

Design Patterns

for



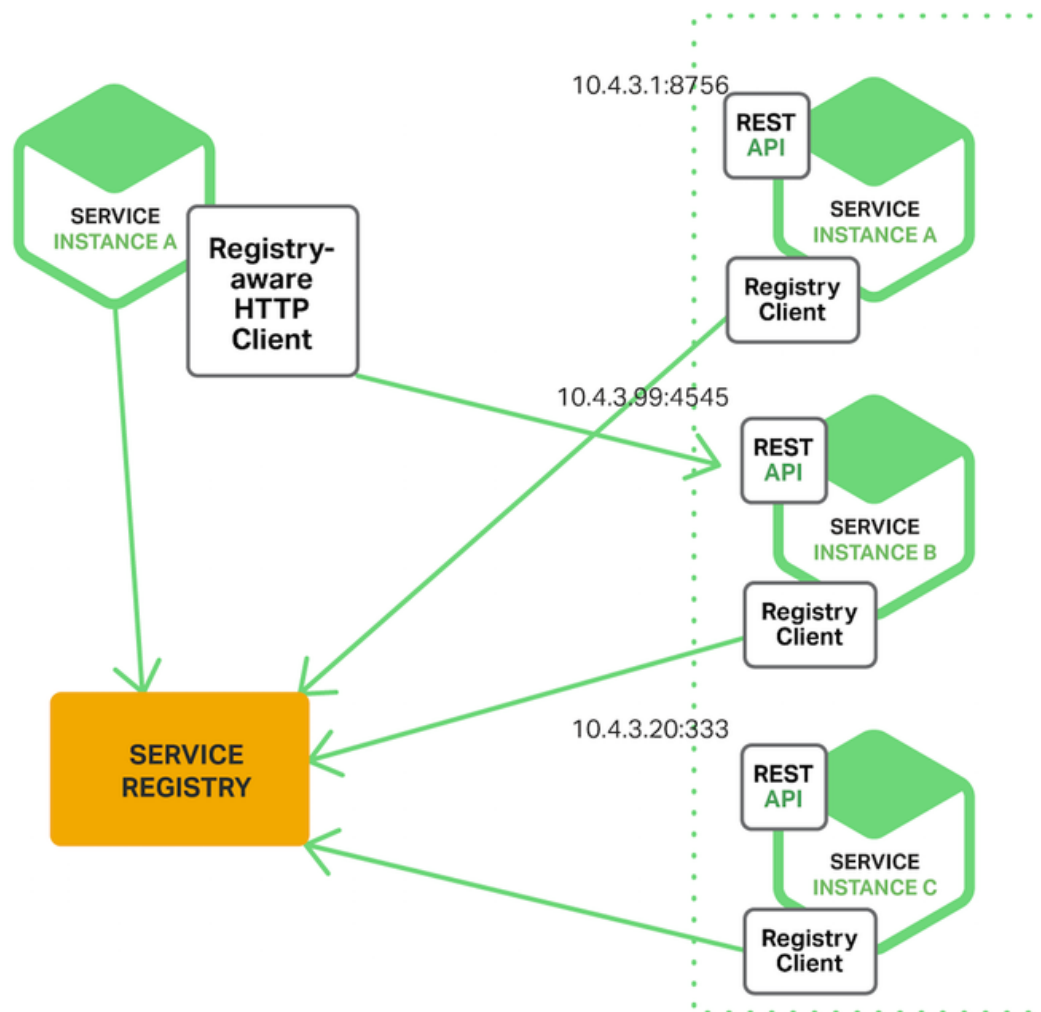
Designing and Implementing

Microservices

The diagram illustrates a microservices architecture. On the left, a **User** interacts with two client applications: a **Client Mobile app** (represented by a smartphone icon) and a **Client Web App** (represented by a desktop monitor icon). Both client apps have bidirectional arrows connecting them to a central **API Gateway**. The **API Gateway** is a large green box containing the text: **API Gateway** (in red), **Security**, **Routing**, **Logging**, **Rate Limiting**, and **Aggregation** (in blue). The **API Gateway** is part of the **Back end**, which is enclosed in a dashed box. To the right of the **API Gateway**, there are three microservices: **Students Microservice** (yellow box), **Courses Microservice** (light gray box), and **Payments Microservice** (purple box). Arrows point from the **API Gateway** to each of these microservices. At the bottom right, there is a blue icon consisting of three dots between two angled brackets, representing a database or additional services.

