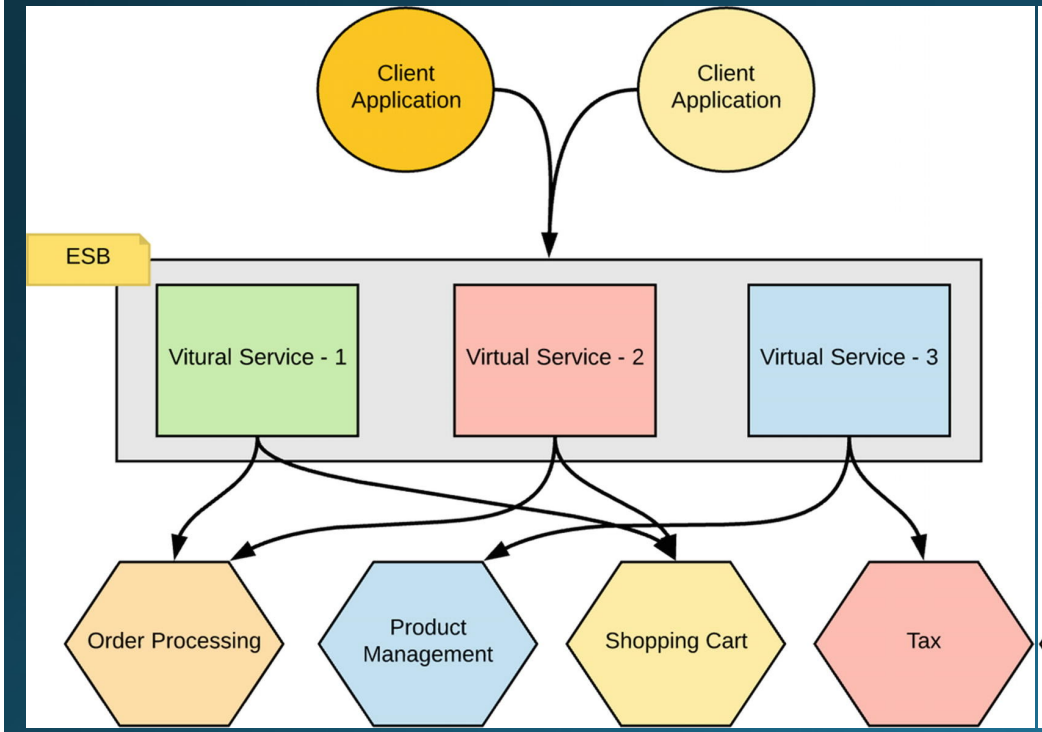
Domain Modeling Techniques for Designing Microservices



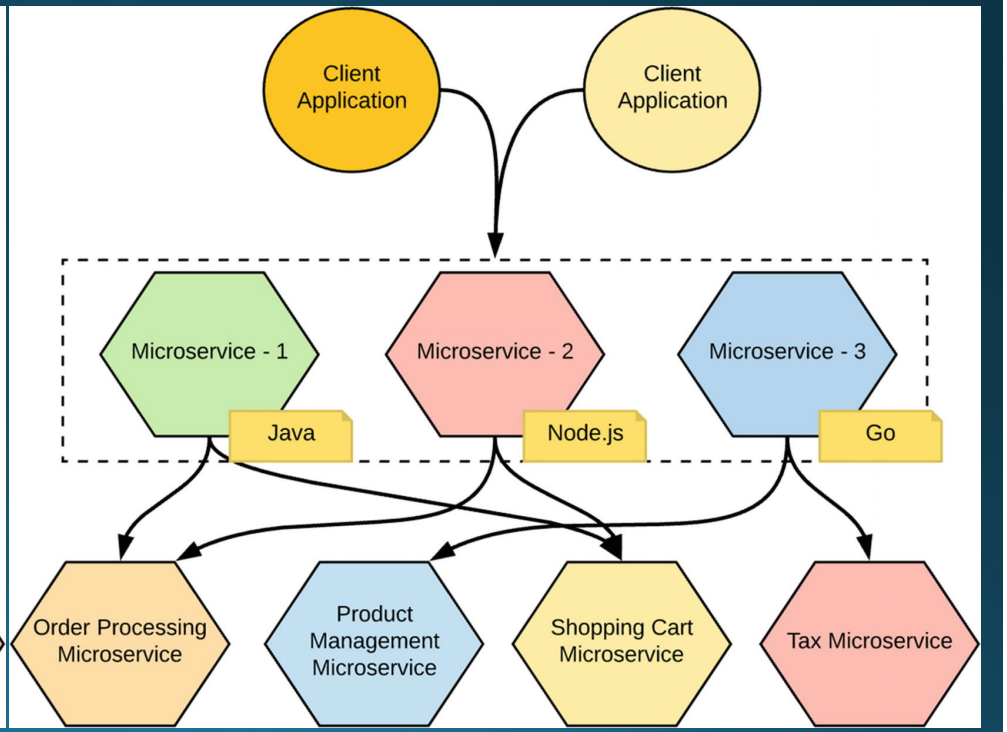
DDD for Microservices will be discussed in detail in the following lecture…

