International Journal of Core Engineering & Management
Volume-6, Issue-08, 2020 ISSN No: 2348-9510

GUIDE TO SUBSCRIBE GOOGLE PUB/SUB IN SALESFORCE USING CLOUD RUN

Chirag Amrutlal Pethad
Stores and Services
PetSmart Inc.
Phoenix, Arizona, USA
chiragpethad@live.com, cpethad@petsmart.com

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 a Cloud Run function in Python language, 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, Cloud Run.

I. INTRODUCTION

Salesforce Event Bus [10], also known as the Platform Events [11] 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 [3] 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 [2] [4] in Salesforce using Push method 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.

II. IDENTITY PROVIDERS AND THEIR ROLE IN USER PROVISIONING

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 as follows.

1. Event Delivery

- Salesforce attempts to deliver events in order, but it is not guaranteed.
- Salesforce may deliver events more than once. So consumers or subscribers must handle potential duplicate events.

International Journal of Core Engineering & Management
Volume-6, Issue-08, 2020 ISSN No: 2348-9510

2. **Event Retention and Replay**
   - Salesforce retains the events for only 72 hours, so consumers may miss some or all events if they are not online during that duration.
   - Salesforce does provide event replay option, but it is limited to last 24 hours only. For any missed events older than 24 hours but within 72 hours retention duration, consumers handle the gaps manually.

3. **Event Size and Volume**
   - The maximum Event size including the payload and metadata is 1MB.
   - Depending on the Salesforce edition there are limits on the number of events that can be published and delivered within a 24 hour duration.
   - There are limits on the number of events that can be published per transaction per hour. Maximum of 1000 events per transaction. Per hour limit varies by Salesforce edition.

4. **Event Processing Limits**
   - Each event can have maximum of 50 subscribers which includes Apex triggers, flows, and external systems as well.
   - Salesforce imposes limits on number of long running concurrent apex transactions, which can impact event processing performance.

5. **Governor Limits and Error Handling**
   - Apex triggers on platform events are subject to Salesforce governor limits, such as CPU time, heap size, and SOQL/DML limits.
   - Errors in triggers can cause event processing failures. Proper error handling and retry mechanisms must be implemented.

6. **Integrations with External Systems**
   - Integrating with external systems can introduce latency and reliability issues. Ensure that external systems can handle the volume and frequency of events.
   - Calling external APIs from Salesforce is subject to API call limits and rate limits imposed by the external system.

7. **Maintenance and Upgrades**
   - Salesforce Platform upgrades and changes to Salesforce API versions can impact event processing and functionality. Ensuring compatibility with the Salesforce changes is required.

8. **Monitoring and Debugging**
   - 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 issues with event processing can be challenging due to asynchronous nature and potential delays in event delivery.

International Journal of Core Engineering & Management
Volume-6, Issue-08, 2020 ISSN No: 2348-9510

III. OVERVIEW OF GOOGLE CLOUD PUB-SUB

Google Cloud Pub/Sub [4] 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. Some of the key features [3] of Google Pub Sub are 1) It handles high throughput and low latency messages. 2) Helps ensure reliable message delivery with at-least once delivery. 3) It integrates with various GCP services and external systems.

IV. OVERVIEW OF GOOGLE CLOUD RUN

Google Cloud Run is a fully managed compute platform that automatically scales stateless containers. It is designed to simplify the process of deploying and running containerized applications in the cloud without the need to manage servers or infrastructure. Cloud Run integrates seamlessly with other Google Cloud services, allowing you to build event-driven architectures. For example, you can trigger a Cloud Run service in response to events from Google Cloud Pub/Sub, Firebase, or Google Cloud Storage. It's key features include.

1. Serverless

Cloud Run abstracts away the underlying infrastructure, allowing you to focus solely on your application code. It automatically handles scaling, load balancing, and infrastructure management, providing a true serverless experience.

2. Scalability

Cloud Run automatically scales your application up or down based on incoming traffic. It can scale down to zero when there is no traffic, ensuring that you only pay for what you use.

