# Architecture Decision Document

Title

Broadcast (Pub-sub) order cancellation event.

Status

Accepted

Context

An order can be cancelled either by user or by system. System can cancel the order in various scenarios like food expiry, fridge going out of order, error during Purchase transaction or any other system or infra failure etc.

Any order cancellation has to be communicated to all relevant services and each service will have to take the relevant action to achieve eventual system consistency for a cancelled order like:

* PaymentService will do payment reversal for a given order cancellation.
* InventoryService will take the required action to compensate the transaction as needed.
* NotificationService will notify user for order cancellation.
* OrderService will mark the order cancelled.
* Coupon/Promotion Service will roll back the coupon applied.
* KitchenService, DeviceManagementService etc.
* TelemetryService will record the order failure.

This communication can either be done synchronously or asynchronously.

Decision

We decided to use the **Even driven architecture** to solve this problem. The order cancellation will be broadcasted (pub-sub) as an event and services participating in order completion will subscribe to these events and take necessary action.

**No orchestration is needed** as Publisher publishes the event once. And each Consumer subscribes to the event and knows what needs to be done.

**Performant -** Because every subscriber service will act in parallel. User get notified faster and can make other food arrangements quickly.

**Reliability**- Event queues and topics are very reliable as compared to in memory data. Generally subscriber will mark an event as completed only after successful processing of the event, so If a process crashes in between then event will still be there in the Subscription queue and can be reprocessed later.

**Evolutionary -** New Services can subscribe to this event in future as needed without any change to other services.

Consequences

Eventual Consistency -

As the system is distributed so getting to a consistent state will take time and might slow down due to error in any of the service.

Error handling -

Error handling becomes complex in event-driven architecture. Below are the ways we will solve for it.

* Use Error queues
* Persisted messages Topics
* Synchronous send - Synchronous send does a blocking wait in the message producer until the broker has acknowledged that the message has been persisted.
* Client Acknowledge mode - with message lock duration.
  + A given message can only be delivered to one processor/Thread at any point in time and this message will be locked so that no other process/thread can read this message. Processor will first process the message and then mark the message as processed and this will delete the event from the Subscription queue. In case processor errors out or crashes then message/event will still remain on subscription queue and can be processed later by any other process or thread.