

Guaranteed Message Delivery: Outbox Design

Version	v0r1
Status	DONE
Responsible	@ Monjure Alam (Unlicensed)
Approving Body	
Date Approved	
Informed	
Consulted With	
AF Decision	
Tracking	
TODO	<ul style="list-style-type: none"><input checked="" type="checkbox"/> Scheduler event format and event name<input checked="" type="checkbox"/> Scheduler event interval<input type="checkbox"/> API Gateway-SQS design update<input checked="" type="checkbox"/> House Keeping of Outbox Events<input type="checkbox"/> Alternate flow on disabling outbox<input type="checkbox"/> Failure alert configuration<input type="checkbox"/> Outbox improvement for data management

- Problem Statement
- Assumptions
- Prerequisites
- Event Processing Flow
- Event Process Architecture
 - Real Time Event Publishing
 - Command Execution and Primary Transaction Boundary
 - Post Commit Event Processor Execution with Secondary Transaction
 - Pending Outbox Events Publishing
 - Event Scheduler
 - Event Replay
 - Exclude event from Outbox Processing
 - Recommendation
 - Option 1: Builder API to create a exclusion event
 - Option 2: Predicate based event exclusion
 - Comparison
- Physical Data Model
 - Data Partitioning and Indexing
- House keeping
- Outbox Improvement for Database Management
 - Secondary transaction flow
 - Implementation details of above secondary transaction flow changes:
 - Replay flow
- Event Model definitions
- Reference

Problem Statement

CMDS Microservices need to have a reliable mechanism to deliver message without loosing any events information.

