Transaction Outbox Pattern

*A microservice should be an autonomous unit that can fulfil most of its assignments with its own data. It can also ask other microservices for missing pieces of information required to complete its tasks.*

*Services also can exchange messages. It’s essential to ensure that the sent message reaches its destination and losing it could yield serious business implications. Proper implementation of communication patterns between services might be one of the most critical aspects when applying microservices architecture.*

*Let’s consider a simple scenario of service A having just finished processing some data. It has committed the transaction. Now it needs to notify service B that it has finished its task and new information is available for fetching. The simplest solution would be just to send a synchronous REST request (most probably POST or PUT) to service B directly after a transaction is committed.*

*This approach has some drawbacks. Arguably, the most important one is a tight coupling between services caused by the synchronous nature of the REST protocol. If any of the services is down because of maintenance or failure, the message will not be delivered. Even if message broker used, we may decouple services, but we may face same issue still.*

*As a first step, we need to introduce a table that stores all messages that are intended for delivery - that’s our message outbox. Then instead of directly doing requests, we just save the message as a row to the new table. Doing an INSERT into the message outbox table is an operation that can be a part of a regular database transaction. If the transaction fails or is rolled back, no message will be persisted in the outbox.*

*In the second step, we must create a background worker process that, in scheduled intervals, will be polling data from the outbox table. If the process finds a row containing an unsent message, it now needs to publish it (send it to an external service or broker) and mark it as sent. If delivery fails for any reason, the worker can retry the delivery in the next round.*

*Ensured message delivery with possible duplicated requests means we’ve got an at-least-once processing guarantee and recipients won’t lose any notifications.*

*Drawbacks –*

* *Implementing the pattern requires writing some boilerplate code.*
* *Polling the outbox table can sometimes put significant stress on your database.* *Decrease the number of database calls by simply increasing the batch size. Nonetheless, with a big number of messages selected, if the request fails, none of them will be marked delivered.*