Outbox Pattern with Couchbase Eventing

Serhat Karabulut

Software Engineer @Trendyol

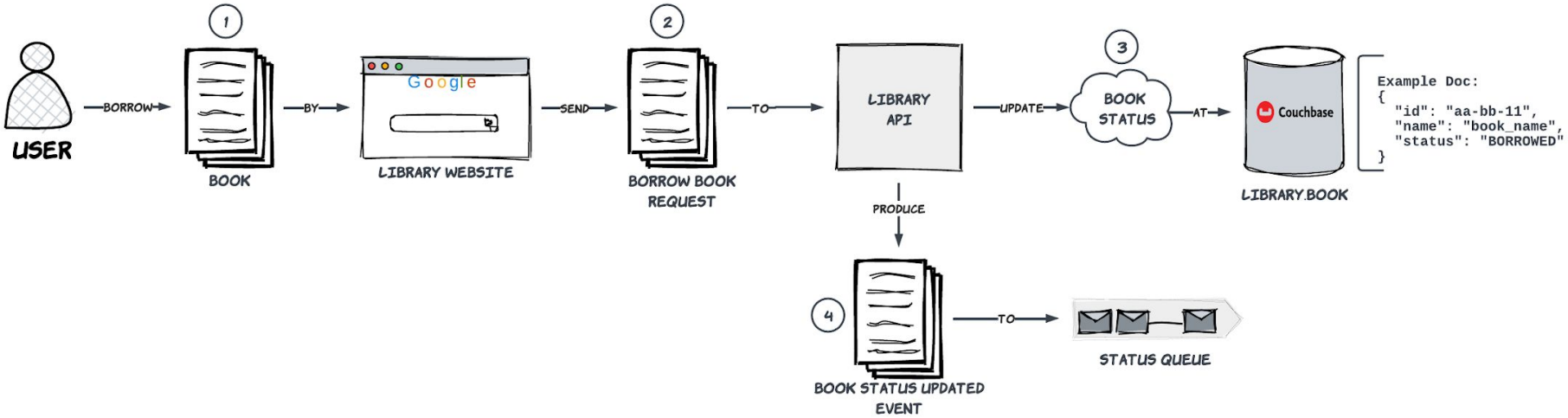
github.com/3n0ugh

Agenda

- Problem
- Outbox Pattern
- DCP
- Kafka Connector
- Couchbase Eventing
- Solution
- Pros
- Cons
- CBEF

Problem

How to atomically update the database and send messages to a message broker?



Outbox Pattern

The Outbox Pattern’s primary goal is to manage asynchronous communication between internal systems while ensuring data integrity.

- The client avoids sending operations directly to internal systems and instead utilizes an outbox.
- The client adds data to the outbox, and this data is later transmitted asynchronously to target systems.
- This ensures reliable asynchronous communication between systems.

DCP

Data Change Patterns (DCP) refer to the ways in which data is modified or updated in a system, especially in the context of a database.

Common patterns include:

- Command Query Responsibility Segregation (CQRS)
- Event Sourcing
- Publish-Subscribe Pattern
- Change Data Capture (CDC)

Kafka Connector

Kafka Connect is a framework in Apache Kafka for connecting external systems such as databases, message queues, and file systems to Kafka. Connectors are the plugins used to achieve this integration.

There are two main types of connectors:

- Source Connectors
- Sink Connectors

Couchbase Eventing Service

The Couchbase Eventing Service is a framework to operate on changes to data in real time. Events are changes to data in the Couchbase cluster.

The changes are:

- Update
- Delete
- Expire

Couchbase Eventing Function

The Eventing Service exposes the ability to write JavaScript code to analyze and manipulate your JSON documents on any type mutations.

