North American Journal of Engineering and Research

Est. 2020

How to Publish and Subscribe to Amazon SQS with Salesforce Apex

Chirag Amrutlal Pethad
Email: ChiragPethad@gmail.com

Abstract

The document outlines the integration of Amazon SQS with Salesforce to enhances event-driven architectures by overcoming limitations of the Salesforce Event Bus. The guide outlines the benefits of asynchronous processing, scalability, and reliability, and provides a step-by-step integration plan, including setting up SQS, configuring IAM, and developing Apex classes for message handling. It emphasizes security considerations, testing, and best practices for maintaining a robust integration.

Keywords—Event Bus, Event Driven Architecture, AWS, Amazon SQS, Integration, Publish, Subscribe, Limits, Scalability.

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. Amazon SQS is a fully managed message queuing service that enables you to decouple and scale micro services, distributed systems, and server-less applications. The integration of Amazon SQS with Salesforce enables businesses to harness the power of real-time asynchronous data processing and avoid the limitations associated with Salesforce Event Bus and enabling more efficient and scalable operations. This white paper provides a detailed guide on integrating Salesforce with Amazon Simple Queue Service (SQS) using Salesforce Apex. It includes an overview of both Salesforce and Amazon SQS, the benefits of integrating these two platforms, step-by-step instructions for setting up the integration, and best practices for maintaining a robust and secure integration for improved operational efficiency, seamless and efficient flow of information.

Overview of Salesforce and Amazon SQS

Salesforce

Salesforce is a cloud-based CRM platform that provides tools and services for managing customer relationships, sales, and marketing efforts. It includes powerful features such as workflow automation, analytics, and integration capabilities.

Amazon SQS

Amazon SQS is a message queuing service that allows you to send, store, and receive messages between software components. SQS helps in building distributed systems, ensuring messages are delivered and processed reliably.

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.

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:
  - 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.
  - 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 latt 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.

Benefits of Integrating Salesforce with Amazon SQS

Key Benefits

- Scalability: Handle high throughput and low-latency messages. Easily scales to handle large volumes of messages and transactions.
- Reliability: Ensure message delivery with at-least-once delivery and reduces the risk of data loss.
- Decoupling: Decouples components, making the system more flexible and easier to maintain.
- Asynchronous Processing: Allows Salesforce to handle tasks asynchronously, improving system performance and user experience.

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 integration strategy involves setting up a Amazon SQS, configuring IAM accounts and policies, and implementing Apex code in Salesforce to publish/subscribe to Amazon SQS. The process includes:

Setting up Amazon SQS

Create a SQS queue and configure necessary IAM policies.

Salesforce Setup

Configure named credentials and remote site settings.

Apex Implementation

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

Step By Step Implementation

Setting up Amazon SQS queue

- Log in to your AWS Management Console.

- Navigate to Amazon SQS.
- Click on "Create Queue".
- Choose the queue type (Standard or FIFO) and configure the queue settings.
- Note down the queue URL.

Configure IAM

Step 1: Create an IAM User

- Navigate to IAM console.
- Click on "Users" and then "Add user".
- Provide a username and select "Programmatic access".
- Click "Next: Permissions".

Step 2: Attach Policies to IAM User

- Attach the "AmazonSQSFullAccess" policy to the user.
- Review and create the user.
- Note down the Access Key ID and Secret Access Key.

Configure Named Credentials in Salesforce

Step 1: Configure Named Credential

- In Setup, navigate to Security > Named Credentials.
- Click New Named Credential.
- Fill out the form as follows:
  1. Label: AmazonSQS
  2. Name: AmazonSQS
  3. URL: https://sqs.<region>.amazonaws.com/
  4. Identity Type: Named Principal
  5. Authentication Protocol: AWS Signature Version 4
  6. AWS Access Key: Access Key ID from IAM user
  7. AWS Secret Key: Secret Access Key from IAM user
- Save the Named Credential.

Configure Remote Site setting in Salesforce

- In Setup, navigate to Security Controls -> Remote Site Settings.
- Add a new remote site with the Amazon SQS URL.

Implement Apex Class for Publishing and Subscribing messages to/from SQS

```
public class MySQSService {
    private static final String NAMED_CREDENTIAL = 'AmazonSQS';
    private static final String QUEUE_URL =
      'https://sqs.<region>.amazonaws.com/<account-id>/<queue-name>';

    public static void sendMessage(String messageBody) {
        HttpRequest req = new HttpRequest();
        req.setEndpoint(NAMED_CREDENTIAL + QUEUE_URL);
        req.setMethod('POST');
        req.setHeader('Content-Type', 'application/x-www-form-urlencoded');

        String body = 'Action=SendMessage&MessageBody='
                      + EncodingUtil.urlEncode(messageBody, 'UTF-8');
        req.setBody(body);
        Http http = new Http();
        HttpResponse res = http.send(req);

        if (res.getStatusCode() != 200) {
            throw new CalloutException(
              'Failed to send message to SQS: ' + res.getBody());
        }
    }

    public static String receiveMessage() {
        HttpRequest req = new HttpRequest();
        req.setEndpoint(NAMED_CREDENTIAL + QUEUE_URL);
        req.setMethod('POST');
        req.setHeader('Content-Type', 'application/x-www-form-urlencoded');

        String body = 'Action=ReceiveMessage&MaxNumberOfMessages=10';
        req.setBody(body);
        Http http = new Http();
        HttpResponse res = http.send(req);

        if (res.getStatusCode() == 200) {
            return res.getBody();
        } else {
            throw new CalloutException(
              'Failed to receive message from SQS: ' + res.getBody());
        }
    }
}
```

Implement Apex Trigger / Batch Job

Implement a trigger or a batch job to handle the processing of message received from SQS.

```
public class MessageProcessor {
    public static void processMessages() {
        String message = MySQSService.receiveMessage();
        System.debug('Received message: ' + message);
    }
}
```

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 policies and permissions for IAM user.
- 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 Amazon SQS.
- Performance Testing: Ensure the system can handle the expected message load.

Best Practices

- Error Handling: Implement robust error handling in your Apex code to manage failed SQS operations.
- Logging: Use Salesforce logging to monitor and troubleshoot issues.
- Scalability: Design your solution to handle large volumes of messages and ensure your Salesforce governor limits are considered.
- Batch: Implement batching to pull messages in bulk.
- Security: Ensure your AWS credentials are securely stored and managed. Use IAM roles and policies to restrict access.
- Monitoring: Monitor the integration using Salesforce debug logs and AWS CloudWatch for SQS metrics.

Conclusion

Integrating Salesforce with Amazon SQS provides a powerful solution for handling asynchronous processing and scaling your operations. By following the steps outlined in this white paper, organizations can set up a robust and secure integration, leveraging the strengths of both Salesforce and Amazon SQS. 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 Amazon SQS and Salesforce.

References

[1] Apex Developer Guide - https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_dev_guide.htm

[2] Amazon SQS Documentation - https://docs.aws.amazon.com/sqs/

[3] Amazon IAM Documentation - https://docs.aws.amazon.com/iam/

[4] Apex Integration - https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_integration_intro.htm

[5] Named Credentials - https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_callouts_named_credentials.htm