American Journal of Computer Architecture 2024, 11(4): 35-41
DOI: 10.5923/j.ajca.20241104.01

Event Driven Architecture in the Salesforce EcoSystem

Prajeet Gadekar

Professional Services, Salesforce Inc, NYC, USA

Abstract This document talks about how Event-driven architecture (EDA) can be implemented in the Salesforce ecosystem. EDA is crucial in today's technology landscape because it allows systems to be highly responsive, scalable, and resilient. In an era where real-time data processing and instant user feedback are paramount, EDA enables applications to react to events as they occur, ensuring timely and efficient handling of information. This architecture is particularly beneficial for microservices and serverless computing, where decoupled services can independently consume and produce events, leading to greater flexibility and easier maintenance. Moreover, EDA supports better fault tolerance and load distribution, as systems can dynamically adjust to varying workloads and failures. As businesses strive to provide seamless and personalized experiences, the ability of EDA to facilitate rapid adaptation and continuous integration of new functionalities becomes indispensable.

Keywords Salesforce, CRM, Event-driven, EDA, Pub-sub

1. Introduction

Event driven communication revolves around the publisher -subscriber model. The publisher or the event generator broadcasts a message on a channel. Any listeners subscribed to the channel receive the message and can act upon it as it suits their needs. It's like a tv station broadcasting signals in an area, a consumer who has the antenna pointed in the right direction will receive the signal.

It gets us out of point to point integrations and decouples the event producers from event consumers thus simplifying communication in a complex connected enterprise.

Our goal here is to explore the options available to implement event driven architecture in Salesforce, compare them and make a recommendation.

2. Event Driven Tech in Salesforce

In this article we are going to talk about the evolution of event driven architecture on the core Salesforce.com platform. We will talk about the different options that are available today and the use cases in which those options should be exercised. Finally we will get into licensing and limits, and declutter the confusion around what limits apply on each one of these and how to extend them.

2.1. Generic Events

This is where it all started, but we will not deep dive as they are now Legacy, very very Legacy. Generic events involved creation of a streaming Channel without a predefined event schema in your Salesforce org and external or internal subscribers with EMP connectors. REST API was used to both push events and monitor the channel.

2.2. PushTopic Events

Salesforce's Journey with events really took off with pushTopics. PushTopic events provide a secure and scalable way to receive notifications for changes to Salesforce data that match a SOQL query that you define. What that means is every time a CRUD operation is performed the PushTopic query is analyzed for impact, and if the change fell within the scope of the query, a notification is generated.

A PushTopic respects the users access based on sharing rules so they will only receive notifications for the records they have access to, this, with the ability to limit the stream of events to only those events that match a subscription filter helps control the volume a user receives.

The main intent when PushTopic was designed was to enable the ability of refreshing the UI of a custom app from Salesforce record changes. Push topic listener is a cometD client based on the Bayeux platform (long polling) so the listener could be a page in the Salesforce application or it could be an application server or a client outside the Salesforce platform.

PushTopics can be created from apex, API or Workbench.

The CRUD part of when the notification should be generated is controlled by these four fields on a PushTopic

- NotifyForOperationCreate
- NotifyForOperationUpdate
- NotifyForOperationDelete

* Corresponding author:
prajeetgadekar@gmail.com (Prajeet Gadekar)
Received: Jul. 9, 2024; Accepted: Jul. 25, 2024; Published: Jul. 31, 2024
Published online at http://journal.sapub.org/ajca

- NotifyForOperationUndelete

The NotifyForFields determines what is evaluated

- All (any change on in the record)
- Referenced (referenced in the query SELECT or WHERE clauses)
- Select (one of the fields in the SELECT clause changes)
- Where (one of the fields in the WHERE clause changes)

Let's take a look at an example:

SELECT Id, Name, BillingCity, BillingState, BillingPostal Code, Phone from Account WHERE NumberOfEmployees > 10000

In this example the payload includes everything in the select clause of the query, however, if the subscriber does not have access to any of these fields it is omitted from the notification the subscriber receives. On the other hand if the subscriber does not have access to the NumberOfEmployees field the subscription will fail. The query should always have an ID field, sub selects and aggregate queries are not supported.