Integration Architecture Examples

Integrating services with ESB in SOA

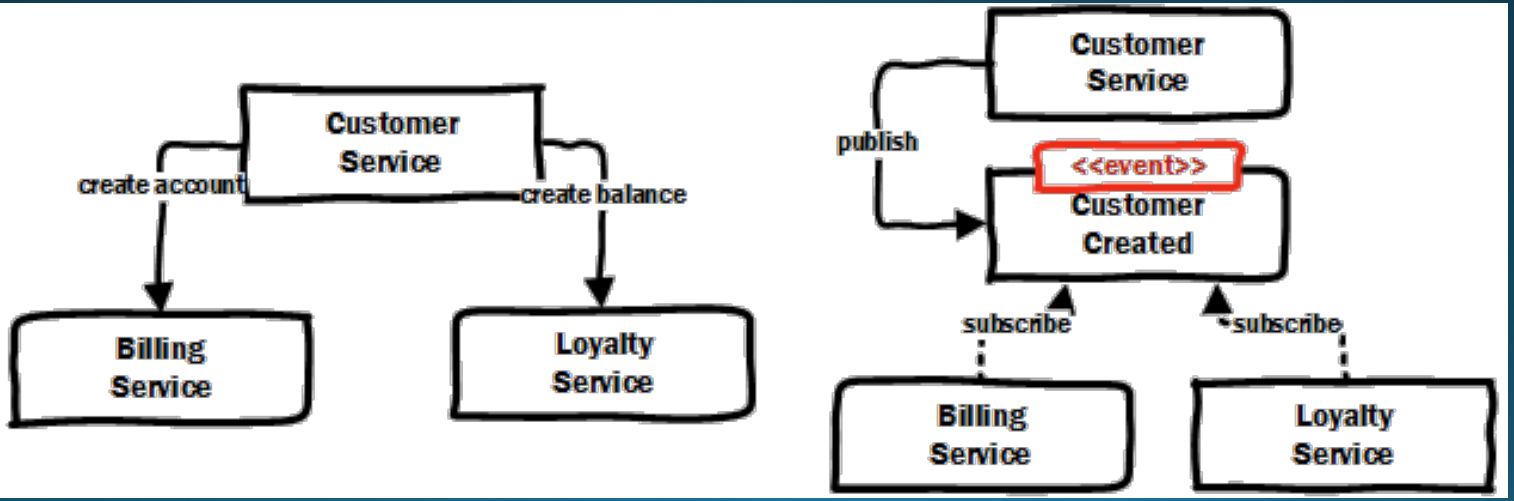


Smart endpoints and dumb pipes