Assumptions

- Distribution Lambda is out of scope from Guaranteed Message Delivery design

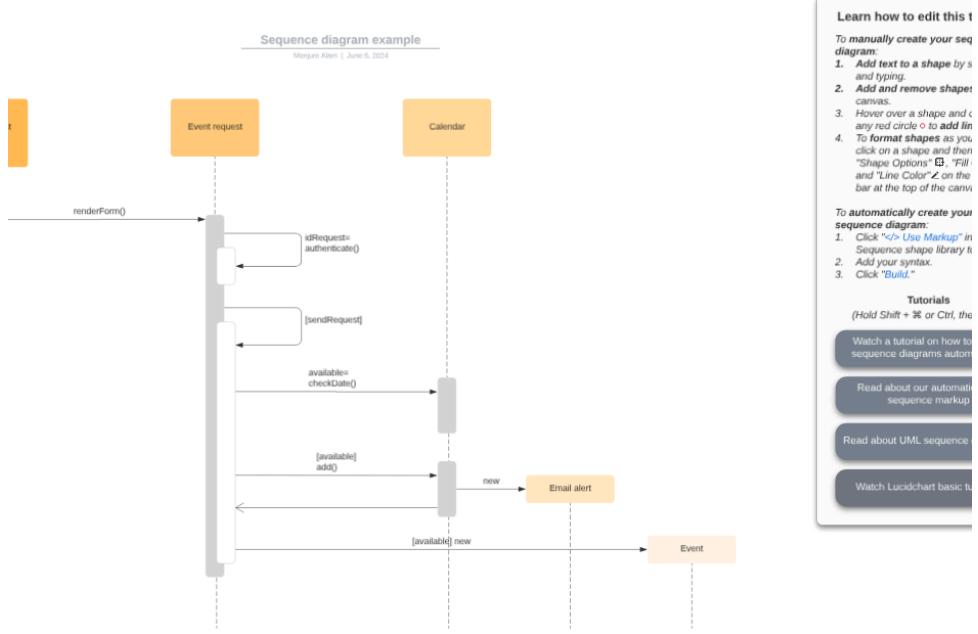
Prerequisites

- Multiple messages needs to be discarded by the MX based on event date-time

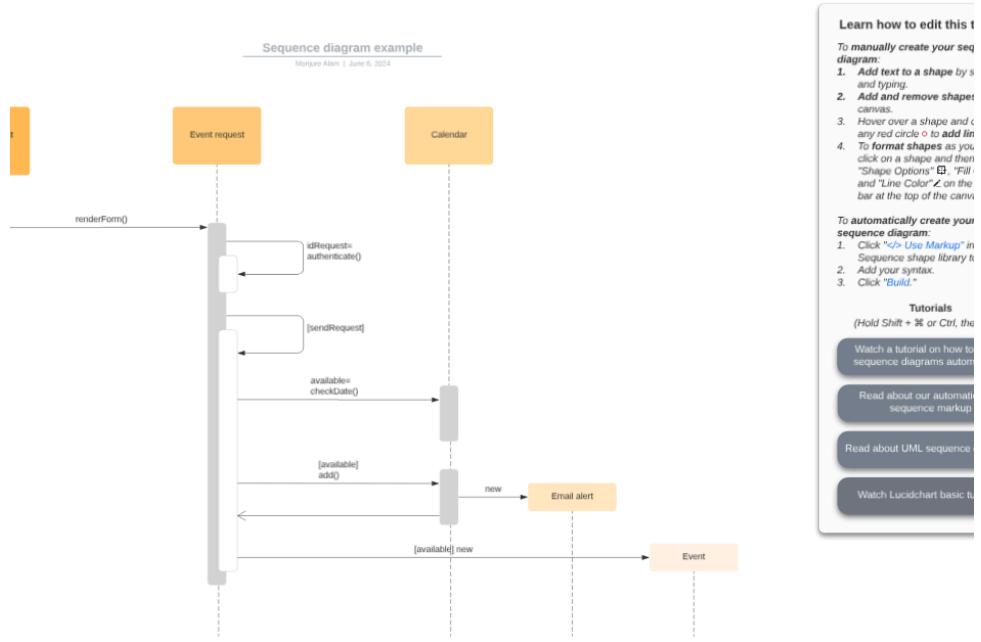
Event Processing Flow

Below sequence diagram outline a very high level flow of the event processing in the context of Outbox Design Pattern.

Below sequence diagram gives a high level flow of standard domain event processing.



Below sequence diagram gives a high level flow of replay of domain event in context of outbox processing.



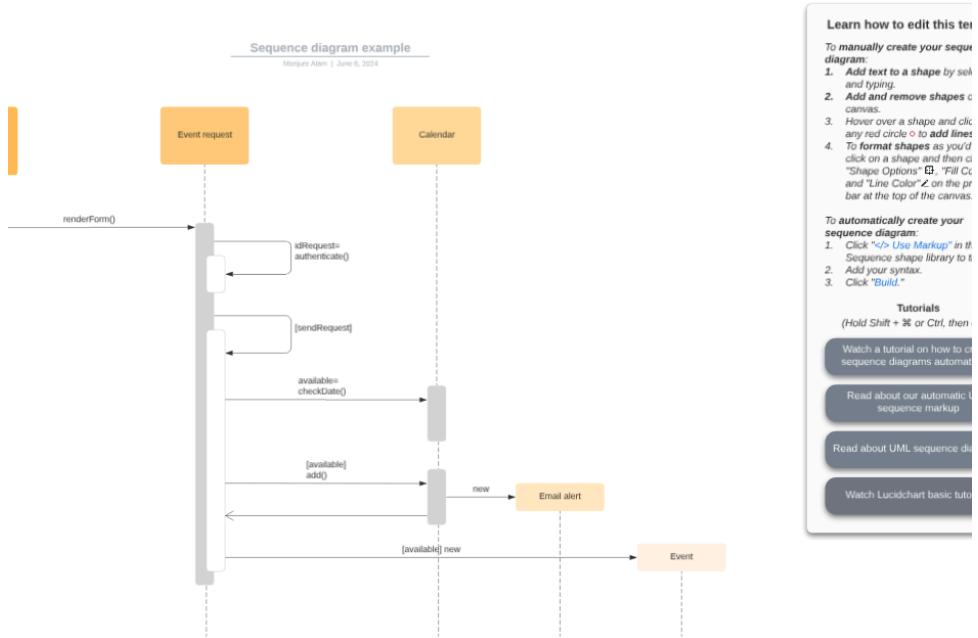
Event Process Architecture

Real Time Event Publishing

In the real time event publishing design there are 2 major components Domain Service and Outbox Event Process is participating in 2 different sets of transactions called primary and secondary.