Filtered subscriptions can be used to further filter the notifications received.

/topic/thisTopic?BillingState='NY'&BillingCity='Newy ork'

Salesforce stores PushTopic events, generic events and standard-volume events for 24 hours and they are available for replay. (High-volume are stored for 72 hours, but we will come to that later)

Unfortunately, PushTopic events are also now considered Legacy, which means no investment from product to improve and a limited support, so unless you have an existing implementation PushTopics should not be in the consideration.

2.3. Platform Events

Platform events are secure and scalable messages that contain data. They are an improvement from generic events and PushTopics. Platform events have a predefined metadata and really opens up the possibilities with publishing and subscribing with Apex Flows API's Etc.

We already know what an event, event message, producers and consumers are. It's important to understand the difference between an event Channel and event Bus. An event Channel is a stream of events on which and even producers send event messages and event consumers read those messages, the channel is for a single platform event or a custom Channel that groups event messages for multiple platform events.

An event Bus is multi-tenant multi-cloud events storage and delivery service for the publish subscribe model. Event bus is based on time ordered log which ensures that event messages are stored, delivered and can be retrieved in the order they are received by Salesforce.

2.3.1. Standard Platform Events

It's Interesting to note that Salesforce uses platform events to run the platform and these are called standard platform events. A good example would be the batch Apex error event which reports errors encountered in batch Apex job executions; you can subscribe to standard platform event streams using the subscription mechanism that the event supports.

Event monitoring using shield has more than fifty types of events that are supported, these are also standard platform events.

2.3.2. Custom Platform Events

These are the platform events that you are able to define and use for your use cases. You define a platform event in the same way that you would Define a custom object. giving it a name and adding custom fields. At a very high level every time a record is created in this newly defined object, you're basically creating a platform event and all the fields that you just defined make the payload of the message for that event.

So in summary a platform event is a special kind of Salesforce entity similar in many ways to an sObject. An event message is an instance of a platform event similar to a record is an instance of a custom or standard object. but unlike a regular object you cannot update or delete event records, you also can't view event records in the Salesforce UI.

Custom platform events can be listened to both internally and externally. Internally just think about it as any other object in salesforce, just like you can have a trigger or a flow that fires on CRUD action, a flow and trigger can also listen to an event.

From a security standpoint a user can be given permission to publish a platform event and to subscribe to a platform event.

From a transaction control perspective you can either publish an event immediately or you can publish it after the database commit, if you choose the prior you cannot roll back the published event by going to a previous setsavepoint() or rollback().

Unlike standard-volume platform events the high-volume platform events are stored for 72 hours and can be replayed using a comedD client.

2.3.2.1. Platform Event Definition

The definition is no different than a custom object definition except that the system appends a __e to create the API name of the event.

Two fields are critical to understand, ReplayID which is an opaque ID assigned to each message. ReplayId field value is populated by the system when the event is delivered to subscribers refers to the position of the event in the event stream.

EventUuid is a UUID that uniquely identifies a platform event message.

From the field types supported, note that a long text area is supported. If you have use cases where you need to create a complex json and send it as part of the payload to avoid a call back from the listener, that's your field. You can use Vlocity

tools like DataRaptor to transform your SOQL output into a JSON.

2.3.2.2. Publishing a Platform Event

From a flow, workflow or a process you can create a platform event by using the 'Create a record' equivalent in these tools. You are basically creating a record in the platform event object that's all that is needed for publishing the event.

From apex it's no different, you are taking a DML action to create a record. All salesforce API's SOAP, REST etc that can create a record in Salesforce can also publish a platform event.

A special mention is required for the pub/sub API's, they are relatively new and based on gRPC.

rpc Publish (PublishRequest) returns (PublishResponse); — publish one event.

rpc PublishStream (stream PublishRequest) returns (stream PublishResponse); — publish multiple events.

