AWS Architecture Blog

Coordinating large messages across accounts and Regions with Amazon SNS and SQS

by Mrudhula Balasubramanyan | on 08 AUG 2022 | in Amazon Elastic Container Service, Amazon Simple Notification Service (SNS), Amazon Simple Queue Service (SQS), Amazon Simple Storage Service (S3), Architecture, AWS Lambda, AWS Step Functions | Permalink | Share

Many organizations have applications distributed across various business units. Teams in these business units may develop their applications independent of each other to serve their individual business needs. Applications can reside in a single Amazon Web Services (AWS) account or be distributed across multiple accounts. Applications may be deployed to a single AWS Region or span multiple Regions.

Irrespective of how the applications are owned and operated, these applications need to communicate with each other. Within an organization, applications tend to be part of a larger system, therefore, communication and coordination among these individual applications is critical to overall operation.

There are a number of ways to enable coordination among component applications. It can be done either synchronously or asynchronously:

- **Synchronous communication** uses a traditional request-response model, in which the applications exchange information in a tightly coupled fashion, introducing multiple points of potential failure.
- **Asynchronous communication** uses an event-driven model, in which the applications exchange messages as events or state changes and are loosely coupled. Loose coupling allows applications to evolve independently of each other, increasing scalability and fault-tolerance in the overall system.

Event-driven architectures use a publisher-subscriber model, in which events are emitted by the publisher and consumed by one or more subscribers.

A key consideration when implementing an event-driven architecture is the size of the messages or events that are exchanged. How can you implement an event-driven architecture for large messages, beyond the default maximum of the services? How can you architect messaging and automation of applications across AWS accounts and Regions?

This blog presents architectures for enhancing event-driven models to exchange large messages. These architectures depict how to coordinate applications across AWS accounts and Regions.

Challenge with application coordination

A challenge with application coordination is exchanging large messages. For the purposes of this post, a large message is defined as an event payload between 256 KB and 2 GB. This stems from the fact that Amazon Simple Notification Service (Amazon SNS) and Amazon Simple Queue Service (Amazon SQS) currently have a maximum event payload size of 256 KB. To exchange messages larger than 256 KB, an intermediate data store must be used.

To exchange messages across AWS accounts and Regions, set up the publisher access policy to allow subscriber applications in other accounts and Regions. In the case of large messages, also set up a central data repository and provide access to subscribers.

Figure 1 depicts a basic schematic of applications distributed across accounts communicating asynchronously as part of a larger enterprise application.

