AWS Architecture Blog

Modernized Database Queuing using Amazon SQS and AWS Services

by Scott Wainner, Anand Komandooru, and Harpreet Virk | on 17 DEC 2021 | in Amazon CloudWatch, Amazon EventBridge, Amazon RDS, Amazon Simple Notification Service (SNS), Amazon Simple Queue Service (SQS), Amazon Simple Storage Service (S3), Architecture, AWS Lambda | Permalink | Share

This blog post was last reviewed/updated August, 2022. The updated version shown below is based on working backwards from a customer need to ensure data consistency post migration, to a modernized microservice architecture.

A queuing system is composed of producers and consumers. A producer *enqueues* messages (writes messages to a database) and a consumer *dequeues* messages (reads messages from the database). Business applications requiring asynchronous communications often use the relational database management system (RDBMS) as the default message storage mechanism. But the increased message volume, complexity, and size, competes with the inherent functionality of the database. The RDBMS becomes a bottleneck for message delivery, while also impacting other traditional enterprise uses of the database.

In this blog, we will show how you can mitigate the RDBMS performance constraints by using Amazon Simple Queue Service (Amazon SQS), while retaining the intrinsic value of the stored relational data.

Problems with legacy queuing methods

Commercial databases such as Oracle offer Advanced Queuing (AQ) mechanisms, while SQL Server supports Service Broker for queuing. The database acts as a message queue system when incoming messages are captured along with metadata. A message stored in a database is often processed multiple times using a sequence of message extraction, transformation, and loading (ETL). The message is then routed for distribution to a set of recipients based on logic that is often also stored in the database.

The repetitive manipulation of messages and iterative attempts at distributing pending messages may create a backlog that interferes with the primary function of the database. This backpressure can propagate to other systems that are trying to store and retrieve data from the database and cause a performance issue (see Figure 1).

