What is a Dead-Letter Queue (DLQ)?

What is a dead-letter queue?

A dead-letter queue (DLQ) is a special type of message queue that temporarily stores messages that a software system cannot process due to errors. Message queues are software components that support asynchronous communication in a distributed system. They let you send messages between software services at any volume and don't require the message receiver to always be available. A dead-letter queue specifically stores erroneous messages that have no destination or which can't be processed by the intended receiver.

Why are dead-letter queues important?

Dead-letter queues (DLQ) exist alongside regular message queues. They act as temporary storage for erroneous and failed messages. DLQs prevent the source queue from overflowing with unprocessed messages.

For example, consider a software that has a regular message queue and a DLQ. The software uses the regular queue to hold messages it plans to send to a destination. If the receiver fails to respond or process the sent messages, the software moves them to the dead-letter queue.

There are two potential causes when messages move to the DLQ pipeline: erroneous message content and changes in the receiver's system.

Erroneous message content

A message will move to a DLQ if the transmitted message is erroneous. Hardware, software, and network conditions might corrupt the sent data. For example, hardware interference slightly changes some of the information during transmission. The unexpected data corruption could cause the receiver to reject or ignore the message.

Changes in the receiver's system

A message might also move to a DLQ if the receiving software has gone through changes that the sender is not aware of. For example, you could attempt to update a customer's information by sending a message for *CUST_ID_005*. However, the receiver could fail to process the incoming message because it's removed the customer from the system's database.

What are the benefits of a dead-letter queue?

Next, we talk about the benefits of dead-letter queues (DLQ).

Reduced communication costs

Regular or standard message queues keep processing messages until the retention period expires. This helps ensure continuous message processing and minimizes the chances of your queue being blocked.

However, if your system processes thousands of messages, a large number of error messages will increase communication overhead costs and burden the communication system. Instead of trying to process failing messages until they expire, it's better to move them to a dead-letter queue after a few processing attempts.

Improved troubleshooting

If you move erroneous messages to the DLQ, this lets your developers focus on identifying the causes of the errors. They can investigate why the receiver couldn't process the messages, apply the fixes, and perform new attempts to deliver the messages.

For example, a banking software might send thousands of credit card applications daily to its backend system for approval. From there, the backend system receives the applications but cannot process all of them because of incomplete information. Instead of making endless attempts, the software moves the messages to the DLQ until the IT team resolves the problem. This allows the system to process and deliver the remaining messages without performance issues.

When should you use a dead-letter queue?

You can use a dead-letter queue (DLQ) if your system has the following issues.

Unordered queues

You can take advantage of DLQs when your applications don’t depend on ordering. While DLQs help you troubleshoot incorrect message transmission operations, you should continue to monitor your queues and resend failed messages.

FIFO queues

Message ordering is important in first-in, first-out (FIFO) queues. Every message must be processed before delivering the next message. You can use dead-letter queues with FIFO queues, but your DLQ implementation should be FIFO as well.

When should you not use a dead-letter queue?

You shouldn’t use a dead-letter queue (DLQ) with unordered queues when you want to be able to keep retrying the transmission of a message indefinitely. For example, don’t use a dead-letter queue if your program must wait for a dependent process to become active or available.

Similarly, you shouldn’t use a dead-letter queue with a first-in, first-out (FIFO) queue if you don’t want to break the exact order of messages or operations. For example, don’t use a dead-letter queue with instructions in an edit decision list (EDL) for a video editing suite. In this instance, by changing the order of edits, you change the context of subsequent edits.

How does a dead-letter queue work?