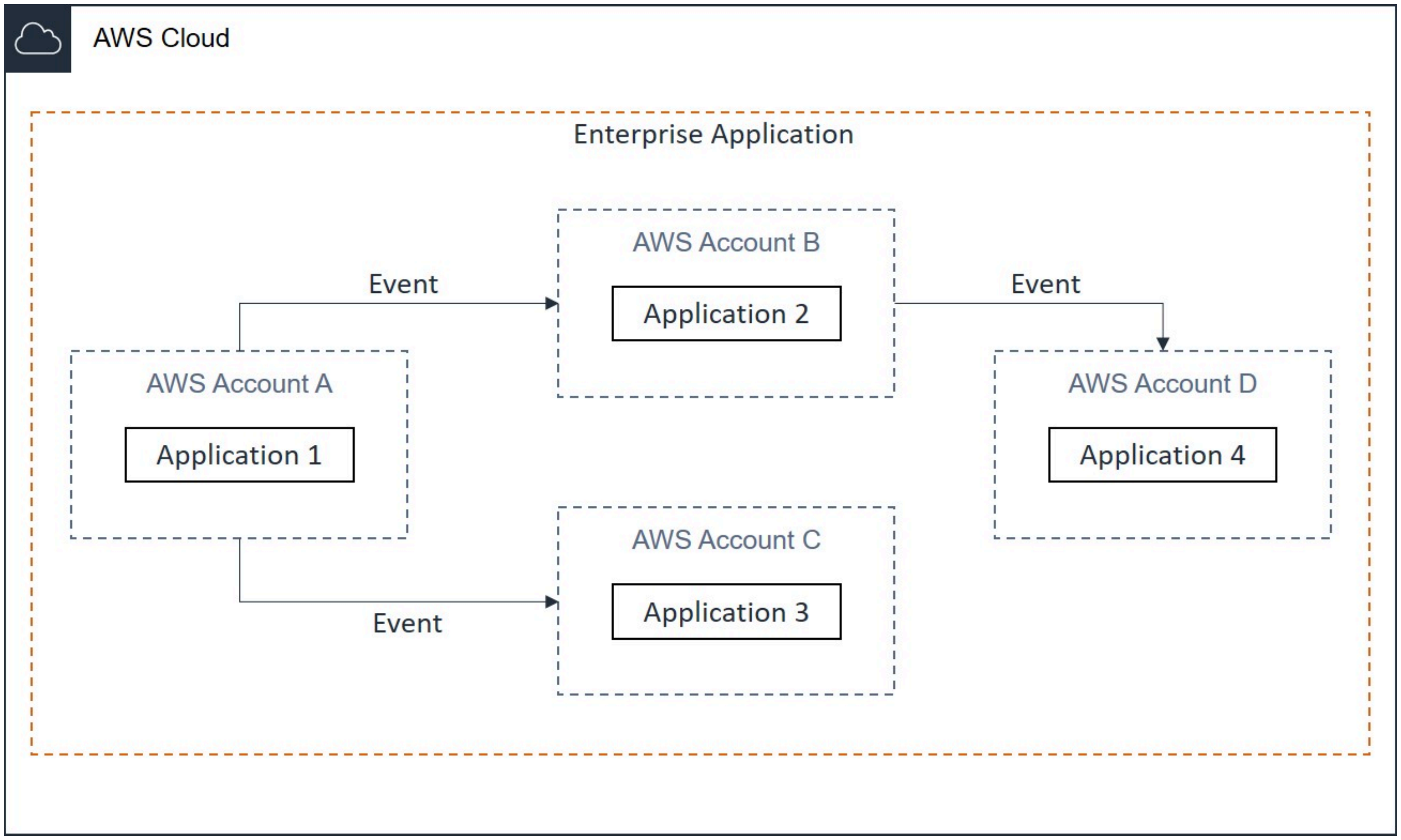

Figure 1. Asynchronous communication across applications

Architecture overview

The overview covers two scenarios:

1. Coordination of applications distributed across AWS accounts and deployed in the **same Region**
2. Coordination of applications distributed across AWS accounts and deployed to **different Regions**

Coordination across accounts and single AWS Region

Figure 2 represents an event-driven architecture, in which applications are distributed across AWS Accounts A, B, and C. The applications are all deployed to the same AWS Region, us-east-1. A single Region simplifies the architecture, so you can focus on application coordination across AWS accounts.

