Have my published Salesforce Platform Events been received by my Mule App?

Insights into Salesforce PubSub with MuleSoft

By Thomas Hoeger (he/him), January 2023

https://sfdc.co/sfdc-mule-pubsub

Salesforce introduced evented data integration around 2015. Whilst it uses internally a Kafka implementation, the “channel” to the outside world was implemented using CometD and Bayeux protocol[1].
In 2021 Salesforce announced a modernization of the evented approach - moving from CometD to gRPC[2] - and calls it now Pub/Sub API[3].

In parallel with announcing the general availability of the Pub/Sub API from Salesforce, MuleSoft also released a connector to easily consume the API in Mulesoft integrations[4].

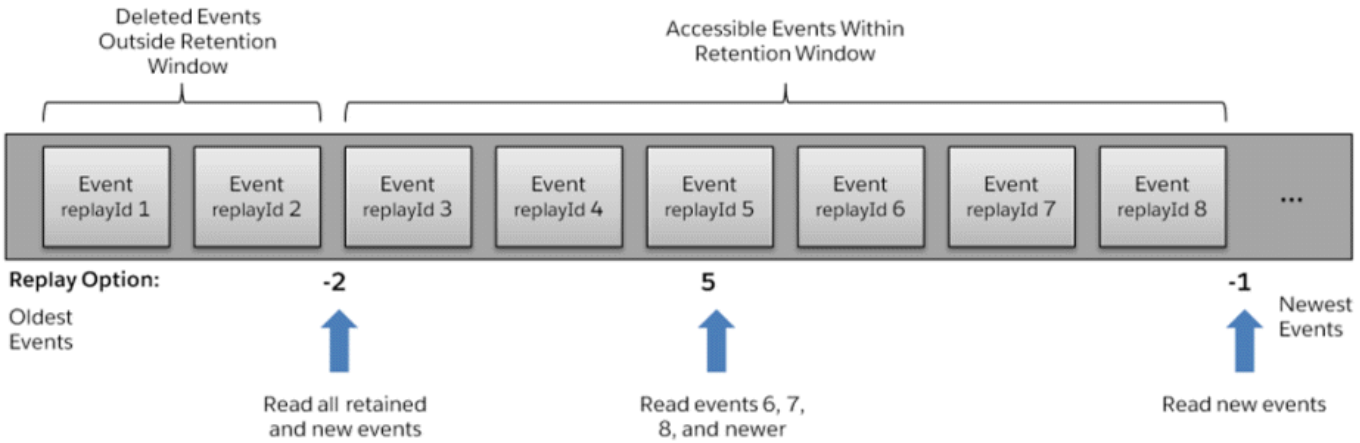
This document discusses in two use cases the changes coming with the MuleSoft PubSub connector, and how this can be used to track that a Platform Event indeed has been received and processed from Mulesoft. Also, an approach to testing the performance of the connector is described, using the tracking approach.

This document assumes the reader is aware of both sides, else it is recommended to read through the information in the footnotes.
First, a short context needs to be set recalling core concepts of the Salesforce Streaming API and the MuleSoft internal storage.

ReplayID and MuleSoft Objectstore

ReplayID

On the Salesforce Event Bus each published event message is assigned an opaque ID contained in the `ReplayId` field. The `ReplayId` field value, which is populated by the system when the event is delivered to subscribers, refers to the position of the event in the event stream[5].
The event stream has a message durability of now 72 hours for the “high volume” platform events-
Using the `ReplayId` it is possible to receive a message from the stream within that period.



[1] See here in the Streaming API Developer Guide
[2] The New Salesforce Event Bus
[3] Pub/Sub API: Building Event-Driven Integrations Just Got Even Easier
[4] Introducing the MuleSoft Connector for Salesforce’s Pub/Sub API
[5] Streaming API - Message Durability

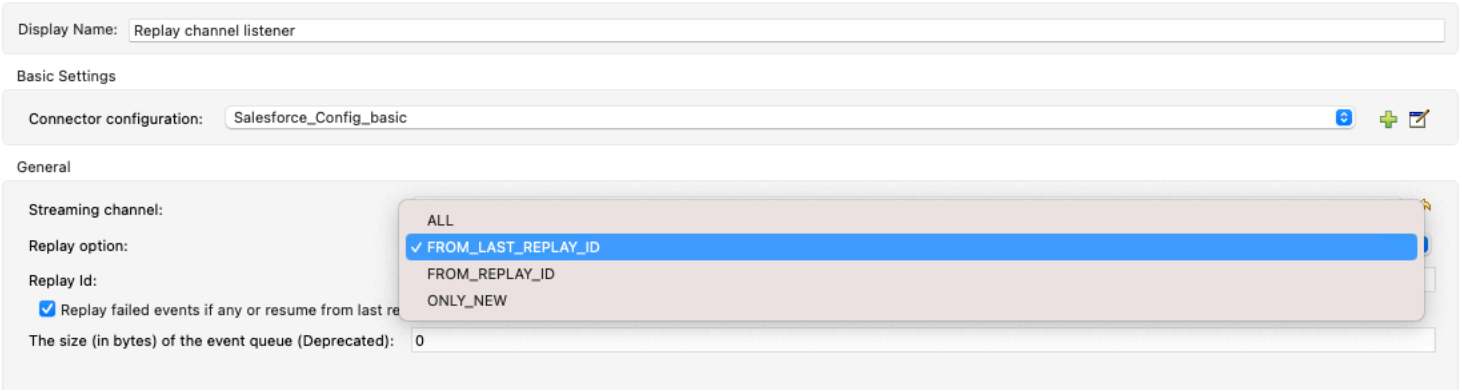
There are also two more acceptable values for the `ReplayId` defined to control the receiver side:

Replay Option	Description	Usage
`ReplayId`	Subscriber receives all stored events after the event specified by its `ReplayId` value and new events.	Catch up on missed events after a certain event message, for example, after a connection failure. To subscribe with a specific replay ID, save the replay ID of the event message after which you want to retrieve stored events. Then use this replay ID when you resubscribe.
-1	(Default if no replay option is specified.) Subscriber receives new events that are broadcast after the client subscribes.	We recommend that clients subscribe with the -1 option to receive new event messages. If clients need to get earlier event messages, they can use any other replay option.
-2	Subscriber receives all events, including past events that are within the retention window and new events.	Catch up on missed events and retrieve all stored events, for example, after a connection failure. Use this option sparingly. Subscribing with the -2 option when a large number of event messages are stored can slow performance.

What does that mean for a MuleSoft subscriber?

The subscriber process needs to carefully track, which replayID it has processed to ensure it does not ~~miss~~[6] or duplicate process events in case of downtime and restart (let it be intentional or due to unintentional process recovery).

