ISSN: 2634 - 8853

Journal of Engineering and Applied Sciences Technology

SCIENTIFIC Research and Community

Review Article

Open Access

Subscribe to Google Pub/Sub Topic from Salesforce using Push Method with Apigee Proxy

Chirag Amrutlal Pethad

PetSmart.com, LLC, Stores and Services Phoenix, Arizona, USA

ABSTRACT

The document outlines the integration of Google Cloud Pub/Sub with Salesforce using a push mechanism by creating an Apigee Proxy API that subscribes to the Pub/Sub and then publishes the messages to a Salesforce REST Api endpoint. Key steps include setting up a Pub-Sub topic, setting up an Apigee Proxy API, 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.

***Corresponding author**

Chirag Amrutlal Pethad, PetSmart.com, LLC, Stores and Services Phoenix, Arizona, USA.

Received: October 03, 2022; **Accepted:** October 10, 2022; **Published:** October 17, 2022

Keywords: Event Bus, Event Driven Architecture, Google PUB-SUB, Integration, Push vs Pull, REST API, Limits, Scalability, Event Replay, Reverse Proxy, Apigee Proxy, Audience, JWT, Token, Authentication, Authorization

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 Push method by configuring Apigee Proxy and leveraging the capabilities of both platforms for improved operational efficiency, seamless and efficient flow of information. The objective is to enable real-time messaging and event-driven architecture in Salesforce by leveraging GCP Pub/Sub for asynchronous communication. This integration allows Salesforce to automatically receive messages pushed from Pub/Sub topics, ensuring efficient and scalable processing of events.

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:

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.

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.

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.

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.

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.

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.

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

Maintenance and Upgrades
- **API Versioning:** Changes to Salesforce API versions can impact event processing. Ensure compatibility with the latest API versions.
- **Platform Upgrades:** Salesforce platform upgrades may introduce changes that impact event bus functionality. Monitor release notes and perform testing during upgrades.

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.

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 Apigee

Apigee, a product of Google Cloud, is a full lifecycle API management platform that allows organizations to design, secure, deploy, monitor, and scale APIs. It serves as a powerful tool for enabling businesses to expose backend services and data to various consumers, whether internal, partner, or external developers, in a secure and scalable manner.

Key Features
- **API Design and Development**
  - Apigee allows you to create API proxies, which are abstractions of your backend services. These proxies provide a controlled interface for consumers while abstracting the underlying implementation.
  - You can apply various policies to APIs, such as security (OAuth, JWT), traffic management (rate limiting, quotas), transformation (XML/JSON conversion), and mediation (routing, orchestration).
- **Security and Governance**
  - Apigee supports OAuth2.0 and OpenID Connect for securing APIs, enabling fine-grained access control.
  - Implement rate limiting, quotas, and spike arrest to protect APIs from abuse and ensure fair usage.
  - Apigee provides features to guard against common security threats like SQL injection, cross-site scripting, and content attacks.
- **Analytics and Monitoring**
  - Apigee offers detailed analytics on API usage, latency, response times, error rates, and more, helping you understand how your APIs are being used and identify performance bottlenecks.
  - You can create custom dashboards to monitor specific metrics relevant to your business needs.
- **Developer Portal**
  - Apigee provides a customizable developer portal where API consumers can discover, explore, and sign up for APIs.
  - Automatically generated or custom API documentation helps developers understand how to use your APIs effectively.
  - Developers can obtain API keys, manage their credentials, and access API metrics through the portal.
- **Lifecycle Management**
  - Manage different versions of your APIs to ensure backward compatibility and smooth transitions.
  - Deploy APIs across different environments (development, test, production) with controls to manage API lifecycle stages.
- **API Monetization**
  - Apigee provides tools to create API products, define rate plans, and monetize your APIs, allowing you to generate revenue directly from API usage.
- **Hybrid and Multi-Cloud Support**
  - Apigee can be deployed on-premises, in a private cloud, or in a hybrid model, providing flexibility in how and where you manage your APIs.
  - Support for multi-cloud deployments ensures that your APIs can be managed consistently across different cloud environments.
- **Extensibility**
  - Use JavaScript, Python, or Node.js scripts within API proxies to extend functionality, or connect with other services like Google Cloud functions or external microservices.
  - Apigee integrates with a wide range of third-party tools and services, including CI/CD pipelines, identity providers, and monitoring tools.

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.

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 Java application to subscribe to the Pub/Sub topic and then publish / forward those messages to Salesforce endpoint. The process includes:

Salesforce Setup

Setup the Connected App, Create a web service to receive messages.