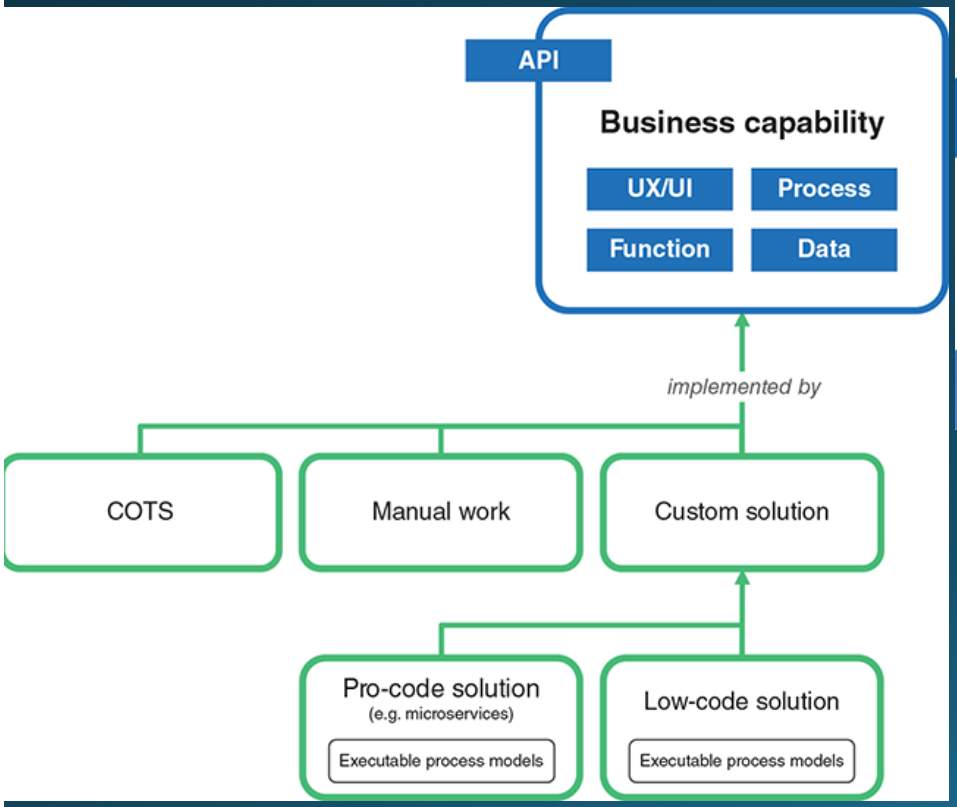


Integration Styles

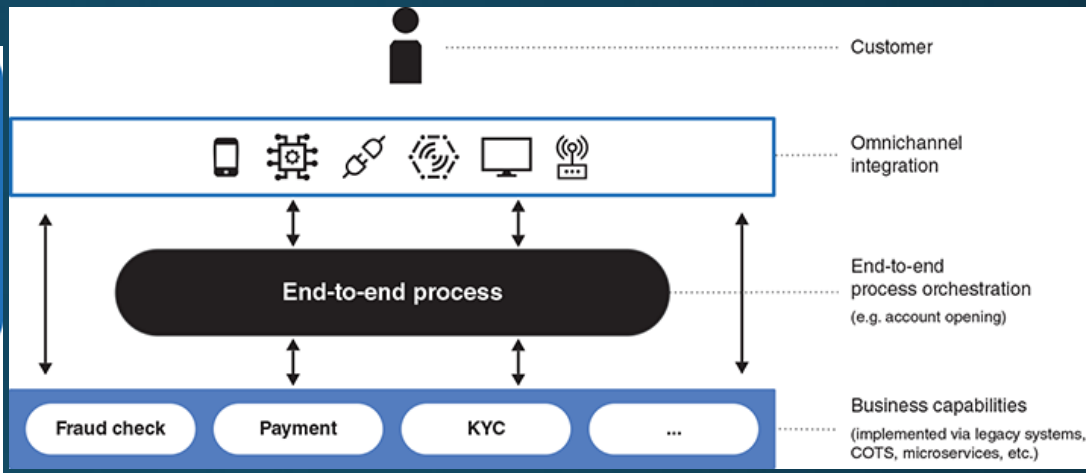
- Web services can be combined in two ways:
  - Orchestration
  - Choreography