The MuleSoft connector for the legacy CometD approach had a built-in approach[7] to keep track of the *received* events and natively allowed a developer to define the replay behavior, using a built-in MuleSoft Objectstore facility. See here the example configuration UI:



[6] "miss" … will be discussed later in this document

[7] MuleSoft Connector Replay Listener Documentation

The new MuleSoft PubSub connector[8] has been streamlined and delegates the responsibility of controlling the replay behavior to the developer. This allows advanced replayID tracking, to correctly ensure that the event has been received *and processed*.
Here is the configuration UI of the PubSub Connector

This shows that it is required to specify a dedicated Object Store.

Mulesoft Object Store

The MuleSoft object store is a facility for storing objects in or across Mule applications. Mule runtime engine (Mule) uses object stores to persist data for eventual retrieval. Internally, Mule uses object stores in various filters, routers, and other message processors that need to store states between messages.[9] Especially for CloudHub deployments it is important to consider the features of Object Store v2[10].
It is important to note that the free Object Store v2 limits usage to 10 transactions per second (TPS) per application. There is a premium add-on available to allow up to 100 TPS per app.[11]

Now, knowing the limits of the free Object Store a decision has to be made on how to deal with it. On one hand, one could purchase the premium add-on. Alternatively, a compromise could be made to say, let's accept the standard and let's work around it.
For this document, the second approach was created and shall be described in the section below.

Keeping track of the received and processed messages

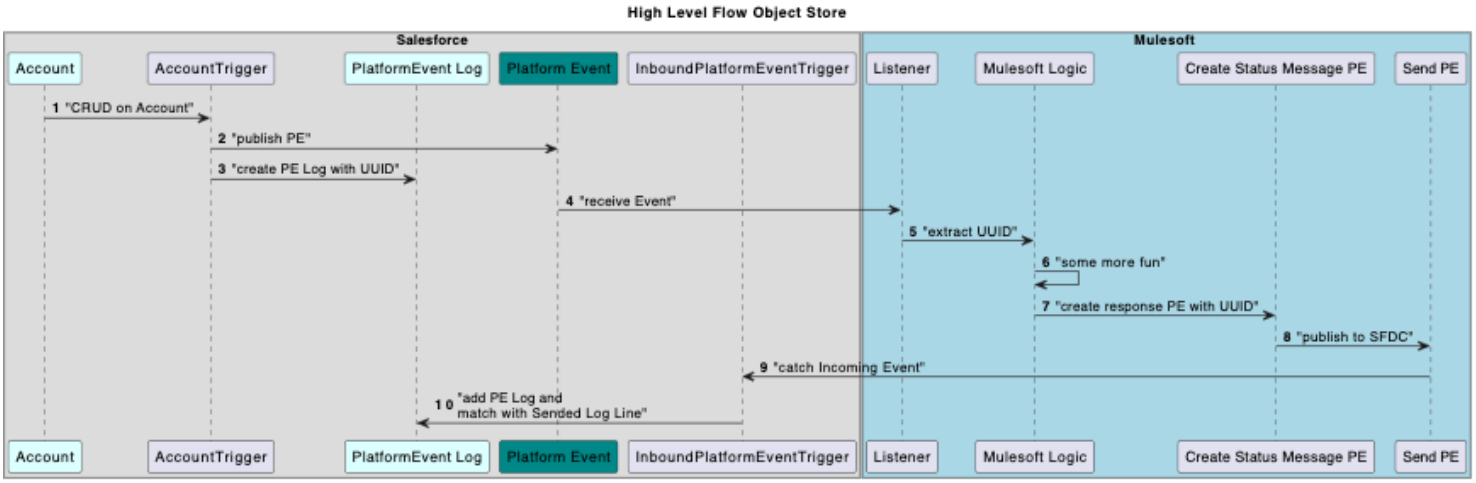
One of the challenges of the salesforce provided event-based integration is, to understand if a particular message has been received by a subscriber (and ideally, been processed)[12]. It is understood that this requirement can only be met with additional logic. Salesforce introduced earlier this year a new functionality, based on the "UUID" field supporting this message tracking[13].

In the following section, an approach to track messages using the new PubSub connector shall be outlined.

---

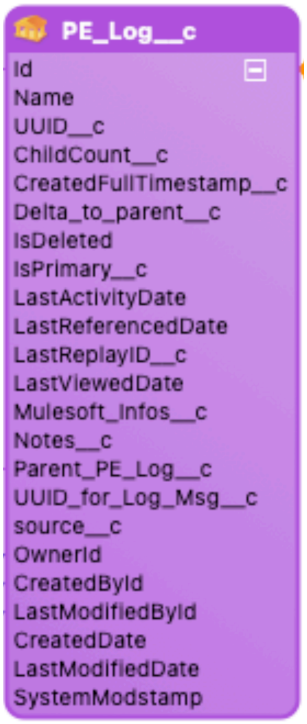
[8] Salesforce Pub/Sub Connector 1.0
[9] Object Store Documentation
[10] Object Store v2 Features
[11] Object Store Note
[12] See Wikipedia article - section "Disadvanatges"
[13] Identify and Match Event Messages with the EventUuid Field

High level, the flow is shown on the drawing below:



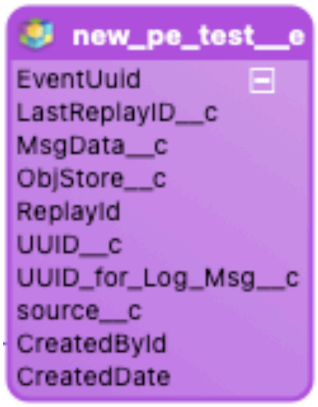
This shows several interesting details that are discussed below.

Let's look first at the Salesforce side



- It uses a custom object, called *PE_Log__c*
- The logic, publishing the platform event, needs to capture the UUID as outlined in the documentation, and writes it to the *PE_Log__c* object[14]

```
32      // add to PE_Log for publish
33      List<PE_Log__c> pelogs = new List<PE_Log__c>();
34
35      // Inspect publishing result for each event
36      for (Database.SaveResult sr : results) {
37          PE_Log__c pl= new PE_Log__c();
38          if (sr.isSuccess()) {
39              System.debug(DebugClass + 'Successfully published event:' + sr);
40              System.debug(DebugClass + 'UUID=' + EventBus.getOperationId(sr));
41              pl.UUID__c = EventBus.getOperationId(sr);
42              pl.IsPrimary__c = true;
43          } else {
44              for(Database.Error err : sr.getErrors()) {
45                  System.debug(DebugClass + 'Error returned: ' +
46                               err.getStatusCode() +
47                               ' - ' +
48                               err.getMessage());
49              }
50          }
51          pelogs.add(pl);
52      }
```



- There is also an Inbound platform event, designated to handle the response message from Mulesoft called *new_pe_test__e*
- A trigger, listening on the platform event *new_pe_test__e* captures the message and adds it to the *PE_Log__c* and, using a helper logic, relates it to the record, created upon publishing (that one has been flagged as "IsPrimary").
  The code for the trigger and helper logic can be found in this git repo

Now, what about the MuleSoft side?