The Primary transaction is around the domain layer and synchronous event process where the secondary transactions are participating in asynchronous out box event publishing operation,

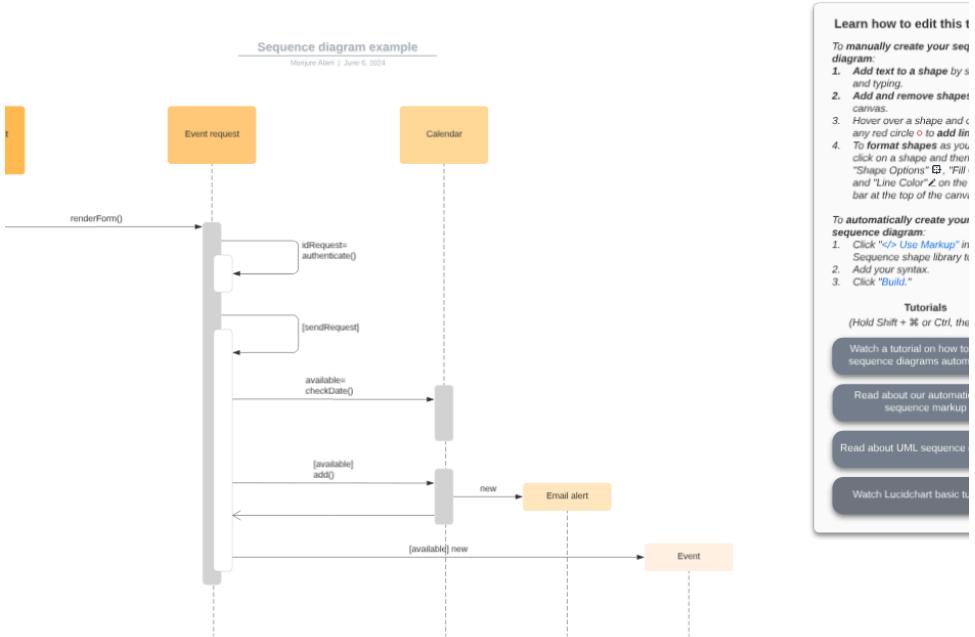
Command Execution and Primary Transaction Boundary



No	Description
1	The MX receives the domain event through standard listener configuration from mx input event queue. On receiving the event mx start executing the command by calling the appropriate domain service. The domain service receives the command and start a database transaction which will be denoted as primary transaction . All of the domain entity will be updated along side with persisting the outbox event in the same MX database within this primary transaction.
2	The domain service will take care of the business operation and will persist the domain entity into MX database.
3	After finishing the domain operation, output if the operation will be published as a Domain Events. Output of the domain event can consists of more than one domain events. This domain events will be published to an application internal event queue which is running inside spring boot application instance.
4	The resulting operation in domain service either could succeed or fail. For success scenario, a synchronous outbox event processor will pick up this event within the same transaction.
5	For failure scenario, it will log the failure details and will rollback the transaction.
6	MX will write failure error log in cloud watch.
7	The domain event which was received in No.1 will now rollback to SQS again for retry. Note: with the current Infra configuration after certain retry the event will be moved to a DLQ for further analysis.
8	Each the event published during No.3 will be picked up here. Event listener will receive each of the event separately and will execute Outbox Event Processor for each of the event. The Event processor will call two of the MX defined components to transform the event to a outbox defined format. <ul style="list-style-type: none"> Event Attribute Extractor: This component will be provide by the MX as they see fit to extract special attributes from the domain event. Event Transformer: This is an optional component which MX can provide if a special transformation required for the domain event. If not provided then processor will use the default CMDS event transformer,
9	The processor will take the output from previous execution(No.8) and will persist the outbox event to a special outbox table in the same MX database. At this stage the state of the event will be PUBLISH_PENDING .
10	The processor will publish the transformed outbox event into application internal queue for further processing.
11	If the above processing are completed without failure, then primary transaction(t1) will be committed the events will be picked up by No.15 asynchronous event processor.
12	For failure scenario, it will log the error and will rollback the ongoing transaction t1.
13	Log the error which was encountered in event processor into cloud watch.
14	The domain event which was received in No.1 will now rollback to SQS again for retry.

No	Description
	Note: with the current Infra configuration after certain retry the event will be moved to a DLQ for further analysis.
15	On successful completion of Primary Transaction(t1), event processor will start picking up the outbox event which was published in No.10 and it will start processing the events asynchronously in a different set of transactions which will be called as Secondary Transaction .
16	Monitoring and Analytics tool will receive the log from cloud watch and will trigger necessary alerts per configuration.