2.3.2.3. Subscribing to a Platform Event

A flow can be started or a paused flow can be resumed on receiving a platform event. Similarly a process can also subscribe to a platform event and get started off it. In Apex a trigger can process a platform event (on create event).

Lightning has the empApi component that works with both Aura and LWC's to subscribe to a streaming channel and listen to events. External clients can implement cometD. Implement your own CometD client or use EMP Connector, an open-source, community-supported tool that implements all the details of connecting to CometD and listening on a channel.

And finally pub/sub API's can also be used to subscribe just like publishing.

2.3.2.4. Custom Channels

You can create custom channels and add event types that will be part of the channel. Unfortunately CDC and Platform Events cannot be part of the same channel.

A channel is a good way to organize your domains, e.g you could have a channel for account management that can include all platform events related to accounts, contacts and their relationships. The subscriber then subscribes to the channel vs subscribing to individual platform events.

Custom channels can also be used to create filtered streaming so the subscriber to the channel only receives a subset of the events generated by a platform event. (filtered subscription where you could add a filtering criteria on a topic is not supported in platform events but achieved through channels)

2.3.2.5. Additional Security

If you have a shield encrypted org your data is encrypted at rest. You can extend that to encrypting your messages stored on the event bus. Before delivery to subscribers messages are decrypted, they are safe as they are delivered over a secure channel using HTTPS and TLS.

2.4. Change Data Capture

Change Data Capture are platform events that are published automatically by salesforce, based on changes to underlying data. The most common use case is real-time data replication establishing a continuous synchronization of new and updated data to the external system.

CDC events generate notifications of Salesforce record changes, including Create, Update, Delete and Undelete Operations.

Change events support most standard and all custom objects. CDC can be added to the default standard channel by simply selecting the entities (Objects)) you are interested in. Metadata API and Tooling API can also be used to create and retrieve object event selections for the default standard or even a custom channel.

2.4.1. Message Structure

```
{
  "schema": "<schema_ID>",
  "payload": {
    "ChangeEventHeader": {
       "entityName": "...",
       "recordIds": "...",
       "changeType": "...",
       "changeOrigin": "...",
       "transactionKey": "...",
       "sequenceNumber": "...",
       "commitTimestamp": "...",
       "commitUser": "...",
       "commitNumber": "...",
       "changedFields": [...]
    },
   "field1": "...",
   "field2": "...",
    . . .
  },
  "event": {
    "replayId": <replayID>
  }
}
```

Figure 1. A sample line CDC message structure

Note that the recordID is in the header, the body is only meant for other fields.

There are cases like bulk updates when one event will have a list of recordID's e.g mass update of a field or mass Delete on an object. There are exceptions to how recordID's are represented in this field when exceptional events occur, e.g when you delete a field in salesforce configuration setup, there is a notification via an event and it is very unique, but let's not get into those details for now to keep it simple.

Based on ChangeType (CRUD operation) the list of fields in the body of the message will differ.

Note that formula fields will not be included in the payload as they are placeholders and calculated realtime when accessed, on the other hand Roll up summary fields are included as they are recalculated as soon as there is a change in child records,

Create: The event message contains all non empty fields including system fields like created date and owner ID.

Update: Only includes the changed fields. Empty fields are included if a value was changed from non empty to empty. LastModifiedDate is included and so is Lastmodified ByID but only if the record is updated by a different user than the last save.

Delete: Empty body, the recordID of the deleted record is in the body.

Undelete: The body includes all non-empty fields from the original record as it was right before deletion. Pay load is similar to a crate.

The approach taken for Update and Delete presented a big challenge for enterprises that relied on an external ID to update downstream applications. This was thankfully fixed in Winter 20, you can now fix some fields to be always included in the body.

If you use pub/sub API's for subscription the payload is more favorable in that even empty fields are included in a Create, and the update includes all fields vs only changed fields, so you can always get a full sync of the record without worrying about missed events if any.

2.4.2.1. GAP Messages

Salesforce will sometimes send a gap event which only has the header information such as change type and record ID. It doesn't include fields in the body.