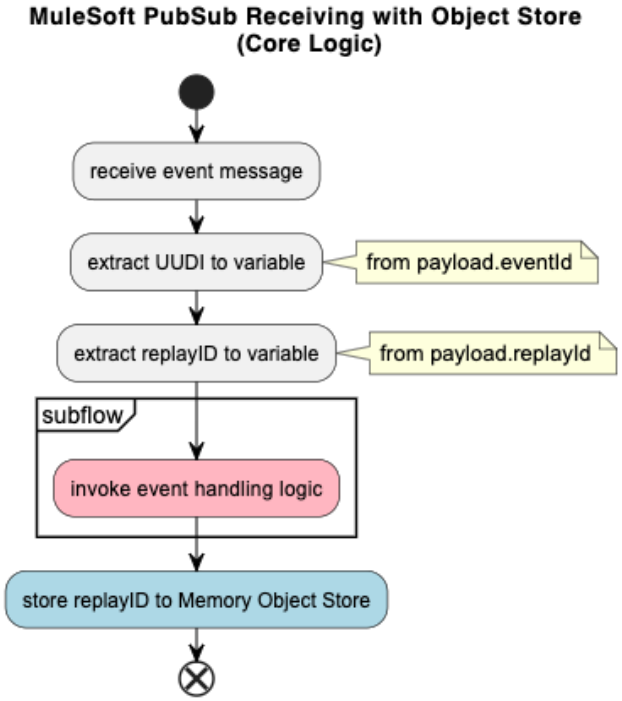
The MuleSoft event listener App contains two major functionalities:

(1) Listen to the platform Event channel

(2) Publish the response to the Salesforce Pub Sub Bus

[14] Full Trigger Source Code here

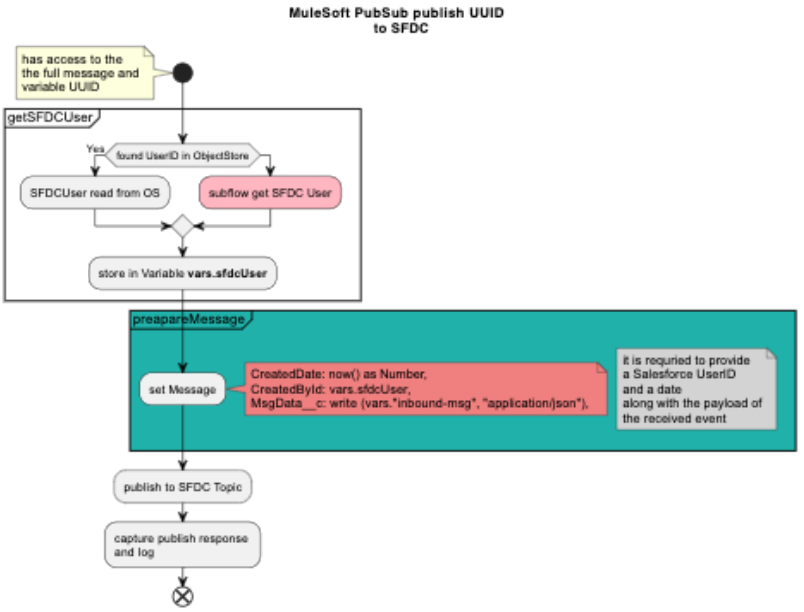
The core flow looks like shown on the following drawing:



In fact - with that, the goal to receive the platform event message - has been achieved.
Now, Salesforce needs to be informed about that.

Publishing event to Salesforce

That is done in the event handling logic as shown before.
A closer look at this part shows this logic:



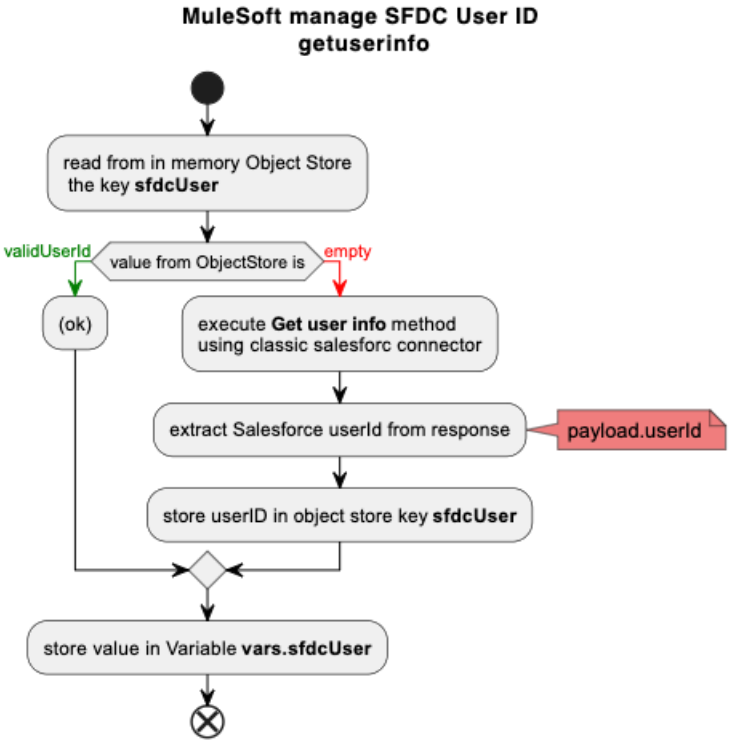
This brings us to one more thing - in order to publish via the MuleSoft PubSub connector it is required to set two fields in the message with valid content:

(a) `CreatedDate`: this needs to be in Unix time
(b) `CreatedById`: this needs to be a valid Salesforce User ID … it is recommended to use the same user as subscribing to with the PubSub Connector.

And here is one caveat - the PubSub Connector does not easily allow retrieving the Salesforce userID.
To handle this, one option would be persisting the Salesforce userID value in the config file. However, this is additional overhead and redundant. So, it is preferred to retrieve it and persist it in the in-memory object store as part of the mule application.

For this, first it is recommended to use the in-memory object store and create a new key to store the Salesforce userID.

With this, a logic like this (using the classic Salesforce connector) has been implemented in a dedicated sub-flow:



Mulesoft Pub Sub Connector - Message durability and Message Replay

The Salesforce event bus stores messages for 72 hours. The responsibility of keeping track of which messages have been processed is in the hand of the receiver. As shown in the previous section, using a combination of Salesforce and MuleSoft it is possible to keep track of the processed messages.

