Available online www.jsaer.com

***Journal of Scientific and Engineering Research*, 2020, 7(3):318-323**

Research Article

ISSN: 2394-2630
CODEN(USA): JSERBR

How to Subscribe to Google Pub/Sub Topic from Salesforce using Pull Mechanism with Apex

Chirag Amrutlal Pethad

PetSmart.com, LLC, Stores and Services, Phoenix, Arizona, USA,
ChiragPethad@live.com,
ChiragPethad@gmail.com,
Cpethad@petsmart.com

Abstract: The document outlines the integration of Google Cloud Pub/Sub with Salesforce using a push mechanism. Key steps include setting up a Pub-Sub topic and subscription, creating an Apex REST service in Salesforce to handle incoming messages, and implementing OAuth 2.0 for secure authentication. It emphasizes security considerations, testing, and best practices for error handling and scalability, ultimately enhancing Salesforce applications' responsiveness and reliability through real-time messaging.

Keywords: Event Bus, Event Driven Architecture, Google PUB-SUB, Integration, Push vs Pull, REST API, Limits, Scalability, Event Replay

Introduction

Salesforce Event Bus, also known as the Platform Events framework, is a powerful tool for enabling event-driven architectures within Salesforce. However, there are several limitations to consider when using Salesforce Event Bus. The integration of Google Cloud Pub/Sub with Salesforce enables businesses to harness the power of real-time data processing and avoid the limitations associated with Salesforce Event Bus. This white paper provides a step-by-step guide to setting up and subscribing to Google Pub/Sub in Salesforce using Pull mechanism leveraging the capabilities of both platforms for improved operational efficiency, seamless and efficient flow of information.

Limitations Of Salesforce Event Bus

Salesforce Event Bus, also known as the Platform Events framework, is a powerful tool for enabling event-driven architectures within Salesforce and integrating with external systems. However, there are several limitations to consider when using Salesforce Event Bus:

A. Event Delivery

• No Guaranteed Order: While Salesforce attempts to deliver events in order, it does not guarantee the order of event delivery.

• At-Least-Once Delivery: Events may be delivered more than once. Consumers must handle potential duplicate events.

B. Event Retention and Replay

• Retention Period: Platform events are retained for 72 hours. If consumers are offline for longer than this period, they may miss events.

• Limited Replay Options: Replay of events is limited to the last 24 hours. For events older than 24 hours but within the 72-hour retention period, consumers must handle gaps manually.

C. Event Size and Volume

• Payload Size: The maximum size of a platform event message is 1 MB. This includes the payload and metadata.

• Volume Limits: There are limits on the number of events that can be published and delivered within a 24-hour period, depending on the Salesforce edition and licensing:

O There are limits on the number of events that can be published and delivered within a 24-hour period, depending on the Salesforce edition and licensing.

O Standard Volume Platform Events: 50,000 events per 24-hour period.

D. Event Publishing Limits

There are limits on the number of events that can be published per transaction and per hour. Exceeding these limits will result in errors:

• Per Transaction: 1,000 events

• Per Hour: Limits vary by Salesforce edition and license count.

Identify applicable funding agency here. If none, delete this text box.

E. Event Processing Limits

• Subscriber Limits: Each event can have a maximum of 50 subscribers (including Apex triggers, flows, and external systems).

• Concurrency Limits: Salesforce imposes limits on the number of concurrent long-running Apex transactions, which can impact event processing performance.

F. Platform Events and Triggers

• Governor Limits: Apex triggers on platform events are subject to Salesforce governor limits, such as CPU time, heap size, and SOQL/DML limits.

• Error Handling: Errors in triggers can cause event processing failures. Proper error handling and retry mechanisms must be implemented.

G. Integration and External Systems

• External System Dependencies: Integrating with external systems can introduce latency and reliability issues. Ensure that external systems can handle the volume and frequency of events.

• API Limits: Calling external APIs from Salesforce is subject to API call limits and rate limits imposed by the external system.

H. Maintenance and Upgrades

• API Versioning: Changes to Salesforce API versions can impact event processing. Ensure compatibility with the latt API versions.

• Platform Upgrades: Salesforce platform upgrades may introduce changes that impact event bus functionality. Monitor release notes and perform testing during upgrades.

I. Monitoring and Debugging

• Limited Monitoring Tools: Salesforce provides limited built-in tools for monitoring platform events. Additional third-party tools or custom monitoring solutions may be needed for comprehensive monitoring and alerting.

• Debugging Challenges: Debugging issues with event processing can be challenging due to asynchronous nature and potential delays in event delivery.

