Event Driven

Status

draft

Context

Event-driven architecture is a way to design loosely coupled system.

In this ADR, we take a look at approaches on how to turn a request/reply based system into an event-driven architecture.

We will also explore the usecase and when to choose this over request/reply system.

The choice is not exclusive, as both architecture can live together.

Decisions

Batch Events

Outbox Pattern

Event Notification

Instead of passing a heavy payload, send small payload that can be serialized as the first MVP. They should contain mostly reference ids and possible the URLs for the consumer to call.

As the name mention, it is meant for notifying the consumer that an event occured, without assuming that the consumer will take action.

The opposite of those is event carrier state transfer.

Event Carrier State Transfer

Event carrier state transfer contains a larger payload. However, passing states can have unexpected consequences, especially if the state is prone to change.

Take an order created notification by example. We pass a payload that contains the details of the order notification. However, there could be delay in processing and publishing, and hence the status of the order could be cancelled. When that happens, there are no reason to actually send the event anymore.

For most cases, it is also better for the consumer to decide when to call the API to fetch the latest data from the publisher, instead of relying on the stale data that was sent.

CQRS

Event Sourcing

Cloudevents

Consumer vs background tasks

For most scenario, we don't need to do publish the message externally. Events can be published and consumed within the system.

Consequences

https://www.ben-morris.com/event-driven-architecture-and-message-design-anti-patterns-and-pitfalls/