However - there is one other functionality to be discussed - the *Replay option.* It had been mentioned already earlier in this document here.

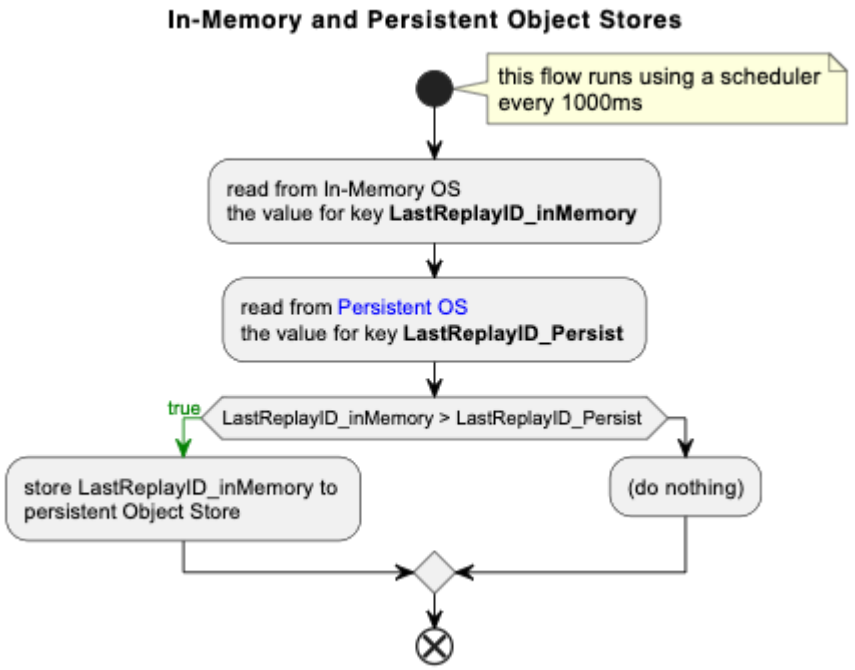
In fact - it is recommended to design the pub-sub receiver flow so that it can reasonably recover after an intended or unintended restart. Specifically, this means, the receiver should be configured to pick up events from the last processed `ReplayId`[15] only. And the only way the receiver can get this is by reading a key from the *persistent* object store.

Buffering the standard object store for higher transaction access

As described above, when reading and processing an event the last processed event is stored to the in-memory object store, using a key *LastReplayID_inMemory*.

[15] There are org wide Salesforce governer limits for reading events from the streaming API. See High-Volume Platform Event Default Allocations

In order to transfer that value to the persistent cloud hub object store, a scheduler flow has been created with this logic

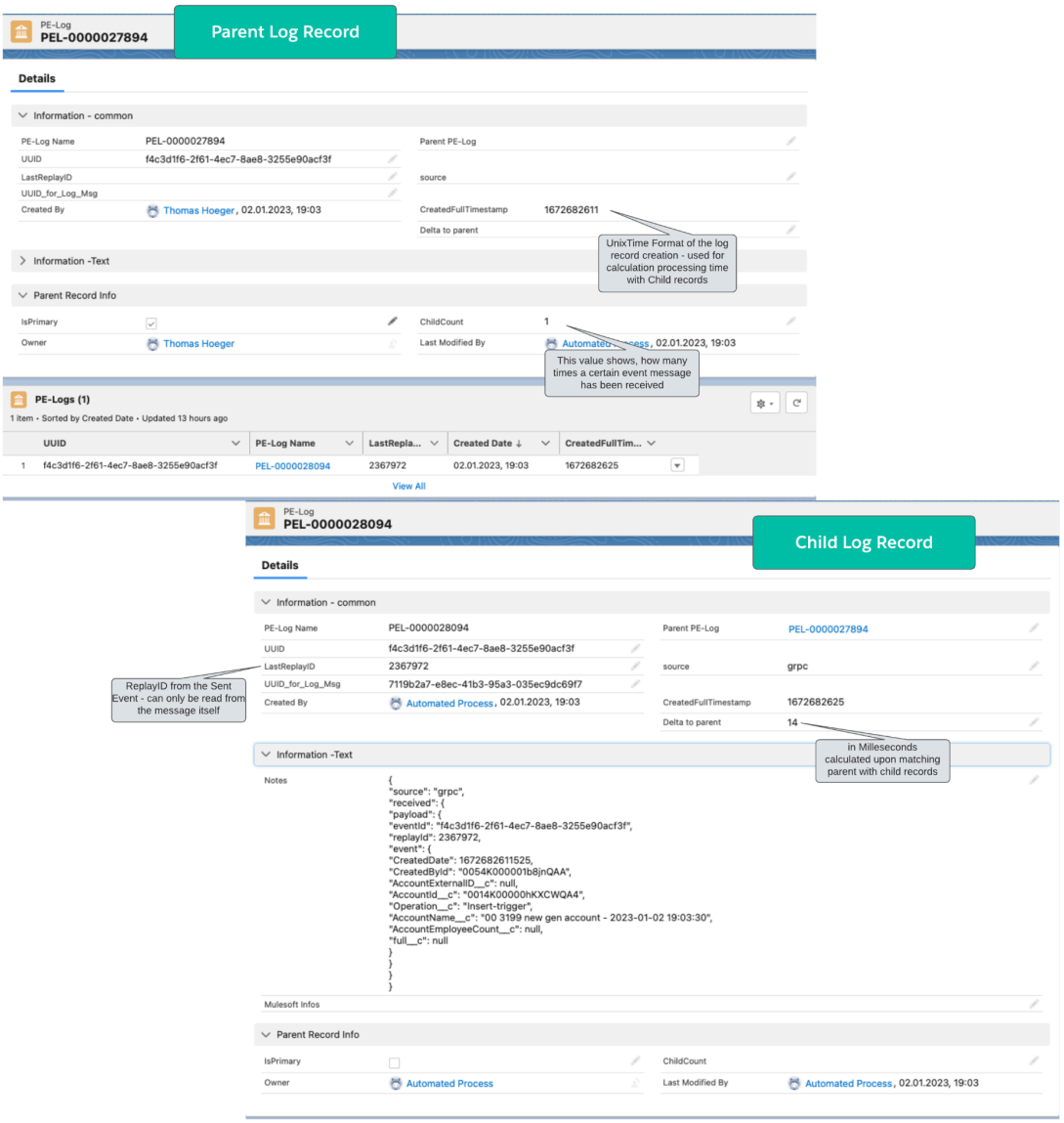


The consideration is as follows:

- The standard object store allows max 10 TPS. The scheduler would write every second - so this is 1 TPS
- If there are processed more events in a second, and it would come to an unexpected termination, than it can happen that after a restart events are re-read from the topic and processed a second time. So it is recommended to ensure that the event processing logic is idempotent unless intended (e.g. for the Event log it is important to log it as often as it is touched)

Tracking Example

The screenshot below shows a representation of a fully processed and acknowledged platform event



With this approach it is possible to understand, which events have been processed in a persistent way, also allowing to see the event message data if needed for debugging
Having timestamps additionally allows for us to get t an initial view of the processing performance of the pub-sub receiver.