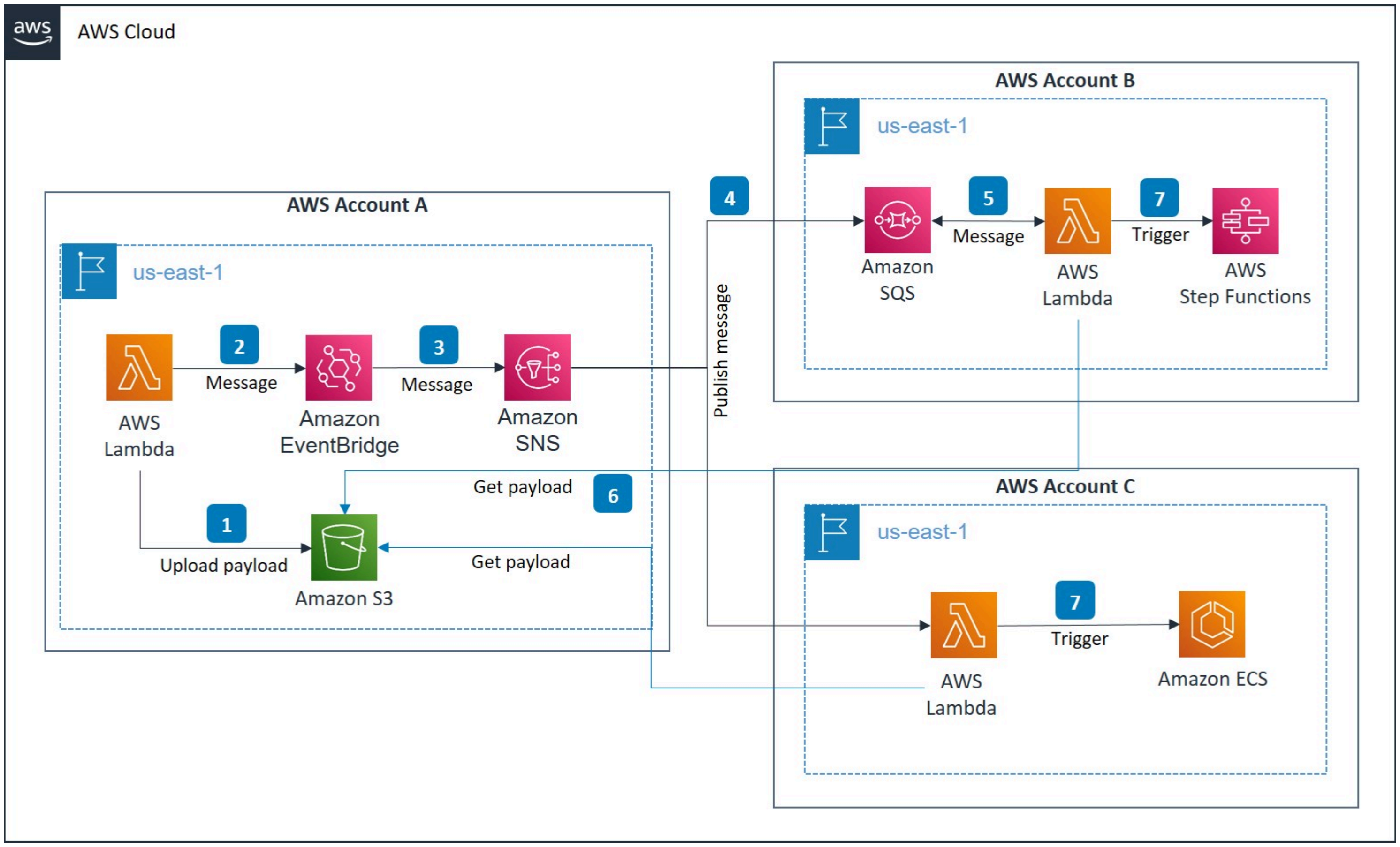

Figure 2. Application coordination across accounts and single AWS Region

The application in Account A (Application A) is implemented as an AWS Lambda function. This application communicates with the applications in Accounts B and C. The application in Account B is launched with AWS Step Functions (Application B), and the application in Account C runs on Amazon Elastic Container Service (Application C).

In this scenario, Applications B and C need information from upstream Application A. Application A publishes this information as an event, and Applications B and C subscribe to an SNS topic to receive the events. However, since they are in other accounts, you must define an access policy to control who can access the SNS topic. You can use sample Amazon SNS access policies to craft your own.

If the event payload is in the 256 KB to 2 GB range, you can use Amazon Simple Storage Service (Amazon S3) as the intermediate data store for your payload. Application A uses the Amazon SNS Extended Client Library for Java to upload the payload to an S3 bucket and publish a message to an SNS topic, with a reference to the stored S3 object. The message containing the metadata must be within the SNS maximum message limit of 256 KB. Amazon EventBridge is used for routing events and handling authentication.

The subscriber Applications B and C need to de-reference and retrieve the payloads from Amazon S3. The SQS queue in Account B and Lambda function in Account C subscribe to the SNS topic in Account A. In Account B, a Lambda function is used to poll the SQS queue and read the message with the metadata. The Lambda function uses the Amazon SQS Extended Client Library for Java to retrieve the S3 object referenced in the message.

The Lambda function in Account C uses the Payload Offloading Java Common Library for AWS to get the referenced S3 object.

Once the S3 object is retrieved, the Lambda functions in Accounts B and C process the data and pass on the information to downstream applications.

This architecture uses Amazon SQS and Lambda as subscribers because they provide libraries that support offloading large payloads to Amazon S3. However, you can use any Java-enabled endpoint, such as an HTTPS endpoint that uses Payload Offloading Java Common Library for AWS to de-reference the message content.

Coordination across accounts and multiple AWS Regions

Sometimes applications are spread across AWS Regions, leading to increased latency in coordination. For existing applications, it could take substantive effort to consolidate to a single Region. Hence, asynchronous coordination would be a good fit for this scenario. Figure 3 expands on the architecture presented earlier to include multiple AWS Regions.

