

Design Patterns

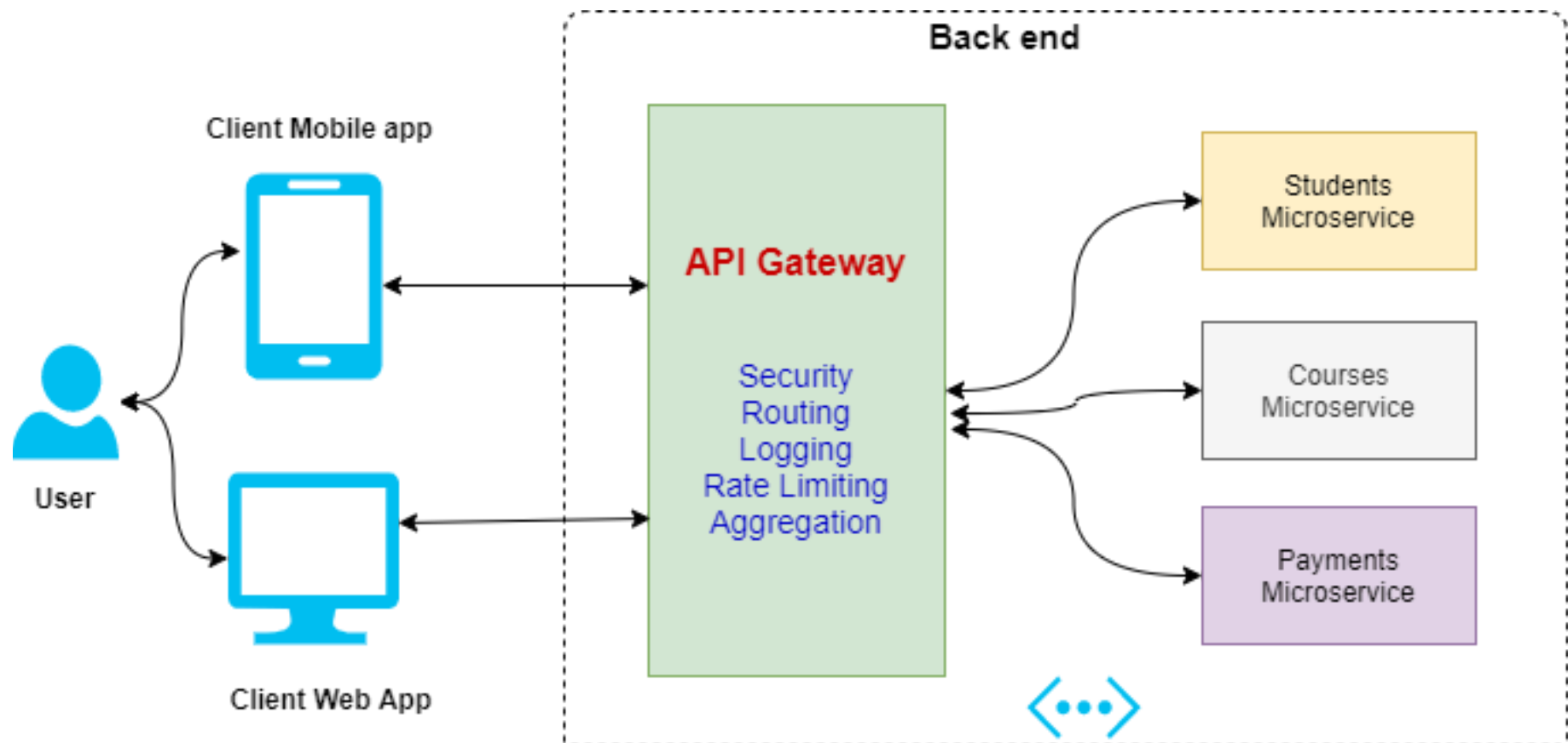
for



Designing and Implementing

Microservices

GATEWAY PATTERN

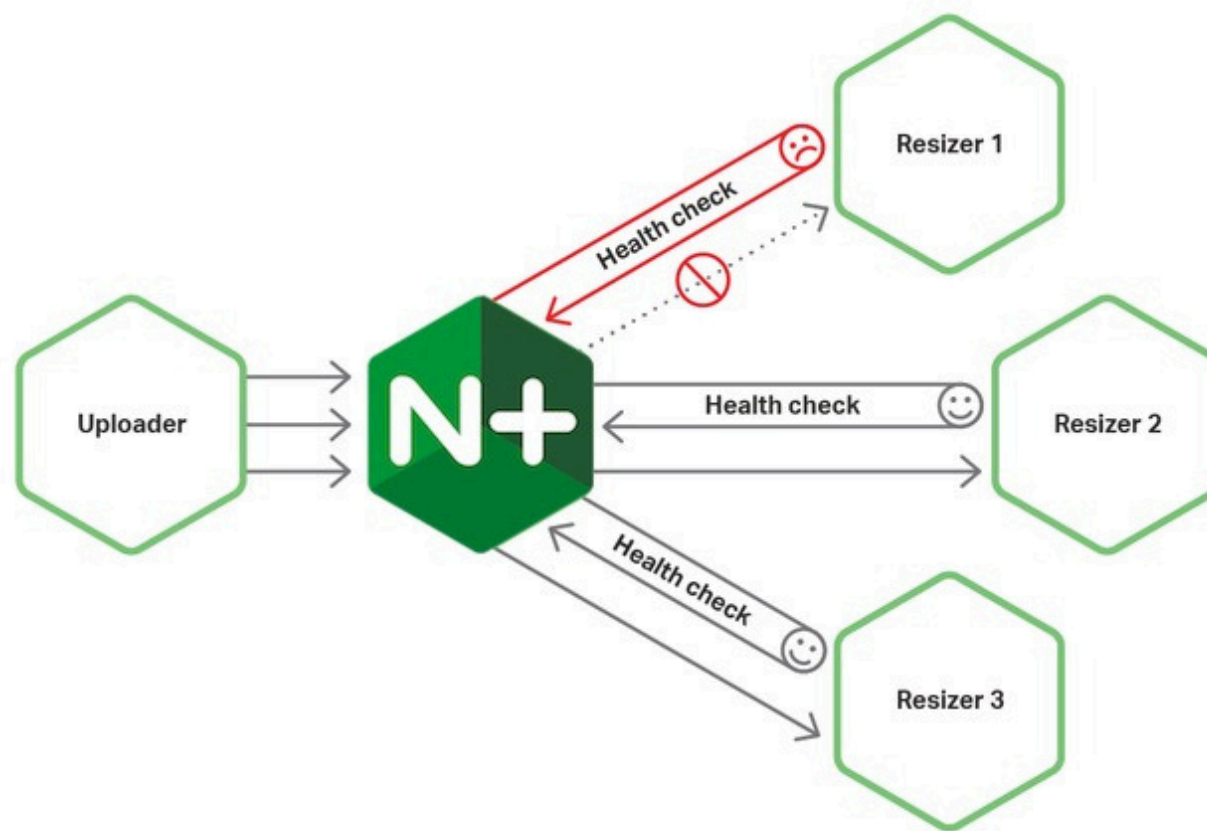


Use an API Gateway to handle client requests and route them to the appropriate microservices. This centralized authentication, load balancing, and routing logic.

The diagram illustrates the Service Registry architecture. A central orange box labeled **SERVICE REGISTRY** is connected to three green boxes on the right, each representing a **SERVICE INSTANCE** (A, B, and C). Each instance contains a **REST API** and a **Registry Client**. Arrows show the **Registry Client** of each instance connecting to the **SERVICE REGISTRY**. A **Registry-aware HTTP Client** (green box) is shown on the left, connected to the **SERVICE REGISTRY** and the **REST API** of **SERVICE INSTANCE A**. IP addresses are listed next to the instances: 10.4.3.1:8756 for Instance A, 10.4.3.99:4545 for Instance B, and 10.4.3.20:333 for Instance C.

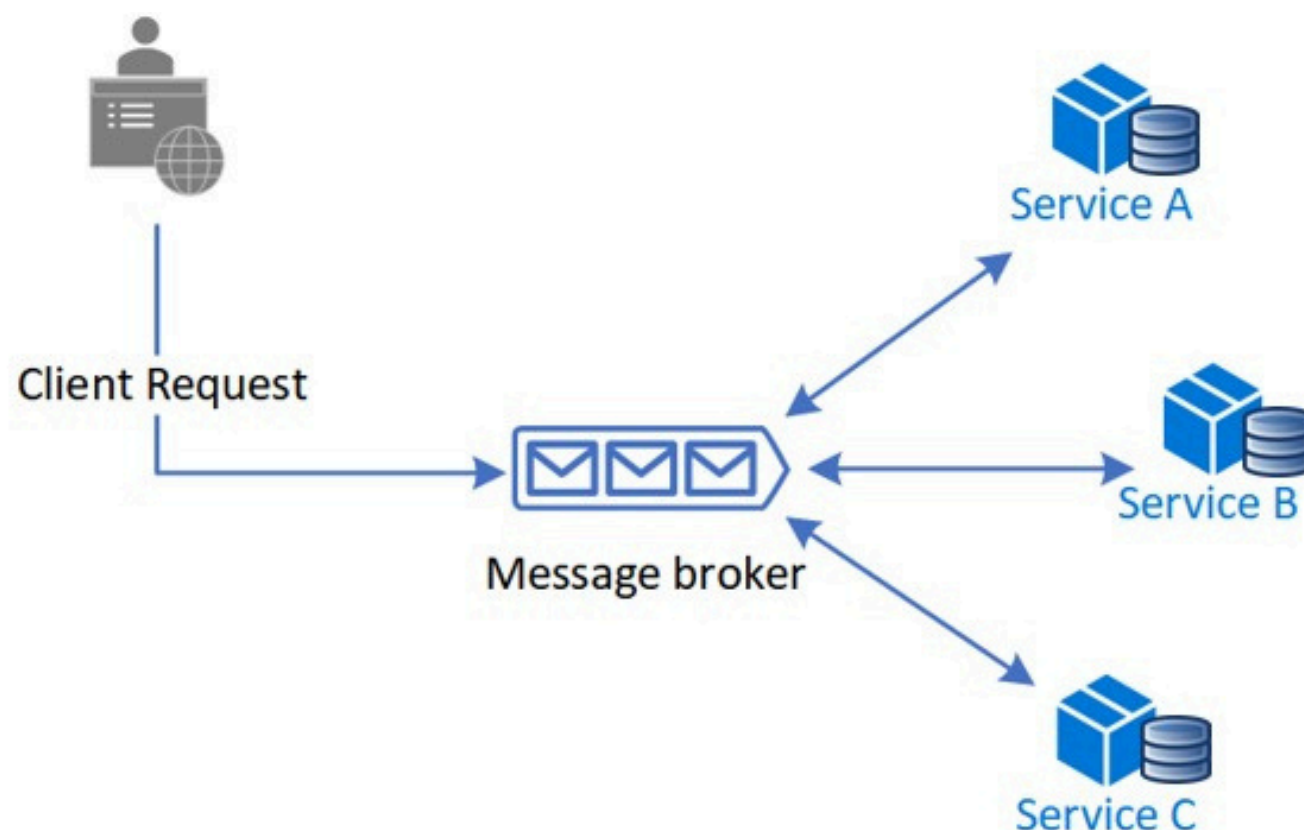
Implement a service registry to automatically locate and register microservices. This helps in dynamic discovery and communication between services.