Some common situations in which this happens is — when the message size exceeds the limit of 1 mb, some custom fields are converted to a different type or format, internal error in Salesforce or some internal cleanup like archiving of activities (tasks and calendar events) that happen a the database level not at the application server level.

A best practice on receiving a gap event is to do a call back with the record ID's to retrieve the latest and greatest version of the record. (some gap events can be ignored on a case by case basis)

Gap Overflow events happen when a single transaction generates over 100k changes. For anything over 100k the system generates a gap overflow event.

Change Data Capture for all practical purposes are like specialized Platform Events, they have all the same properties and are treated exactly like Platform events, just that they are published automatically by the system. They also support encryption on the channel if desired.

3. Method & Case Studies

For evaluation of these different options we created a sandbox, a non-production test instance of Salesforce and applied these methods. As of the time of study generic events are no longer supported on the platform so the evaluation of generic events was limited to project references. We also collected feedback from client projects that used one of these methods to implement.

3.1. Case Studies

We are outlining two case studies based on real life implementations for real clients. Client names have been anonymized.

3.1.1. Use Case for a Leading Retail Company

Background: A leading retail company (anonymized as "RetailCo") faced challenges in synchronizing data across its various systems, including inventory management, customer relationship management (CRM), and order processing. The existing architecture relied heavily on batch processing, leading to delays and inefficiencies in real-time data updates.

Challenge: RetailCo needed a solution to ensure real-time data synchronization across its systems to improve customer experience, streamline operations, and enhance decision-making. The goal was to implement an event-driven architecture that could handle high volumes of transactions and provide immediate updates across all platforms.

Solution: RetailCo decided to leverage Salesforce Platform Events to build an event-driven architecture. Platform Events enabled the company to publish and subscribe to events, ensuring real-time communication between different systems.

1. Event Definition: RetailCo defined custom Platform Events to represent key business processes, such as OrderPlaced, InventoryUpdated, and CustomerUpdated. These events encapsulated the necessary data to be shared across systems.
2. Event Publishing: When a customer placed an order, the order management system published an OrderPlaced event. This event included details such as order ID, customer information, and product details.
3. Event Subscription: The inventory management system subscribed to the OrderPlaced event. Upon receiving the event, it immediately updated the inventory levels and published an InventoryUpdated event. Similarly, the CRM system subscribed to both OrderPlaced and CustomerUpdated events to keep customer records up-to-date.
4. Real-Time Processing: By using Platform Events, RetailCo achieved real-time data synchronization. The event-driven architecture ensured that all systems were immediately aware of changes, reducing latency and improving operational efficiency.

Results:

Improved Customer Experience: Customers received real-time updates on their orders, enhancing transparency and satisfaction.

Operational Efficiency: The real-time data flow reduced manual interventions and errors, streamlining operations.

Scalability: The event-driven architecture easily handled high transaction volumes, supporting RetailCo's growth.

Conclusion: By implementing Salesforce Platform Events, RetailCo successfully transitioned to an event-driven architecture. This transformation enabled real-time data synchronization, improved customer experience, and enhanced

operational efficiency. The flexibility and scalability of Platform Events provided RetailCo with a robust foundation for future growth and innovation.

3.1.2. Use Case for a Global Financial Services Company

Background: A global financial services company (anonymized as "FinanceCo") was struggling with data consistency across its various systems, including customer relationship management (CRM), transaction processing, and compliance monitoring. The existing architecture relied on periodic data synchronization, leading to delays and potential data discrepancies.

Challenge: FinanceCo needed a solution to ensure real-time data consistency across its systems to enhance customer service, streamline operations, and meet regulatory requirements. The objective was to implement an event-driven architecture that could capture and propagate data changes instantly.

Solution: FinanceCo chose to utilize Salesforce Change Data Capture (CDC) to build an event-driven architecture. CDC allowed the company to capture changes to Salesforce records and propagate these changes to external systems in real-time.