Caveats

The above described approach illustrates the functionality of the pub-sub protocol and the Salesforce and MuleSoft capabilities.
The approach was built as a prototype, with that several assumptions and considerations shall be outlined here

Salesforce side

Beside the apex code in triggers and the helper class also a new object solely for tracking has been created. This can grow significantly in a high-volume scenario. So, there are further considerations on cleaning this object up on a regular basis.

Also … if there is a desire for permanent logging of the whole platform event history, salesforce core might not be the best place with the current capabilities.[16]

Integration approach for the response

In fact, the processing update to Salesforce is done via the Pub/Sub API as well. For the sake of this approach using publishing to the Pub/Sub API was chosen, also with the intent to showcase the details to be taken care of when using this new MuleSoft Connector
More generally, one could argue on the approach and if a request/response approach using a REST api call would be better. This topics deserves a dedicated discussion, no being the scope of this document.

Just to note, for time measurements as shown above, this async approach does introduce overhead, since also the reaction time of the subscriber trigger is included in the calculation of the delta. So, it is more relevant here to showcase a relative comparison of timing behavior.

BatcheventSize - How does the work?

One of the key advancements of the PubSub Receiver is that it allows controlling the amount of messages received and processed in a given period of time.

Using the tracking setup described above an initial test using a Salesforce developer org and MuleSoft Cloud Hub was executed.

As a test case, in Salesforce 500 records were created using an Apex code. After all the processing, in Salesforce the “delta” time between creating the log entry during record creation and the log entry from the confirmation message was analyzed, as well as the MuleSoft worker statistics were reviewed

Current summary:

- Using BatcheventSize=1 vs 100 triples the time until salesforce records the processing, which is still in the area milliseconds
- A 0.1 vCore worker is sufficient to handle the load - there is not seen a performance improvement when using a 0.2 core worker

Please find below the test measurements:

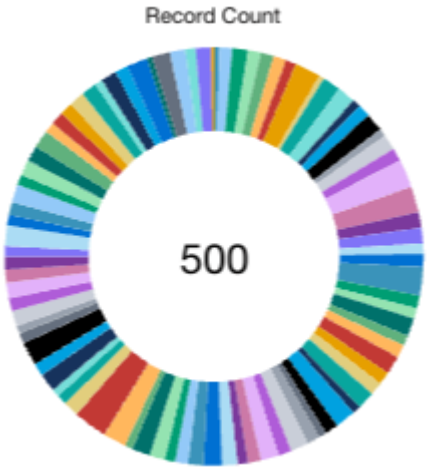
[16] There are considerations if salesforce would building something - pls check with the PM Tyson Read

Test case 1

Mulesoft Worker 0.1 vCores, pubsubBatchSize =1

The records had been created in Salesforce at 15:42:26

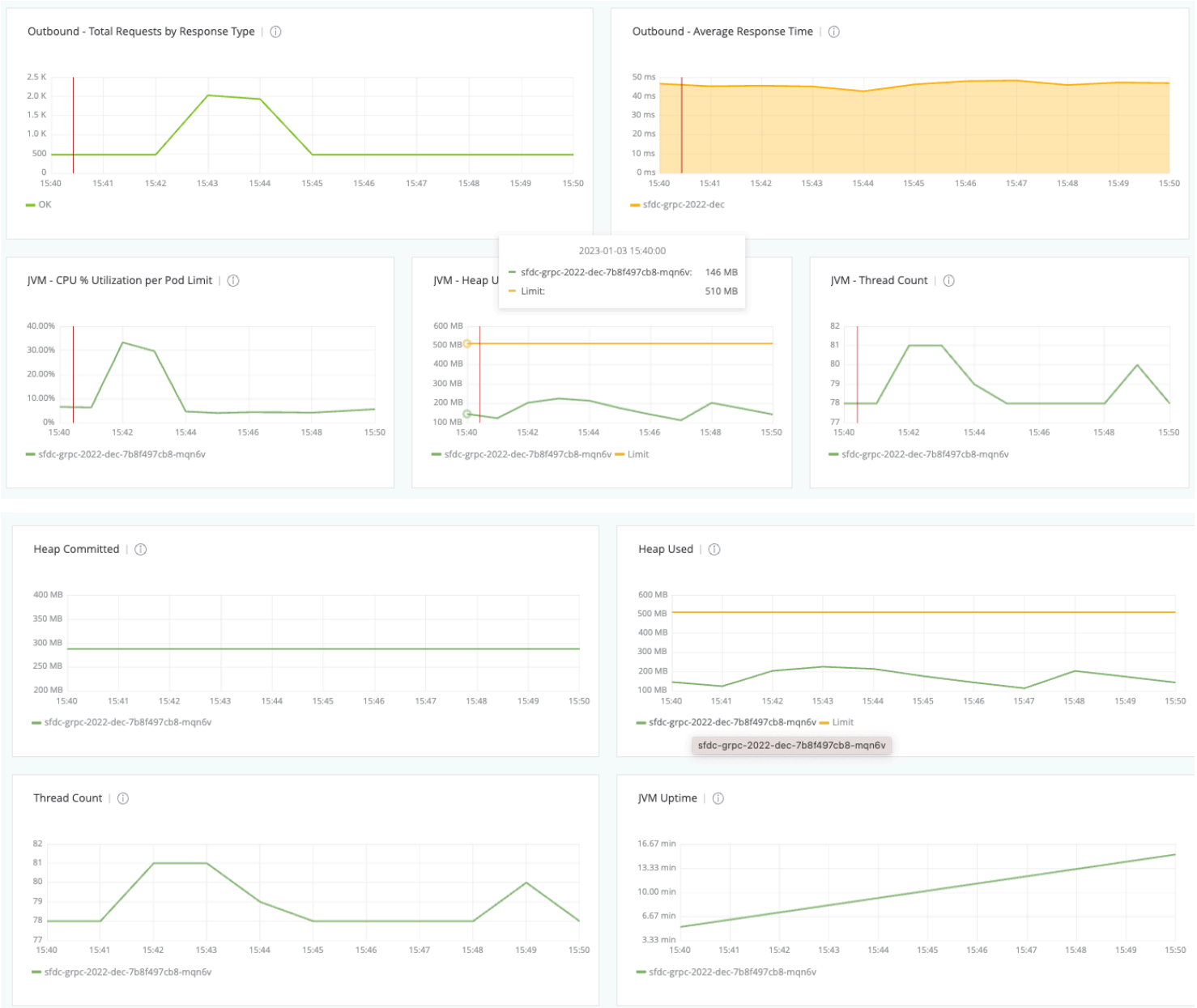
In Salesforce we see Delta times from 3 to 93 - as shown here