Setting up Apigee Proxy

Create the API proxy in Apigee to connect to the Salesforce API

Setting up Google Cloud Pub/Sub

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

Step by Step Implementation

Create a Service Account User with required permissions in Salesforce

Step 1: Create New User

- Log in to your Salesforce instance with an account that has administrative privileges.
- Click on the gear icon in the top right corner to open the Setup menu.
- In the Quick Find box on the left, type Users and select Users under the Users section.
- Click on the New User button.
  - **First Name:** Service (or a name indicating the account's purpose)
  - **Last Name:** Account
  - **Email:** Provide a valid email address (for notifications and password reset)
  - **Username:** Must be in the format of an email address (unique across all Salesforce instances)
  - **User License:** Select the appropriate license type, typically Salesforce or API Only.
  - **Profile:** Assign a profile that provides the necessary permissions (e.g., System Administrator or a custom profile with specific API permissions).
- Ensure the Generate new password and notify user immediately option is checked if you want the login credentials to be sent to the email address specified. Click Save.

Step 2: Assign Permissions

- After creating the user, you may need to assign additional permissions via Permission Sets to grant specific access required for the service account.
- In the Setup menu, type Permission Sets in the Quick Find box and select Permission Sets.
- Create a new Permission Set or use an existing one.
- Assign the necessary permissions (e.g., API access, specific object permissions).
- Go to the Users section of the Permission Set and assign the newly created service account user to this Permission Set.
- If the service account will be used for API access, ensure the profile or permission set assigned to the user includes API Enabled permissions.
- Navigate to the System Permissions section within the Profile or Permission Set and ensure API Enabled is checked.

Step 3: Set-Up Connected App

- If the service account will be used with OAuth for authentication, you need to create a Connected App.
- In the Setup menu, type App Manager in the Quick Find box and select App Manager.
- Click New Connected App.
- Fill in the required fields, such as Connected App Name, API Name, Contact Email.
- Under API (Enable OAuth Settings), check the Enable OAuth Settings box.
- Provide the Callback URL and select the necessary OAuth scopes (e.g., Full Access, Perform requests on your behalf at any time). Set a Callback URL (you can use https://login.salesforce.com/services/oauth2/callback or your Apigee proxy URL).
- Add required OAuth scopes, like full, api, or refresh_token.
- Save the Connected App and note the Consumer Key and Consumer Secret for API integration.

Create and Configure Salesforce Service to Receive Pub/Sub Messages

Step 1: Create a Salesforce APEX REST Service

- In Salesforce, navigate to Setup.
- Go to Apex Classes and click "New" to create a new Apex class that will handle incoming Pub/Sub messages.

```
//@RestResource(urlMapping='/PubSubHandler/*')
global with sharing class PubSubHandler {
    @HttpPost
    global static void doPost() {
        RestRequest req = RestContext.request;
        RestResponse res = RestContext.response;

        String requestBody = req.requestBody.toString();
        Map<String, Object> message = (Map<String, Object>)
              fiJSON.deserializeUntyped(requestBody);
        String messageData = (String) message.get('message').get('data');

        Blob decodedBlob = EncodingUtil.base64Decode(messageData);
        String decodedMessage = decodedBlob.toString();

        processMessage(decodedMessage);

        res.responseBody = Blob.valueOf('Message processed successfully');
        res.statusCode = 200;
    }

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

- Grant access to PubSubHandler for the Service Account

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.

Setup Api Proxy in Apigee

Step 1: Create a New API Proxy:

- Log in to your Apigee account.
- Go to the "API Proxies" section in the Apigee Edge management console.

- Click "Create Proxy."
- Choose "Reverse proxy" as the proxy type.
- Proxy Name: Give your proxy a name (e.g., SalesforceProxy).
- Base Path: Set the base path (e.g., /salesforce).
- Target URL: Enter the Salesforce API endpoint (e.g., https://yourInstance.salesforce.com).
- Virtual Hosts: Select the virtual hosts (e.g., default).

Step 2: Verify / Validate JWT Token

Use Apigee's VerifyJWT policy to validate the JWT token that comes in the Authorization Header of the Pub/Sub push request. This involves checking the signature, token's claims (like the issuer, audience, and subject), and ensuring the token is not expired.