CIRCUIT BREAKER PATTERN



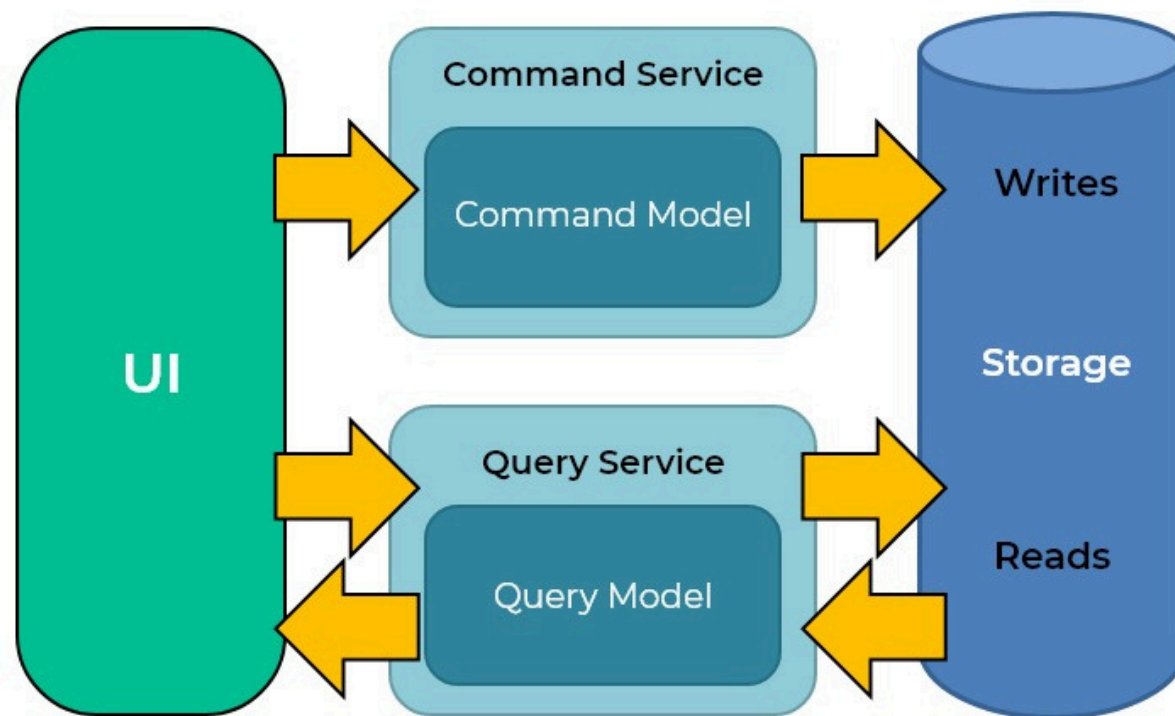
Prevent cascading failures by using a circuit breaker that can temporarily stop requests to a failing service and provide fallback mechanisms.

