Microservice Architecture (/index.html)

Supported by Kong (https://konghq.com/)

pattern (/tags/pattern) service collaboration (/tags/service collaboration) implementing queries (/tags/implementing queries)

Pattern: Command Query Responsibility Segregation (CQRS)

Context

You have applied the Microservices architecture pattern (../microservices.html) and the Database per service pattern (database-per-service.html). As a result, it is no longer straightforward to implement queries that join data from multiple services. Also, if you have applied the Event sourcing pattern (event-sourcing.html) then the data is no longer easily queried.

Problem

How to implement a query that retrieves data from multiple services in a microservice architecture?

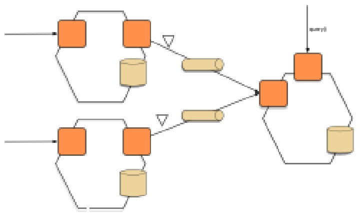
Solution

Define a view database, which is a read-only replica that is designed to support that query. The application keeps the replica up to data by subscribing to Domain events (domain-event.html) published by the service that own the data.

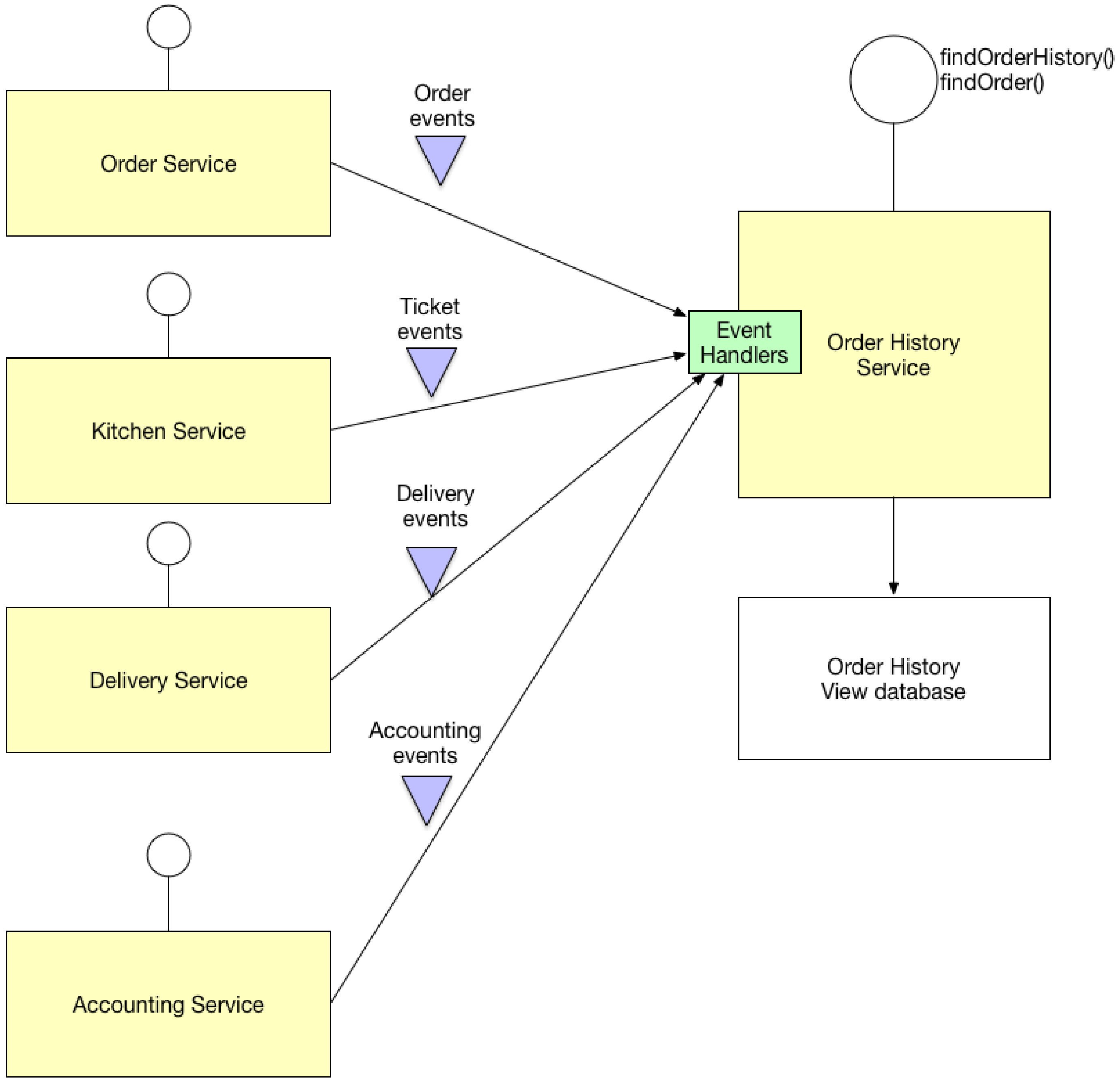
Want to learn more about this pattern?

Take a look at my self-paced, online bootcamp (https://chrisrichardson.net/virtual-bootcamp-distributed-data-management.html) that teaches you how to use the Saga, API Composition, and CQRS patterns to design operations that span multiple services.