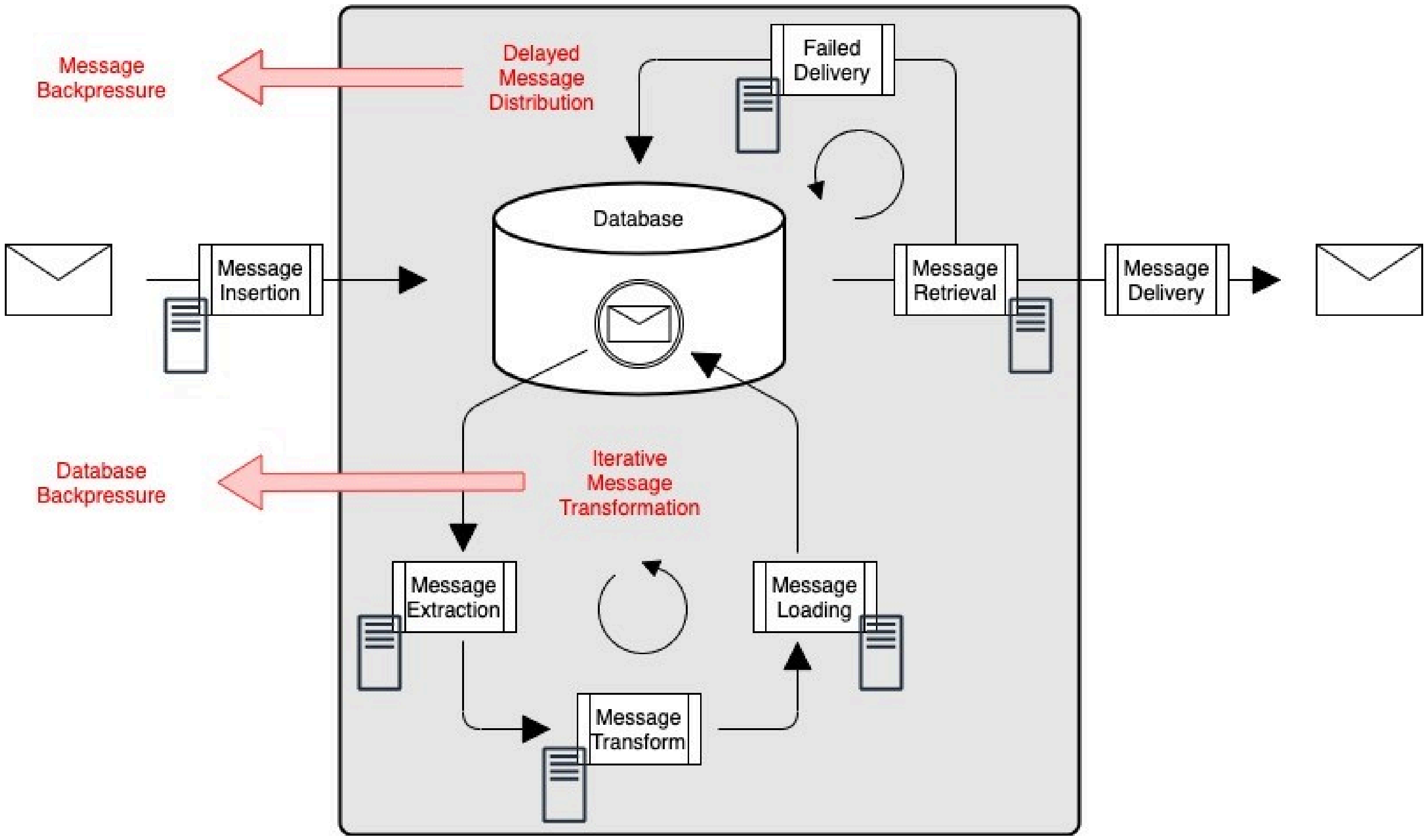

Figure 1. A relational database serving as a message queue.

There are several scenarios where the database can become a bottleneck for message processing:

Message metadata. Messages consist of the *payload* (the content of the message) and *metadata* that describes the attributes of the message. The metadata often includes routing instructions, message disposition, message state, and payload attributes.

- The message metadata may require iterative transformation during the message processing. This creates an inefficient sequence of read, transform, and write processes. The iterative read/write process of metadata consumes the database IOPS, and forces the database to scale vertically (add more CPU and more memory).
- A new paradigm emerges when message management processes exist outside of the database. Here, the metadata is manipulated without interacting with the database, except to write the final message disposition. Application logic can be applied through functions such as AWS Lambda to transform the message metadata.

Message large object (LOB). A message may contain a large binary object that must be stored in the payload.

- Storing large binary objects in the RDBMS is expensive. Manipulating them consumes the throughput of the database with iterative read/write operations.
- An alternative approach offers a more efficient message processing sequence. The large object is stored external to the database in universally addressable object storage, such as Amazon Simple Storage Service (Amazon S3).

Message fan-out. A message can be loaded into the database and analyzed for routing, where the same message must be distributed to multiple recipients.

- Messages that require multiple recipients may require a copy of the message replicated for each recipient. The replication creates multiple writes and reads from the database, which is inefficient.
- A new method captures only the routing logic and target recipients in the database. The message replication then occurs outside of the database in distributed messaging systems, such as Amazon Simple Notification Service (Amazon SNS).

Message queuing. Messages are often kept in the database until they are successfully processed for delivery. If a message is read from the database and determined to be undeliverable, then the message is kept there until a later attempt is successful.

- An inoperable message delivery process can create backpressure on the database where iterative message reads are processed for the same message with unsuccessful delivery. This creates a feedback loop causing even more unsuccessful work for the database.
- Try a message queuing system such as Amazon MQ or Amazon SQS, which offloads the message queuing from the database. These services offer efficient message retry mechanisms, and reduce iterative reads from the database.

Sequenced message delivery. Messages may require ordered delivery where the delivery sequence is crucial for maintaining application integrity.

- The application may capture the message order within database tables, but the sorting function still consumes processing capabilities. The order sequence must be sorted and maintained for each attempted message delivery.
- Message order can be maintained outside of the database using a queue system, such as Amazon SQS, with first-in/first-out (FIFO) delivery.

Message scheduling. Messages may also be queued with a scheduled delivery attribute. These messages require an event driven architecture with initiated scheduled message delivery.