SAGA PATTERN



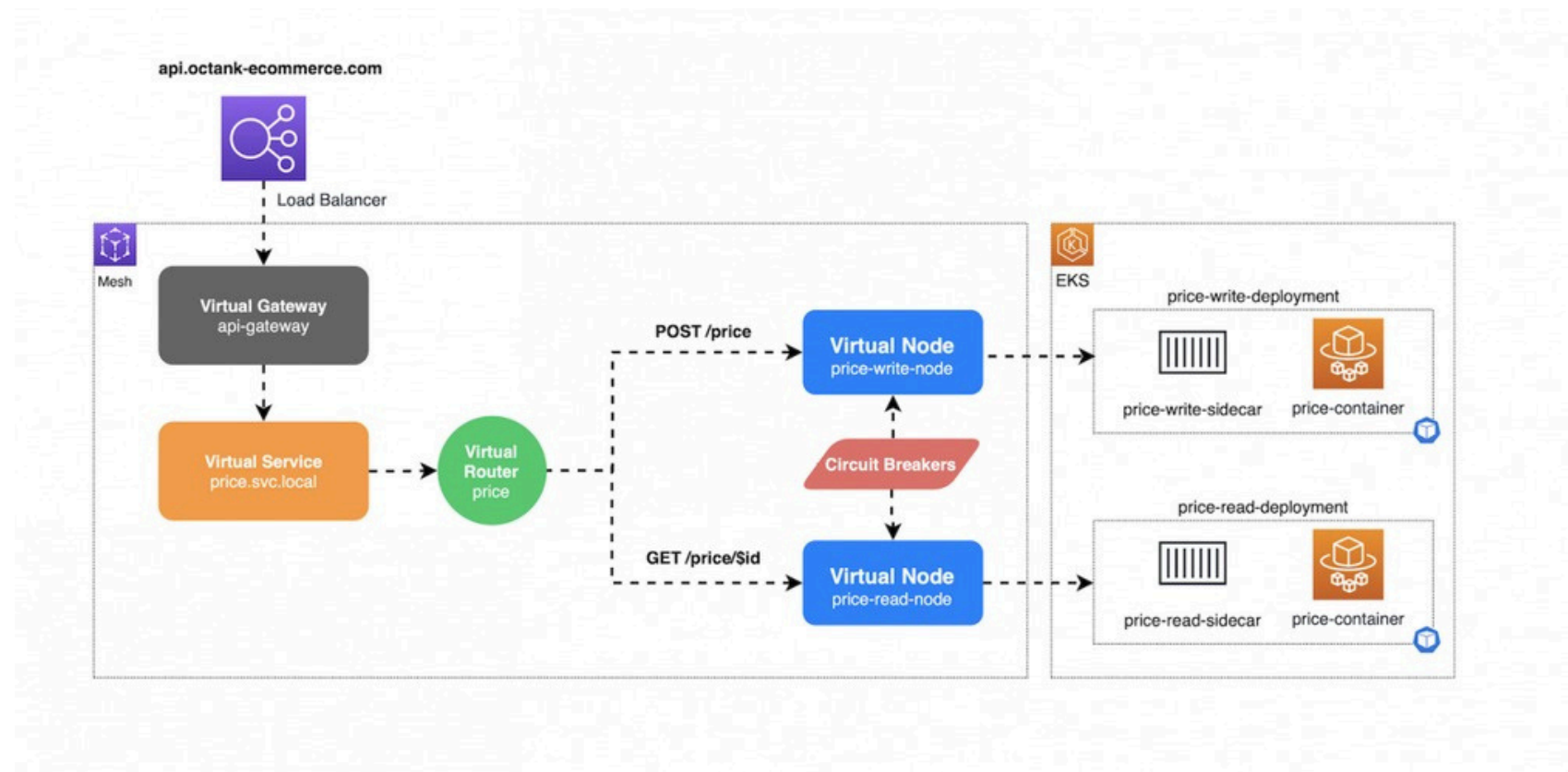
Manage long-lived transactions across multiple microservices by breaking them down into a sequence of smaller, local transactions.

CQRS PATTERN



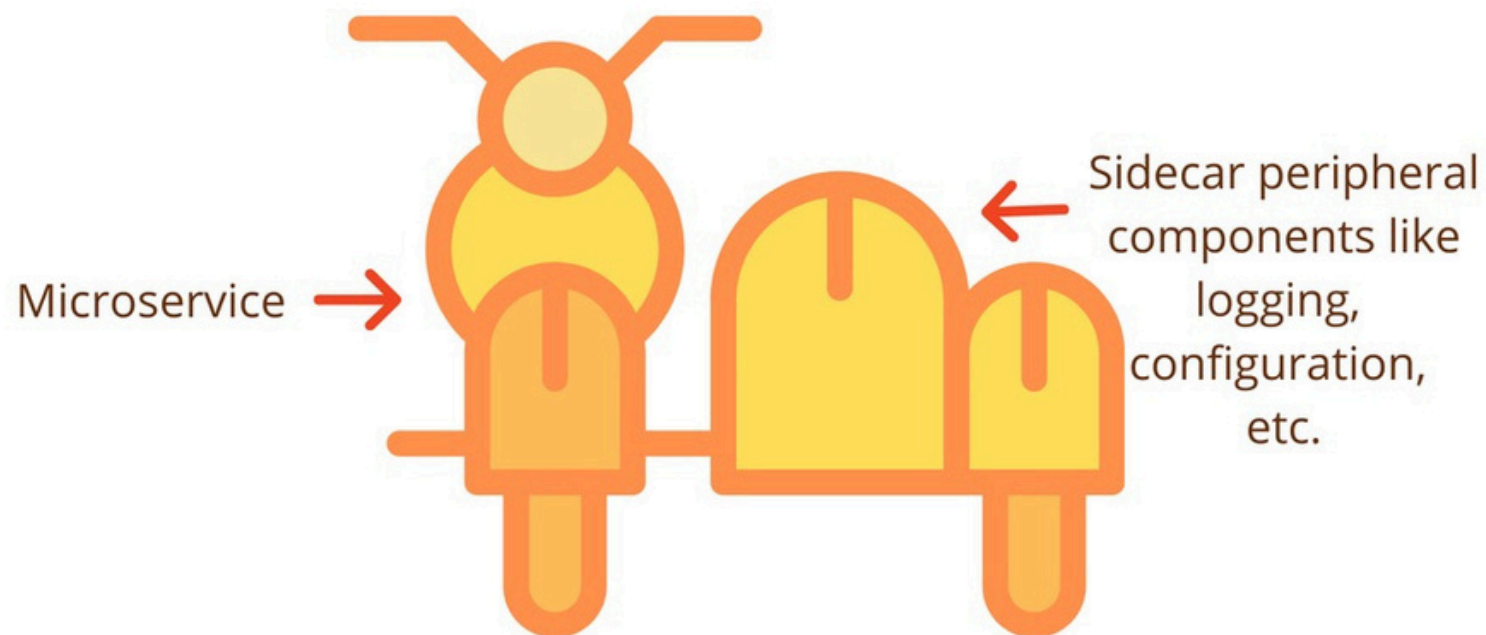
Separate the read and write responsibilities of a system, allowing for optimized performance and scalability.