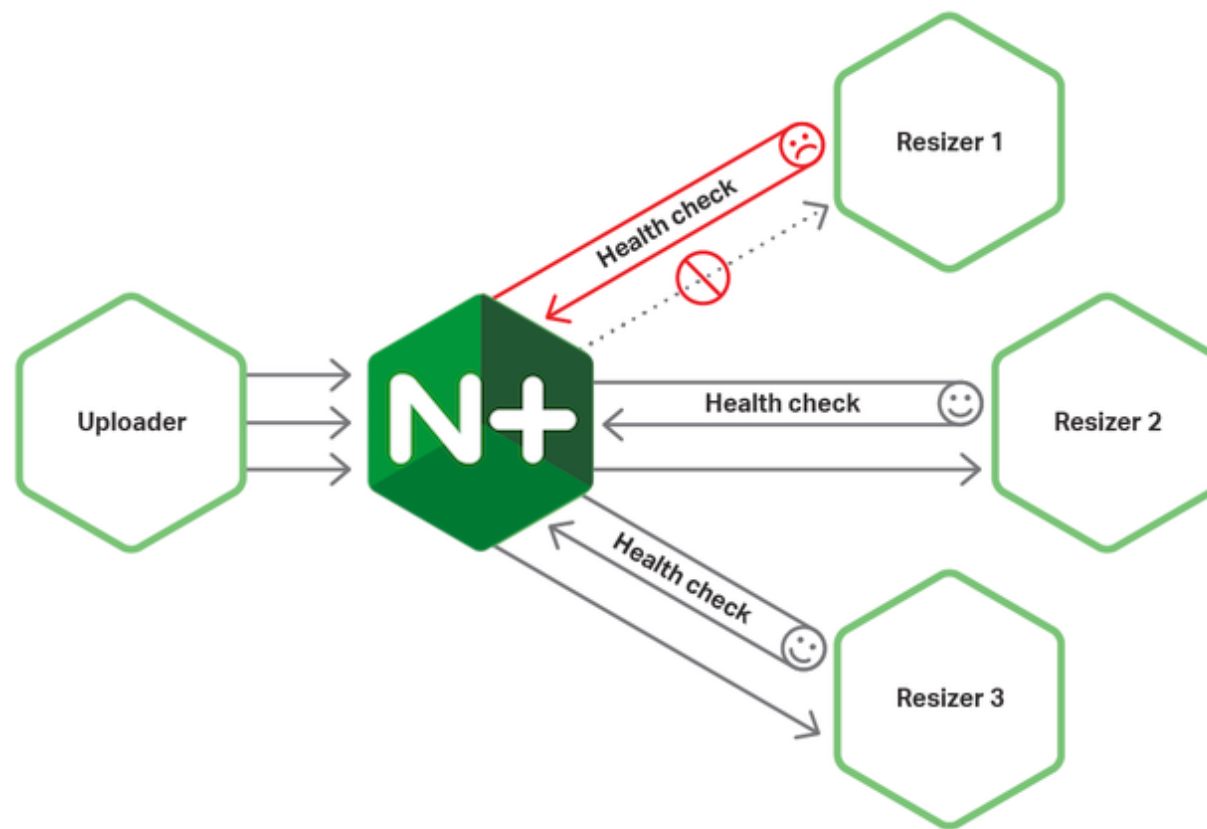
Use an API Gateway to handle client requests and route them to the appropriate microservices. This centralized authentication, load balancing, and routing logic.

SERVICE REGISTRY PATTERN



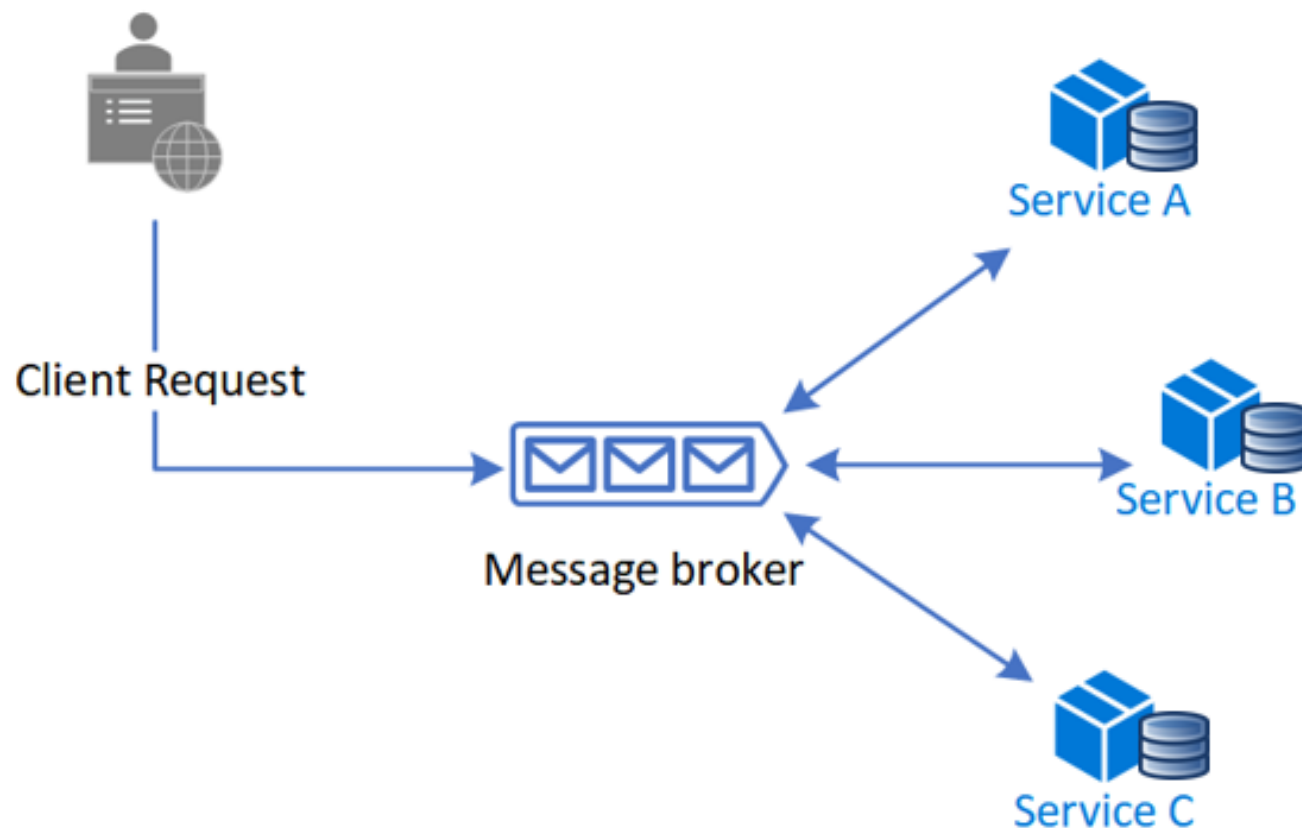
Implement a service registry to automatically locate and register microservices. This helps in dynamic discovery and communication between services.