1. Change Event Definition: FinanceCo enabled Change Data Capture for key Salesforce objects, such as Account, Transaction, and ComplianceCase. This setup ensured that any changes to these objects would generate change events.
2. Change Event Publishing: When a record in Salesforce was created, updated, deleted, or undeleted, a corresponding change event was published. For example, updating an Account record triggered an AccountChangeEvent, containing details of the change.
3. Event Subscription: External systems, including the transaction processing and compliance monitoring systems, subscribed to these change events. Upon receiving an AccountChangeEvent, the transaction processing system updated its records to reflect the latest account information. Similarly, the compliance monitoring system subscribed to TransactionChange Event to ensure real-time compliance checks.
4. Real-Time Data Flow: By leveraging CDC, FinanceCo achieved real-time data propagation. The event-driven architecture ensured that all systems were immediately updated with the latest data changes, reducing latency and improving data accuracy.

Results:

Enhanced Customer Service: Real-time updates ensured that customer service representatives had the most current information, improving response times and customer satisfaction.

Operational Efficiency: The real-time data flow minimized manual data reconciliation efforts, reducing errors and operational overhead.

Regulatory Compliance: Immediate propagation of data changes enabled FinanceCo to meet stringent regulatory requirements, ensuring timely and accurate compliance reporting.

Conclusion: By implementing Salesforce Change Data Capture, FinanceCo successfully transitioned to an event-driven architecture. This transformation enabled real-time data consistency, enhanced customer service, and improved operational efficiency. The robust and scalable nature of CDC provided FinanceCo with a reliable foundation for future innovation and growth.

4. Evaluation

If you have read until this point, it’s pretty clear the only options for your consideration are Platform Events and Change Data Capture. The daily/monthly transaction limits on them fall into the same bucket, so that does not impact.

4.1. Change Data Capture vs Platform Events

If you have read until this point, it’s pretty clear the only options for your consideration are Platform Events and Change Data Capture. The daily/monthly transaction limits on them fall into the same bucket, so that does not impact.

It’s really up to the use case CDC should be used for

- Synchronization of downstream applications or databases Near real time.
- Especially useful when you have multiple downstream applications and want to ignore some events based on the origin of the change in Salesforce (if the change originated by and API by the same system)
- Subscribe to mass changes in a scalable way.

Platform events give you a much granular control on how and when events are published. Their payload is fully customizable.

4.1.1. Licensing and Limits

Let’s talk a little bit about the limits. Let’s first talk about limits you will likely not have to worry about for enterprise clients

Max # of Concurrent CometD clients across all channels and all event types — 2000 for unlimited and 1000 for Enterprise editions.

Max # of Custom channels — 100 for both performance and enterprise.

Max # of CDC entities — 5 for everyone (But you almost always have to buy add on licenses if you are serious about CDC).

Event Delivery — 50000 for Unlimited and 25000 for Enterprise in a 24 hour period.

The Limit on event delivery is a little tricky, it includes all external subscribers whether using CometB or pub/Sub API’s, but they also include lightning components or VF using empApi.

And remember if you deliver one event to five subscribers it is counted as five deliveries. So it’s almost always a best practice to implement a hub and spoke in an enterprise where one listener is listening to Salesforce events and distributing them to everyone in the enterprise.

Table below summarizes the limits for both CDC and Platform events for an unlimited org.

Note that all of these are common limits so 50000 deliveries apply to Platform events and Change Data Capture combined.

The limit on publishing is super high, 250k/hour not per day, so you can use Platform events internally subscribing through flows, processes and triggers without worrying about limits.

