

[Microservice Architecture \(/index.html\)](/index.html)

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

🔖 [pattern \(/tags/pattern\)](/tags/pattern) 🔖 [transactional messaging \(/tags/transactional messaging\)](/tags/transactional%20messaging) 🔖 [service design \(/tags/service design\)](/tags/service%20design) 🔖 [inter-service communication \(/tags/inter-service communication\)](/tags/inter-service%20communication)

Pattern: Transactional outbox

Also known as

Application events

Context

A service command typically needs to update the database **and** send messages/events. For example, a service that participates in a saga (</patterns/data/saga.html>) needs to atomically update the database and sends messages/events. Similarly, a service that publishes a domain event (<domain-event.html>) must atomically update an aggregate (<aggregate.html>) and publish an event.

A service must atomically update the database and send messages in order to avoid data inconsistencies and bugs. However, it is not viable to use a traditional distributed transaction (2PC) that spans the database and the message broker to atomically update the database and publish messages/events. The message broker might not support 2PC. And even if does, it's often undesirable to couple the service to both the database and the message.

But without using 2PC, sending a message in the middle of a transaction is not reliable. There's no guarantee that the transaction will commit. Similarly, if a service sends a message after committing the transaction there's no guarantee that it won't crash before sending the message.

In addition, messages must be sent to the message broker in the order they were sent by the service. They must usually be delivered to each consumer in the same order although that's outside the scope of this pattern. For example, let's suppose that an aggregate is updated by a series of transactions T_1 , T_2 , etc. This transactions might be performed by the same service instance or by different service instances. Each transaction publishes a corresponding event: $T_1 \rightarrow E_1$, $T_2 \rightarrow E_2$, etc. Since T_1 precedes T_2 , event E_1 must be published before E_2 .

Problem

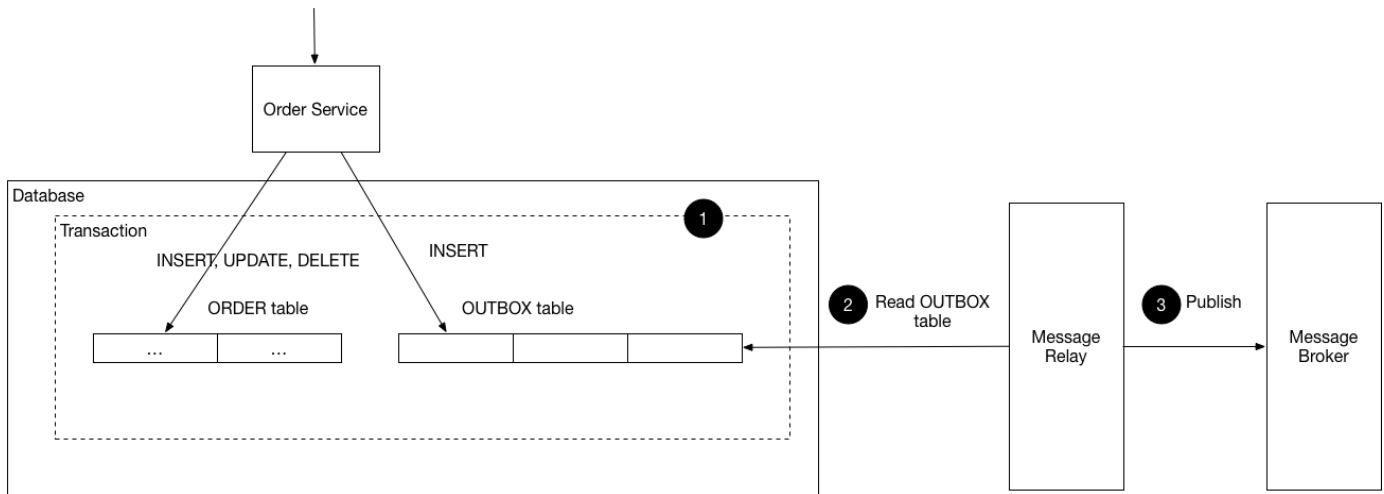
How to reliably/atomically update the database and send messages/events?