```
<VerifyJWT name="Verify-JWT-Token">
    <Algorithm>RS256</Algorithm>
    <Source>request.header.Authorization</Source>
    <PublicKey>
        <JWKS uri="<https://www.googleapis.com/oauth2/v3/certs>"/>
    </PublicKey>
    <Audience><https://api.example.com/pubsub></Audience>
    <Issuer><https://accounts.google.com></Issuer>
    <SubjectMatch>
        <Pattern>service-account-email@project-id.iam.gserviceaccount.com</Pattern>
    </SubjectMatch>
</VerifyJWT>
```

- **Source:** The JWT is expected in the Authorization header as Bearer Token.
- **PublicKey:** This points to the Google public keys used to verify the JWT signature.
- **Audience:** Should match the audience set in your Pub/Sub subscription.
- **Issue:** Typically https://accounts.google.com for Google signed tokens.
- **SubjectMatch:** This ensure the token was issued for the correct service account.

Step 3: Extract and Validate the Service Account Email

After verifying the JWT, you can extract the service account's email from the sub (subject) claim of the JWT. This email identifies the service account used by Pub/Sub to push messages to your Apigee proxy. This policy stores the service account email in the serviceAccountEmail variable.

```
<ExtractVariables name="Extract-Service-Account-Email">
    <Source>jwt.Claims.sub</Source>
    <Variable name="serviceAccountEmail"/>
</ExtractVariables>
```

Step 4: Verify Service Account Permissions

In Apigee, you don't have built-in support to check specific IAM permissions of a service account directly. So, we can create a ServiceCallout policy to call the Google IAM API to check if the service account has the required permissions.

```
<ServiceCallout name="Check-Permissions">
    <Request>
        <Set>
            <Verb>POST</Verb>
            <Headers>
                <Header name="Authorization">Bearer {access_token}</Header>
                <Header name="Content-Type">application/json</Header>
            </Headers>
            <Payload contentType="application/json">
                {
                    "permissions": [
                        "pubsub.subscriptions.consume"
                    ],
                    "resource": "projects/project-id/subscriptions/subscription-id"
                }
            </Payload>
        </Set>
    </Request>
    <Response>permissionsResponse</Response>
    <HTTPTargetConnection>
        <URL><https://cloudresourcemanager.googleapis.com/v1/projects/project-id:testIamPermissions></URL>
    </HTTPTargetConnection>
</ServiceCallout>
```

- **Access_token:** This should be a valid OAuth token with IAM scope, which you can generate using a service account.
- **Permissions:** The specific IAM permissions you want to check.
- **Resource:** The resource (e.g., Pub/Sub subscription) for which you're checking permissions.
- Evaluate the Response to check if the service account has required permissions.
- If the JWT is invalid or the service account lacks the required permissions, you should handle this appropriately, such as by returning an HTTP 403 Forbidden status or logging the event for security audits.

Step 5: Handle the Pub/Sub Message

After the JWT is validated, you can extract the Pub/Sub message payload and process it as needed. Typically, the message payload will be in the request body, which you can extract and handle using an ExtractVariables or AssignMessage policy.

```
<ExtractVariables name="Extract-PubSub-Message">
    <Source>request</Source>
    <JSONPayload>
        <Variable name="pubsubMessageData" type="string">
            <JSONPath>$.message.data</JSONPath>
        </Variable>
        <Variable name="pubsubMessageAttributes" type="string">
            <JSONPath>$.message.attributes</JSONPath>
        </Variable>
    </JSONPayload>
    <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
</ExtractVariables>
```

- The $.message.data field will contain the base64-encoded message data.
- You might need to decode this data depending on your processing needs.

Step 6: Configure the Proxy to Call Salesforce for OAuth Access Token

In the Apigee proxy, add an ServiceCallout policy under the "PreFlow" to handle token requests to Salesforce. Use the Salesforce Consumer Key, Consumer Secret, and Token Endpoint (typically https://login.salesforce.com/services/oauth2/token).

```
<ServiceCallout name="Check-Permissions">
    <Request>
        <Set>
            <Verb>POST</Verb>
            <Headers>
                <Header name="Authorization">Bearer {access_token}</Header>
                <Header name="Content-Type">application/json</Header>
            </Headers>
            <Payload contentType="application/json">
                {
                    "permissions": [
                        "pubsub.subscriptions.consume"
                    ],
                    "resource": "projects/project-id/subscriptions/subscription-id"
                }
            </Payload>
        </Set>
    </Request>
    <Response>permissionsResponse</Response>
    <HTTPTargetConnection>
        <URL><https://cloudresourcemanager.googleapis.com/v1/projects/project-id:testIamPermissions></URL>
    </HTTPTargetConnection>
</ServiceCallout>
```

- This policy authenticates with Salesforce and retrieves the access token which will be used to call the Salesforce Target Endpoint.
- The response gets stored in the response variable which can then be used to add the Authorization in the request header.

Step 7: Configure the Proxy to Set the Request Headers and Payload.

In the Apigee proxy, add an Assign Message policy under the "PreFlow" for Target Endpoint to set the Authorization header

and Payload to the Salesforce request.