BULKHEAD PATTERN



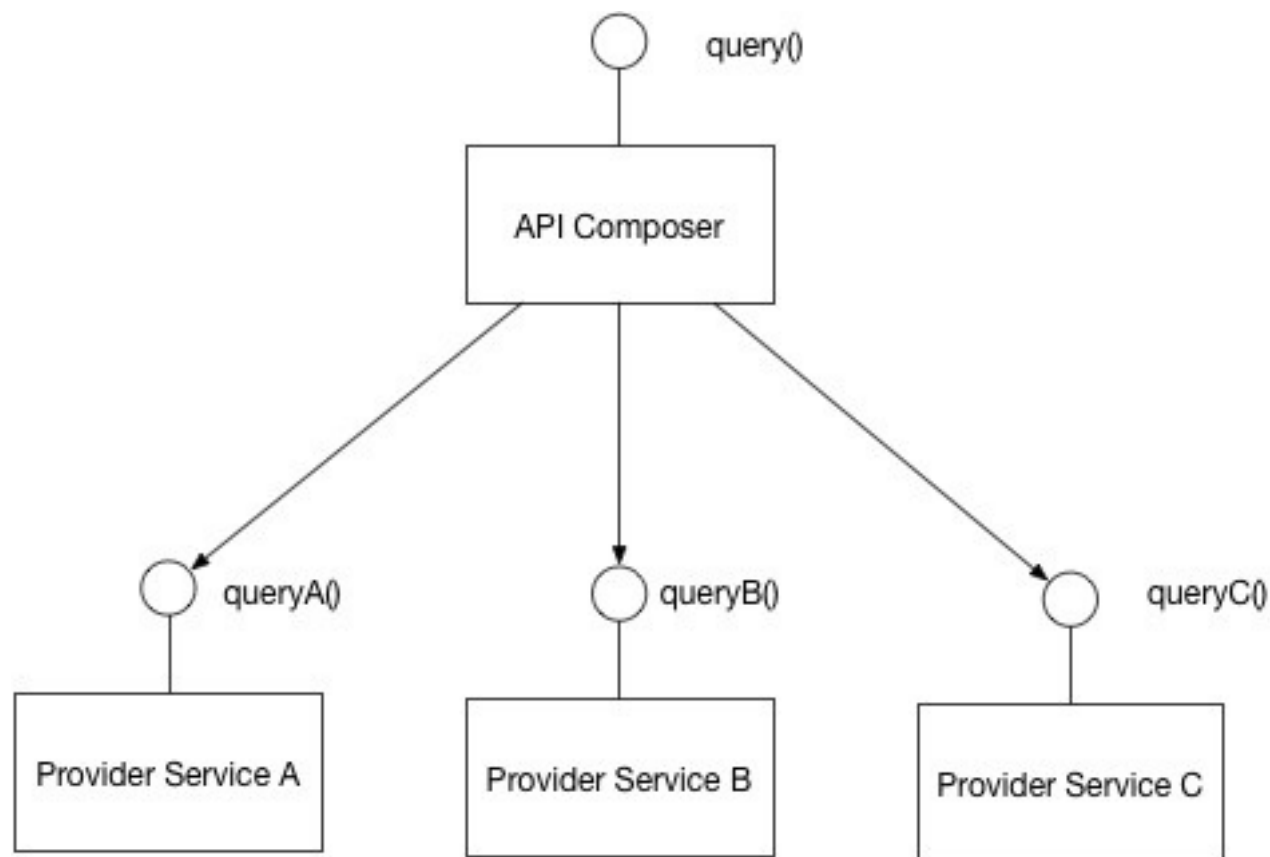
Isolate failures within separate sections to prevent them from affecting the entire system.

SIDECAR PATTERN



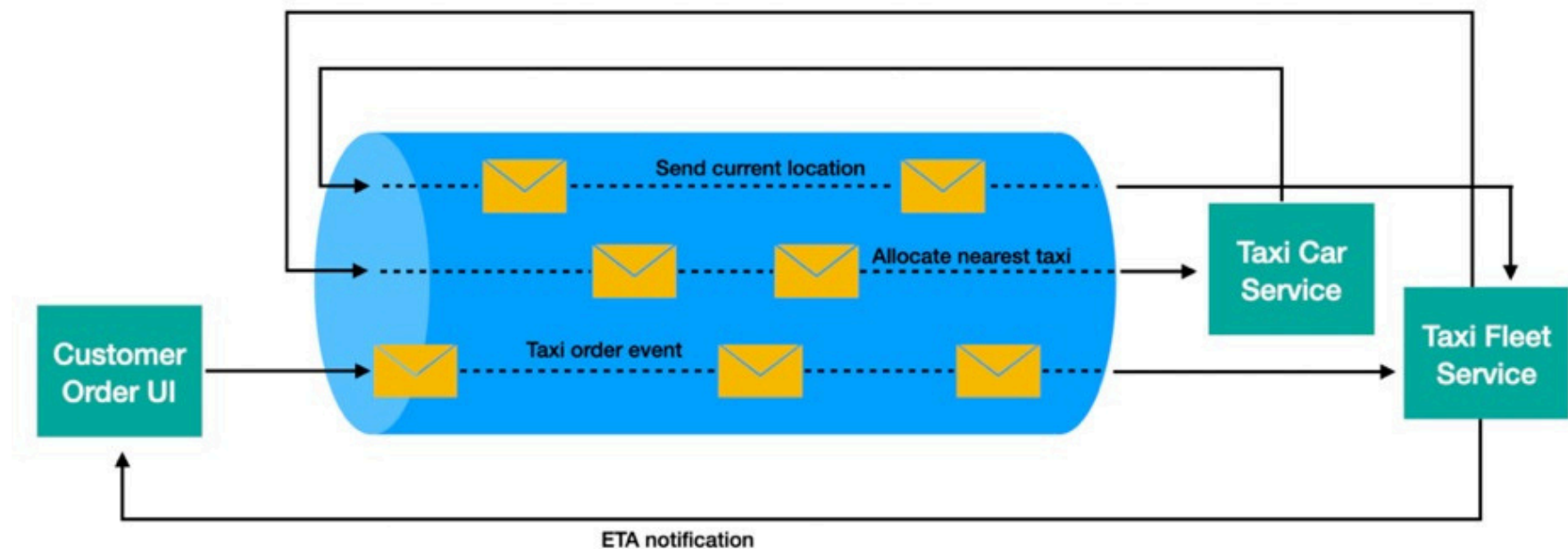
Attach a separate microservice (sidecar) to handle specific tasks like monitoring, logging, or authentication.