3. Language and Framework Agnostic

Since Cloud Run runs containers, it supports any programming language or framework that can be containerized. This flexibility allows developers to use the tools and languages they are most comfortable with.

4. Event Driven Architecture

Cloud Run integrates seamlessly with other Google Cloud services, allowing you to build event-driven architectures. For example, you can trigger a Cloud Run service in response to events from Google Cloud Pub/Sub, Firebase, or Google Cloud Storage.

5. REST Endpoints

Cloud Run exposes your application as a fully managed HTTP/HTTPS endpoint. This makes it easy to build and expose web applications, APIs, or microservices that can be accessed over the internet.

V. OVERVIEW OF SALESFORCE APEX

Salesforce provides an object oriented programming language called Apex [1] to perform code driven customizations on Salesforce platform to meet business requirements. It enables developers to execute flows and transaction control statements on the Salesforce platform. It also enables creation of web services, email services, and implement complex business processes. It is highly scalable, robust and supports integrations with external systems via REST and SOAP architecture.

International Journal of Core Engineering & Management
Volume-6, Issue-08, 2020 ISSN No: 2348-9510

VI. 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 the following steps.

1. Salesforce Setup

In this step we create a Connected App and create a REST Web Service [5] to receive the Pub Sub Event messages in Salesforce for further processing.

2. Setting up Google Cloud Run

In this step we create the Cloud Run Service, we develop and deploy the Cloud Run Function that connects to the Salesforce REST Web Service.

3. Setting up Google Cloud Pub-Sub

In this step we create a Pub-Sub Topic with a Push Subscription that delivers message to the Cloud Run Service.

VII. STEP BY STEP IMPLEMENTATION

Here is the detailed step by step instruction to implement all the necessary steps in Salesforce and Google Cloud for the integration to be successful.

1. Salesforce Setup

We start by setting up a new User / Service account in Salesforce with appropriate permissions. We also create an APEX REST Web Service to allow external systems to communicate / integrate with Salesforce.

A. Setup a new Service Account User.

- Log into your Salesforce instance with your Administrator account.
- Go to Setup and Select Users under Users section.
- Click on New User button and provide below detail.
- First Name: Service (or name indicating the account's purpose).
- Last Name: Account
- Email: Provide a valid email address (for notifications and password reset).
- Username: Unique username across all Salesforce instances globally.
- User License: Select Salesforce as the License.
- Profile: Assign a relevant profile that provides necessary permissions for one of the standard profile or create a custom profile according to your business requirements.
- Check the "Generate new password and notify user immediately" option and Click Save.
- Assign additional Permissions using Permission Sets to grant specific access for the Service account based on business requirements.

B. Setup Connected App [9]

- In the setup menu Select the App Manager.
- Click on New Connected App button.
- Fill in the required fields such as App Name, API name, and Contact email.
- Select the Enable OAuth Settings option.
- Set the Callback URL to "https://login.salesforce.com/services/oauth2/callback".

International Journal of Core Engineering & Management
Volume-6, Issue-08, 2020 ISSN No: 2348-9510

- Add "Full" option for the required OAuth Scopes.
- Save the connected app and capture the Consumer Key and Secret for Google Cloud Integration.

C. Create and Configure Salesforce APEX REST Web Service to receive Pub Sub Messages

- In the setup menu Select Apex Classes [1].
- Click New to create a new Apex Class as follows.

```
//@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
    }
}
```

Figure 1: Apex REST service to receive and process incoming messages.

- Grant access to the Service account for the Apex Class.

2. Setting up Google Cloud Run Service

In this step, we create a Cloud Run Service, Develop and Deploy a Cloud Run Function in Python that integrates with the Salesforce APEX REST endpoint to forward the Pub Sub message to Salesforce for further processing.

A. Create a Cloud Run Service

- Navigate to Cloud Run in the GCP Console.
- Click "Create Service" and select the appropriate project.
- Choose a Region close to your Salesforce instance considering the latency.
- Deploy a simple HTTP server as a placeholder to initialize the service.