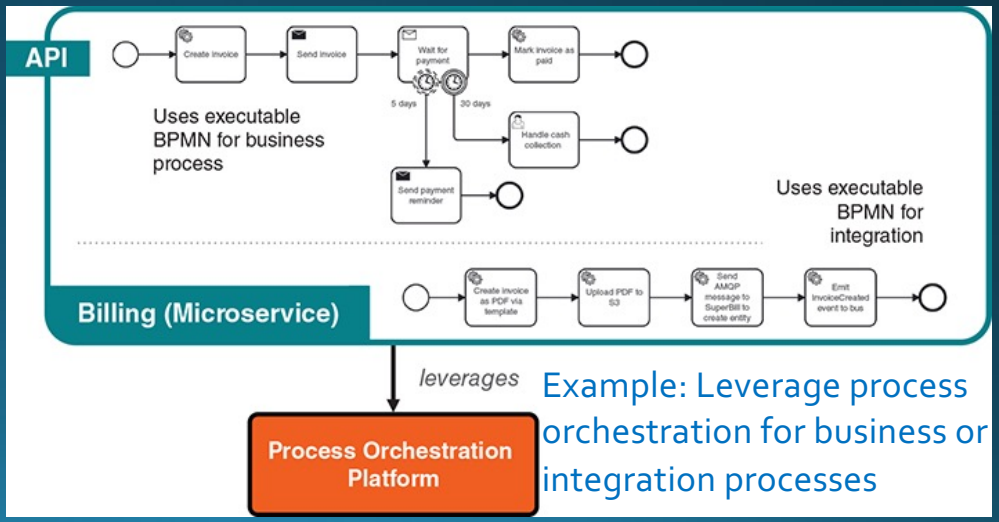


Implementing Business Capability with Orchestration



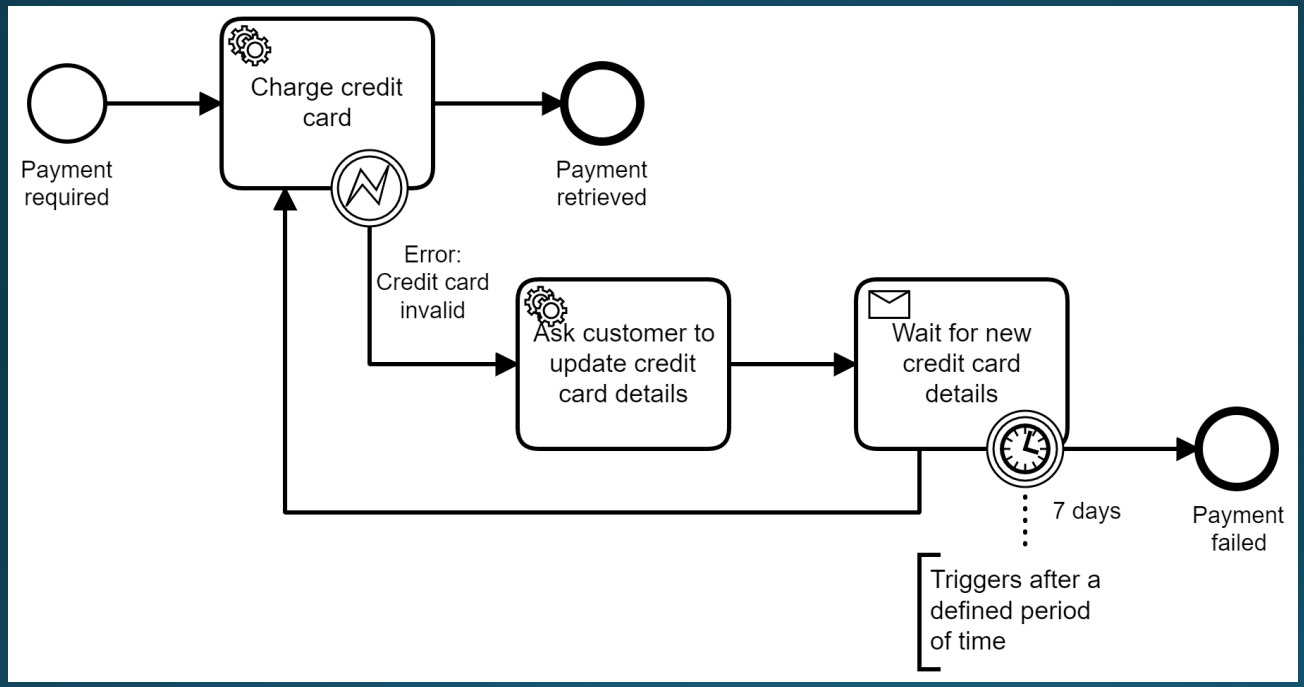
Different options for implementing business capabilities