API COMPOSITION PATTERN



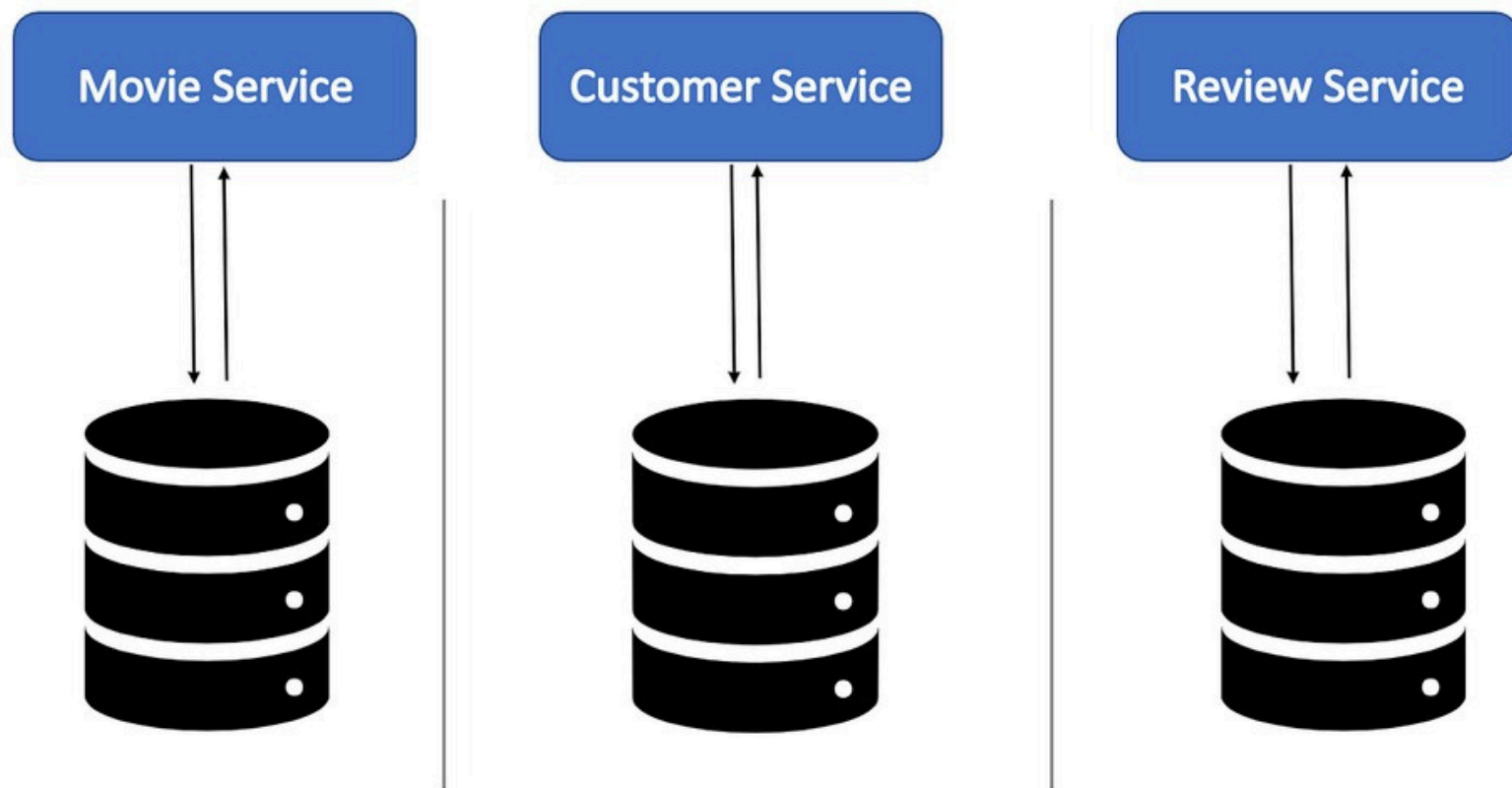
Combine multiple microservices to create a more complex and feature-rich API for clients.