B. Create Cloud Run Function

- We will create a function in the Python language that will subscribe to the Pub Sub messages. You can choose any other language e.g. Node.js that are supported by Cloud Run.
- The function will parse the incoming Pub Sub message.
- The function authenticates with Salesforce using OAuth2.0.

International Journal of Core Engineering & Management
Volume-6, Issue-08, 2020 ISSN No: 2348-9510

- And the function will make an HTTP POST call to the Salesforce APEX REST Api to publish the message to Salesforce App for further processing.
- The function uses Salesforce Connected App Credentials (Consumer Key and Secret) to authenticate and obtain the access token to call the REST Api.
- The Cloud function will be developed as follows.

```python
import base64
import json
import requests
from flask import Flask, request

app = Flask(__name__)

@app.route('/', methods=['POST'])
def handle_pubsub_message():
    pubsub_message = request.get_json()
    message_data = base64.b64decode(
      pubsub_message['message']['data']).decode('utf-8')
    message = json.loads(message_data)

    # Authenticate with Salesforce
    salesforce_url = "<https://your-salesforce-instance.salesforce.com/"
     + "services/apexrest/PubSubHandler"
    access_token = get_salesforce_token()

    headers = {
        'Authorization': f'Bearer {access_token}',
        'Content-Type': 'application/json'
    }

    # Send data to Salesforce
    response = requests.post(salesforce_url, headers=headers, json=message)
    return response.text, response.status_code

def get_salesforce_token():
    # Implement OAuth2.0 flow to get an access token from Salesforce
    return "your_access_token"

if __name__ == "__main__":
    app.run(debug=True)
```

Figure 2: Python Cloud Run Function

C. Deploy the Function to Cloud Run

- Deploy the function to Cloud Run and ensure the service is configured to receive HTTP POST requests as follows.

```
gcloud run deploy salesforce-subscriber --source . --region your-region
```

Figure 3: G-Cloud Command to deploy the Cloud Run Service.

3. Setting up Google Cloud Pub Sub [2]

We also create a Pub Sub Topic and a Push Subscription that triggers the Cloud Run function / service endpoint.

A. Create a Pub Sub Topic

- Navigate to Pub/Sub section in Google Cloud Console.
- Click “Create Topic”.

International Journal of Core Engineering & Management
Volume-6, Issue-08, 2020 ISSN No: 2348-9510

- Enter a unique Topic ID such as "MyPubSubTopic".
- Un-Select the "Add a default subscription" option.
- Click "Create".

B. Create a Push Subscription [6]

- Navigate to Subscriptions under Pub/Sub section.
- Click "Create Subscription".
- Enter a unique Subscription ID such as "MySalesforcePubSubSubscription".
- Select the Pub Sub Topic we created in previous step.
- Select "Push" as the Delivery Type.
- Enter the Cloud Run Service URL for the Endpoint URL. E.g. "https://your-service-url.run.app".

C. Setup IAM Permissions

- Navigate to IAM and Admin section in Google Cloud Console.
- Create a new Service account under Service Accounts section.
- Navigate to IAM section and Grant "Cloud Pub/Sub Subscriber" and "Cloud Run Invoker" role / access to the Service account / principal.

4. Testing the Deployment

And finally, we Publish a test message to validate end to end processing of the message by monitoring Cloud Run Logs and Salesforce Debug logs.

A. Publish a Test Pub Sub Message

- We can use Google Cloud Console or G-Cloud CLI to publish a test message to the Pub Sub Topic. Below is the G-Cloud CLI Command example.

```
gcloud pubsub topics publish salesforce-updates --message='{"field1": "value1", "field2": "value2"}'
```

Figure 4: G-Cloud Command to Publish a Pub Sub Message on a Topic name "salesforce-update".

B. Monitor Cloud Run Logs

- Monitor and Check Cloud Run logs to make sure the message was successfully received by the Cloud Run function and then successfully processed and forwarded to Salesforce APEX REST Api.

C. Monitor Salesforce Developer Console Logs

- Monitor and Check Salesforce Developer Console logs or Debug Logs under setup to make sure the REST Api was successfully invoked and the message was successfully received and processed.