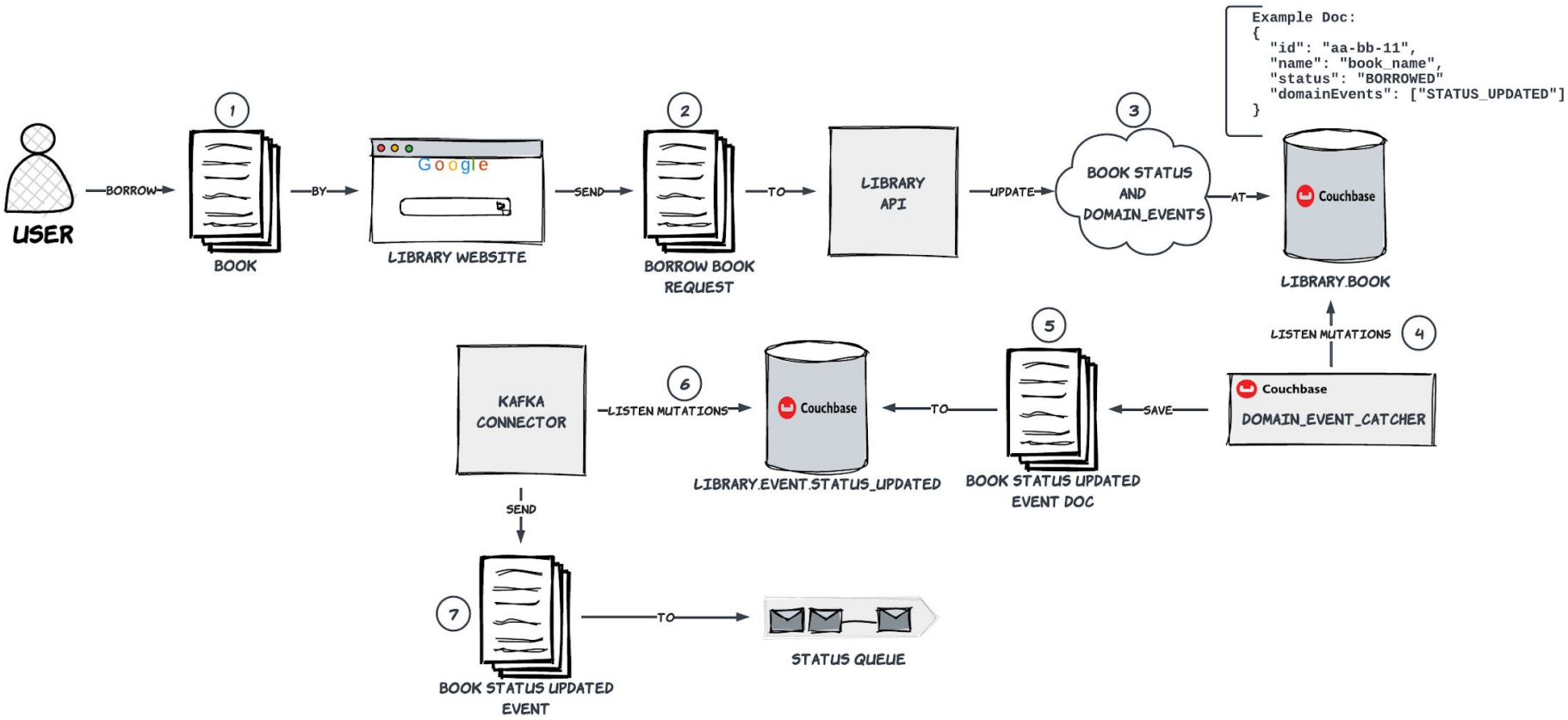
There are two handlers to manage that document changes:

- **OnUpdate:** After any document create or update process calls this handler.
- **OnDelete:** After any document expiration or delete process calls at this handler.

Couchbase Eventing Function vs SQL Triggers

	Couchbase Eventing Function	**SQL Triggers**
Architecture	Distributed and Serverles	Database Specific
Language	Javascript	SQL
Event Model	Event Driven	Database Events
Data Access	Full Access to Data	Restricted Access
Use Cases	Business Logic	Data Integrity and Validation

Solution


Example Doc:
{
"id": "aa-bb-11",
"name": "book_name",
"status": "BORROWED"
"domainEvents": ["STATUS_UPDATED"]
}
USER
BORROW
1
BOOK
BY
Google
LIBRARY WEBSITE
SEND
2
BORROW BOOK REQUEST
TO
LIBRARY API
UPDATE
3
BOOK STATUS AND DOMAIN_EVENTS
AT
Couchbase
LIBRARY.BOOK
LISTEN MUTATIONS
4
Couchbase
DOMAIN_EVENT_CATCHER
SAVE
5
BOOK STATUS UPDATED EVENT DOC
TO
Couchbase
LIBRARY.EVENT.STATUS_UPDATED
LISTEN MUTATIONS
6
KAFKA CONNECTOR
SEND
7
BOOK STATUS UPDATED EVENT
TO
STATUS QUEUE