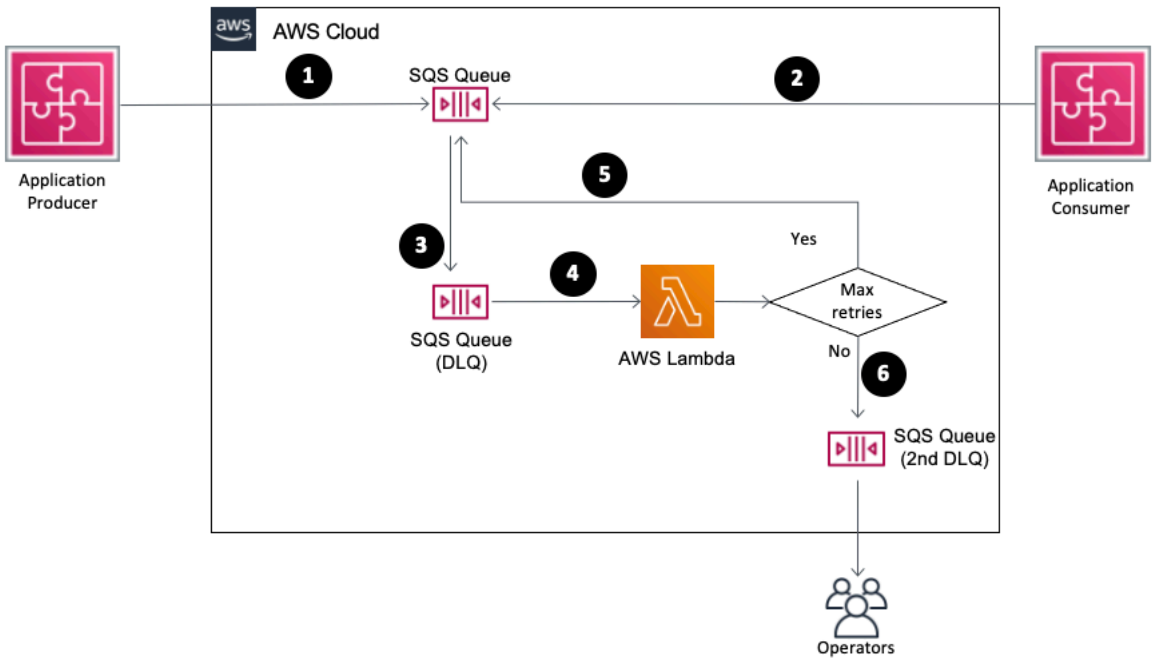
For the most part, a dead-letter queue (DLQ) works like a regular message queue. It stores erroneous messages until you process them to investigate the reason for the error.

Next, we discuss the redrive policy for DLQs and how messages move in and out of DLQs.

Creating a redrive policy

Software moves messages to a dead-letter queue by referring to the redrive policy. The redrive policy consists of rules that determine when the software should move messages into the dead-letter queue. Mainly by defining the maximum retry count, the redrive policy regulates how the source queue and dead-letter queue interact with each other.

For example, if your developer sets the maximum retry count to one, the system moves all unsuccessful deliveries to the DLQ after a single attempt. Some failed deliveries may be caused by temporary network overload or software issues. This sends many undelivered messages to the DLQ. To get the right balance, developers optimize the maximum retry count to ensure the software performs enough retries before moving messages to the DLQ.



Moving messages into the dead-letter queue

Delivery attempts between the sender and receiver can fail for several reasons:

- The receiver fails to receive the message because it doesn't exist.
- The message contains errors.
- The message exceeds the queue or message length limits. For example, some receivers can't process messages that exceed a specific size.
- The message's *time to live* (TTL) has expired. TTL is a value that indicates how long a particular data packet is valid on the network.

Moving messages out of the dead-letter queue

When messages move into the dead-letter queue, developers inspect the erroneous messages to determine the causes. Messages in the DLQ might contain valuable insights to prevent future recurrences of similar issues. After developers analyze and remediate the issues, the system moves the messages out of the DLQ and into the source queue. This allows the sender to continue processing the messages.

How can AWS support your dead-letter queue requirements?

Amazon Simple Queue Service (Amazon SQS) provides a scalable approach for exchanging messages between distributed systems at scale. Developers use Amazon SQS to build reliable web applications with fully managed standard queues and first-in, first-out (FIFO) queues.

Here are other benefits of Amazon SQS:

- Amazon SQS allows systems to create an unlimited number of message queues
- Developers can transfer messages in batches to achieve cost efficiency
- Amazon SQS supports message locking, preventing multiple computers from simultaneously processing the same message

Get started with Amazon Web Services (AWS) by creating an AWS account today.

Next Steps on AWS

Learn

What Is AWS?

What Is Cloud Computing?

What Is Agentic AI?

Cloud Computing Concepts Hub

AWS Cloud Security

What's New

Blogs

Press Releases

Resources

Getting Started

Training

AWS Trust Center

AWS Solutions Library

Architecture Center

Product and Technical FAQs

Analyst Reports

AWS Partners

Developers

Builder Center

SDKs & Tools

.NET on AWS

Python on AWS

Java on AWS

PHP on AWS

JavaScript on AWS

Help

Contact Us

File a Support Ticket

AWS re:Post

Knowledge Center

AWS Support Overview

Get Expert Help

AWS Accessibility

Legal

Back to top ↑