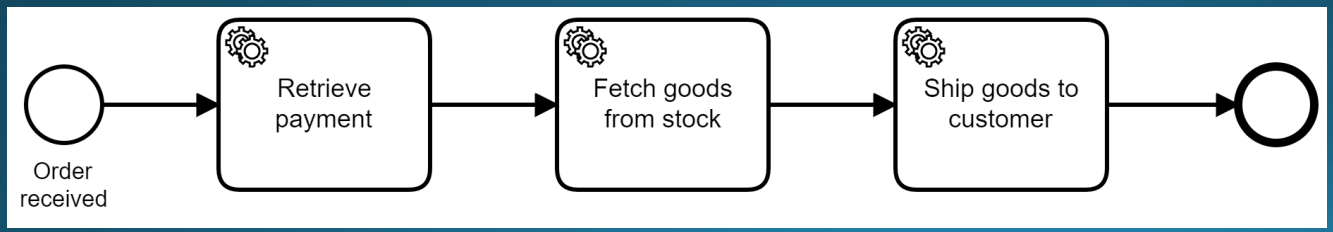




Example: Leverage process orchestration for business or integration processes

Examples of Process Model

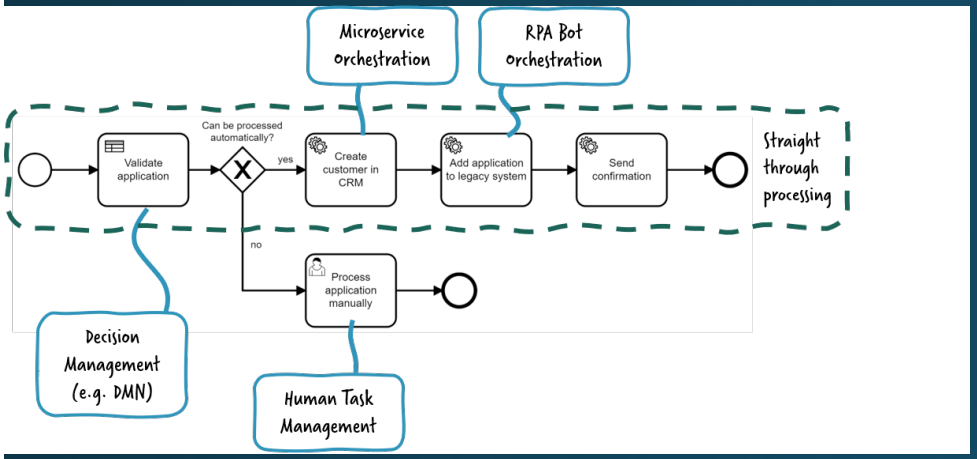


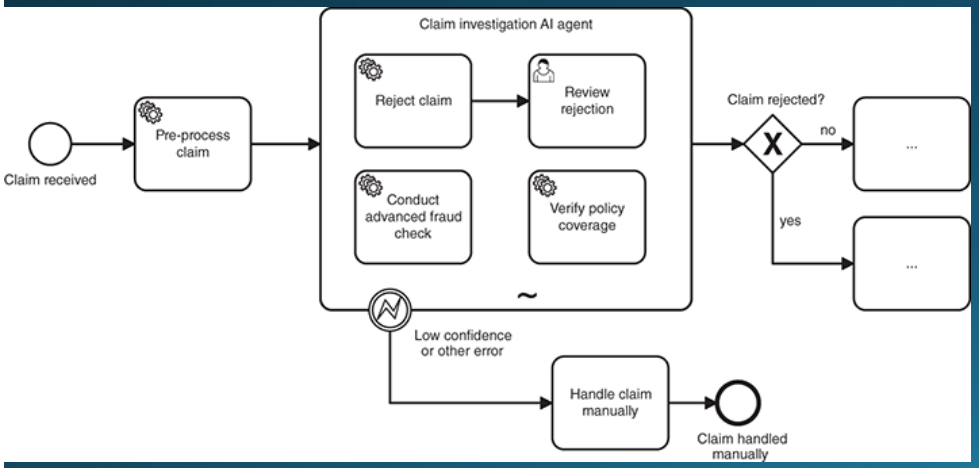


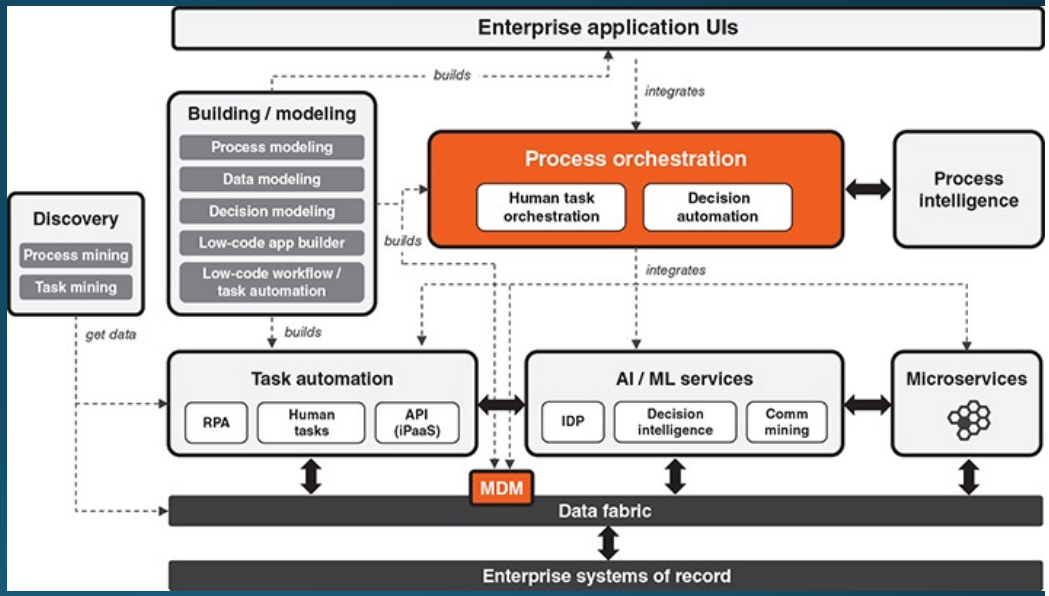
The Usual Integration Chaos

- **Integration via database**: a service accesses some other services' database directly in order to communicate, often without the other service knowing it.
- **Naive point-to-point integrations**: Two components communicate directly with each other, often via REST, SOAP or messaging protocols, without properly clarifying all aspects around remote communication.
- **Database triggers**: Additional logic is invoked whenever you write something to the database.
- **Brittle tool chains**: For example moving comma separated text files (CSV) via FTP.
- The story above may lead to a home grown workflow engine. Home grown workflow engines have severe shortcomings.