CDC & PE limits	Platform Events	CDC	One Add On License
Maximum number of platform event definitions that can be created in an org	100	NA	
Maximum number of concurrent CometD clients (subscribers) across all channels and for all event types	2000		
Maximum number of Process Builder processes and flows that can subscribe to a platform event	4000	NA	
Maximum number of active Process Builder processes and flows that can subscribe to a platform event	2000	NA	
Maximum number of custom channels that can be created. This allocation is separate from the one for custom change data capture channels.	100	100	
Maximum number of distinct platform events that can be added to a channel as part of channel membersIf the same platform event is added to multiple channels, it's counted once toward the allocation.	50	50	
Maximum number of entities, including standard and custom objects, that you can select for Change Data Capture.	NA	5	Unlimited
Event Publishing: maximum number of event notifications published per hour. (Applies to all publishing methods, including Apex, Pub/Sub API and other APIs, flows, and Process Builder processes.)	250000	NA as event us published by SF	275000/Hour
Event Delivery: maximum number of delivered event notifications in the last 24 hours, shared by all clients. (Applies to CometD and Pub/Sub API clients, empApi Lightning components, and event relays only.)	50000		Additional 100000 = 150000/day, Enforced monthly at 4.5M/Month

Figure 2. Common CDC and Platform event limits

The Add-on license adds 100k additional deliveries at a time. But that's not it. With even one add-on license, the limit is not enforced daily any more, it is enforced monthly up to a limit of 4.5M/Month. So if your business has highs and lows through the week or a month it works to your advantage. If your volumes are low on weekends, you have 4.5M for almost 22 days in a month so almost 200k/day. Another perk is the limit is now a soft limit, so if you go over in a month you will get a call from the AE to discuss your needs vs stopping your event deliveries. This license also allows you to set up CDC on as many objects as you want, removing the restriction of five objects.

The Add-on license is not very expensive and if your client is a large enterprise with a SELA with Salesforce it is usually fairly inexpensive.

4.2. Practical Tips and Considerations

1. When you need related record data, try to use a platform event instead of a CDC. That will save a call back from the target to get the additional information
2. Field attributes are validated, so, type, length, required etc is enforced when creating a platform event so make sure your logic accounts for it.
3. 99% of use cases are set up with "Publish After Commit". Be wary of trying to expedite your external processes with 'Publish Immediately". The immediate option is asynchronous and not bound to the transaction. If you have a call back from a subscriber to query the record that was created you may find the record does not exist because it was not committed yet.
4. After insert trigger is also not a safe place for "Publish Immediately", your subscriber may still receive it before record is committed in salesforce.
5. Change Data Capture is by default "Publish After Commit", even if you have multiple DML's to the same object record in a transaction there will be only one CDC event generated with the payload that is the committed state of the object at the end of the transaction.
6. Change Data Capture will only send the diff on fields with more than 1000 characters, so long texts etc will have to be reconstructed based on the diff and previous value.
7. If your target needs a payload in a specific format use a text area field in your platform event to build a JSON structure. (Either apex or Vlocity tools to convert to a JSON format)
8. By using only one client to subscribe to all events and using filters, your subscriptions are optimized. (a Hub to listen to events and distribute further in your enterprise)
9. Create a custom channel with filtered streaming if one of your subscribers needs to receive a subset of events. This way you don't have to create a separate event for multiple use cases where events are overlapping.

5. Summary

Both Platform Events and Change Data Capture have a place of their own. In large enterprises you will typically see that both have been used. A multitude of factors like use cases, licensing considerations, limits etc are considerations in determining the best fit for a given situation and use case.

Pushtopic events are still supported but there is no investment to improve them, so platform events should be prioritized over them.

Purely within the platform, apex triggers can be used to implement EDA. Aside from this Anypoint MQ and Mulesoft connectors can also support event driven interactions with external systems.

REFERENCES

[1] Standard platform event list https://developer.salesforce.com/docs/atlas.en-us.platform_events.meta/platform_events/platform_events_objects_list.htm.

[2] Including fields in the body, winter 20 fix https://help.salesforce.com/s/articleView?id=release-notes.rn_change_data_capture_enrichment.htm&release=222&type=5.

[3] Best practices on handling gap events https://developer.salesforce.com/docs/atlas.en-us.change_data_capture.meta/change_data_capture/cdc_replication_steps.htm#section_handle_gap_event.

[4] Best practices for handling overflow events https://developer.salesforce.com/docs/atlas.en-us.change_data_capture.meta/change_data_capture/cdc_replication_steps.htm#section_handle_overflow_event.