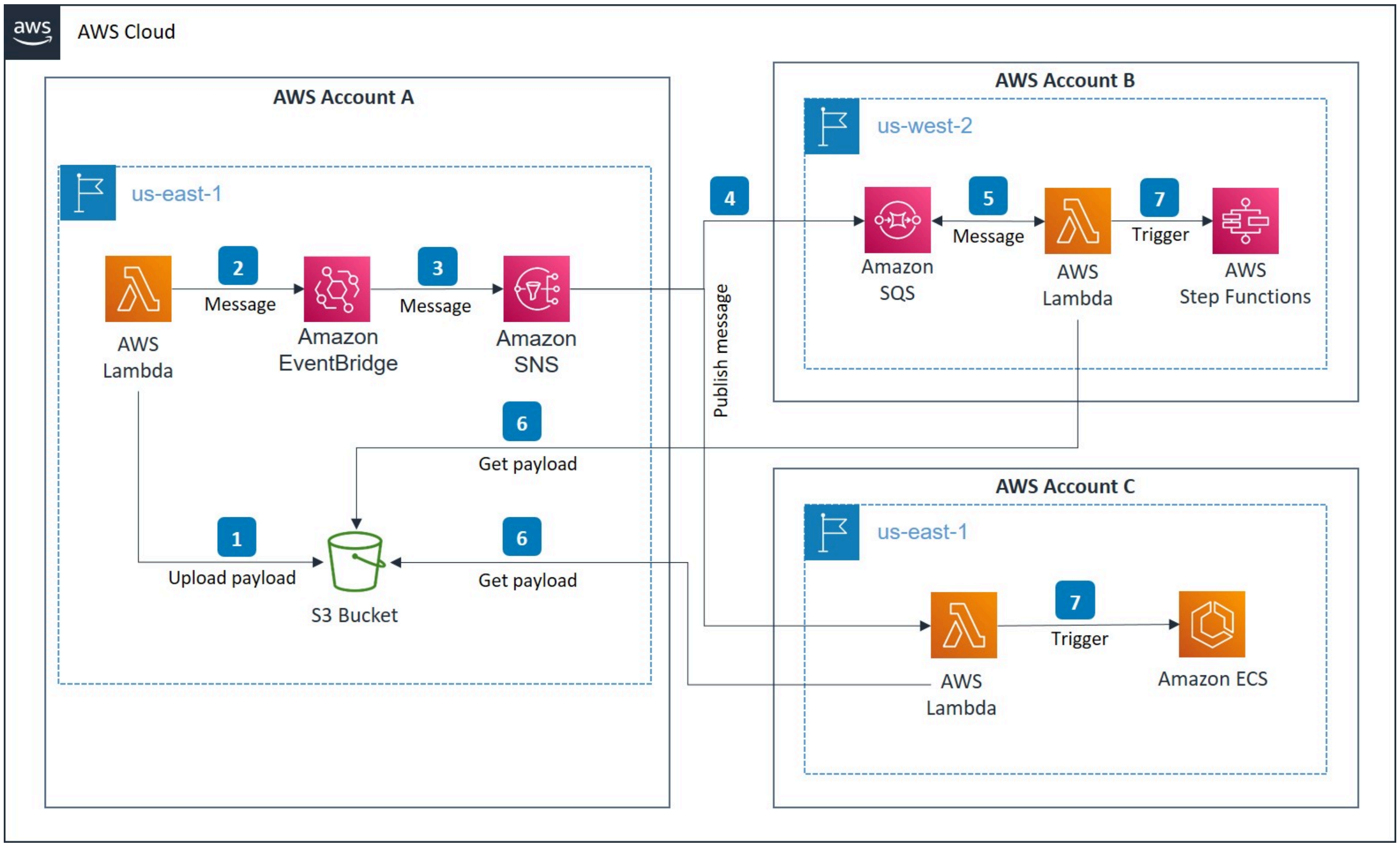

Figure 3. Application coordination across accounts and multiple AWS Regions

The Lambda function in Account C is in the same Region as the upstream application in Account A, but the Lambda function in Account B is in a different Region. These functions must retrieve the payload from the S3 bucket in Account A.

To provide access, configure the AWS Lambda execution role with the appropriate permissions. Make sure that the S3 bucket policy allows access to the Lambda functions from Accounts B and C.

Considerations

For variable message sizes, you can specify if payloads are always stored in Amazon S3 regardless of their size, which can help simplify the design.

If the application that publishes/subscribes large messages is implemented using the AWS Java SDK, it must be Java 8 or higher. Service-specific client libraries are also available in Python, C#, and Node.js.

An Amazon S3 Multi-Region Access Point can be an alternative to a centralized bucket for the payloads. It has not been explored in this post due to the asynchronous nature of cross-region replication.

In general, retrieval of data across Regions is slower than in the same Region. For faster retrieval, workloads should be run in the same AWS Region.

Conclusion

This post demonstrates how to use event-driven architectures for coordinating applications that need to exchange large messages across AWS accounts and Regions. The messaging and automation are enabled by the Payload Offloading Java Common Library for AWS and use Amazon S3 as the intermediate data store. These components can simplify the solution implementation and improve scalability, fault-tolerance, and performance of your applications.

Ready to get started? Explore SQS Large Message Handling.