VIII. LIMITATIONS AND CHALLENGES

When using Cloud Run to subscribe to Google Cloud Pub/Sub and publish to a Salesforce REST API, there are several challenges and limitations to consider. Here's a breakdown of potential issues you might face:

International Journal of Core Engineering & Management
Volume-6, Issue-08, 2020 ISSN No: 2348-9510

1. **Cold Start and Latency**
   - Cold Start Delays: Cloud Run instances may experience a delay when spinning up if there hasn't been recent activity. Since Pub/Sub pushes messages to Cloud Run in an event-driven fashion, if the service hasn't been invoked recently, there can be cold start delays, causing latency in processing Pub/Sub messages.
   - Latency Impact on Salesforce API calls: Any delay in Cloud Run startup could increase the time it takes to make API calls to Salesforce, potentially causing timeouts or performance bottlenecks, especially in high-throughput scenarios.

2. **Rate Limiting and Throttling**
   - Salesforce API Limits: Salesforce imposes strict rate limits on API requests (e.g., daily limits, concurrent API call limits). If Cloud Run processes a high volume of messages from Pub/Sub, you may exceed Salesforce API limits, causing errors and requiring complex retry logic.
   - Exponential Back off or Retries: Implementing exponential back off for retries when Salesforce rate limits are hit can be tricky, and poorly handled retries may compound the problem by adding load.

3. **Authentication Challenges**
   - OAuth2.0 Tokens: Connecting Cloud Run to the Salesforce REST API typically requires an OAuth 2.0 token. Properly managing token refresh (due to token expiration) and securely storing client credentials is crucial. Implementing token management in a server less architecture like Cloud Run can be challenging, especially when scaling across multiple instances.
   - Service Account Permissions: The Cloud Run service account needs proper IAM permissions to access Pub/Sub and other Google Cloud services, and security needs to be carefully managed to avoid over-provisioning of permissions.

4. **Handling Pub Sub Message Ordering**
   - Out of Order Processing: Pub/Sub does not guarantee message ordering by default unless you use message ordering with a specific ordering key. This could result in out-of-order data being sent to Salesforce, which may require additional logic to handle correctly in your Cloud Run service.
   - Duplicate Message Processing: Pub/Sub can deliver duplicate messages in rare cases, so idempotency is important when sending data to Salesforce to avoid creating duplicate records or inconsistent states.

5. **Timeouts and Retries**
   - Pub Sun Acknowledgement Deadlines: Cloud Run instances need to acknowledge Pub/Sub messages within a specific timeframe (default is 10 seconds, can be extended). If the processing of Salesforce requests takes too long (due to network issues or Salesforce response delays), you might miss the acknowledgement deadline, resulting in message re-delivery.
   - Salesforce Timeout: The Salesforce API also has its own timeout limits. Long-running operations on Cloud Run may cause timeouts, leading to failed transactions, and Pub/Sub message retries.

International Journal of Core Engineering & Management
Volume-6, Issue-08, 2020 ISSN No: 2348-9510

6. **Concurrency and Scaling**
   - Auto-Scaling of Cloud Run: Cloud Run scales automatically based on incoming requests. However, if Pub/Sub sends a high volume of messages, Cloud Run could scale up quickly, possibly leading to a flood of Salesforce API requests that exceed limits.
   - Concurrency Handling: Cloud Run allows multiple requests per instance by default. Handling concurrency issues (like multiple requests trying to send the same data to Salesforce) may require additional synchronization logic.

7. **Error Handling and Logging**
   - Handling Salesforce Errors: If a Salesforce request fails (due to a 4xx or 5xx response), you need robust error handling, retries, and possibly dead-letter queue (DLQ) implementations to ensure messages are not lost.
   - Logging and Debugging: With serverless architectures, logging and debugging can be more difficult than in traditional environments. You will need to rely on tools like Google Cloud Logging and monitoring solutions to effectively debug and trace errors.

