Microservice Architecture (/index.html)

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

Pattern: Command Query Responsibility Segregation (CQRS)

Context

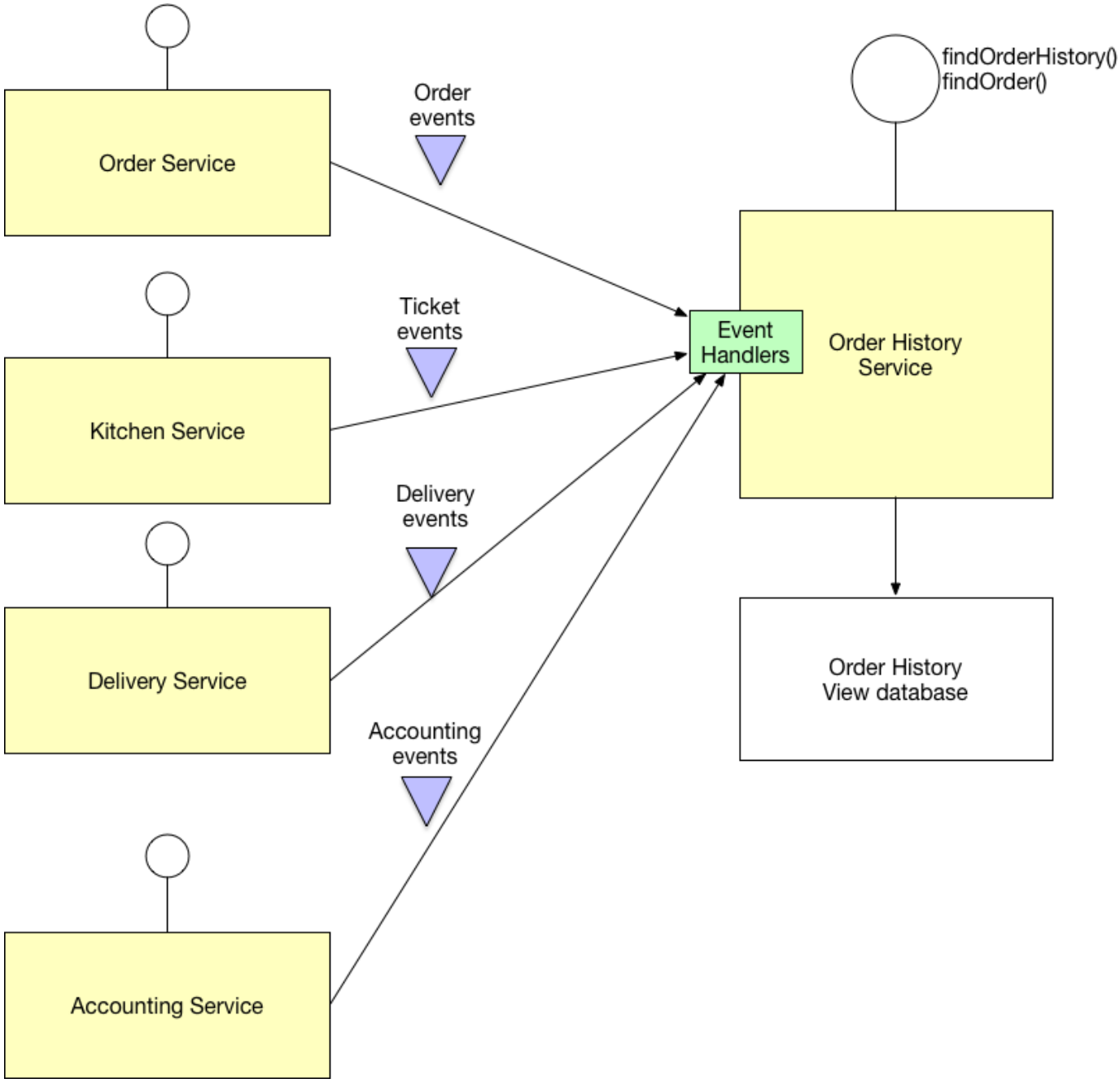
You have applied the Microservices architecture pattern (../microservices.html) and the Database per service pattern (database-per-service.html). As a result, it is no longer straightforward to implement queries that join data from multiple services. Also, if you have applied the Event sourcing pattern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



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.
- My book Microservices patterns (/book) describes this pattern in a lot more detail.