Use cases are typically mixed in real-life examples

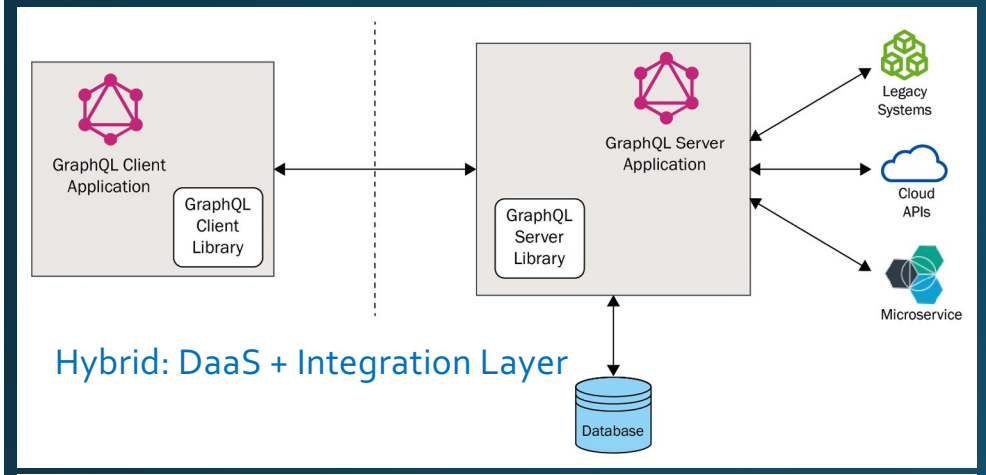




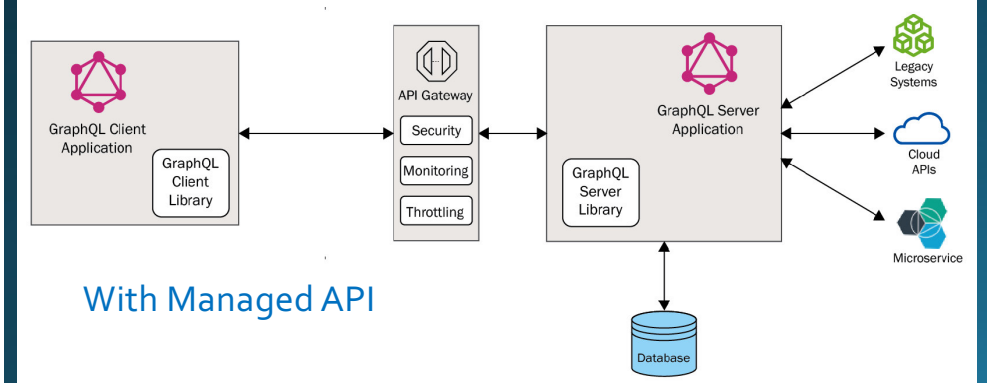


Technologies around process orchestration required
To digitalize modern business

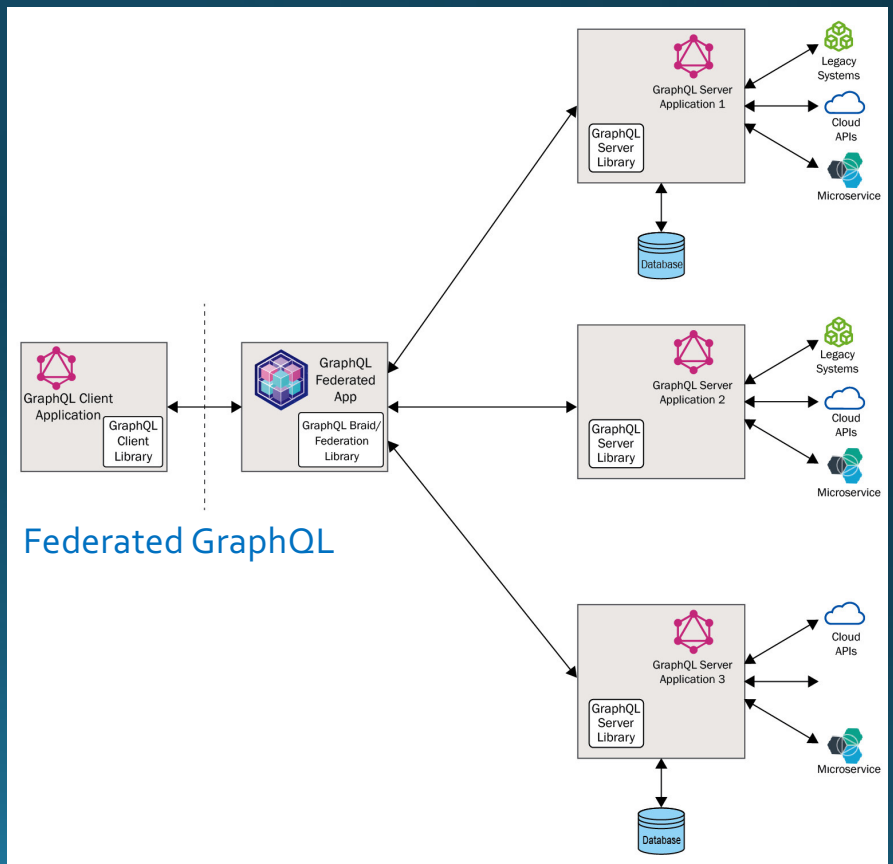
Using GraphQL in the Solution Architecture