- The database often uses trigger mechanisms to initiate message delivery. Message delivery may require a synchronized point in time for delivery (many messages at once), which can cause a spike in work at the scheduled interval. This impacts the database performance with artificially induced peak load intervals.
- Event signals can be generated in systems such as Amazon EventBridge, which can coordinate the transmission of messages.

Message disposition. Each message maintains a message disposition state that describes the delivery state.

- The database is often used as a logging system for message transmission status. The message metadata is updated with the disposition of the message, while the message remains in the database as an artifact.
- An optimized technique is available using Amazon CloudWatch as a record of message disposition.

Modernized queuing architecture

Decoupling message queuing from the database improves database availability and enables greater message queue scalability. It also provides a more cost-effective use of the database, and mitigates backpressure created when database performance is constrained by message management.

The modernized architecture uses loosely coupled services, such as Amazon S3, AWS Lambda, Amazon Message Queue, Amazon SQS, Amazon SNS, Amazon EventBridge, and Amazon CloudWatch.

Figure 2 depicts a message queuing architecture that uses Amazon SQS for message queuing and AWS Lambda for message routing, transformation, and disposition management. An RDBMS is still leveraged to retain metadata profiles, routing logic, and message disposition. The ETL processes are handled by AWS Lambda, while large objects are stored in Amazon S3. Finally, message fan-out distribution is handled by Amazon SNS, and the queue state is monitored and managed by Amazon CloudWatch and Amazon EventBridge.