Understanding Google Cloud Pub Sub

Google Cloud Pub/Sub is a fully-managed real-time messaging service that allows you to send and receive messages between independent applications. It decouples services that produce events from services that process events, enhancing scalability and reliability.

A. Key Features

• Scalability: Handle high throughput and low-latency messages.

• Reliability: Ensure message delivery with at-least-once delivery.

• Flexibility: Integrate with various GCP services and external systems.

Overview Of Salesforce Apex

Apex is a strongly-typed, object-oriented programming language used by developers to execute flow and transaction control statements on the Salesforce platform. It enables the creation of web services, email services, and complex business processes.

A. Key Features

• Scalability: Handle large volumes of data and transactions.
• Robustness: Build complex logic and automation workflows.
• Integration Capabilities: Interact with external systems via REST and SOAP APIs.

Implementation Plan

The implementation plan involves setting up a Google Cloud Pub/Sub topic, configuring service accounts and permissions, and implementing Apex code in Salesforce to subscribe to the Pub/Sub topic. The process includes:

A. Setting up Google Cloud Pub/Sub

Create a Pub-Sub topic and configure necessary IAM roles.

B. Salesforce Setup

Configure named credentials and remote site settings.

C. Apex Implementation

Develop Apex classes to handle authentication, subscription, and message processing.

Step By Step Implementation

A. Create a Pub/Sub Topic

• Go to Google Cloud Console
• Navigate to Pub/Sub section
• Create a New Topic

B. Configure IAM permissions

Step 1: Create a Service Account

• Navigate to IAM & Admin > Service Accounts.
• Click + CREATE SERVICE ACCOUNT.
• Enter a name and description for the service account, then click CREATE.
• Assign the required roles to the service account, such as Pub/Sub Subscriber.

Step 2: Create a Key for the Service Account

• In the Service Accounts page, find your new service account.
• Click the Actions column (three dots) for your service account and select Manage keys.
• Click ADD KEY > Create new key.
• Select JSON and click Create to download the JSON key file.

C. Configure Named Credentials in Salesforce

Step 1: Upload the JSON Key file

• Go to Setup in Salesforce
• Navigate to Files and upload the JSON key file. Make note of the document ID.

Step 2: Configure Named Credential

• In Setup, navigate to Security > Named Credentials.
• Click New Named Credential.
• Fill out the form as follows:

1. Label: GoogleServiceAccount
2. Name: GoogleServiceAccount
3. URL: https://oauth2.googleapis.com
4. Identity Type: Named Principal
5. Authentication Protocol: JWT
6. JWT Signing Algorithm: RS256
7. Issuer: service-account-email (from JSON key file)

8. Subject: service-account-email (from JSON key file)
9. Audience: https://oauth2.googleapis.com/token
10. Token endpoint: /token
11. Client Credentials Source: Paste the private key from the JSON key file.

D. Configure Remote Site setting in Salesforce

• In Setup, navigate to Security Controls -> Remote Site Settings.
• Add a new remote site with the Pub/Sub API URL.

E. Implementing OAuth2.0 Authentication in Salesforce

We start with an Apex class GoogleAuthenticator.

```
public with sharing class GoogleAuthenticator {
    private static final String TOKEN_URL = '<https://oauth2.googleapis.com/token>';
    private static final String SCOPE = '<https://www.googleapis.com/auth/pubsub>';
}
```

Next, we implement logic / methods to generate JWT Signature and Assertion in the GoogleAuthenticator apex class.

```
    private String generateJwtAssertion() {
        // Implement JWT generation using the service account JSON key file
        // This can be done using libraries or manually constructing the JWT token
        // For simplicity, pseudocode is provided below

        // Load the private key from the Named Credential or JSON file
        String privateKey = '...'; // replace with actual key

        // Create JWT header and payload
        String header = '{"alg":"RS256","typ":"JWT"}';
        String payload = '{"iss":"service-account-email","scope":"'
            + SCOPE + '","aud":"'
            + TOKEN_URL + '","exp":expiration-time,"iat":issue-time}';

        // Encode header and payload using Base64
        String encodedHeader = EncodingUtil.base64Encode(Blob.valueOf(header));
        String encodedPayload = EncodingUtil.base64Encode(Blob.valueOf(payload));

        // Create the JWT signature using the private key
        String signature = createJwtSignature(encodedHeader + '.'
          + encodedPayload, privateKey);

        // Combine header, payload, and signature to form the JWT token
        return encodedHeader + '.' + encodedPayload + '.' + signature;
    }

    private String createJwtSignature(String data, String privateKey) {
        // Implement the RSA256 signature creation
        // For simplicity, pseudocode is provided
        return 'signature'; // replace with actual signature generation code
    }
```

