A Brief Description on the Choice of API Messaging Pattern Implementation

Adrian Langseth

October 24, 2020

1 Introduction

As the project is aimed to be a *proof-of-concept*, the chief goal is to find whether an implementation is feasible, often through a functional prototype evidencing the implementational possibility of a concept. In the case of the possibility for external entities to subscribe to certain data flows from specific sensors, the explored possibilities were a proprietary Publish-Subscribe(PubSub) design pattern, a Publish-subscribe design pattern implemented through the GraphQL-subscriptions module and a Data Distribution Service(DDS).

2 PubSub

The publishers of the data flows, the sensors, have no need of information regarding the entities which subscribe to the data flows. Likewise, the subscribers express interest in the data flows necessary and have no further need of information regarding the origin of the data. This loose coupling is a primary component of the publish-subscribe messaging pattern, where publishers and subscribers have no direct interest in knowledge of each other, only in the messages passed. In the context of the subscription part of the project, a publish-subscribe messaging pattern would mean that the sensors classify the data flows, and the external entities subscribe to the specific data flow classes which are of their interest. The sensors have no need to know who the subscribers are, or if there are any, and the external entities have no interest in the specific sensors, only in the data flows.

From the messaging pattern follows a scalability because of the lack of a explicit messaging between the publishers and subscribers. The publishers classify its data onto endpoints and the subscribers receive from the classes in which it has expressed interest. This means that new sensors can be added without large overhead by setting them as contributors to existing classes. New classes can also be made to accommodate for new data flows from the new sensors. As no new coupling from a sensor to all external entities are necessary, such as is necessary with some other messaging patterns, these operations are scalable and will not add much overhead to a increasingly larger system. From the subscriber-side, adding new external entities and new class subscriptions for existing subscribers is as simple as allowing for a new connection onto the class endpoint.

2.1 Proprietary PubSub

A proprietary implementation of PubSub was assessed to require much more work to create and implement than the GraphQL implementation. Such a solution was therefore deemed an inefficient use of the groups time and was therefore discarded.

2.2 GraphQL PubSub

For this project, the standard version of PubSub was implemented. This implementation was selected as it was the one already supported by the middleware chosen for our API, GraphQL. The selection of this version was based on ease of implementation, as the GraphQL version was functionally sufficient, and based on the project being a broader proof-of-concept research, meaning to implement enough to prove feasibility and describing future possibility of further implementation to enhance scalability and performance.

3 Further Work: DDS

During a sprint planning meeting, the customer representative specifically mentioned the DDS implementation of the publish-subscribe pattern, as it is the implementation already in use internally to the customer. As DDS is an implementation of the publish-subscribe pattern, it can be seamlessly implemented in place of the solution implemented in the proof-of-concept project.

DDS has reliability of the data stream as one of its primary goals, and is therefore popular within applications which are mission- or business-critical, such as air-traffic controls. The fully-developed system is aimed to be used by seafood producers, meaning the data on the health of the fish may be business- and mission critical. Hence, the reliability of the time-sensitive data stream is of high priority and implementing DDS to the project in place of the GraphQL PubSub would be an effective upgrade.

The scope of the project is variable with time and in the short-term, the system is to be used primarily for research, and later a possibility for commercial application. Therefore, scalability is a also a concern. Scalability is already a feature of PubSub, but is further addressed by DDS[**?**]. The further upgrade to DDS from GraphQL PubSub would allow for a enhanced scalability for future data- and user-growth.

4 Conclusion

For the purposes of this project, a publish-subscribe messaging pattern was implemented through the GraphQL-subscriptions module. The implementation used is functional and allows for subscription to specific data flows, and therefore proves the feasibility of subscriptions in an open-source capacity. For a large-scale implementation, it is recommended to implement DDS in its place. As these are different implementations of the same concept, the DDS can slot into the place of the current implementation with little alteration needed. With scalability being a chief attribute of the envisioned final system, inserting DDS should be considered.