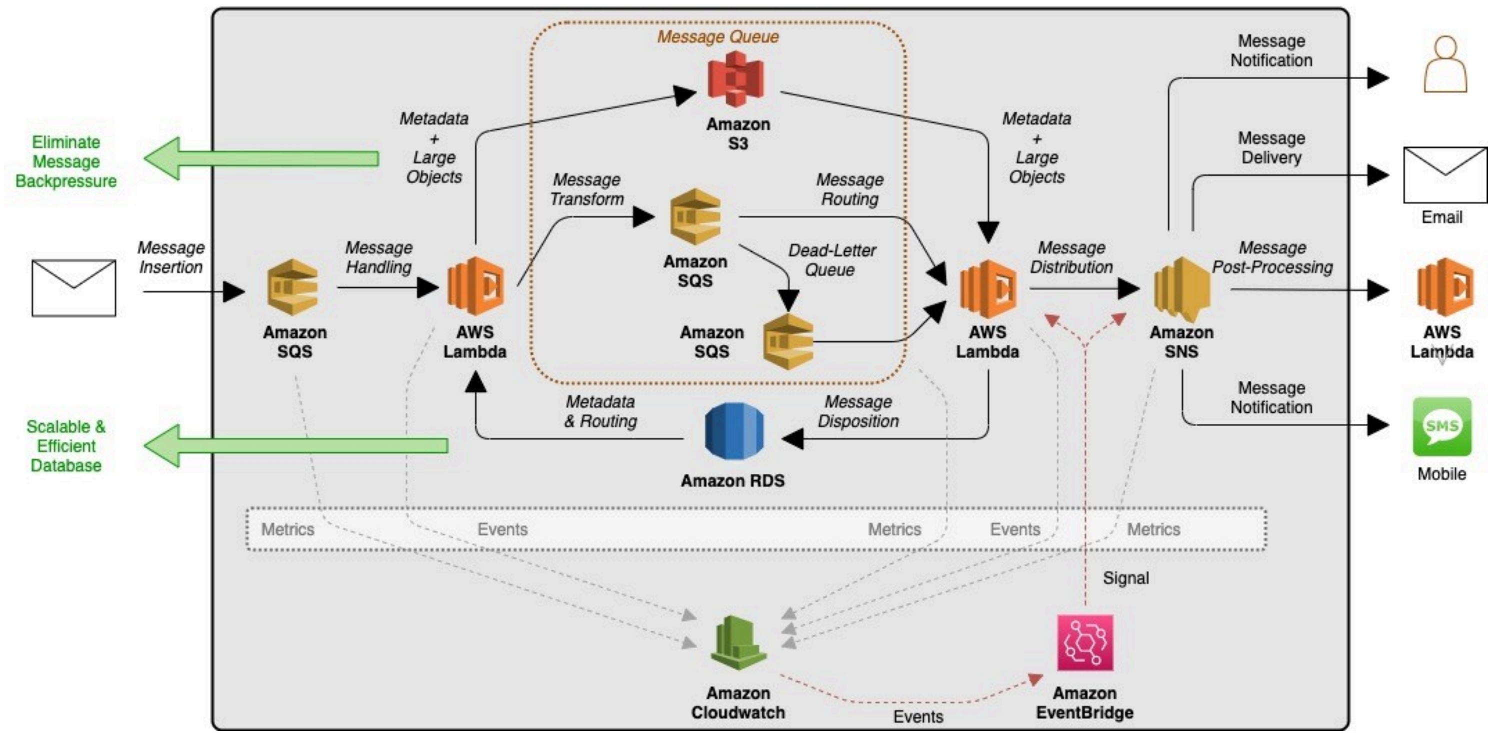

Figure 2. Modernized queuing architecture using Amazon SQS

The message fan-out distribution will help with use cases where the transformed data needs to be shared with another microservice application, as shown in Figure 3.

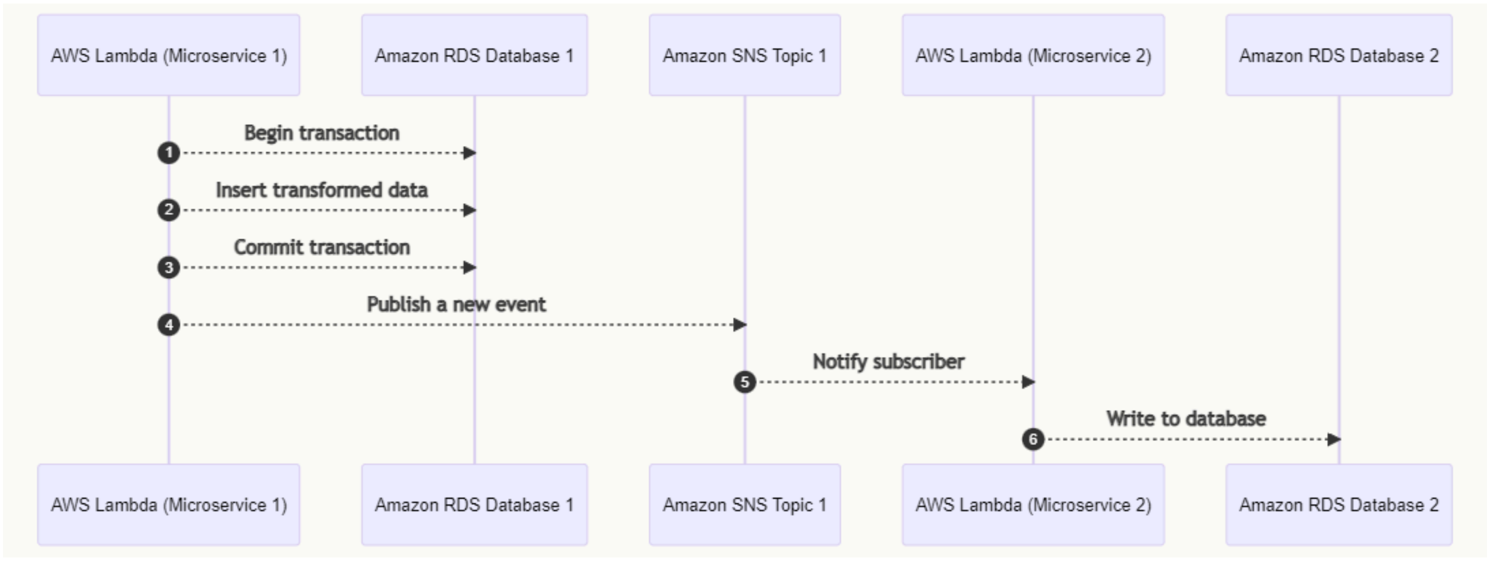

Figure 3. A common approach to sharing data, where it is possible for Step 3 to succeed, but Step 4 to fail. This is due to a network error resulting in data inconsistency.