Post Commit Event Processor Execution with Secondary Transaction



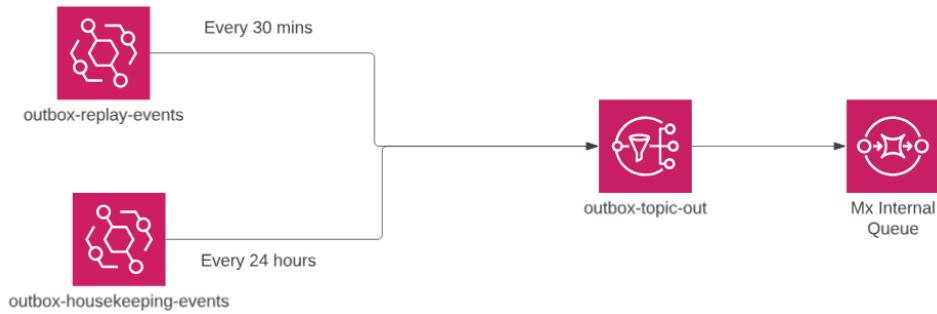
No	Description
1	Outbox event listener will receive the Outbox events in a asynchronous process post commit of the primary transaction.
2	Each of the event will be processed in a separate set of transactions in comparison to the primary transaction. This transaction will be called as Secondary Transactions . For each of the events a new secondary transaction will be initialised.
3	Outbox event processor will call the event publisher to publish the event into MX output topic. The event publisher component will be provided by the MX and MX will configure the target topic.
4	Based on the outcome of the No.3 process it will log the process as success or failure with the appropriate meta data for observability.
5	Update the outbox event state in database based on the No.3 outcome. If the publishing of the event succeed then state will be PUBLISH_SUCCEED In case of publishing failure the state will be PUBLISH_FAILURE . Post database persistence the transaction will be committed.
6	If the commit is successful then the process will be considered as successful.
7	In case of any failure during persistence or during transaction commit, the event will be logged and transaction will be rollback.
8	Processor will log the failure into standard log stream for monitoring and alert.
9	Step-1-8 will be repeated here for another event.
10	Predefined alert will be configured in Sumologic to provide insight with the event processor.

Pending Outbox Events Publishing

The second part of the outbox design is to take care of the event which is in pending or failure state. After certain interval will the process will be executed to find out the pending or failure events for retry.

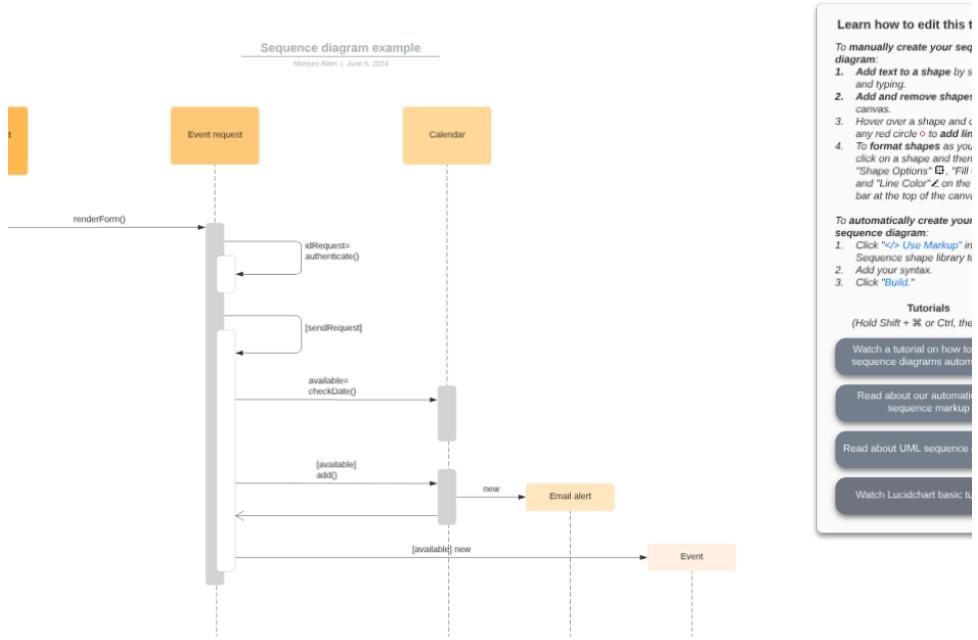
To achieve this we need a event scheduler to trigger the retry process after certain interval and a process which will be responsible for publishing the message and update the state of the message.