Pros

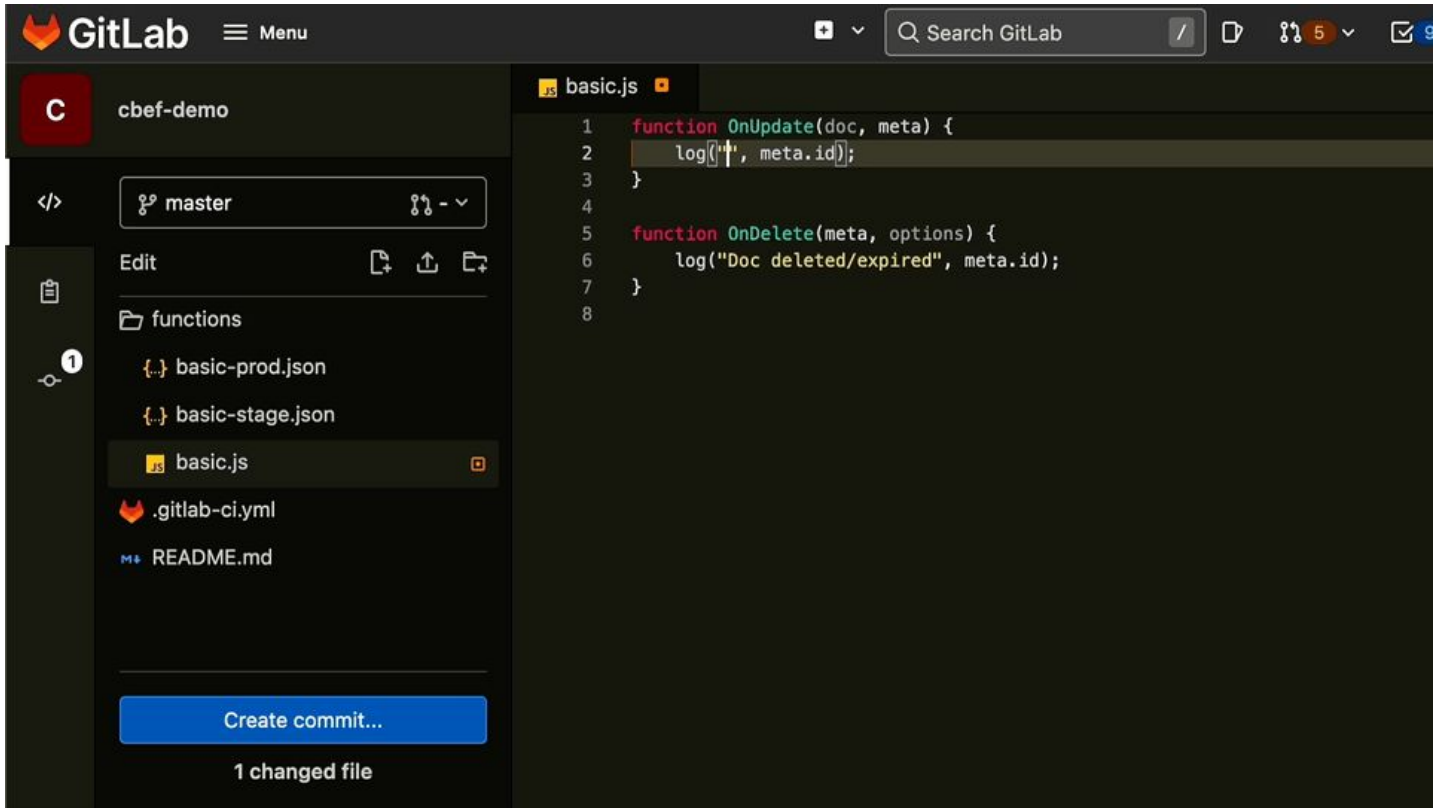
- Consistency (At least Once send message)
- More decoupling
- Availability

