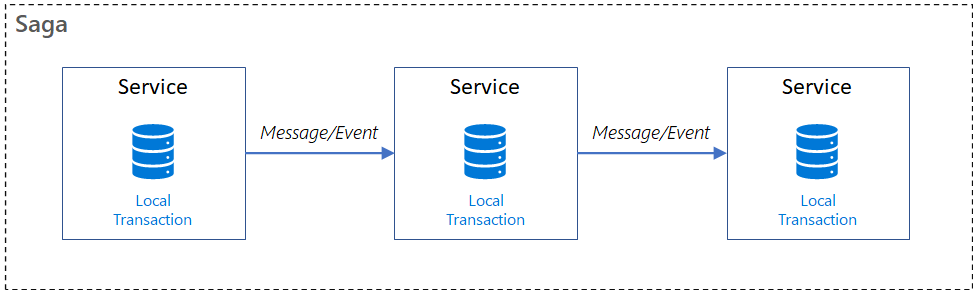
**Saga distributed transactions pattern**

**Saga means series of events in English**. The Saga design pattern is a way to manage data consistency across microservices in distributed transaction scenarios.

The Saga pattern provides transaction management using a sequence of **local transactions**. **A saga is a sequence of transactions where each transaction updates the database and publishes a message or event to the messaging topic** **for the next action which itself is a transaction**. **If a local transaction fails, the saga executes a series of compensating transactions that undo the changes that were made by the preceding local transactions**



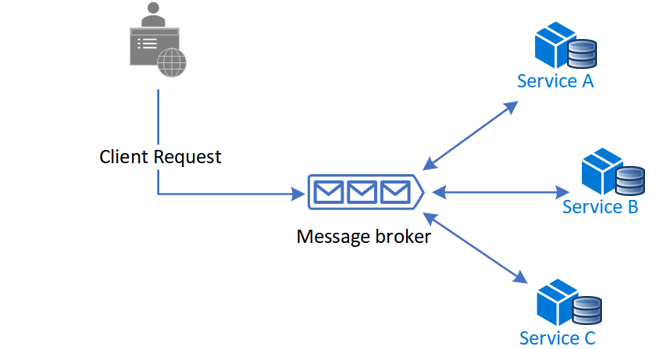
In Saga patterns:

* *Compensable transactions* are transactions that can potentially be reversed by processing another transaction with the opposite effect.
* A *pivot transaction* is the go/no-go point in a saga. If the pivot transaction commits, the saga runs until completion. A pivot transaction can be a transaction that is neither compensable nor retryable, or it can be the last compensable transaction or the first retryable transaction in the saga.
* *Retryable transactions* are transactions that follow the pivot transaction and are guaranteed to succeed.

There are two common saga implementation approaches, *choreography* and *orchestration*. Each approach has its own set of challenges and technologies to coordinate the workflow.

**Choreography**

Choreography is a way to coordinate sagas where participants exchange events without a centralized point of control. **With choreography, each local transaction publishes domain events that trigger local transactions in other services.**



**Advantages of Choreography Pattern**

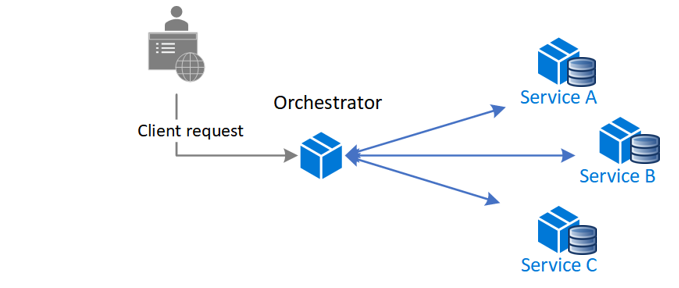
* Good for simple workflows that require few participants and don't need a coordination logic.
* Doesn't require additional service implementation and maintenance.
* Doesn't introduce a single point of failure, since the responsibilities are distributed across the saga participants.

**Disadvantages of Choreography Pattern**

* Workflow can become confusing when adding new steps, as it's difficult to track which saga participants listen to which commands.
* **There's a risk of cyclic dependency between saga participants** because they have to consume each other's commands.
* Integration testing is difficult because all services must be running to simulate a transaction.

**Orchestration**

Orchestration is a way to coordinate sagas where a centralized controller tells the saga participants what local transactions to execute. **The saga orchestrator handles all the transactions and tells the participants which operation to perform based on events.** The orchestrator executes saga requests, stores and interprets the states of each task, and handles failure recovery with compensating transactions.



**Advantages of Orchestration Pattern**

* Good for complex workflows involving many participants or new participants added over time.
* Suitable when there is control over every participant in the process, and control over the flow of activities.
* Doesn't introduce cyclical dependencies, because the orchestrator unilaterally depends on the saga participants.
* Saga participants don't need to know about commands for other participants. Clear separation of concerns simplifies business logic.

**Disadvantages of Orchestration Pattern**

* Additional design complexity requires an implementation of a coordination logic.
* There's an additional point of failure, because the orchestrator manages the complete workflow.