Event Scheduler



No	Description
1	<p>An event scheduler will be preconfigured in AWS Eventbridge which will publish the scheduler event in the structure defined in Outbox: Implementation Guidelines after every 30 minutes. The event will be published to a common topic.</p> <p>Input Path:</p> <pre>Input Path { "eventId": "\$.id", "time": "\$.time" }</pre> <p>Input Transformer:</p> <pre>Input Transformer { "eventHeader": { "transactionId": <eventId>, "correlationId": <eventId>, "eventName": "OutboxEventReplayRequested", "eventDateTime": <time> }, "eventBody": "{\"retryCountThreshold\":null,\"timeLimitInSecond\":null,\"eventNumberLimit\":null}", "eventErrors":null }</pre>
2	Respective MX queue will subscribe to this common scheduler event topic
3	MX will receive the Scheduler Event and will trigger the event replay processor.
4	Connection with the MX DB to retrieve the event and update the latest event state .
5	Required info/error will be logged in cloud watch.
6	Monitoring and Alert setup will continuously analyse the logs for preconfigured alerts.
7	The event processor will publish the domain event in the MX topic.

Event Replay



No	Description
1	The MX receives the scheduler event which will trigger the outbox event process to replay the pending and failed events.
2	In the first stage of processing, the processor will fetch the events from the database which state is in PUBLISH_PENDING or PUBLISH_FAILURE. The processor can return 0 to n number of events. <ul style="list-style-type: none"> PUBLISH_FAILURE events will be ignored after a certain retry count. This threshold value will be configurable.
3	Each of the events will be processed in a isolated transaction.
4	The process will publish the event with the help of event published which will be configured and controlled by the respective MX.
5	Outcome of step 4 will be logged in the cloud watch log.
6	Update the outbox event state in database based on the step 3 outcome. If the publishing of the event succeed then then state will be PUBLISH_SUCCEED. In case of publishing failure the state will be PUBLISH_FAILURE. Post database persistence the transaction will be committed.
7	Processor will log the failure into standard log stream for monitoring and alert.
8	Step 3-7 will be repeated here for the subsequent event.
9	Predefined alert will be configured in Sumologic to provide inside with the event processor.

Exclude event from Outbox Processing

Outbox design needs to provide a mechanism where MX can decide whether certain events do not need Outbox Processing and be published directly ignoring the outbox design.

Options are presented here to deal with the process.

Recommendation

Option 1: Builder API to create a exclusion event

Option 1: Builder API to create a exclusion event

In this option Outbox will expose a Builder API which MX need to apply on the event for which the MX wants to be excluded from Outbox Processing.

This Builder API will set a exclusion flag in the event. When Outbox receives the event, it will look for the exclusion flag to decide whether the current event can be excluded from Outbox Processing.

Option 2: Predicate based event exclusion

In this option the MX will provide a Predicate Function which will decide which event needs Outbox Processing and which one does not. During processing Outbox will call the MX provided predicate to decide the Processing path.

Comparison

	Option 1: Flag based event exclusion	Option 2: Predicate based event exclusion
Code Reusability	+ Reusable component	+ Reusable component
Ease of Implementation	+ Easy to implement	+ Easy to implement
Maintainability	+ Easy to maintain	+ Easy to maintain
Separation of Concern	+ Event exclusion concern is separated individual domain	⚠ Event exclusion concern is not separate and it is managed by the predicate

Physical Data Model



IELTS Online

Test Report Form

ACADEMIC

NOTE It is recommended that the candidate's language ability as indicated in this Test Report Form be re-assessed **after two years** from the date of the test.
To find out more about IELTS Online, IELTS band scores and the CEFR levels, please visit ielts.org/scores

Centre Number Date Candidate Number

Candidate Details

Family Name	Arora			 Age: 37 Height: 75 Weight: 75	Full Time Manager <input checked="" type="checkbox"/> <input type="checkbox"/> <input type="radio"/> <input checked="" type="checkbox"/>
First Name(s)	Pulkit				
Candidate ID	HID0023667770				
Name James Maria Saul					
Date of Birth	17/05/1997	Sex (M/F)	M	Scheme Code	Private Candidate
Country or Region of Origin	INDIA				
Country of Nationality	INDIA				
First Language	English				