Delta to parent	Record Count
3	1
4	6
5	4
6	6
7	7
8	2
9	8
10	5
11	6
12	3
13	6
14	6

Delta to parent	Record Count
93	2
92	5
91	6
90	5
89	5
88	5
87	4
86	10
85	2
84	7
83	7
82	3

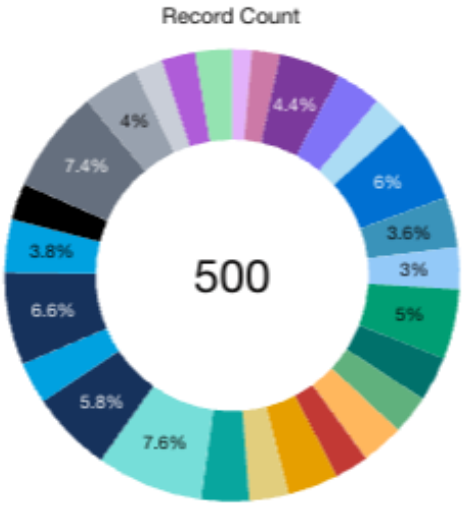
And MuleSoft shows these counter



Test case 2

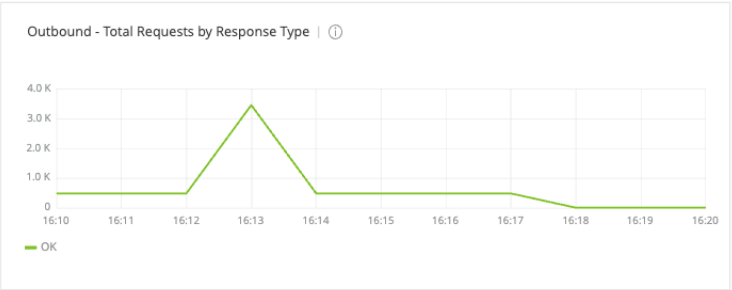
Mulesoft Worker 0.1 vCores, pubsubBatchSize =100
The records had been created in Salesforce at 16:13:04
In Salesforce we see Delta times from 3 to 28 - as shown here

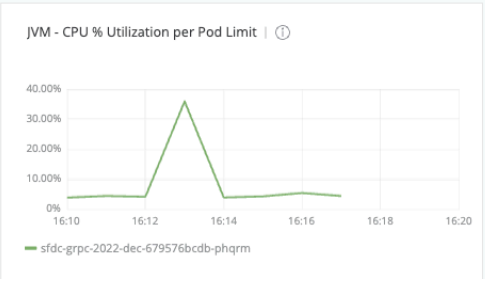
And MuleSoft shows these counters

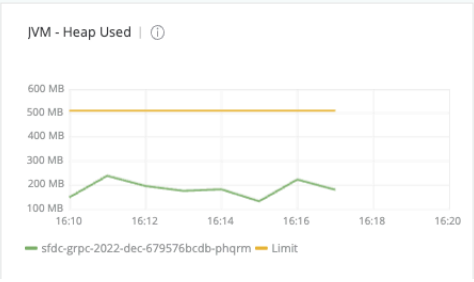


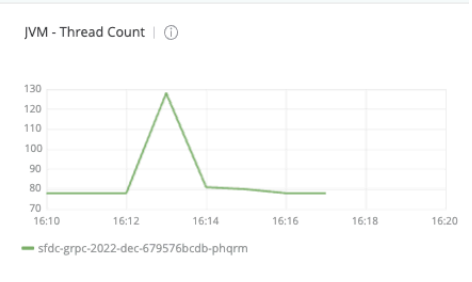
Delta to parent	Record Count
1	13
2	12
3	10
4	20
5	37
7	13
8	19
9	33
10	15
11	29

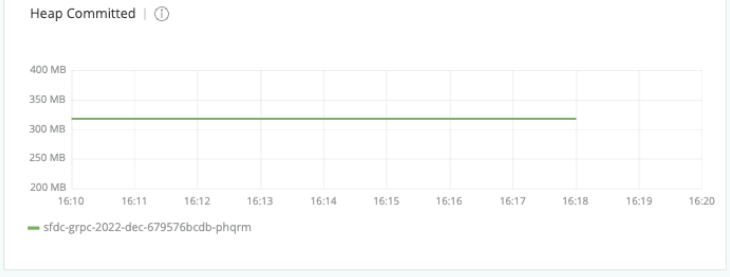
Delta to parent	Record Count
28	7
27	10
26	22
25	16
24	12
23	30
22	18
21	15
20	25
19	16