Hybrid: DaaS + Integration Layer

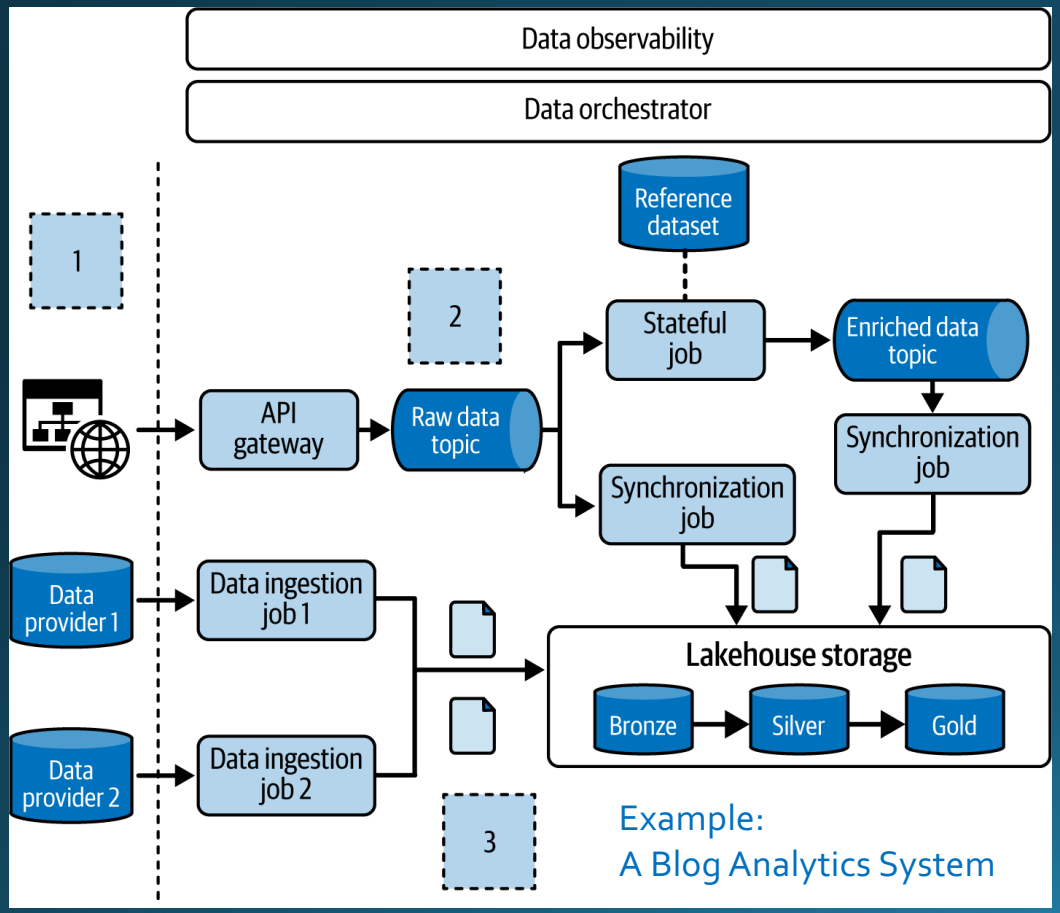


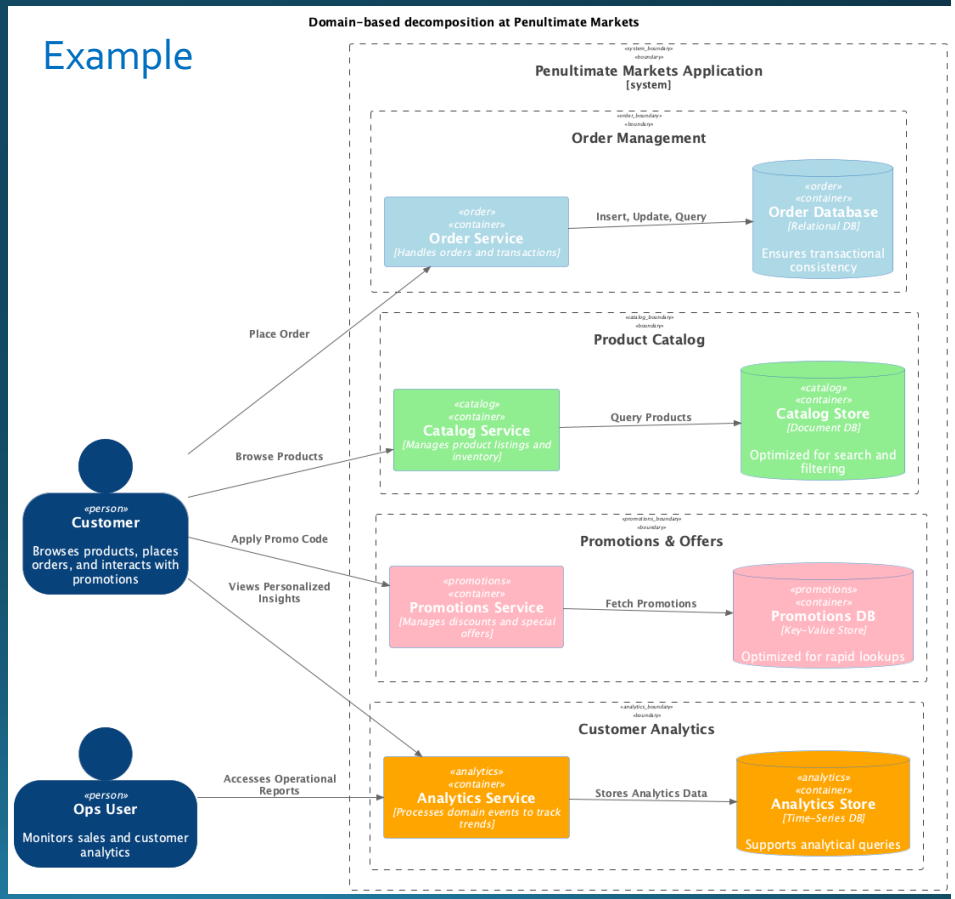
With Managed API



Federated GraphQL

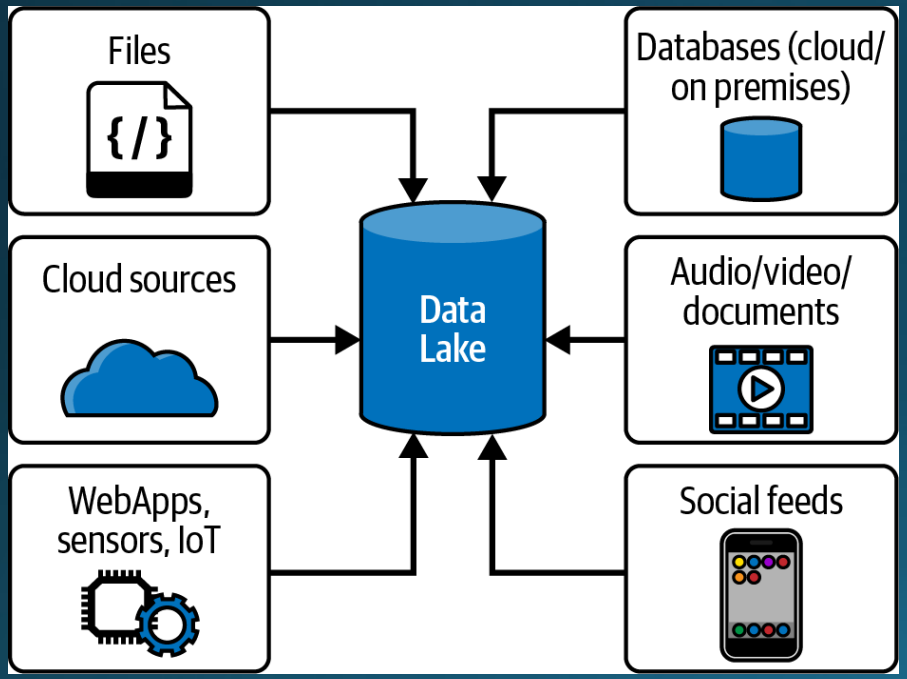
More Complexity in Data-Centric Architecture

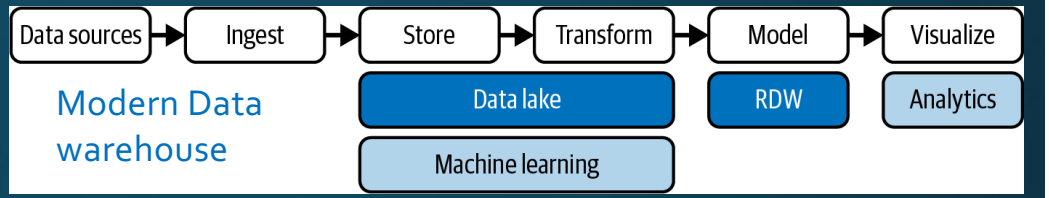


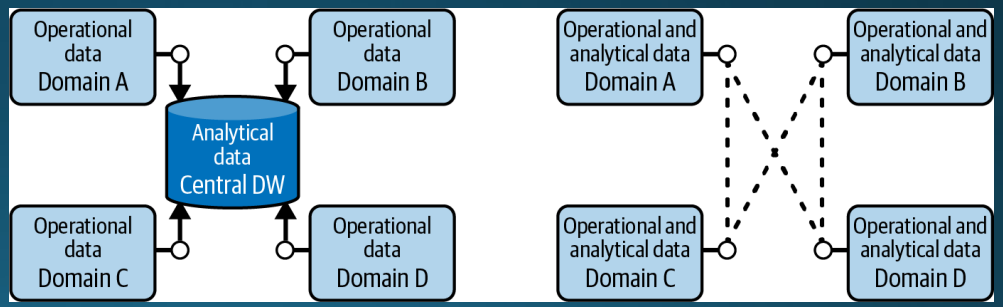


The Evolution of Modern Data Architectrue

- Data Architecture may utilize microservices architecture
- Design of Microservices should be aware of the data architecture and its evolution







Data mesh: data is kept within several domains within a company

Further Reading

- Hexagonal Architecture
  - Search online by yourself
- The Twelve-Factor App Methodology
  - https://12factor.net/