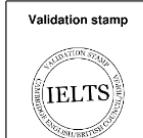
Test Results

Listening Reading Writing Speaking Overall Band Score CEFR Level

Administrator Comments

Practice Test Only. Not to be used for admissions purposes

Recognising organisations must verify this score at ielts.org/verify



Date Test Report Form Number

Table Name	Column Name	Mandatory	Data Type	Constraint	Description
outbox_event					
	outbox_event_uuid	Y	UUID	Primary Key: pk_outbox_event	
	transaction_uuid	Y	UUID	Index: idx_02_outbox_event on column transaction_uuid, event_name	
	event_name	Y	VARCHAR(256)	Index: idx_02_outbox_event on column transaction_uuid, event_name	
	event_datetime	Y	TIMESTAMPZ		
	payload	Y	TEXT		
	publish_state	Y	ENUM	[PUBLISH_PENDING, PUBLISH_SUCCEEDED, PUBLISH_FAILURE] Index: idx_01_outbox_event on column publish_state, updated_datetime	
	retry_count	Y	INTEGER		
	created_datetime	Y	TIMESTAMPZ		
	updated_datetime	Y	TIMESTAMPZ	Index: idx_01_outbox_event on column publish_state, updated_datetime	
	concurrency_version	Y	INTEGER		
outbox_event_attribute					
	outbox_event_attribute_uuid	Y	UUID	Primary Key: pk_outbox_event_attribute	
	outbox_event_uuid	Y	UUID	Foreign Key: fk_01_outbox_event_attribute_event_outbox_uuid_outbox_event_outbox_event_uuid Unique Key: uk_01_outbox_event_attribute_event_outbox_uuid_attribute_key	
	attribute_key	Y	VARCHAR(256)	Unique Key: uk_01_outbox_event_attribute_event_outbox_uuid_attribute_key	
	attribute_value	N	TEXT		
	updated_datetime	Y	TIMESTAMPZ		

Data Partitioning and Indexing

Partitioning of the outbox_event table is required based in the publish_state and also updated_datetime. This will serve the below purpose

- Events can be kept in a different partition which is already published and also based on certain time interval. This will help query to perform better on pending and failed event.



IELTS Online

Test Report Form

ACADEMIC

NOTE It is recommended that the candidate's language ability as indicated in this Test Report Form be re-assessed **after two years** from the date of the test.
To find out more about IELTS Online, IELTS band scores and the CEFR levels, please visit ielts.org/scores

Centre Number Date Candidate Number

Candidate Details

Family Name	Arora			 Age: 37 Height: 75 Weight: 75	Full Time Manager <input checked="" type="checkbox"/> <input type="checkbox"/> <input type="radio"/> <input checked="" type="checkbox"/>
First Name(s)	Pulkit				
Candidate ID	HID0023667770				
Name James Maria Saul					
Date of Birth	17/05/1997	Sex (M/F)	M	Scheme Code	Private Candidate
Country or Region of Origin	INDIA				
Country of Nationality	INDIA				
First Language	English				

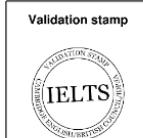
Test Results

Listening Reading Writing Speaking Overall Band Score CEFR Level

Administrator Comments

Practice Test Only. Not to be used for admissions purposes

Recognising organisations must verify this score at ielts.org/verify

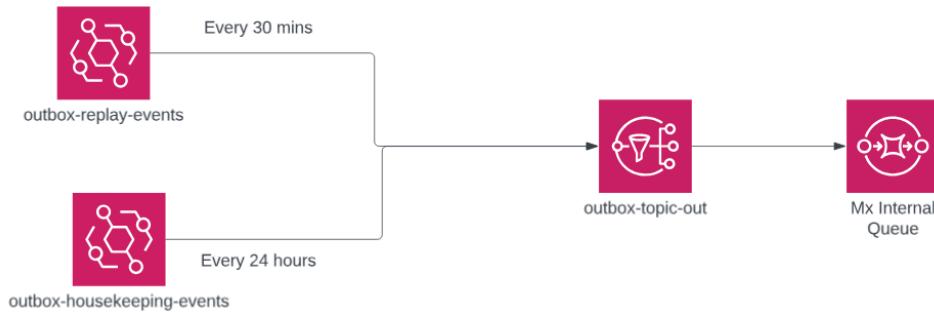


Date Test Report Form Number

House keeping

During house keeping activity outbox events will be deleted based on the published states and the last updated datetime.