CIRCUIT BREAKER PATTERN



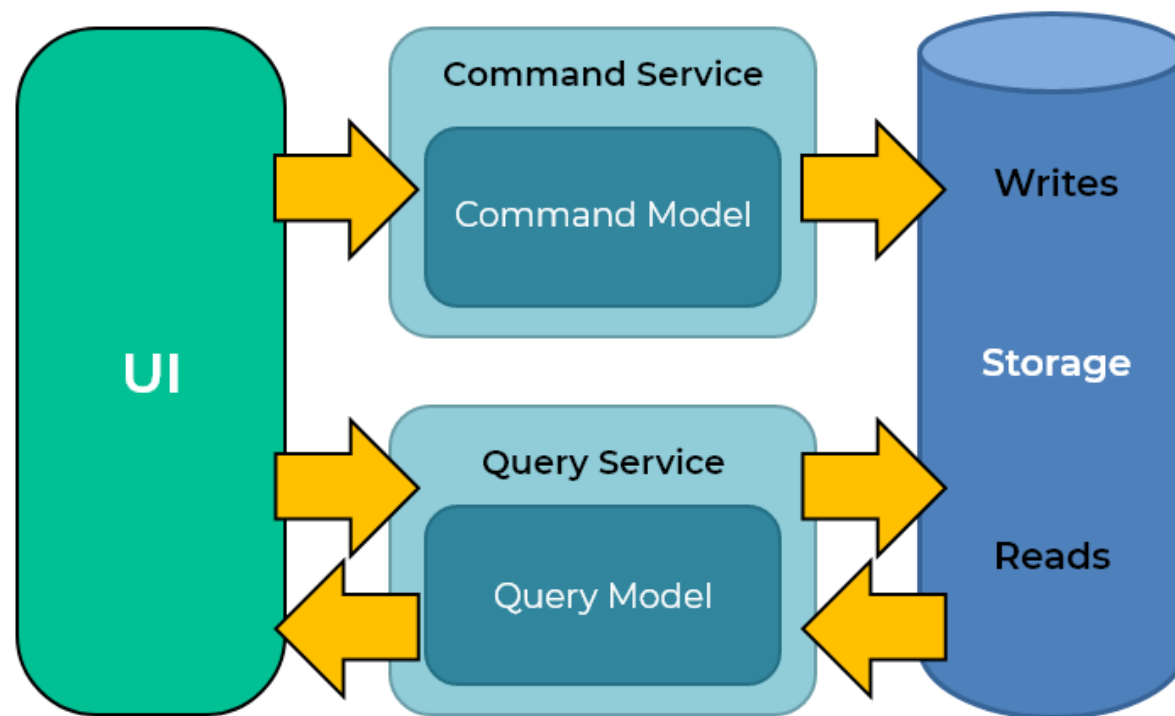
Prevent cascading failures by using a circuit breaker that can temporarily stop requests to a failing service and provide fallback mechanisms.

SAGA PATTERN



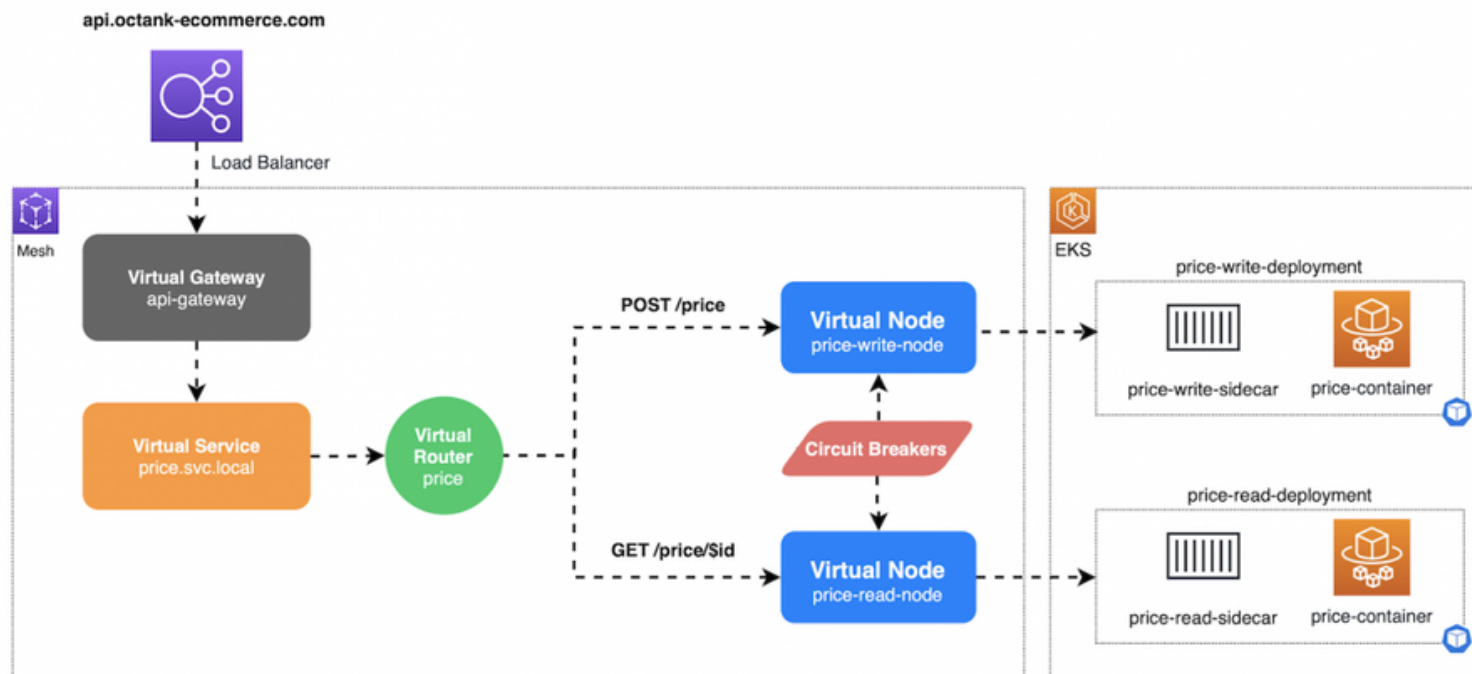
Manage long-lived transactions across multiple microservices by breaking them down into a sequence of smaller, local transactions.