8. **Cost Considerations**
   - Cloud Run Costs: While Cloud Run is cost-effective for intermittent workloads, high-frequency Pub/Sub messages may cause Cloud Run to scale significantly, leading to increased costs.
   - Salesforce API Call Costs: Depending on your Salesforce edition, excessive API calls may also incur additional costs or require upgrading your plan to handle higher API request limits.

9. **Security Concerns**
   - Data Security: Ensuring secure transmission of data between Cloud Run and Salesforce (e.g., using HTTPS, proper OAuth scopes) is crucial.
   - Service Account and Credentials: Careful management of service accounts, API keys, and OAuth credentials is important to avoid security vulnerabilities.

IX. BEST PRACTICES

When implementing Cloud Run to subscribe to Google Cloud Pub/Sub and publish to the Salesforce REST API, it's important to follow best practices to ensure scalability, reliability, and maintainability. Below are key best practices to guide your implementation:

1. **Efficient Cloud Run Setup [7]**
- Use Container-First Design: Ensure your Cloud Run service is optimized for containerized environments. Keep the container lightweight, only including necessary dependencies.
- Minimize Cold Start Latency: To reduce cold start times, optimize the container image (e.g., use a minimal base image like alpine). Avoid heavyweight dependencies, and pre-load essential resources during startup to minimize delays in processing Pub/Sub messages.
- Warm Start Strategy: To avoid cold starts, you could use Cloud Scheduler to send periodic "pings" to keep your Cloud Run instance warm if low-latency response times are critical.

2. **Optimize Pub Sub Integration [8]**
- Set Appropriate Acknowledge Deadlines: Use an appropriate Pub/Sub acknowledgement deadline that accounts for the time needed to process messages, including Salesforce API calls.

International Journal of Core Engineering & Management
Volume-6, Issue-08, 2020 ISSN No: 2348-9510

The deadline can be set to up to 600 seconds to accommodate longer operations if needed.

- Use Dead Letter Topics (DLTs): Configure dead-letter topics to capture messages that fail multiple times (e.g., after exceeding a retry limit) instead of dropping them. This helps with debugging and ensures no messages are lost.
- Enable Message Ordering (if Required): If message order is important (e.g., for sequential updates in Salesforce), enable Pub/Sub message ordering using an ordering key to ensure messages are processed in the correct sequence.

3. Handle Salesforce API Limits Gracefully [12]

- Respect Rate Limits: Salesforce imposes strict API call limits. Use throttling mechanisms within Cloud Run to ensure the service doesn't overwhelm Salesforce with too many requests, especially when Pub/Sub sends a large volume of messages.
- Implement Exponential Backoff for Retries: In the event of errors such as rate limits (HTTP 429) or Salesforce API timeouts (HTTP 5xx), implement exponential backoff and retry mechanisms to avoid continuously hitting the Salesforce API at high frequency.
- Batch API Requests: If possible, batch multiple Pub/Sub messages into a single API call to Salesforce (e.g., using Salesforce Bulk API) to reduce the number of API requests and avoid rate limiting issues.

4. Implement Robust Authentication and Security

- Use OAuth 2.0 with Token Refresh: Use Salesforce's OAuth 2.0 protocol for authenticating API calls. Implement automatic token refresh mechanisms in your Cloud Run service to handle token expiration without manual intervention.
- Secure Environment Variables: Store sensitive information (e.g., Salesforce client credentials, OAuth tokens) as encrypted environment variables using Google Cloud Secret Manager. Avoid hardcoding sensitive information in the container image or codebase.
- Use Service Accounts with Least Privilege: Assign the Cloud Run service a Google Cloud service account with the minimum permissions required to subscribe to Pub/Sub and other necessary resources. Avoid using overly permissive roles to reduce security risks.

5. Optimize for Scalability and Performance [7]

- Control Concurrency: By default, Cloud Run instances handle multiple concurrent requests. Set the appropriate concurrency level (-concurrency flag) based on the complexity and processing time of each Pub/Sub message. If each message requires significant processing (e.g., heavy Salesforce API interaction), you may want to reduce concurrency or set it to 1 for sequential processing.
- Auto-Scaling Configuration: Cloud Run automatically scales based on incoming traffic. Set appropriate maximum instance limits (-max-instances) to prevent Cloud Run from over-scaling and overwhelming the Salesforce API with too many concurrent requests.

