Introduction to Sagas

In this lesson you will learn about Sagas, what are they and how they work in the context of a microservices architecture.

What is the Saga algorithm?

So, what is a Saga? A saga is a long-lived transaction that can be written as a sequence of transactions that all complete successfully or the compensating transactions are executed to amend a partial execution. This concept was introduced in a paper published in 1987 by Hector Garcia-Molina and Kenneth Salem.

In essence, a saga helps us to think about our multi microservice business process as if it is really one transaction composed of series of independent sub transactions and corresponding compensating transactions. In a saga, all these sub transactions must complete successfully, or if there are failures, the compensating transactions are executed to roll back everything.

Let's see how this works.

How does a Saga work?

Let's bring back our Inventory and Identity microservices. Here, our long-lived operation, is our purchase process. The process starts by asking Inventory to grant 5 potions to some user. Inventory goes ahead, runs a local transaction in its database, and comes back with a successful response.

Then, the process makes a call to Identity asking it to debit 25 gil from the same user and, in a similar way, Identity makes the database update in its own local transaction, after which it replies with a successful response. If all the participants completed their part with no issues, we consider the overall purchase process as completed successfully.

Notice that even if we cannot consider the overall purchase process as an ACID transaction anymore, the individual changes to each database can keep using ACID transactions. Now, to provide some form of atomicity to the purchase transaction, we make use of compensating transactions.

So, for instance, let's say Identity is not able to debit 25 gil from the user, because the user just doesn't have enough gil. This triggers a series of additional transactions to compensate what was done before. In this simple scenario, there's only one compensating transaction, where we ask Inventory to take back the 5 potions it had granted to the user before.

After the compensating transactions are completed, we consider the overall purchase transaction as rolled back.

The main benefit of a saga as compared with the two-phase commit algorithm, is that there is no database locking anymore. Instead of waiting for all other participants, each microservice performs the requested operation and moves on. Also, if asynchronous messaging is used to communicate with the saga participants, the fact that any of them is temporarily unavailable won't impact the success of the

overall process, because of the asynchronous communication benefits we discussed in previous modules.

Now, there are two ways things can get coordinated within a saga: choreography and orchestration. Let's explore these two types of sagas in the next lesson.