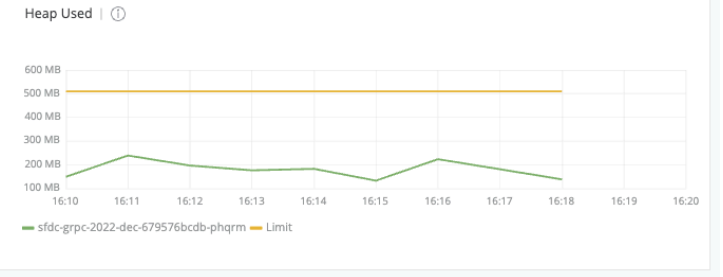


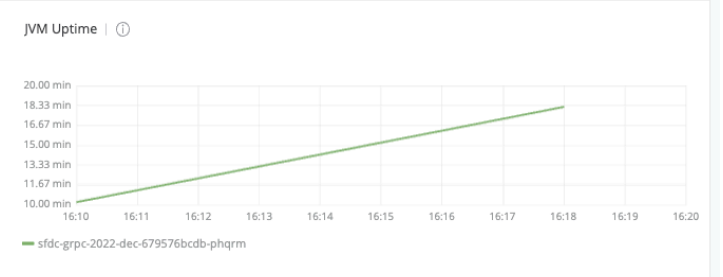










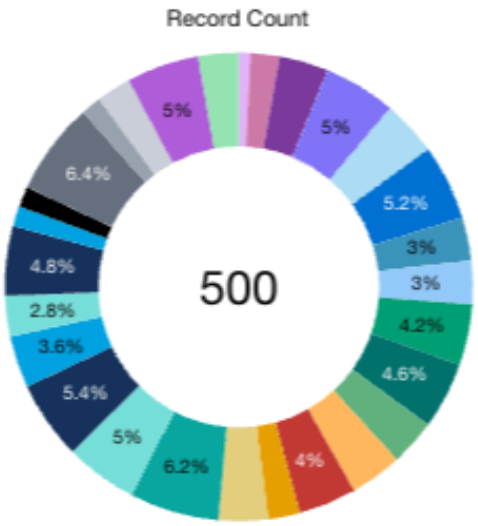


Test case 3

Mulesoft Worker 0.1 vCores, pubsubBatchSize =500

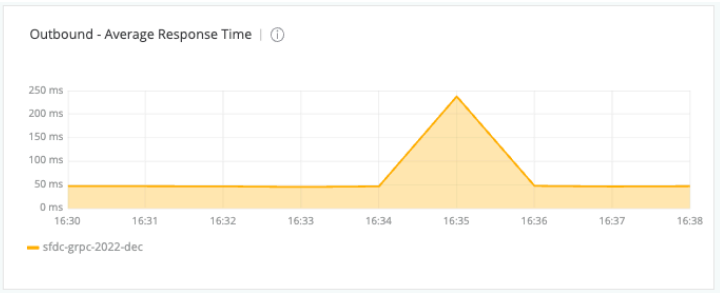
The records had been created in Salesforce at 16:34:30

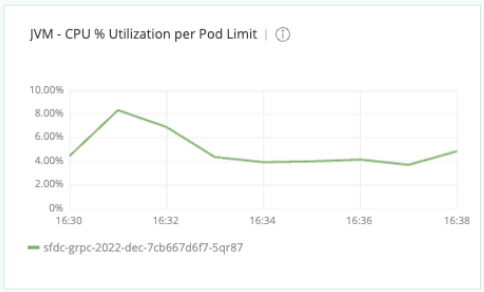
In Salesforce we see Delta times from 1 to 28 - as shown here

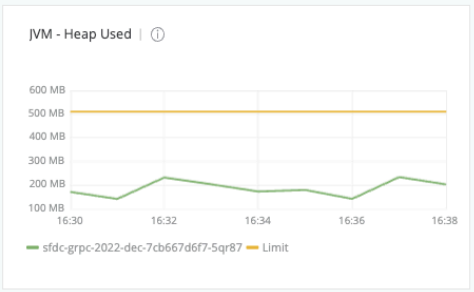


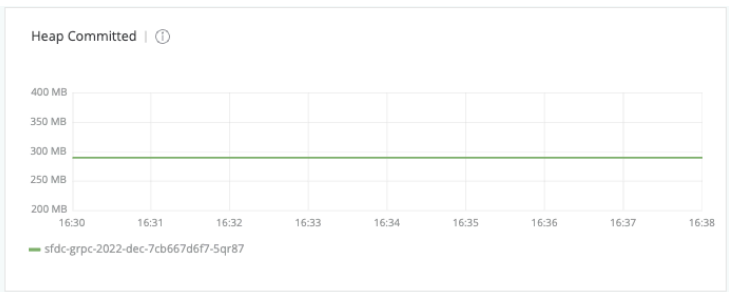
Delta to parent	Record Count
1	14
2	25
3	12
4	7
5	32
6	7
7	7
8	24
9	14
10	18

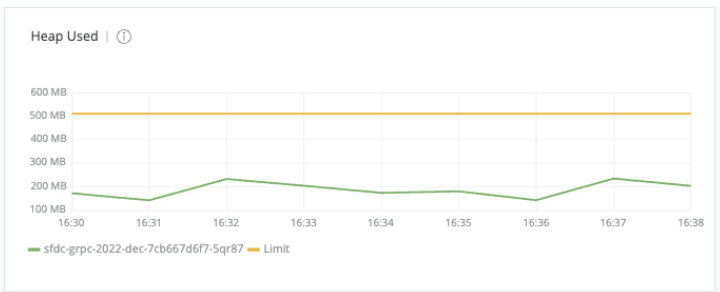
Delta to parent	Record Count
28	4
27	10
26	17
25	25
24	19
23	26
22	15
21	15
20	21
19	23

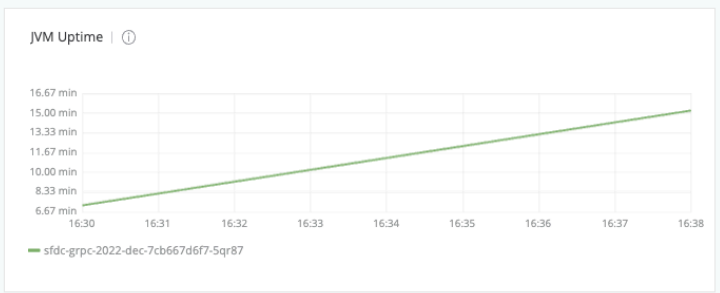










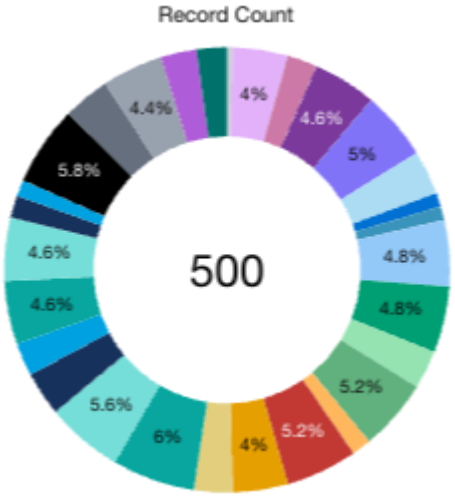


Test case 4

Mulesoft Worker 0.2 vCores, pubsubBatchSize =500

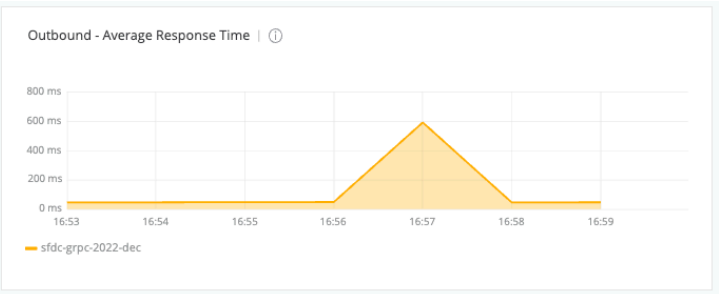
The records had been created in Salesforce at 16:56:45