CQRS PATTERN



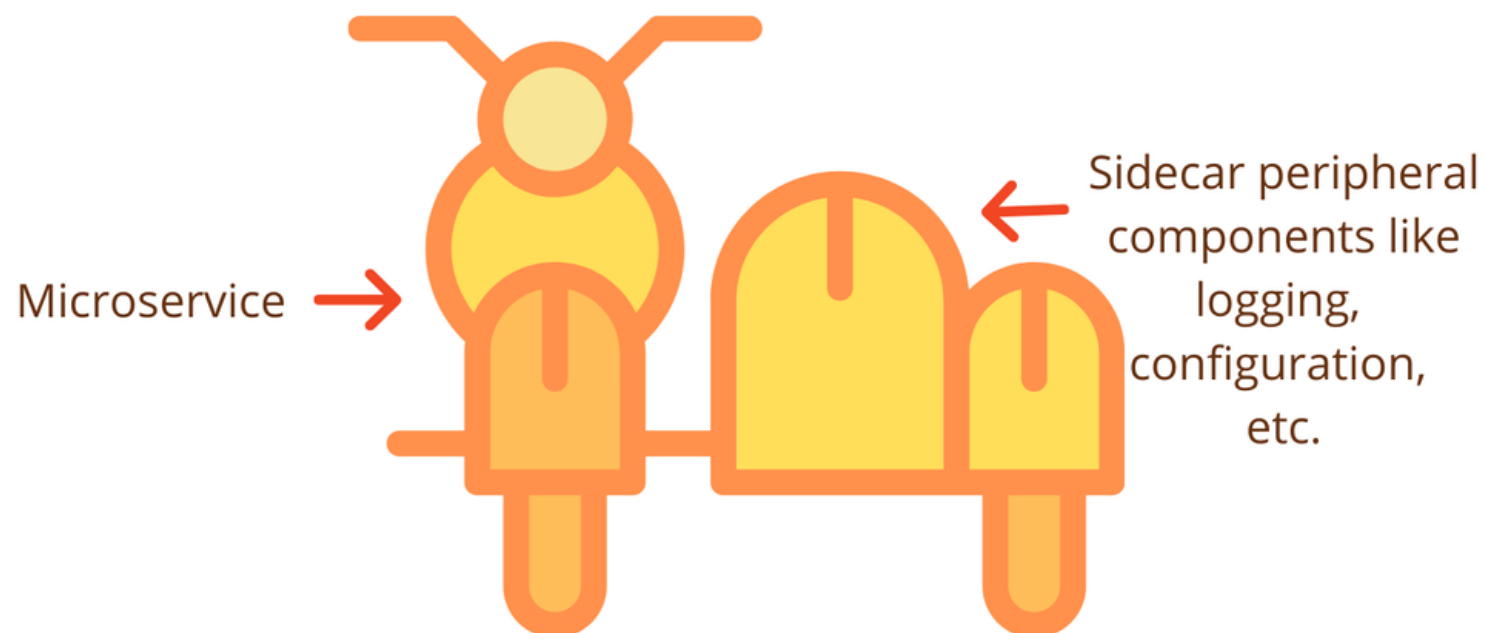
Separate the read and write responsibilities of a system, allowing for optimized performance and scalability.

BULKHEAD PATTERN



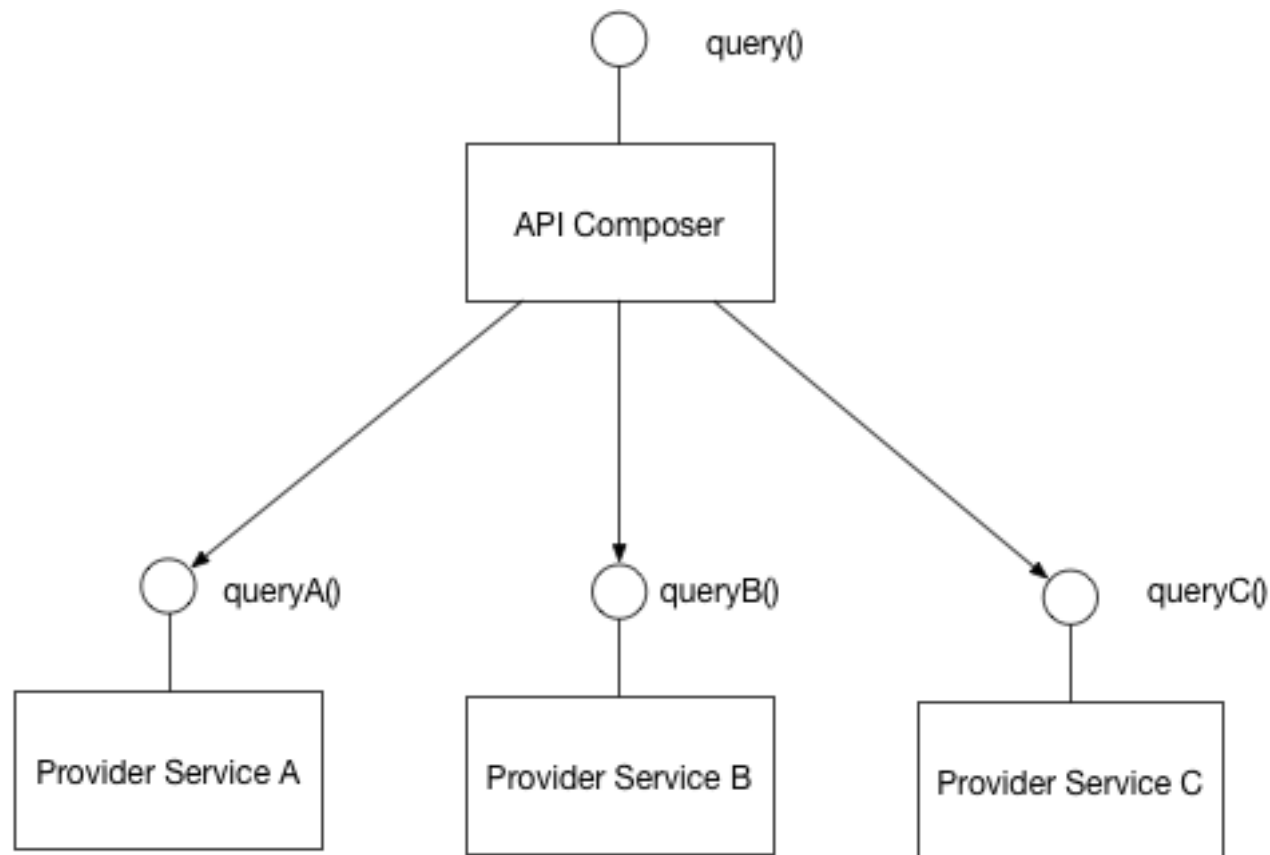
Isolate failures within separate sections to prevent them from affecting the entire system.