```
<AssignMessage continueOnError="false" enabled="true" name="AM-RequestHeader">
    <DisplayName>AM-RequestHeader</DisplayName>
    <Set>
        <Headers>
            <Header name="Authorization">Bearer {accessToken}</Header>
            <Header name="Content-Type">application/json</Header>
        </Headers>
        <Payload contentType="application/json">{pubsubMessageData}</Payload>
    </Set>
    <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
    <AssignTo type="request"/>
</AssignMessage>
```

Step 8: Set the Salesforce API Endpoint as the target URL in the Target Endpoint Configuration.

```
<TargetEndpoint name="default">
    <Description/>
    <FaultRules/>
    <PreFlow name="PreFlow">
        <Request>
            <Step>
                <Name>AM-RequestHeader</Name>
            </Step>
        </Request>
        <Response/>
    </PreFlow>
    <PostFlow name="PostFlow">
        <Request/>
        <Response/>
    </PostFlow>
    <Flows/>
    <HTTPTargetConnection>
        <Properties/>
        <URL>https://<<Your_Salesforce_Custom_Domain_URl>>/services/apexrest</URL>
    </HTTPTargetConnection>
</TargetEndpoint>
```

Create a Pub/Sub Topic

- Go to Google Cloud Console
- Navigate to Pub/Sub section
- Create a New Topic and provide a name for your topic (e.g., my-topic).

Create a Subscription with Push Delivery

- In the Google Cloud Console, navigate to the Pub-Sub topic you created.
- Choose "Create Subscription".
- **Name:** Give your subscription a name (e.g., apigee-subscription).
- **Topic:** Select the topic you created earlier.
- **Delivery Type:** Choose "Push" for the delivery type.
- **Endpoint URL:** Enter the URL of your Apigee proxy (e.g., https://your-apigee-environment.com/ salesforce).
- **Authentication:** Optionally, configure authentication. You can use a Google service account JWT or an OIDC token to secure the endpoint.
- **Acknowledge Deadlines:** Set the acknowledge deadline. This determines how long Pub/Sub waits before retrying the delivery of a message.
- **Retry Policy:** Configure the retry policy if needed, including the maximum number of retries and minimum backoff time.

Authentication Setup (Optional)

- **Service Account Authentication:** If you use service account tokens for authentication, ensure that the Apigee proxy validates the JWT or token. This can be done by verifying the token's signature and claims (like aud or iss).
- **OIDC Token:** Alternatively, you can configure Pub/Sub to send an OIDC token in the request. Apigee can verify this token against an identity provider.
- Provide the API Proxy URL as the Audience and ensure Apigee proxy validates the Audience information.

Deploy and Test the API Proxy

- Deploy the Apigee proxy to desired environment.
- Test it by publishing a message to the Pub-Sub topic associated with your push subscription.
- Check if Apigee correctly validates the JWT and processes the Pub/Sub message.
- Use Apigee's logging and debugging tools to monitor incoming requests, verify JWT validation, and troubleshoot any issues.
- Check if the Salesforce Endpoint was called and the message was successfully processed by the Salesforce REST API.

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.
- **Send Test Messages:** Use the Google Cloud Console to publish test messages to the Pub/Sub topic.
- **Monitor Salesforce Logs:** Check Salesforce debug logs to verify that the messages are received and processed correctly.
- **Validate Data:** Ensure the processed data is correctly updated or created in Salesforce as per your business logic.
- **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 Processing:** Implement batching to process messages in bulk in Apex.

Conclusion

Integrating Google Cloud Pub/Sub with Salesforce 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 using Apigee Proxy [1-7].

References

1. (2024) Apex Developer Guide - https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_dev_guide.htm.
2. (2024) Google Pub/Sub - https://cloud.google.com/pubsub/docs.
3. (2024) Google Pub/Sub Architecture - https://cloud.google.com/pubsub/architecture.
4. (2024) Google Pub/Sub basics - https://cloud.google.com/pubsub/docs/pubsub-basics.
5. (2024) Apex Integration - https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_integration_intro.htm.
6. (2024) Apigee Reference Documentation - https://docs.apigee.com/api-platform/reference/apigee-reference.
7. (2024) Google Pub/Sub Push Subscription - https://cloud.google.com/pubsub/docs/create-subscription .