The regular price is $395/person but use coupon JUNVCEJE to sign up for $195 (valid until February 1st, 2023)



(https://chrisrichardson.net/virtual-bootcamp-distributed-data-management.html)



Examples

- My book's FTGO example application has the `Order History Service` (https://github.com/microservices-patterns/ftgo-application#chapter-7-implementing-queries-in-a-microservice-architecture), which implements this pattern.
- There are several Eventuate-based example applications (http://eventuate.io/exampleapps.html) that illustrate how to use this pattern.

Resulting context

This pattern has the following benefits:

- Supports multiple denormalized views that are scalable and performant
- Improved separation of concerns = simpler command and query models
- Necessary in an event sourced architecture

This pattern has the following drawbacks:

- Increased complexity

- Potential code duplication
- Replication lag/eventually consistent views

Related patterns

- The Database per Service pattern (database-per-service.html) creates the need for this pattern
- The API Composition pattern (api-composition.html) is an alternative solution
- The Domain event (domain-event.html) pattern generates the events
- CQRS is often used with Event sourcing (event-sourcing.html)

See also

- Eventuate (http://eventuate.io), which is a platform for developing transactional business applications.

Learn more

- My book Microservices patterns (/book) describes this pattern in a lot more detail
- Take a look at my self-paced, online bootcamp (https://chrisrichardson.net/virtual-bootcamp-distributed-data-management.html) that teaches you how to use the Saga, API Composition, and CQRS patterns to design operations that span multiple services.

pattern (/tags/pattern) service collaboration (/tags/service collaboration) implementing queries (/tags/implementing queries)

Tweet

Follow @MicroSvcArch

ALSO ON **MICROSERVICES**

Idempotent Consumer
3 years ago · 3 comments
Pattern: Idempotent Consumer Context In an enterprise application, it's …

API gateway pattern
a year ago · 1 comment
An API gateway acts a single entry point into the application, routing and …

Sagas
5 years ago · 38 comments
Implement transactions using a saga, which is sequence of local …

Self-c
3 years
Design
can res
synchr

15 Comments

1 **Login ▾**

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

6 Share Best Newest Oldest

An0nym0usC0ward

3 years ago

That's not CQRS. And CQRS doesn't solve the problem stated in this pattern.

You use CQRS when handling commands to update a data store so that writing and reading operations could be symmetric would be excessively expensive. Or when you need to work with a predefined data store which is more just an event log, but need to expose a query API that would need to aggregate the data in the log. Then you use CQRS. Whether the data comes from one or multiple microservices is not relevant.

When maintaining the data store so that it is fit for the kind of querying you want to perform is sufficiently cheap, you always do it, rather than just logging the events - it's cheaper overall, you only update the data store once per command, whereas especially with microservices and web based apps you typically query the data store orders of magnitude more often.

You don't necessarily implement CQRS at the database level, or at the database query level. Conceptually, you always implement it by using two different models, one for updates and one for queries. You populate a model specifically designed to make updates easy from commands, then serialize it, and you populate a distinctly designed model, designed to make the life of querying clients easy, from data store queries, and the put it on the wire and send it to clients. How you populate each of the two distinct models, whether you use a database view, an in-memory cache, aggregate SQL queries or some other mechanism is irrelevant. As long as you stick to the two distinct models pattern it's CQRS.

2 0 • Reply • Share ›

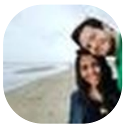
DM

David Martines ➜ An0nym0usC0ward

🕒 8 months ago

I agree that this pattern, as described, might be more accurately titled "View Database". It is related to CQRS in that it's an implementation of separating the Query from the Command. Conceptually, CQRS doesn't require this particular implementation.

1 0 • Reply • Share ›



David Gonzalez Shannon

🕒 4 years ago

This looks expensive. Do you have an article about monetary costs of running a microservice architecture with database per service?

1 database/service + backups + 1 database / view = $$$$

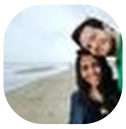
2 0 • Reply • Share ›

Abdelrahman Salem ➜ David Gonzalez Shannon

🕒 4 years ago

well that shouldn't be an issue, as microservices will have smaller hardware specs as they suppose to achieve a single mission

6 0 • Reply • Share ›



David Gonzalez Shannon ➜ Abdelrahman Salem

🕒 4 years ago

I guess you're right, although we might have different opinions on what we consider expensive. If we applied Database per service to our 20 microservices, using instances with decent network speed, it would cost around 3k/month on aws, which isn't a farfetched budget for a database, but still... I'm not sure if that added complexity is worth it.

0 0 • Reply • Share ›

Florin Marin ➜ David Gonzalez Shannon

🕒 4 years ago

The pattern is Database per

Service, regarding deployment you are applying service per host I think. Anyway it all depends on your infrastructure but you can lower the cost if you use 2-3 database servers with replication that will hold all your 20 databases for your microservices. If you dont have too much traffic you can go for a database hosted in cloud solution and you will pay for read/writes

0 0 • Reply • Sh

JP **José Pereira**

2 years ago

message-oriented approach can be used instead of events?

1 0 • Reply • Share ›

Tridib Bolar

2 years ago

"The application keeps the replica up to data by subscribing to Domain events published by the service that own the data" (2nd sentence of Solution paragraph) - I believe it must be 'up to date' instead of 'up to data'

0 0 • Reply • Share ›