Handling message duplication

In this lesson you will learn about message duplication and how to use idempotent consumers to properly deal with this issue.

Message duplication

To understand the issue of message duplication let's go through the scenario where gil is debited from the user's wallet.

As you know, as part of our purchase process, a debit gil command will eventually be sent to the Identity microservice via our message broker. Identity will go ahead and debit 5 gil from the user in its local database, resulting in, say, a new total balance of 50 gil for this user.

Identity will then try to send a gil debited message back to the broker, but something goes wrong and such message never arrives to the broker, either because Identity or the broker has network connectivity problems or because either of them just crashes in the middle of the operation.

Since the broker has not received an acknowledgement of the message it sent, it decides to redeliver it. Therefore, it sends the same message again to Identity. The service treats this as if it was a brand new message and proceeds to debit 5 more gil from the user's wallet, leaving him with a new total balance of 45 gil. And, this time, the gil debited event is able to reach the message broker.

As you can see, the main issue here is that the service consumes every message as if it was a new message, and each time it is received by the service it results in more gil debited from the user's wallet. Basically, there is no idempotency in how the consumer processes the message, which results in an incorrect gil balance for the user.

What can we do about this?

Idempotent consumer

Here's where the idea of having an idempotent consumer comes into place. When any service consumes a message from the message broker, it usually includes a unique identifier, a MessageId. This MessageId comes as part of the header of the message and will be included any time the broker redelivers it to any consumer.

So, this time, when Identity receives the message, it first asks the question: Have I seen this MessageId before? This likely translates into the service querying its own database, where it would have stored any previously received MessageIds.

Since this is the first time it receives this message, it can't find it in the DB, so it proceeds to debit the 5 gil from the user and in this local transaction it includes the MessageId, either as part of the updated User record or document or perhaps in a separate table or collection where it stores processed messages.

Then the service tries to send the gil debited message, but again, such message never reaches the broker. With no acknowledgement from the service, the broker decides to redeliver the message, so it

sends it again to Identity, but since this is the same message as before, it includes the exact same MessageId used the first time.

So again, the first thing Identity does with this message is confirm if it has seen it already by checking its local database. It successfully finds the MessageId and decides that there is no need to process such message again, so it does not perform any database updates. And then it publishes the gil debited event which this time successfully arrives to the broker.

As you can see, when an idempotent consumer processes the same message twice (or any number of times really) it is the same as if it processed it only once. And this brings the expected benefit of not causing any inconsistencies in the service's local transactions, like the modification of the user's gil balance.

In the next lesson we will see how to modify our Inventory and Identity microservices to become idempotent consumers.