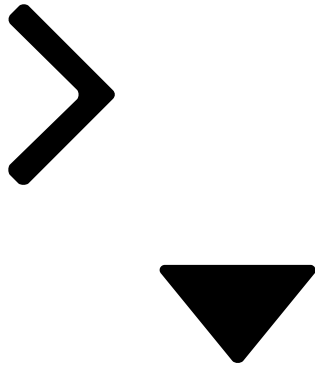


## 1. [Lightning Platform API Basics](#)



## 2. [Use Streaming API](#)

# Use Streaming API

## Learning Objectives

After completing this unit, you'll be able to:

- Describe the primary benefit that push technology offers over pull technology.
- Create a PushTopic and receive event notifications.
- Define a platform event and derive the subscription channel.
- Broadcast a message with generic streaming.
- Specify replay options for durable streaming.

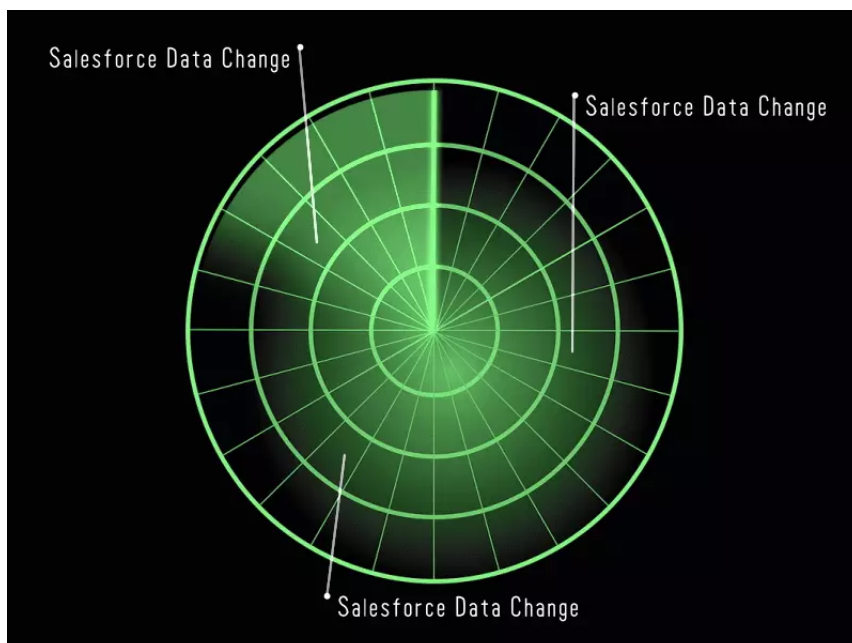
## Streaming Events

To conclude our survey of Salesforce's data APIs, let's look at an API that serves an entirely different use case. Streaming API lets you push a stream of notifications from Salesforce to client apps based on criteria that you define. How does pushing notifications differ from the pull paradigm that our other APIs use, in which the client app requests, or pulls, data from Salesforce? Let's examine the problem from a ship captain's point of view.

Imagine that you're sailing the high seas, and you want to keep an eye out for oncoming hazards, other ships, and islands rich with treasure. You put a sailor in the crow's nest to keep an active lookout. Now put on your developer hat again. Let's say you're writing an app using REST or SOAP API that periodically checks to see if any accounts have been updated. You can use a similar solution and keep an active lookout by constantly requesting account data and checking to see if it matches the old data.


Now imagine that you're on your ship again, but this time you have access to a shiny, new radar display. You don't need to keep a sailor in the crow's nest, because whenever an object of interest approaches, the display beeps.

Streaming API is your radar. It lets you define events and push notifications to your client app when the events occur. You don't have to keep an active lookout for data changes—you don't have to constantly poll Salesforce and make unnecessary API requests.



Tracking data changes in Salesforce is especially useful when you have business data stored in a system external to Salesforce. You can use Streaming API to keep your external source in sync with your Salesforce data with PushTopic events and Change Data Capture events. Also, Streaming API lets you

process business logic in an external system in response to data changes in Salesforce. For example, you can use Streaming API to notify a fulfillment center whenever an opportunity is updated.



**Note**

For more information about Change Data Capture, see the [Change Data Capture Basics](#) Trailhead module.

In addition to data changes, you can use Streaming API to broadcast custom notifications with platform events and generic streaming. For example, an app can generate platform event notifications for orders that an order fulfillment service processes. Or an app can listen to generic events and display a message whenever a system maintenance window is about to start or when a new offer is available to your users.