SIDECAR PATTERN



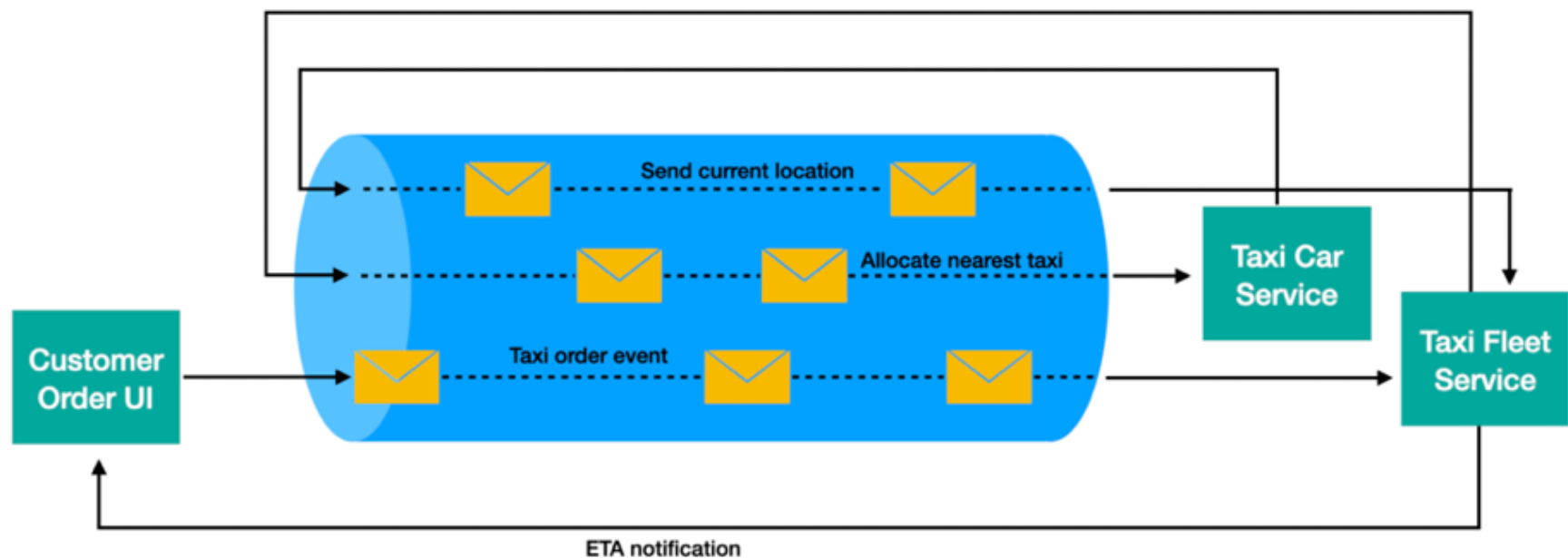
Attach a separate microservice (sidecar) to handle specific tasks like monitoring, logging, or authentication.

