Ingestion and Workflow

In microservices, a transaction can span multiple services. The workflow must be reliable since it cannot leave them uncompleted. Similarly, we need to control the rate of incoming requests in order to avoid overloading the network.

# Challenges

## Load Leveling

Too many requests can overwhelm the system. It is important to control the flow by buffering or queuing them for processing.

## Guaranteed Delivery

To avoid dropping client requests, we must guarantee delivery of the messages.

## Error Handling

Errors in one service basically breaks done the transaction. Therefore, we must implement recovery mechanism.

# Ingestion

Provisioning enough resources to handle the maximum load would be extremely inefficient. A better approach is to have a buffer to handle the maximum ingestion rate over short periods such that that backend service only needs to handle the maximum sustained load. That way, the backend shouldn’t need to handle large spikes. There are couple of ways to achieve this, one is with a queue and another with a stream.

## Queue

In queue, individual consumer removes the message from the queue such that the next consumer won’t see it. In a way, multiple consumers compete for processing messages.

## Stream

In a stream, things are a bit different. Think of it as the follow. Each consumer is responsible for a partition of the queue. It is like each consumer has its own queue.

# Workflow

## Handling Failures

There is in general two types of failures: transient and non-transients. In transient failure, it is an error that probably happens for a limited amount of times. The fix would be to retry the call. Hence, it is critical to make operation idempotent. However, in non-transient failure, an error is unlikely to go away. In this case, we need to undo steps (i.e. compensating transactions).

## Idempotent Operations

To avoid losing requests, we must guarantee that all requests are processed at least once. However, to avoid duplicate work, we should also make the operations idempotent in case requests are processed twice or more. Idempotent methods do not produce additional side-effects after the first call. Therefore, it is safe to call them multiple times.

## Compensating Transactions

Undoing steps might not be trivial. Sometimes it requires a manual process or an external system. A better approach would have a separate microservices handle the cancellation process. Picture it as a cancel request coming from the client.