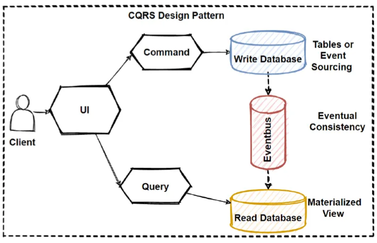
Microservice Design Patterns

CQRS Pattern



→ CQRS = Command Query Responsibility Separation

→ The CQRS pattern is used to separate the read and write operations of an application into different models.

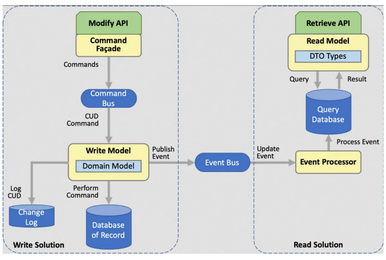
It allows for different models to be optimized for their respective tasks.

By separating the read and write operations, we can choose the appropriate storage and processing mechanisms for each.

→ For example, consider a banking application where users can view their account balance, transaction history, and make transfers.

The read operations (like view account balance) can be optimized for high availability and low latency, while the write operations (e.g., make a transfer) can be optimized for data consistency and accuracy.

→ Here is how a CQRS pattern looks like in Microservice architecture:-



→ CQRS is best suited for scenarios where read and write operations are heavily used and need to be optimized separately.

CQRS pattern is used in systems that require high scalability and performance, such as social media platforms.

→ In microservice architecture, CQRS pattern is typically implemented using event sourcing.

→ Another example we can: E-commerce App

Let's say the e-commerce app has a feature where users can search for products.

In this case, the Command side would handle the creation, updation, and deletion of products. This would involve services that allow users to add products to the system, update product details, and delete products from the system.

On the other hand, the Query side would be responsible for handling user requests for product information. This would involve services that allow users to search for products based on various criteria, such as name, category, price range, etc. The Query side would be optimized for fast read operations, and the data would be stored in a denormalized form to improve query performance.

To implement this using CQRS in a Spring Boot microservice architecture, we could have separate microservices for the Command and Query sides. The Command microservice would handle product creation, updates and deletion. And, it would publish events to a Kafka topic wherever a product is created, updated or deleted.

The Query microservice would subscribe to this Kafka topic and update its own data store with the latest product information. It would expose a set of REST endpoints that allows users to search for products based on various criteria. The Query microservice could use Elasticsearch or another search engine to provide fast search capabilities.

By separating the Command and the Query sides, we can optimize each side for its specific use case. The Command side can focus on ensuring data consistency and enforcing business rules, while the Query side can focus on providing fast, responsive queries to users.

→ Task for Implementation