API COMPOSITION PATTERN



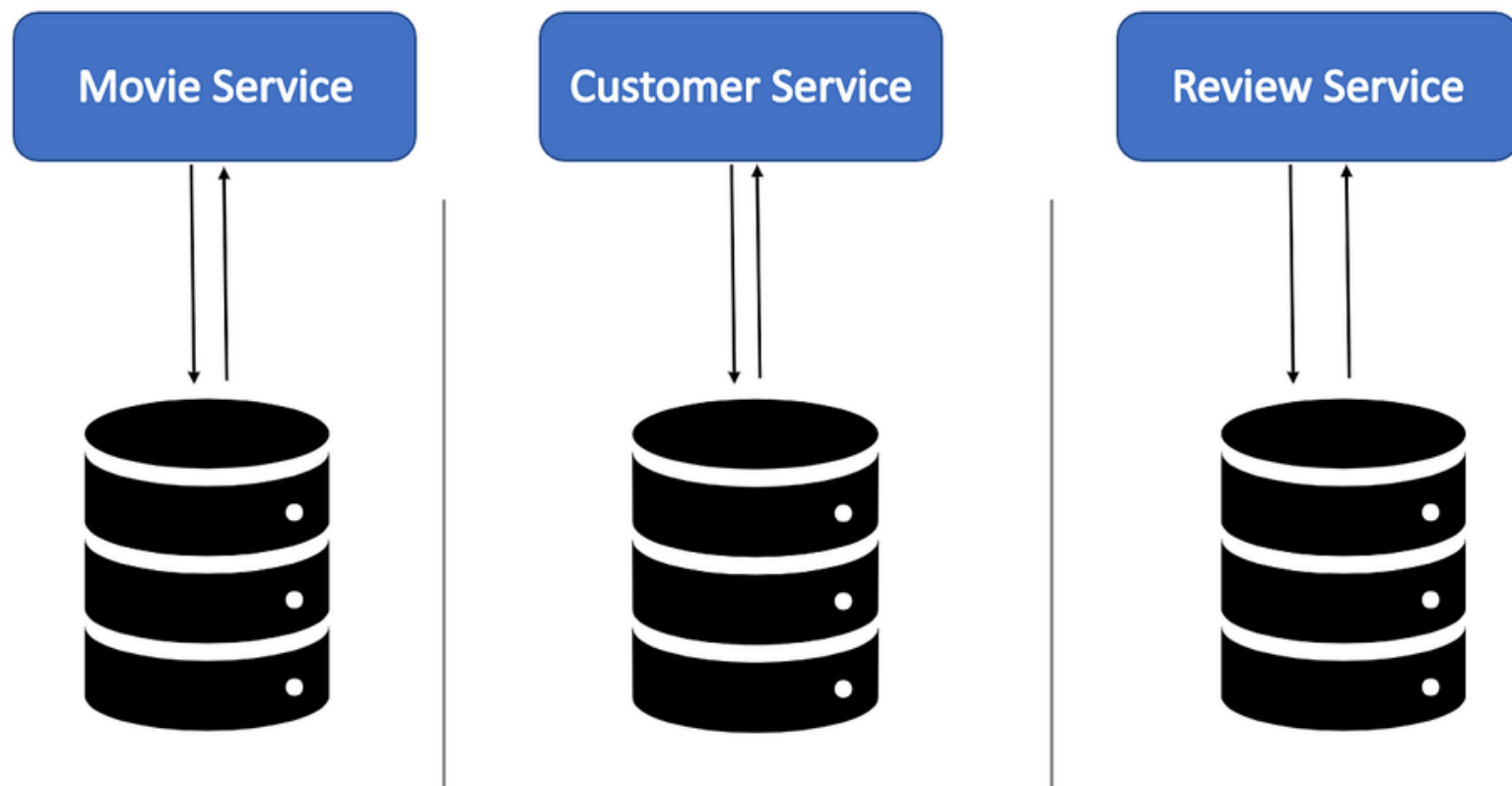
Combine multiple microservices to create a more complex and feature-rich API for clients.

EVENT-DRIVEN ARCHITECTURE PATTERN



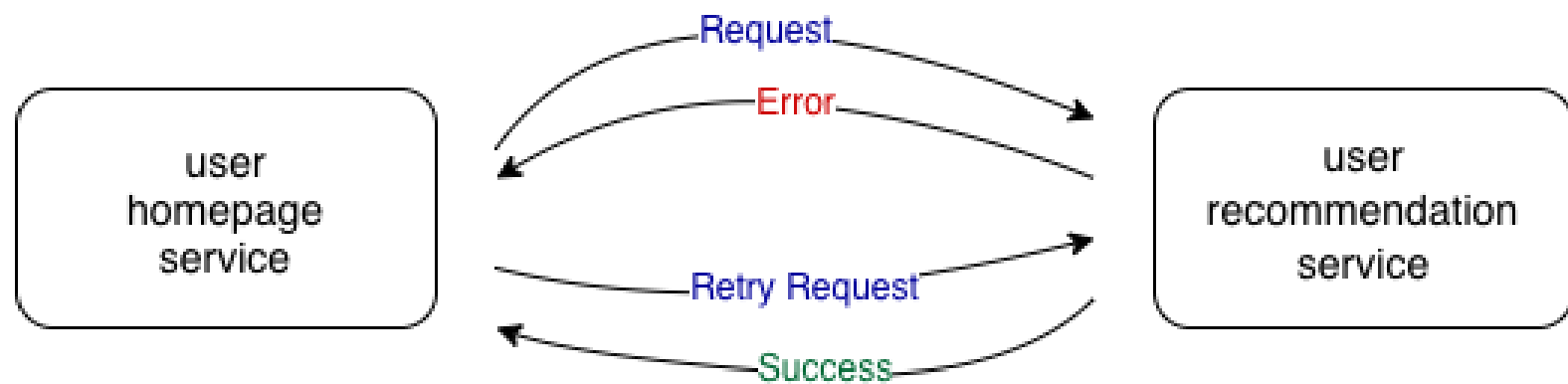
Communicate between microservices through events, enabling loose coupling and scalability.