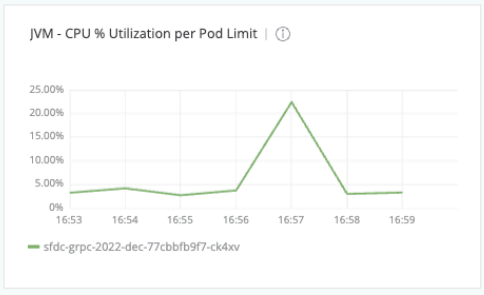
In Salesforce we see Delta times from 1 to 30 as shown here

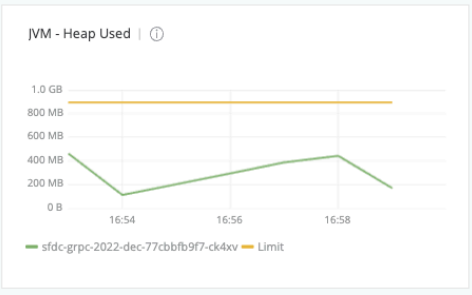


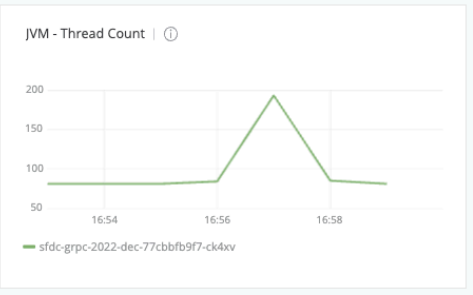
Delta to parent	Record Count
1	11
2	13
3	22
4	17
5	29
6	6
7	8
8	23
9	23
10	12

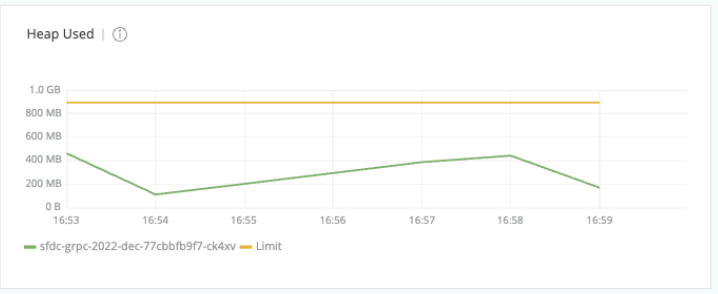
Delta to parent	Record Count
30	2
29	20
28	11
27	23
26	25
25	16
24	5
23	5

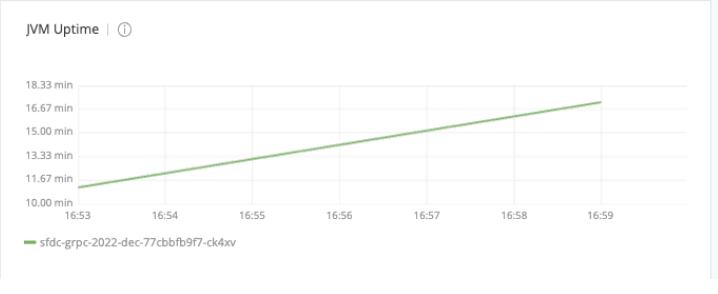












Appendix

Example MuleSoft Worker Log messages

Example Log messages during pub-sub receiver startup - showing that the replay I was read correctly:

```
Python
2023-01-02T17:58:30.82Z INFO [xbjdp] PubSubConnection [MuleRuntime].uber.02:
[sfdc-grpc-2022-dec].uber@org.mule.runtime.module.extension.internal.runtime.source.ExtensionMessageSource.lambda$
null$17:435 @7e46a13d - PartnerConnection validated. Validating Pub/Sub API Connection. Arguments:
{definedPlatformEvents={new_pe_test__e=new_pe_test, InboundAccountPE__e=InboundAccountPE,
AccountUpdatePE__e=AccountUpdatePE, CasePE_Out__e=CasePE Out, Cloud_News__e=Cloud News},
connection=com.mulesoft.connector.Salesforce.pubsub.internal.connection.PubSubConnection@395b85ba (sessionId:
00D4K0000039G1S!ARoAQKTY****N8D, instanceUrl: https://pe-demo-001-dev-ed.my.Salesforce.com, tenantId:
00D4K0000039G1SUAU, apiVersion: 54.0, partnerConnection: com.sforce.soap.partner.PartnerConnection@666d1039,
metadataConnection: com.sforce.soap.metadata.MetadataConnection@44e47e85).}.

2023-01-02T17:58:30.876Z INFO [xbjdp] PubSubConnection [MuleRuntime].uber.02:
[sfdc-grpc-2022-dec].uber@org.mule.runtime.module.extension.internal.runtime.source.ExtensionMessageSource.lambda$
null$17:435 @7e46a13d - Pub/Sub API connection validated. TopicInfo retrieved successfully. Arguments:
{requestedTopicInfo=topic_name: "/event/new_pe_test__e"
tenant_guid: "CORE/prod/00D4K0000039G1SUAU"
can_publish: true
can_subscribe: true
schema_id: "c00JfRi-Fq2tNZTYq6Cuhw"
rpc_id: "2e60db94-8fcf-4764-8cc0-9a14d05bb21d"
, connection=com.mulesoft.connector.Salesforce.pubsub.internal.connection.PubSubConnection@395b85ba (sessionId:
00D4K0000039G1S!ARoAQKTY****N8D, instanceUrl: https://pe-demo-001-dev-ed.my.Salesforce.com, tenantId:
00D4K0000039G1SUAU, apiVersion: 54.0, partnerConnection: com.sforce.soap.partner.PartnerConnection@666d1039,
metadataConnection: com.sforce.soap.metadata.MetadataConnection@44e47e85).}.

2023-01-02T17:58:30.911Z INFO [xbjdp] CloudObjectStore [MuleRuntime].uber.02:
[sfdc-grpc-2022-dec].uber@org.mule.runtime.module.extension.internal.runtime.source.ExtensionMessageSource.lambda$
null$17:435 @7e46a13d - Object store last attempt was less than 10000ms before, will not attempt until next window

2023-01-02T17:58:33.918Z INFO [xbjdp] SubscribeChannelSource [MuleRuntime].uber.02:
[sfdc-grpc-2022-dec].uber@org.mule.runtime.module.extension.internal.runtime.source.ExtensionMessageSource.lambda$
null$17:435 @7e46a13d - Replay id validation passed. Startup continues. Arguments:
{topic=/event/AccountUpdatePE__e, replayId=2367429}.

2023-01-02T17:58:33.922Z INFO [xbjdp] ExtensionMessageSource [MuleRuntime].uber.02:
[sfdc-grpc-2022-dec].uber@org.mule.runtime.module.extension.internal.runtime.source.ExtensionMessageSource.lambda$
null$17:435 @7e46a13d - Message source 'subscribe-channel-listener' on flow 'sfdc-grpcFlow' successfully started
```