## PushTopics

A PushTopic is an sObject that contains the criteria of events you want to listen to, such as data changes for a particular object. You define the criteria as a SOQL query in the PushTopic and specify the record operations to notify on (create, update, delete, and undelete). In addition to event criteria, a PushTopic represents the channel that client apps subscribe to.

We’ll dive deeper when we create our own PushTopic.

### Supported Objects in PushTopic Queries

PushTopic queries support all custom objects and some of the popular standard objects, such as Account, Contact, and Opportunity. For a complete list of supported standard objects, see the *Streaming API Developer Guide* in the Resources section.

## PushTopics and Notifications

A PushTopic enables you to define the object, fields, and criteria you’re interested in receiving event notifications for. The following example shows a PushTopic defined and inserted in Apex. After this PushTopic is created, you can subscribe to this PushTopic channel to track changes on accounts whose billing city is San Francisco. This PushTopic specifies that the Id, Name, Phone fields are returned in each event notification. By default, notifications are sent for create, update, delete, and undelete operations that match the query’s criteria.

```
PushTopic pushTopic = new PushTopic();
pushTopic.Name = 'AccountUpdates';
pushTopic.Query = 'SELECT Id, Name, Phone FROM Account WHERE BillingCity=\'San Francisco\'';
pushTopic.ApiVersion = 37.0;
insert pushTopic;
```

Copy

At the minimum, define the PushTopic name, query, and API version. You can use default values for the remaining properties. By default, the fields in the `SELECT` statement field list and `WHERE` clause are the ones that trigger notifications. Notifications are sent only for the records that match the criteria in the `WHERE` clause. Notifications include the fields from the `SELECT` clause. To change which fields trigger notifications, set `pushTopic.NotifyForFields` to one of these values.

NotifyForFields Value	Description
All	Notifications are generated for all record field changes, provided the evaluated records match the criteria specified in the WHERE clause.
Referenced (default)	Changes to fields referenced in the SELECT and WHERE clauses are evaluated. Notifications are generated for the evaluated records only if they match the criteria specified in the WHERE clause.
Select	Changes to fields referenced in the SELECT clause are evaluated. Notifications are generated for the evaluated records only if they match the criteria specified in the WHERE clause.
Where	Changes to fields referenced in the WHERE clause are evaluated. Notifications are generated for the evaluated records only if they match the criteria specified in the WHERE clause.

To set notification preferences explicitly, set the following properties to either `true` or `false`. By default, all values are set to `true`.

```
pushTopic.NotifyForOperationCreate = true;
pushTopic.NotifyForOperationUpdate = true;
pushTopic.NotifyForOperationUndelete = true;
pushTopic.NotifyForOperationDelete = true;
```

Copy

If you create an account, an event notification is generated. The notification is in JSON and contains the fields that we specified in the PushTopic query: Id, Name, and Phone. The event notification looks similar to the following.

```
{
  "clientId": "1xd19o32njygi1gj47kgfaga4k",
  "data": {
    "event": {
      "createdDate": "2016-09-16T19:45:27.454Z",
      "replayId": 1,
      "type": "created"
    },
    "subject": {
      "Phone": "(415) 555-1212",
      "Id": "001D000000KneakIAB",
      "Name": "Blackbeard"
    }
  },
  "channel": "/topic/AccountUpdates"
}
```

Copy

The notification message includes the channel for the PushTopic, whose name format is `/topic/PushTopicName`. When you create a PushTopic, the channel is created automatically.

## PushTopic Queries

Let's take a moment to dive into the query we just defined for our PushTopic. PushTopic queries are regular SOQL queries, so if you're familiar with SOQL, you don't need to learn a new format. The format of the query is:

```
SELECT <comma-separated list of fields> FROM <Salesforce object> WHERE <filter criteria>
```

Copy

To ensure that notifications are sent in a timely manner, the following requirements apply to PushTopic queries.

- The `SELECT` statement's field list must include Id.
- Only one object per query is allowed.
- The object must be valid for the specified API version.

Certain queries aren't supported, such as aggregate queries or semi-joins.

## Custom Notifications with Platform Events

Use platform events to publish and subscribe to custom notifications with a predefined schema. Unlike PushTopic and Change Data Capture events, platform events aren't tied to Salesforce records and aren't autopublished by Salesforce. Instead, you define the schema of a platform event message by creating a platform event and adding fields. Also, clients publish platform events using declarative tools on the Lightning Platform, Apex, or APIs.