DATABASE PER SERVICE PATTERN



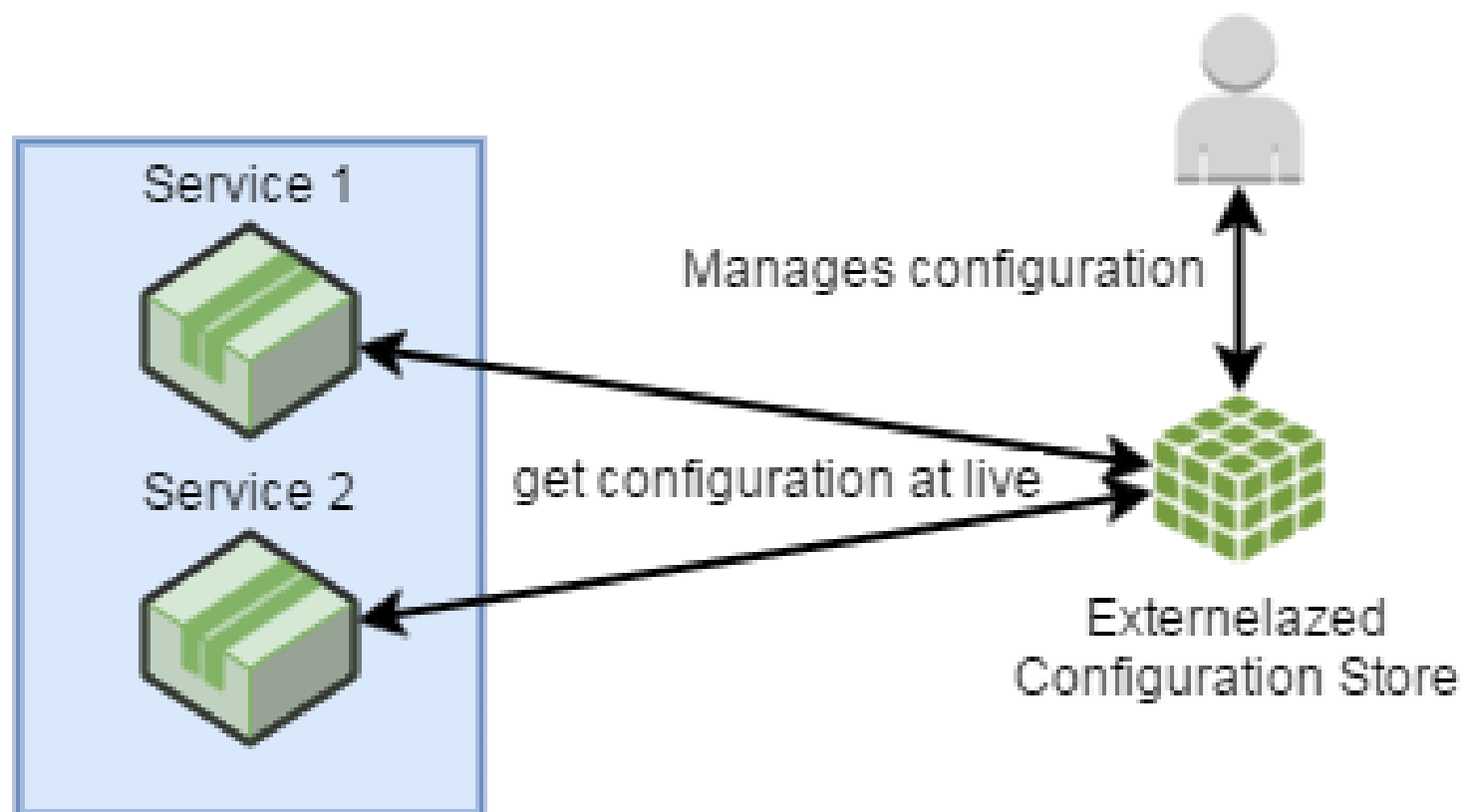
Each microservice has its own dedicated database to ensure loose coupling and autonomy.

RETRY PATTERN



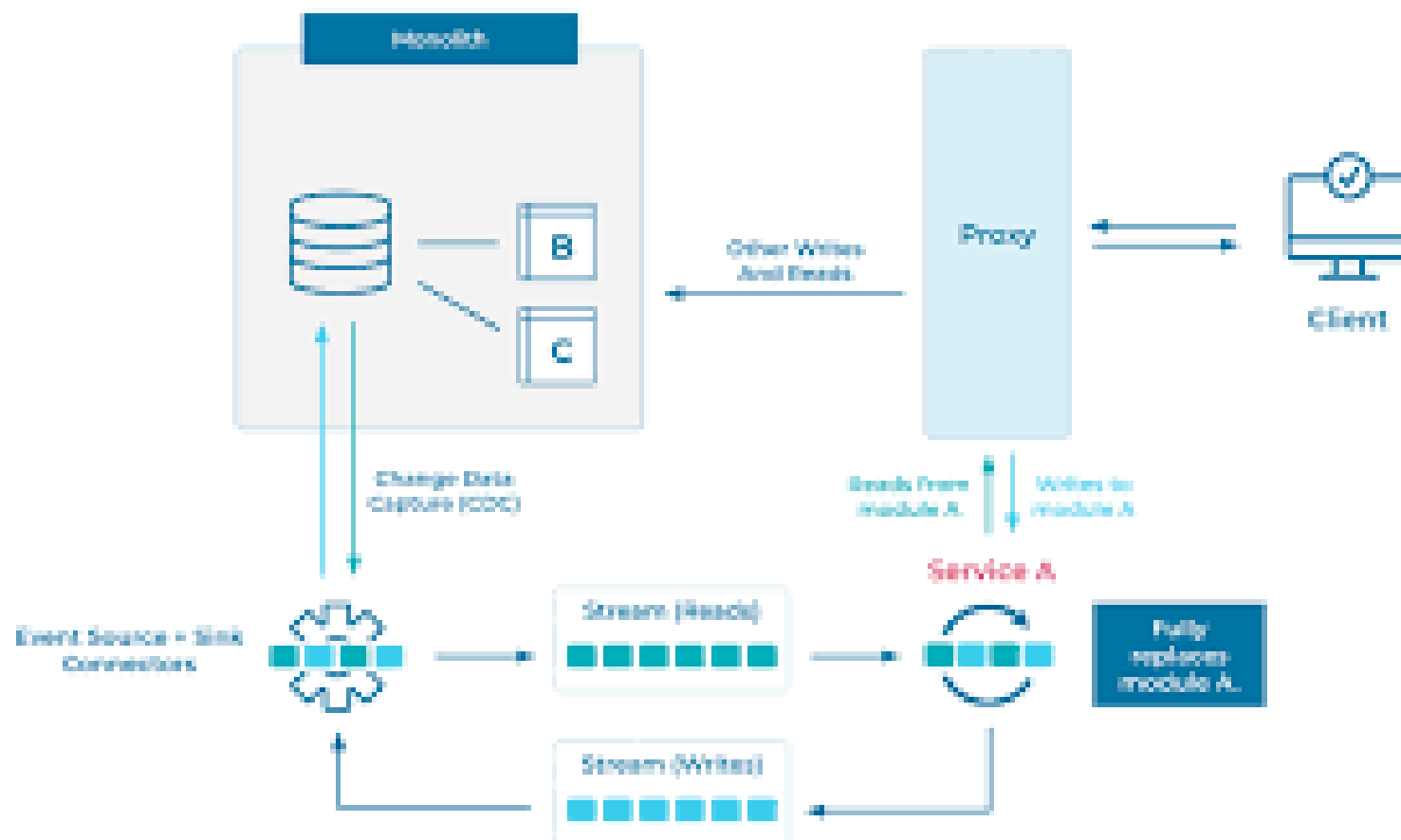
Automatically retry failed operations to improve the chances of success.