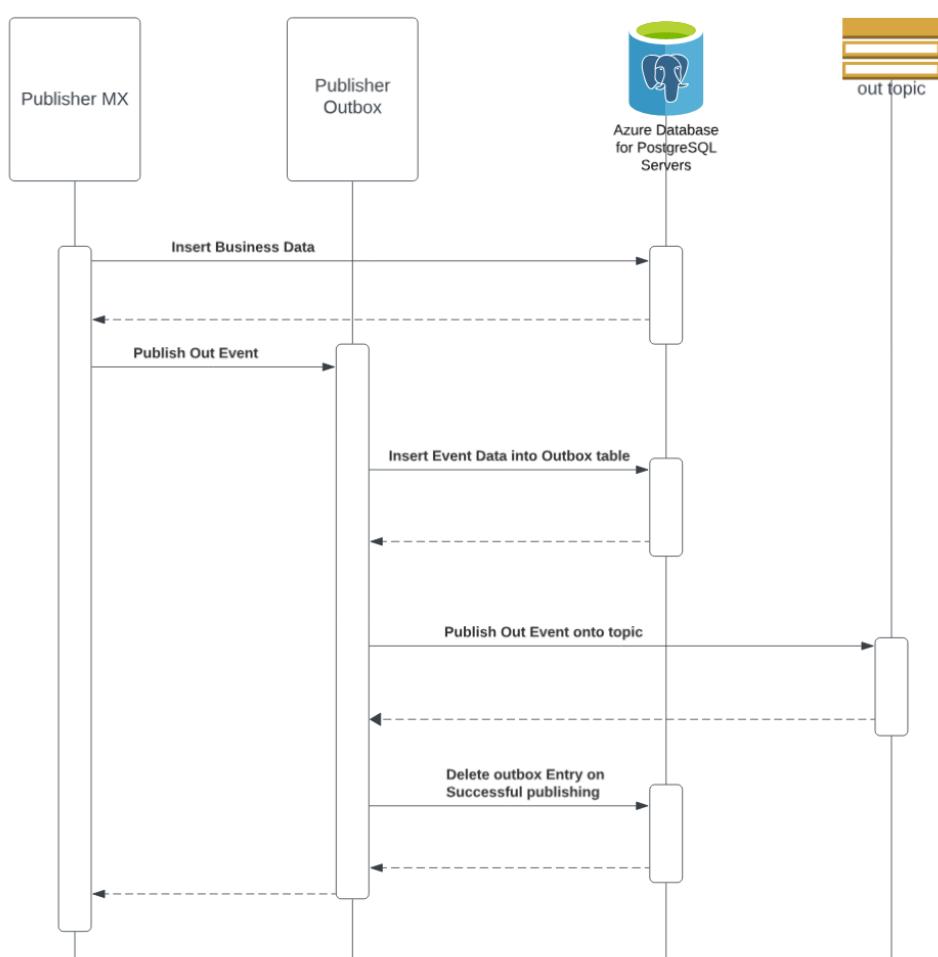
The house keeping job will be triggered using a Scheduler Event (for every x hours), which will delete the records in PUBLISH_SUCCESS state and are older than <configured_interval>



No	Description
1	<p>An event scheduler will be preconfigured in AWS Eventbridge which will publish the scheduler event in the structure defined in after every X hours. The event will be published to a common topic.</p> <p>Input Path:</p> <pre> Input Path { "eventId": "\$.id", "time": "\$.time" } </pre> <p>Input Transformer:</p> <pre> Input Transformer { "eventHeader": { "transactionId": <eventId>, "correlationId": <eventId>, "eventName": "OutboxDeleteEventRequested", "eventDateTime": <time> }, "eventBody": "{\"timeLimitInMinutes\":null, \"eventNumberLimit\":null}", "eventErrors":null } </pre>
2	Respective MX queue will subscribe to this common scheduler event topic with a filter containing the event name.
3	MX will receive the Scheduler Event and will trigger the delete event processor.
4	<p>Connection with the MX DB to delete events which are older than pre-configured time limit and (number of records to be deleted)*. The <code>timeLimitInMinutes</code> present in <code>eventBody</code> takes precedence over pre-configured limit also don't delete current day*</p> <p>*DELETE access required for mx user on <code>outbox_event</code> and <code>outbox_event_attribute</code> tables</p> <p><code>timeLimitInMinutes</code> will be added platform specific parameter once https://bitservice.atlassian.net/browse/IMOD-31093 is implemented. Till then, this will be handled as Mx specific parameter</p>
5	Required info/error will be logged in cloud watch.
7	The event processor will publish " <code>OutboxDeletionCompleted</code> " with number of records and from & to timestamp of events deleted to the topic. Persistence ignore for this one?