The versioned schema of a platform event enables subscribers to deterministically parse events. Each schema version corresponds to a unique schema ID, which is included in the event notification message.

### Defining a Platform Event

To define a platform event in the user interface, in Setup, enter `Platform Events` in the Quick Find box, then select **Platform Events**. Adding fields to a platform event is similar to how you add fields to a custom object. A subset of field types is supported.

The API name of a platform event contains the `_e` suffix. For example, if you create a platform event with the label `Order Event`, the API name is `Order_Event_e`.

Once you define a platform event, a channel name is automatically provided. The channel name is based on the event's API name and the format is `/event/Event_Name`. For example, `/event/Order_Event_e`.

### Publishing Platform Events

You can publish platform events using these declarative or programmatic tools on the Lightning Platform.

- Process Builder using the Create a Record action
- Flow using a Create Records element
- Apex `EventBus.publish()` method
- REST API `subjects` resource
- SOAP API `create()` call

For more details, refer to the Platform Events documentation in the Resources section.

## Subscribing to Platform Events

Streaming API provides the subscription mechanism for multiple types of events, including platform events. In addition to Streaming API, you can subscribe to platform events using the Lightning Platform.

- Process Builder using a process that starts when a platform event occurs
- Flow that waits for a platform event to occur
- Apex trigger
- Streaming API using the CometD messaging library

This example is a platform event message for the order event.

```
{
  "data": {
    "schema": "dffQ2QLzDNHqwB8_sHMxdA",
    "payload": {
      "CreatedDate": "2018-08-22T12:11:40.517Z",
      "CreatedById": "005D0000001cSZs",
      "Order Number c": "12345",
      "Has Shipped _c": true
    },
    "event": {
      "replayId": 1
    }
  },
  "channel": "/event/Order_Event__e"
}
```

Copy

For more details, refer to the Platform Events documentation in the Resources section.

## Custom Notifications with Generic Streaming

Before you set sail on your own, let's spend a few minutes reviewing generic streaming. Streaming API supports sending notifications with a generic payload that aren't tied to Salesforce data changes.

Use generic streaming to send and receive custom notifications with arbitrary payloads and no predefined schema. Generic streaming lets you publish notifications to a targeted set of users. To use generic streaming, you need:

- A streaming channel that defines the channel
- One or more clients subscribed to the channel
- The Streaming Channel Push resource to monitor and invoke events on the channel

You can create a streaming channel for generic streaming either through the Streaming Channels app in the user interface, or through the API. A streaming channel is represented by the StreamingChannel sObject, so you can create it through Apex, REST API, or SOAP API. The format of the channel name for generic streaming is `/u/ChannelName`. For example, this Apex snippet creates a channel named `Broadcast`.

```
StreamingChannel ch = new StreamingChannel();
ch.Name = '/u/Broadcast';
insert ch;
```

Copy

Alternatively, you can opt to have Salesforce create the streaming channel dynamically for you if it doesn't exist. To enable dynamic streaming channels in your org, from Setup, enter `User Interface` in the Quick Find box, then select **User Interface**. On the User Interface page, select the **Enable Dynamic Streaming Channel Creation** option.

You can subscribe to the channel by using a CometD client. (The Resources section links to a sample walkthrough in the Streaming API Developer Guide.)

To generate events, make a POST request to the following REST resource. Replace `XX.0` with the API version and *Streaming Channel ID* with the ID of your channel.

```
/services/data/vXX.0/subjects/StreamingChannel/Streaming Channel ID/push
```

Copy



### Note


To obtain your channel ID, run a SOQL query on StreamingChannel, such as: `SELECT Id, Name FROM StreamingChannel`

Example REST request body.

```
{
  "pushEvents": [
    {
      "payload": "Broadcast message to all subscribers",
      "userIds": []
    }
  ]
}
```

```
}  
  ]  
}
```

Copy



**Note**

Instead of broadcasting to all subscribers, specify a list of subscribed users to send notifications to by using the optional `userIds` field. Also, you can use the GET method of the Streaming Channel Push REST API resource to get a list of active subscribers to the channel.

The event notification that the subscribed client receives looks similar to the following.

```
{  
  "clientId": "1p145y6g3x3nmnlodd7v9nhi4k",  
  "data": {  
    "payload": "Broadcast message to all subscribers",  
    "event": {  
      "createdDate": "2016-09-16T20:43:39.392Z",  
      "replayId": 1  
    }  
  },  
  "channel": "/u/Broadcast"  
}
```

Copy

Notice that this event notification contains the `replayId` field. Generic event notifications are also stored for 24 hours and can be retrieved using the `replayId` value starting in API version 37.0.