CONFIGURATION EXTERNALIZATION PATTERN



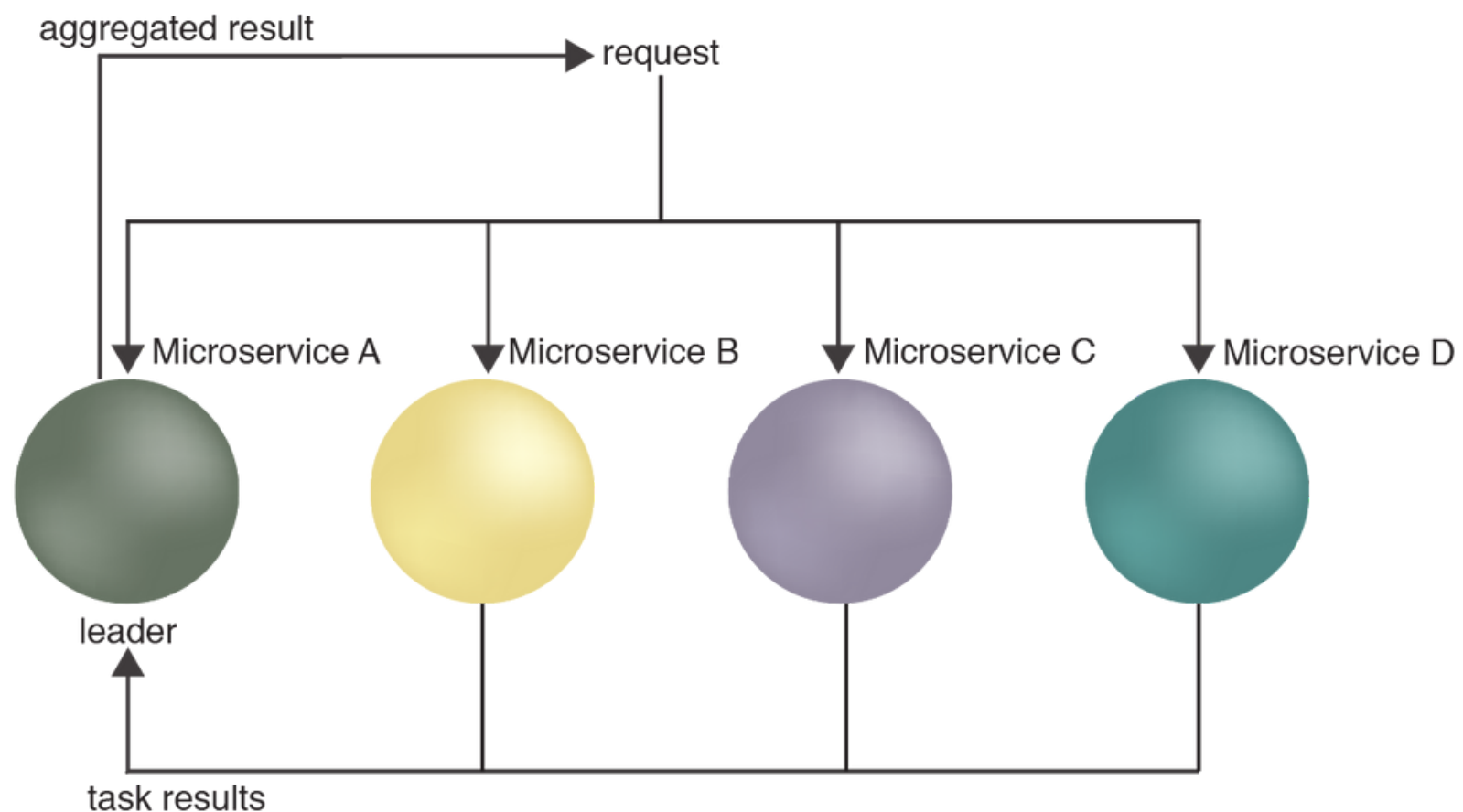
Store configuration settings outside the codebase for easier management and updates.

STRANGLER FIG PATTERN



Gradually replace components of a legacy system with new ones until the old system is "strangled" and replaced entirely.

LEADER ELECTION PATTERN



Designate a leader among instances of a microservice for tasks like coordination and decision-making.

Microservices

Thank
you!



FOLLOW

For More