Outbox Improvement for Database Management

Changes are done to below flow as per design decision taken here [D0005 Outbox Data Volume Management](#). Below is the updated flow diagram after implementing changes in outbox table record management.



There will be changes in below 2 flows

Secondary transaction flow

No	Description
1	Outbox event listener will receive the Outbox events in a asynchronous process post commit of the primary transaction.
2	Each of the event will be processed in a separate set of transactions in comparison to the primary transaction. This transaction will be call as Secondary Transactions . For each of the events a new secondary transaction will be initialised.
3	Outbox event processor will call the event publisher to publish the event into MX output topic. The event publisher component will be provided by the MX and MX will configure the target topic.
4	Based on the outcome of the No.3 process it will log the process as success or failure with the appropriate meta data for observability.
5	Depending on the No.3 outcome, there can be 2 possibilities. If the publishing of the event succeed then outbox table entry for that event is deleted. In case of publishing failed , the state will be PUBLISH_FAILURE . It will be persisted in outbox table with PUBLISH_FAILURE state. Post database persistence the transaction will be committed.
6	If the commit is successful then the process will be considered as successful.
7	In case of any failure during persistence or during transaction commit, the event will be logged and transaction will be rollback.
8	Processor will log the failure into standard log stream for monitoring and alert.
9	Step-1-8 will be repeated here for another event.
10	Predefined alert will be configured in Sumologic to provide inside with the event processor.

Implementation details of above secondary transaction flow changes:

In **EventPersistenceService** class new method is added(i.e., delete(event)) :

```
public void delete(final OutboxEventV1 event) {
    // Event will not be deleted if this event is marked as outbox ignored - because it doesn't get persisted in the primary transaction
    if (!outboxIgnoreManager.isOutboxIgnore(event)) {
        outboxEventRepository.deleteById(event.getOutboxEventUuid());
    }
}
```

In **OutboxEventProcessor** class when the event is getting published successfully then we are not changing the publish_state to PUBLISH_SUCCEED instead we are deleting the record from the database by calling above delete method. If the publishing of event is not successful then we are updating the entry with publish_state as PUBLISH_FAILURE and persisting the data into database

```
public void process(final OutboxEventV1 event) {
    try {
        snsEventPublisher.publish(event);
        eventPersistenceService.delete(event);
    } catch (Exception e) {
        event.setPublishState(PublishState.PUBLISH_FAILURE);
        event.setRetryCount(event.getRetryCount() + 1);
        eventPersistenceService.save(event);
    }
}
```

Replay flow

No	Description
1	The MX receives the scheduler event which will trigger the outbox event process to replay the pending and failed events.
2	In the first stage of processing, the processor will fetch the events from the database which state is in PUBLISH_PENDING or PUBLISH_FAILURE. The processor can return 0 to n number of events. <ul style="list-style-type: none"> PUBLISH_FAILURE events will be ignored after a certain retry count. This threshold value will be configurable.
3	Each of the events will be processed in a isolated transaction.
4	The process will publish the event with the help of event published which will be configured and controlled by the respective MX.
5	Outcome of step 4 will be logged in the cloud watch log.
6	Depending on the No.3 outcome, there can be 2 possibilities. If the publishing of the event succeed then outbox table entry for that event is deleted. In case of publishing failed , the state will be PUBLISH_FAILURE . It will be persisted in outbox table with PUBLISH_FAILURE state.

No	Description
	Post database persistence the transaction will be committed.
7	Processor will log the failure into standard log stream for monitoring and alert.
8	Step 3-7 will be repeated here for the subsequent event.
9	Predefined alert will be configured in Sumologic to provide inside with the event processor.

Event Model definitions

Events Definition 1.0 OAS 3.0

Private

Mx Event Event published by Mx

POST /MxDomainEvent_InMemory_Existing Mx Domain Event model

Outbox Event

POST /Outbox_DbEvent Model for outbox event

Scheduler Event

POST /OutboxEventScheduler Scheduler Event model

Schemas

OutboxEventSchedulerV1 >

MxDomainEventV1 >

OutboxEventV1 >

OutboxEventAttributeV1 >

BaseHeader >

BaseEventErrors >

Reference

- <https://reflectoring.io/spring-boot-application-events-explained/>
- <https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/event/EventListener.html>
- <https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/ApplicationEventPublisher.html>
- D0005 Outbox Data Volume Management