Forces

- 2PC is not an option
- If the database transaction commits messages must be sent. Conversely, if the database rolls back, the messages must not be sent
- Messages must be sent to the message broker in the order they were sent by the service. This ordering must be preserved across multiple service instances that update the same aggregate.

Solution

A service that uses a relational database inserts messages/events into an *outbox* table (e.g. `MESSAGE`) as part of the local transaction. An service that uses a NoSQL database appends the messages/events to attribute of the record (e.g. document or item) being updated. A separate *Message Relay* process publishes the events inserted into database to a message broker.



Result context

This pattern has the following benefits:

- 2PC is not used
- Messages are guaranteed to be sent if and only if the database transaction commits
- Messages are sent to the message broker in the order they were sent by the application

This pattern has the following drawbacks:

- Potentially error prone since the developer might forget to publish the message/event after updating the database.

This pattern also has the following issues:





- The Message Relay might publish a message more than once. It might, for example, crash after publishing a message but before recording the fact that it has done so. When it restarts, it will then publish the message again. As a result, a message consumer must be idempotent, perhaps by tracking the IDs of the messages that it has already processed. Fortunately, since Message Consumers usually need to be idempotent (because a message broker can deliver messages more than once) this is typically not a problem.

Related patterns

- The Saga ([saga.html](#)) and Domain event ([domain-event.html](#)) patterns create the need for this pattern.
- The Event sourcing ([event-sourcing.html](#)) is an alternative solution
- There are two patterns for implementing the message relay:
 - The Transaction log tailing ([transaction-log-tailing.html](#)) pattern
 - The Polling publisher ([polling-publisher.html](#)) pattern

Learn more

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

 pattern (</tags/pattern>)  transactional messaging (</tags/transactional messaging>)  service design (</tags/service design>)  inter-service communication (</tags/inter-service communication>)

Tweet

Follow @MicroSvcArch

Copyright © 2023 Chris Richardson • All rights reserved • Supported by Kong (<https://konghq.com/>).

ALSO ON MICROSERVICES

<div>API gateway pattern</div> <div>a year ago · 1 comment</div> <div>An API gateway acts a single entry point into the application, routing and ...</div>	<div>Decompose by business capability</div> <div>3 years ago · 1 comment</div> <div>Decompose an application into services by defining services corresponding to ...</div>	<div>Remote Procedure Invocation (RPI)</div> <div>6 years ago · 1 comment</div> <div>Pattern: Remote Procedure Invocation (RPI) Context You have applied the ...</div>	<div>Exception handling</div> <div>6 years ago · 1 comment</div> <div>Pattern: Exception handling Context the M...</div>
--	--	--	--

Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS

?

Name

3

Share

Best

NewestOldest

KS

Kennedy Sigauke

—

🚩

🕒 4 months ago

After the event is processed by the message relay, how is the outbox table updated atomically?

0

0

• Reply • Share ›

AF

Alexander Fedkin

➔ Kennedy Sigauke

—

🚩


🕒 2 days ago

There is no way to update outbox table atomically (i.e. if and only if message was successfully sent to broker). This is the reason why exactly-once delivery is impossible and message broker should be idempotent

0

0

• Reply • Share ›



Chris Richardson

Mod

—

🚩

➔ Alexander Fedkin


🕒 2 days ago

See the resulting context section. The consumer can detect and discard messages that were published more than once.

1

0

• Reply • Share ›



avinash kumar

—

🚩

🕒 3 years ago

What do you mean by message relay ?

0

0

• Reply • Share ›



Zariga

🕒 3 years ago

Inspired by this concept drafted
leanpub.com/microops

0 0 • Reply • Share ›



Jijie Chen