A well-known design pattern called Transactional Outbox can be used to implement reliable messaging and ensure data consistency across multiple independent systems.

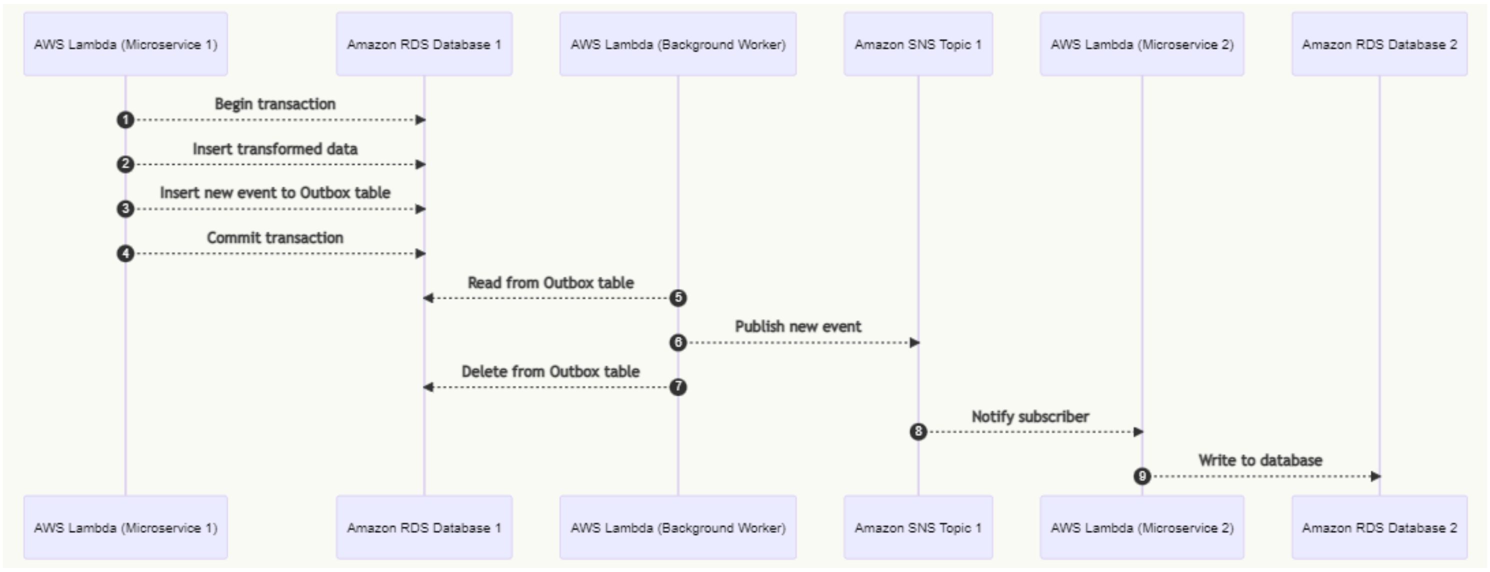

Figure 4. Ensure data consistency by implementing a Transactional Outbox design pattern

Conclusion

In this blog, we show how queuing functionality can be migrated from the RDMBS while minimizing changes to the business application. The RDBMS continues to play a central role in sourcing the message metadata, running routing logic, and storing message disposition. However, AWS services such as Amazon SQS offload queue management tasks related to the messages. AWS Lambda performs message transformation, queues the message, and transmits the message with massive scale, fault-tolerance, and efficient message distribution.

Read more about the diverse capabilities of AWS messaging services:

- Amazon Simple Queue Service

- Understanding how AWS Lambda scales with Amazon SQS standard queue
- Amazon MQ

By using AWS services, the RDBMS is no longer a performance bottleneck in your business applications. This improves scalability, and provides resilient, fault-tolerant, and efficient message delivery.

Read our blog on modernization of common database functions:

- Migrating a Database Workflow to Modernized AWS Workflow Services