Finally, we implement the logic / method to make the callout to retrieve the Access Token.

```
public String getAccessToken() {
    HttpRequest req = new HttpRequest();
    req.setEndpoint(TOKEN_URL);
    req.setMethod('POST');
    req.setHeader('Content-Type', 'application/x-www-form-urlencoded');

    String body = 'grant_type=urn:ietf:params:oauth:grant-type:jwt-bearer' +
                  '&assertion=' + generateJwtAssertion();
    req.setBody(body);

    Http http = new Http();
    HttpResponse res = http.send(req);

    if (res.getStatusCode() == 200) {
        Map<String, Object> jsonResponse =
          (Map<String, Object>) JSON.deserializeUntyped(res.getBody());
        return (String) jsonResponse.get('access_token');
    } else {
        throw new AuthenticationException('Failed to get access token: ' + res.getBody());
    }
}
```

F. Access Google Pub/Sub with the Access Token

```
public with sharing class PubSubService {
    private GoogleAuth auth = new GoogleAuth();

    public void pullMessages() {
        String accessToken = auth.getAccessToken();

        HttpRequest req = new HttpRequest();
        req.setEndpoint(
          '<https://pubsub.googleapis.com/v1/projects/your-project-id'
          + '/subscriptions/your-subscription-id:pull>');
        req.setMethod('POST');
        req.setHeader('Authorization', 'Bearer ' + accessToken);
        req.setHeader('Content-Type', 'application/json');
        req.setBody('{"maxMessages": 10}');

        Http http = new Http();
        HttpResponse res = http.send(req);

        if (res.getStatusCode() == 200) {
            // Process the response and messages
            Map<String, Object> jsonResponse =
              (Map<String, Object>) JSON.deserializeUntyped(res.getBody());
            List<Object> receivedMessages =
              (List<Object>) jsonResponse.get('receivedMessages');
            for (Object messageObj : receivedMessages) {
                Map<String, Object> message = (Map<String, Object>) messageObj;
                String data = (String) message.get('message').get('data');
                // Decode base64 encoded message
                Blob decodedBlob = EncodingUtil.base64Decode(data);
                String decodedMessage = decodedBlob.toString();
                // Handle the message data
                processMessage(decodedMessage);
            }
        } else {
            throw new ServiceException('Failed to pull messages: ' + res.getBody());
        }
    }

    private void processMessage(String message) {
        // Implement message processing logic
    }
}
```

Security Considerations

• Authentication: Use OAuth 2.0 for secure authentication.
• Data Encryption: Ensure data encryption in transit and at rest.
• Secure Storage: Ensure the JSON key file is securely store and access is limited.
• Access Control: Implement proper IAM roles and permissions for the service account.
• Data Encryption: Ensure data encryption in transit and at rest.
• Validation: Validate incoming requests to ensure they are from trusted sources.

Testing And Validation

• Unit Testing: Write unit tests for Apex classes to ensure functionality.
• Integration Testing: Validate end-to-end integration between Salesforce and GCP Pub/Sub.
• Performance Testing: Ensure the system can handle the expected message load.

Best Practices

• Error Handling: Implement robust error handling and retry mechanisms.
• Logging: Use Salesforce logging to monitor and troubleshoot issues.
• Scalability: Design the system to handle growth in message volume.
• Batch: Implement batching to pull messages in bulk.

Conclusion

Integrating Google Cloud Pub/Sub with Salesforce using Apex provides a powerful solution for real-time messaging and event-driven architecture. By following the steps outlined in this white paper, organizations can enhance their Salesforce applications' responsiveness, scalability, and reliability. This white paper serves as a

guide for developers and architects looking to leverage the combined capabilities of Google Cloud Pub/Sub and Salesforce applications.

References

[1]. Apex Developer Guide - https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_dev_guide.htm
[2]. Google Pub/Sub - https://cloud.google.com/pubsub/docs
[3]. Google Pub/Sub Architecture - https://cloud.google.com/pubsub/architecture
[4]. Google Pub/Sub basics - https://cloud.google.com/pubsub/docs/pubsub-basics
[5]. Apex Integration - https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_integration_intro.htm
[6]. Named Credentials - https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_callouts_named_credentials.htm
[7]. Google Pub/Sub Pull Subscription - https://cloud.google.com/pubsub/docs/create-subscription