EVENT-DRIVEN ARCHITECTURE PATTERN



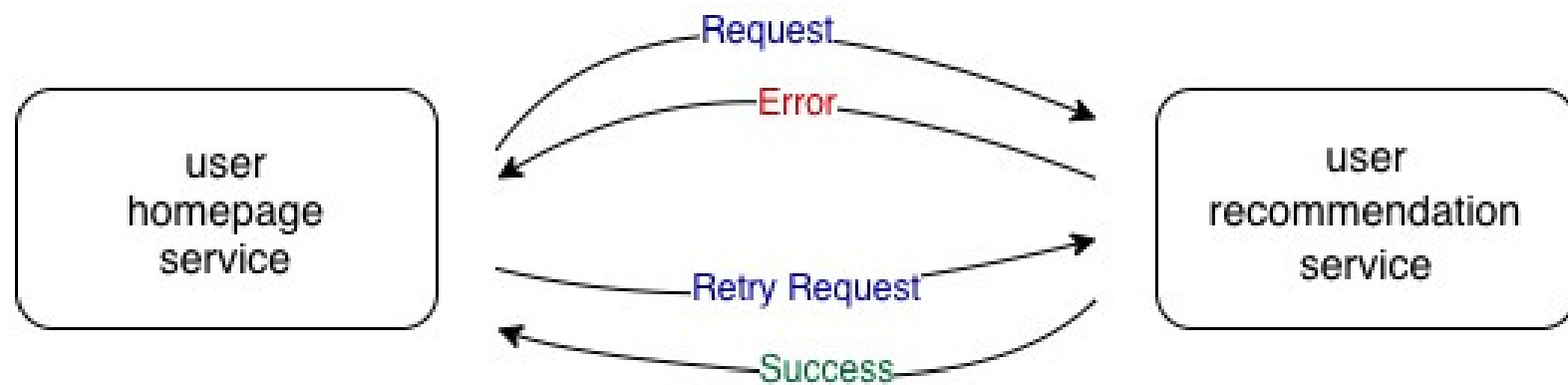
Communicate between microservices through events, enabling loose coupling and scalability.

DATABASE PER SERVICE PATTERN



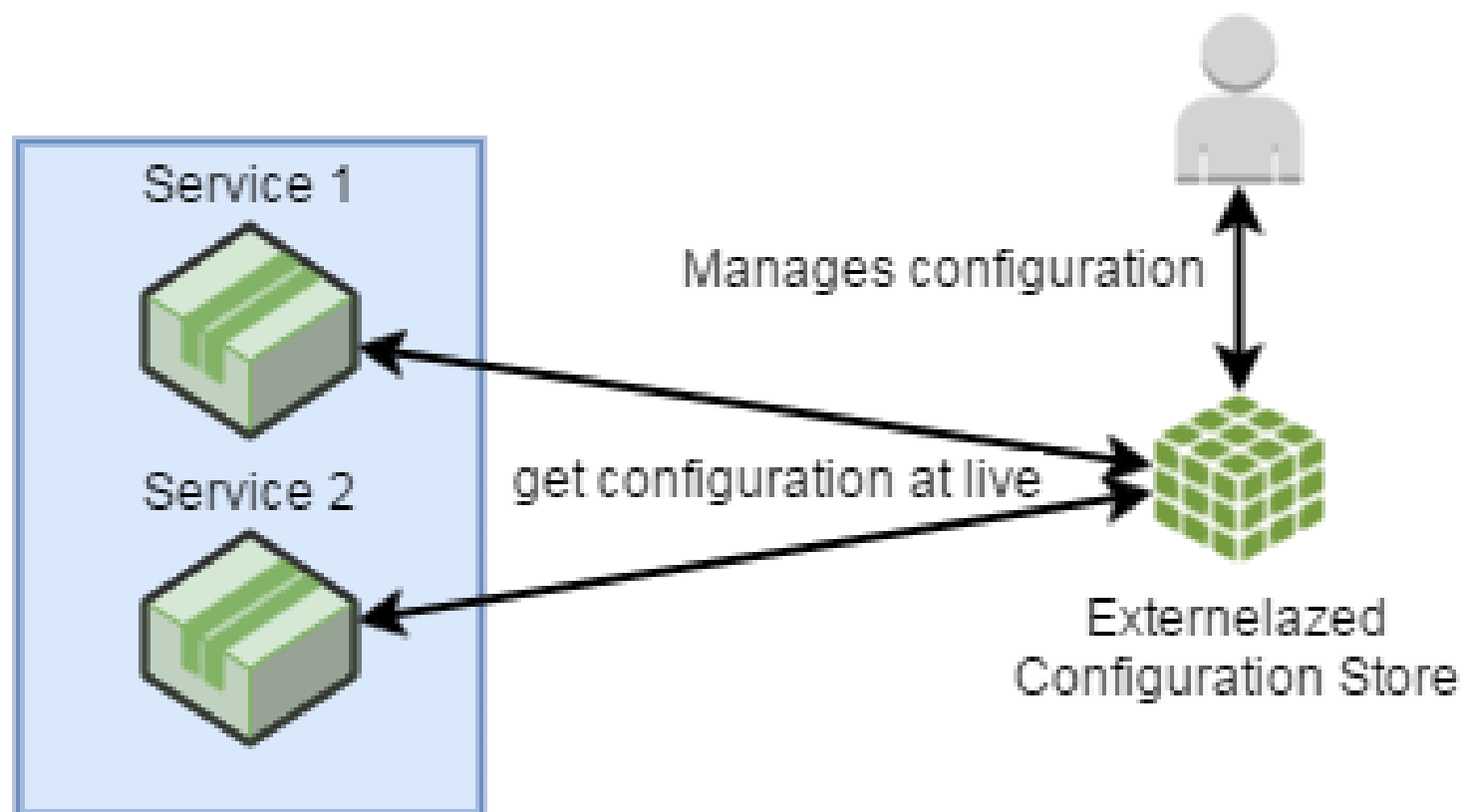
Each microservice has its own dedicated database to ensure loose coupling and autonomy.