6. Ensure Idempotency

- Idempotent API Requests: Pub/Sub may occasionally deliver the same message multiple times, so it's crucial that Salesforce API requests are idempotent. Design your API calls to Salesforce (e.g., upserts instead of inserts) to ensure that multiple executions of the same message don't cause duplicate records or data corruption.
- Message Deduplication: If necessary, implement a deduplication mechanism (e.g., use message IDs or timestamps) to track which messages have already been processed to avoid processing the same message multiple times.

International Journal of Core Engineering & Management
Volume-6, Issue-08, 2020 ISSN No: 2348-9510

7. **Monitor and Log Everything**
- Structured Logging: Use structured logging (e.g., JSON format) to capture detailed logs from both Pub/Sub and Salesforce interactions. These logs should include message IDs, status codes, and response times to make debugging easier.
- Error Tracking and Alerts: Set up error tracking and alerts using Google Cloud Operations (formerly Stack driver). Monitor metrics such as error rates, failed API calls to Salesforce, and Pub/Sub message backlog to quickly identify issues.
- Cloud Run and Pub/Sub Monitoring: Use Google Cloud Monitoring to track important metrics like Cloud Run CPU and memory usage, Pub/Sub subscription throughput, and Salesforce API success rates.

8. **Ensure Reliable Error Handling**
- Graceful Failures: Ensure that your Cloud Run service handles errors gracefully. For example, if a Salesforce API call fails, return an appropriate HTTP status code (e.g., 500) to allow Pub/Sub to retry the message.
- Retries and Circuit Breakers: Implement retry logic with circuit breakers to prevent flooding Salesforce with retries in case of ongoing failures. This helps protect against cascading failures in case Salesforce is temporarily down or overloaded.
- Dead Letter Queues (DLQs): For messages that fail after multiple retries, use Pub/Sub's dead-letter topics to offload them for further investigation rather than discarding the messages.

9. **Test Thoroughly**
- Simulate High Traffic: Test your Cloud Run service under high traffic conditions by simulating a large number of Pub/Sub messages. Ensure that the service scales effectively without overwhelming Salesforce or breaching API limits.
- Integration Testing with Salesforce: Conduct end-to-end tests with Salesforce's sandbox environment to verify that the API interactions behave as expected. Simulate different error conditions such as timeouts, rate limits, and network issues to validate your error-handling logic.

10. **Cost Management**
- Optimize Cloud Run Usage: Monitor Cloud Run usage and set appropriate autoscaling limits to avoid unnecessary scaling that can lead to high costs. For long-running tasks, consider breaking them up or using Pub/Sub push subscriptions instead of pull, which can reduce the time Cloud Run needs to stay active.
- Monitor API Usage: Track your Salesforce API usage regularly to ensure you're not exceeding your daily API limits and potentially incurring additional costs.

X. 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.

International Journal of Core Engineering & Management
Volume-6, Issue-08, 2020 ISSN No: 2348-9510

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. [Google Pub Sub Push Subscription - https://cloud.google.com/pubsub/docs/create-subscription.
7. Cloud Run Best Practices - https://cloud.google.com/run/docs/tips/general
8. Google Pub Sub Best Practices - https://cloud.google.com/pubsub/docs/subscribe-best-practices
9. Connected App in Salesforce - https://help.salesforce.com/s/articleView?id=sf.connected_app_create.htm&language=en_US&type=5
10. Salesforce Event Bus - https://help.salesforce.com/s/articleView?id=release-notes.rn_messaging_event_bus_section.htm&release=238&type=5
11. Salesforce Platform Events - https://developer.salesforce.com/docs/atlas.en-us.platform_events.meta/platform_events/platform_events_intro.htm
12. Salesforce Platform Events Limits - https://developer.salesforce.com/docs/atlas.en-us.platform_events.meta/platform_events/platform_event_limits.htm