Implementation Pattern: CQRS

# Purpose of this document

|  |  |  |
| --- | --- | --- |
| What |  | This document explains the CQRS Pattern for managing data, and when it should be used |
|  |  |  |
| Whom |  | Database Architects or Developers responsible for a datastore and its API |
|  |  |  |
| When |  | There are performance issues with the datastore and/or the associated API |

# Summary

For most databases, the read/write split is close to 80/20% where most activity is read but write is more computationally intensive. It can be difficult to effectively design the datastore and API to support both kinds of workloads efficiently.

CQRS is a pattern that separates the read and write operations into their own separate APIs, and even their own separate datastores, in order to improve performance. This allows the Database Architect to design a separate schema for each that will optimize the kind of work that is being done, and the API Developer to isolate their validation logic from their query logic for better readability and performance.

[Event-driven architecture](https://docs.google.com/document/d/15SWeFQuVmeYbBuFYwKnFto9PqFJ24Bked2C0yM4UvcI/edit#heading=h.p3vlgyzakx2n) is used to keep the Read and Write datastores in sync: the Write datastore produces events whenever a write operation is completed, which the Read datastore is subscribed to.

This approach comes with a drastic increase in complexity, so is only appropriate in situations where there are already performance problems that could be resolved by isolating read and write operations.

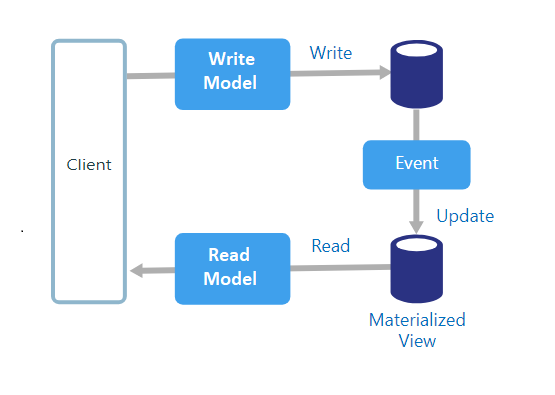
# Challenges with traditional CRUD

In traditional architectures, the same data model is used to query and update a database. That's simple and works well for basic CRUD operations. In more complex applications, however, this approach can become unwieldy. For example, on the read side, the application may perform many different queries, returning data transfer objects (DTOs) with different shapes. Object mapping can become complicated. On the write side, the model may implement complex validation and business logic. As a result, you can end up with an overly complex model that does too much.

CQRS can help resolve these challenges.

# What is CQRS?

The Command and Query Responsibility Segregation (CQRS) pattern separates read and update operations for a datastore. Implementing CQRS in your application can maximize its performance, scalability, and security. The flexibility created by using CQRS allows a system to better evolve over time and prevents update commands from causing merge conflicts at the domain level.



# When should I use CQRS?

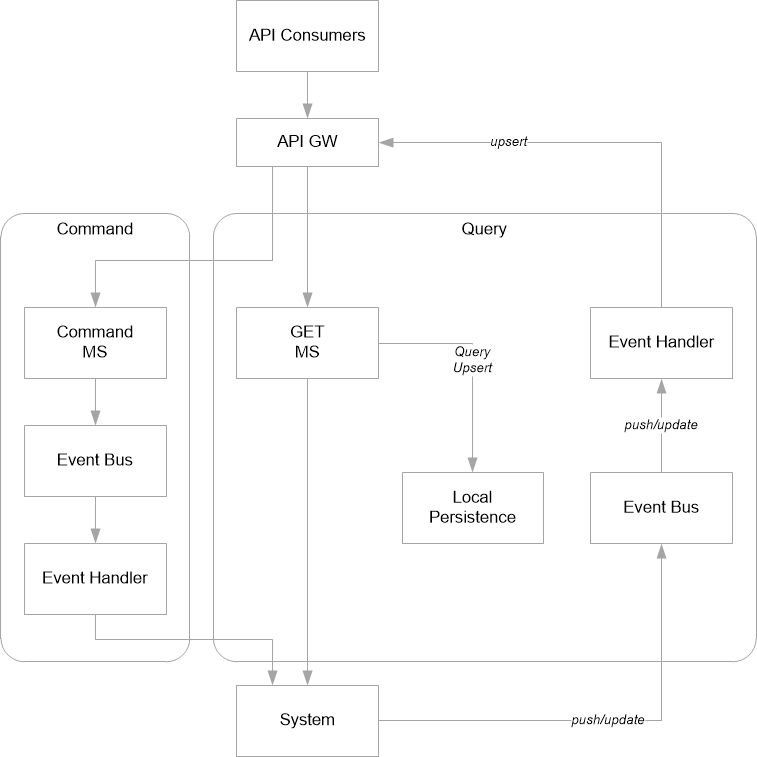
CQRS adds a lot of complexity, and should only be used where the performance value provided would outweigh the cost of the increased complexity. If you answer yes to any one of the following questions, CQRS may be a helpful pattern to adopt:

* The datastore/API is not currently meeting performance needs
* Optimizing the schema further would be difficult to do in a way that worked for both Read and Write operations
* The implementation today executes transformations on the data depending on context, or has multiple ways of representing the data
* The validation logic to submit a write request is taking up a lot of code, and is consuming too much runtime compute
* Permissions need to be managed differently for read and write operations

# How do I implement CQRS?

The PartyManagement API for MDM utilizes this pattern. The high-level steps are:

1. Separate Microservice between Command and Query
2. Implement an Event bus for Command
3. Implement local persistence for Query



# About this document

Authors

* [Katie Peters](mailto:katie.peters@telus.com) [Min Kim](mailto:min.kim@telus.com)

EA Reviewers

External Consultants

Related Links

* [Event-Driven Architecture](https://docs.google.com/document/d/15SWeFQuVmeYbBuFYwKnFto9PqFJ24Bked2C0yM4UvcI/edit#heading=h.p3vlgyzakx2n): used to synchronize data between the read and write datastores

References

* [Martin Fowler - CQRS](https://martinfowler.com/bliki/CQRS.html#:~:text=CQRS%20stands%20for%20Command%20Query,you%20use%20to%20read%20information.): an explanation of the pattern
* [Microsoft - CQRS Pattern](https://docs.microsoft.com/en-us/azure/architecture/patterns/cqrs): an explanation of the pattern