RETRY PATTERN



Automatically retry failed operations to improve the chances of success.

CONFIGURATION EXTERNALIZATION PATTERN

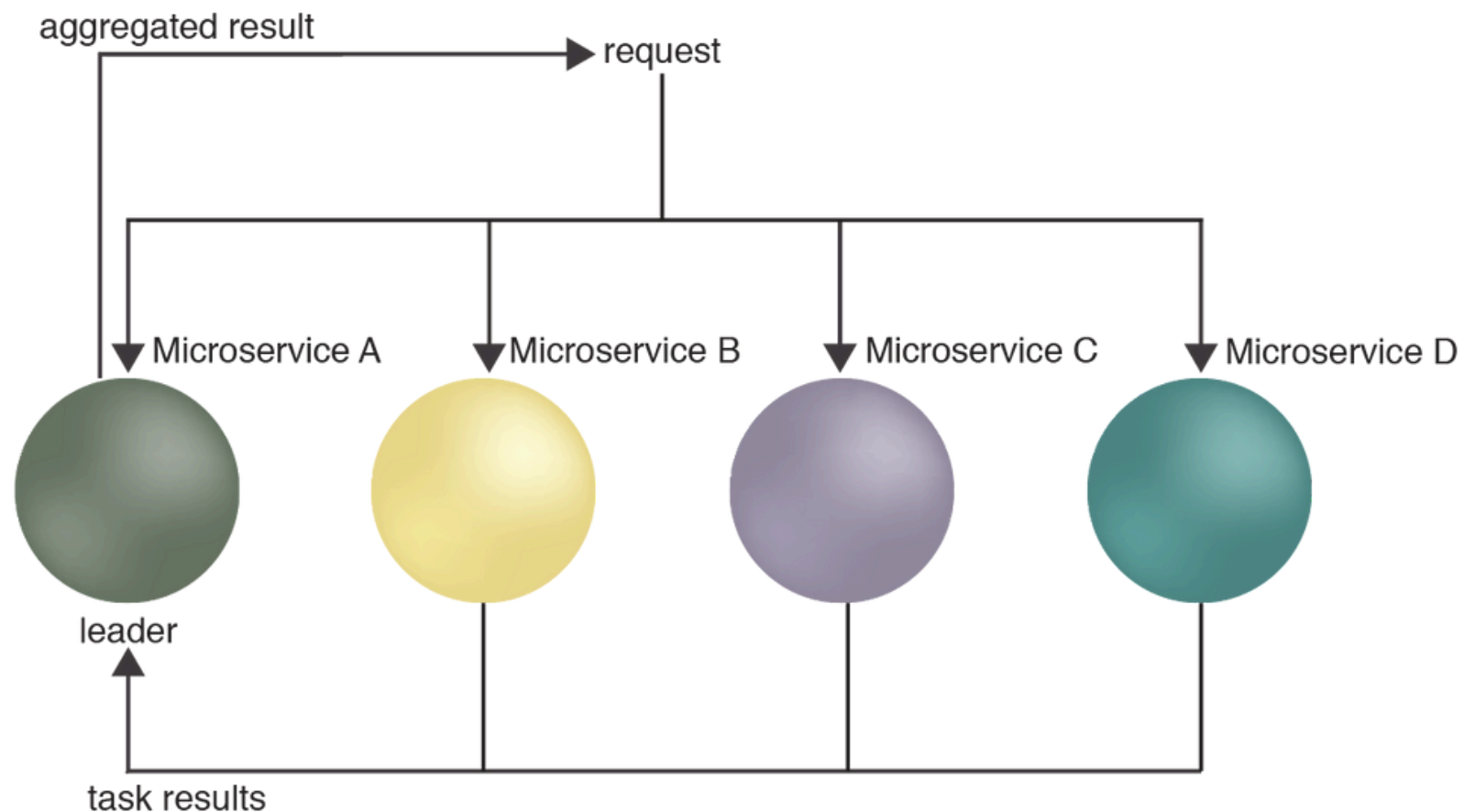


Store configuration settings outside the codebase for easier management and updates.

The diagram illustrates a data integration architecture. At the top left, a box labeled "Microsoft" contains a database icon and two smaller boxes labeled "B" and "C". A double-headed arrow connects the database to these boxes. Below this, a blue cloud icon labeled "Event Source = Sink Connectors" is connected to the database by a double-headed arrow labeled "Change Data Capture (CDC)". To the right of the Event Source are two boxes: "Stream (Reads)" and "Stream (Writes)", each containing five small blue squares. Arrows point from the Event Source to both stream boxes. To the right of the streams is a circular icon labeled "Service A" containing five small blue squares. A double-headed arrow connects the "Stream (Reads)" box to Service A. A double-headed arrow connects Service A to the "Stream (Writes)" box. To the right of Service A is a blue box labeled "Fully replaces module A.". Above Service A is a light blue vertical rectangle labeled "Proxy". A double-headed arrow connects the Proxy to Service A. A double-headed arrow connects the Proxy to a computer icon labeled "Client". A double-headed arrow connects the Proxy to the "Stream (Reads)" box, with the label "Order Writes And Reads" above it. A double-headed arrow connects the Proxy to the "Stream (Writes)" box, with the label "Reads From Module A" above it and "Writes to Module A" below it.

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.

Thank
you!



FOLLOW

For More