Cons

- Might publish a message more than once
- Complexity
- Eventing Function Gitops

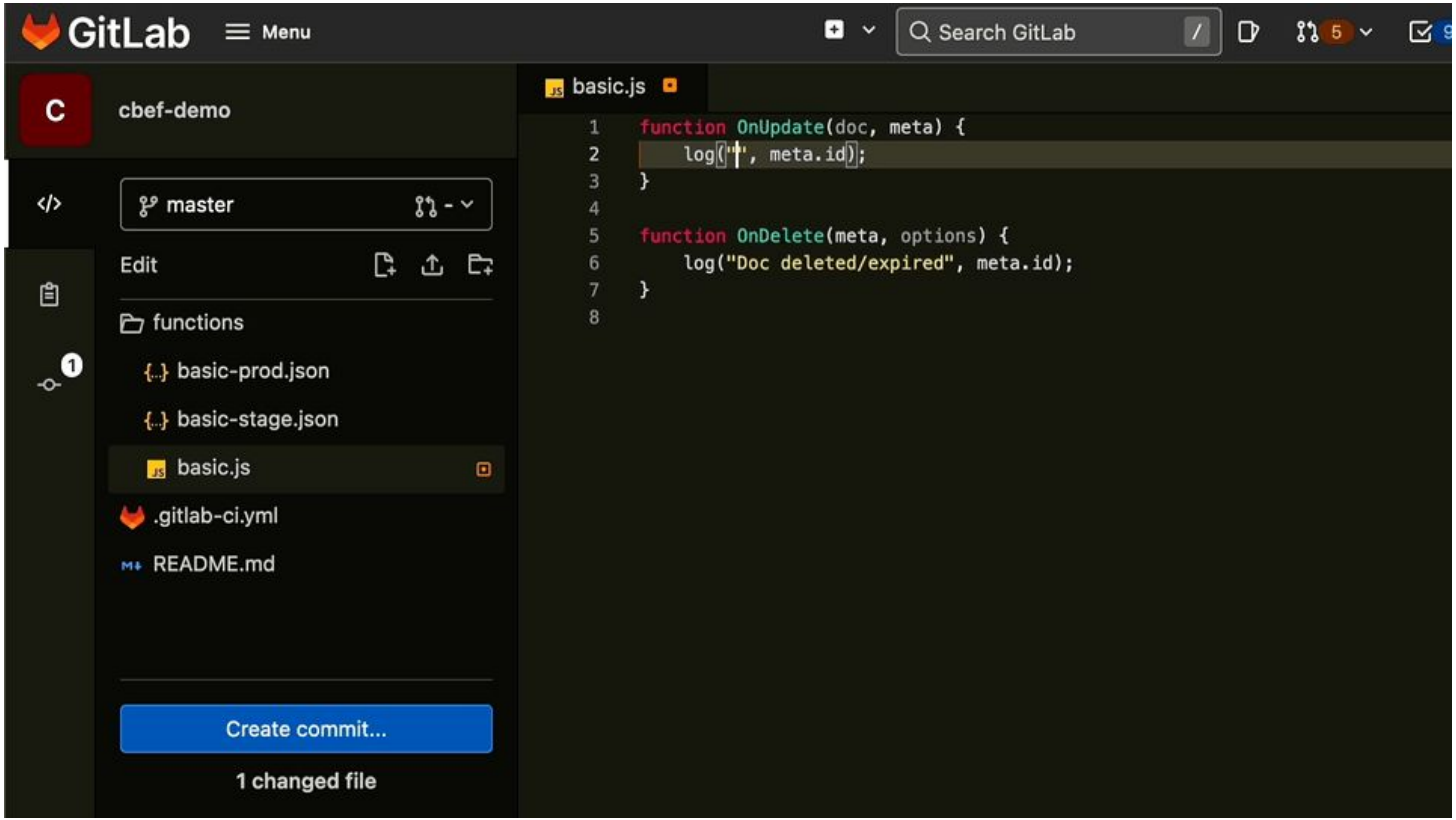
CBEF

https://github.com/Trendyol/cbef



CBEF

https://github.com/Trendyol/cbef



Thank You For Your Attention