Thanks to event notifications, you can set sail in the high seas with confidence and head to the island rich with treasure!

## Retrieve Past Notifications Using Durable Streaming

So far, you’ve learned about the various types of events. What happens if a Salesforce record is created or a custom notification is generated before a client subscribes to an event channel or while a client is disconnected? Before API version 37.0, the client misses the corresponding notification. As of API version 37.0, Salesforce stores event messages. The events are stored for 24 hours, and you can retrieve them at any time during that window. Yay!

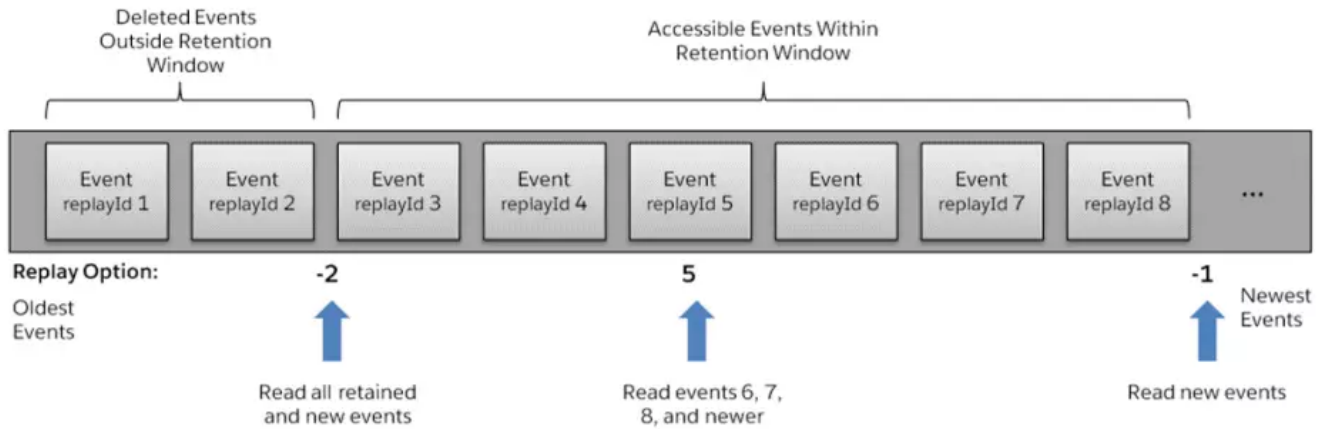
Starting with API version 37.0, each event notification message contains a field called `replayId`. Similar to replaying a video, Streaming API replays the event notifications that were sent by using the `replayId` field.

Each 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. Replay ID values are not guaranteed to be contiguous for consecutive events. For example, the event following the event with ID 999 can have an ID of 1,025. A subscriber can store a replay ID value and use it on resubscription to retrieve events that are within the retention window. For example, a subscriber can retrieve missed events after a connection failure. Subscribers must not compute new replay IDs based on a stored replay ID to refer to other events in the stream.

Table 1. Replay Options

Replay Option	Description	Usage
Replay ID	Subscriber receives all stored events after the event specified by its <code>replayId</code> 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	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.

This diagram shows how event consumers can read a stream of events by using various replay options.



## Resources

### [Streaming API Developer Guide](#)

- [PushTopic Object Reference](#)
- [Streaming Channel Push REST Resource](#)
- [Bayeux Protocol, CometD, and Long Polling](#)

### Trailhead

- [Platform Events Basics](#)
- [Change Data Capture Basics](#)

### [Platform Events Developer Guide](#)

- [Defining Platform Events](#)
- [Publishing Platform Events](#)
- [Subscribing to Platform Events](#)

### [Streaming API Developer Guide](#)

- [Example: Subscribe to and Replay Events Using a Visualforce Page](#)
- [Example: Subscribe to and Replay Events Using a Java Client \(EMP Connector\)](#)

### GitHub

- [Salesforce Durable Streaming Demo in GitHub](#)
- [Enterprise Messaging Platform Connector \(CometD Java Client\)](#)
- [Java and JavaScript CometD Extensions](#)

### [SOQL and SOSL Reference](#)

## Quiz Complete!

+25 points



Lightning Platform API Basics

100%

Progress: 100%

[View more modules](#)