APPLICATION INTEGRATION
IN
AWS

Decoupling

Decoupling in the context of cloud architecture usually means designing the different parts of an application such that they can function independently of each other. Decoupling is so ubiquitous in the modern tech environment that its actually hard even to imagine an example of a software architecture with different parts of the application tightly knit and dependent on one another.

Lets say for example that we were designing a pedometer application which counts the number of steps each day and then displays it on a calendar even if the device is offline. Then, it might benefit us to package the database and front-end application together within the same distribution (.apk, .exe file) such that these two parts of the application are shipped together and run together, without any middlemen. A mock UI of said app is given below:

This however means that the application and the database code are coupled together, and that if an issue were to happen with say, the database, then the front-end application would go down as well and vice versa since separation between the two parts is non-existent.

Now, it is not hard to see how this could be a problem in more modern software paradigms which rely on quite complex architectures with many different moving parts. After all, risking the operation of the entire application every time a small component within it faces issues.

Decoupling is extremely important within an AWS environment in particular, as we often rely on a wide array of offerings, often belonging to different category of services and serving unique purposes and during such situations it becomes almost a necessity to ensure that issues in one service does not lead to the failure of the entire application.

This also allows us to manage and scale AWS services independently of one another, enhancing the flexibility as well as the reliability of our architecture. Now, there are three major offerings provided by AWS that enable us to efficiently decouple our AWS environment. These are as follows: **Simple Queue Service (SQS)**, **Simple Notification Service (SNS)** and **EventBridge**. All three will be discussed in the next section.

SQS, SNS and EventBridge

Let us take a closer look at the three aforementioned services used for the purposes of decoupling:

Simple Queue Service (SQS): SQS is designed for reliable, one-to-one asynchronous communication, where messages are held temporarily in a queue until they are processed. It is ideal for distributed systems that require message decoupling. SQS excels in situations where we need a temporary message holding pool and ordered message processing, especially when the consumer may not be available to process messages immediately. It is primarily used in cases where we want guaranteed message delivery whilst being able to tolerate a little flexibility in processing speed.

Simple Notification Service (SNS): SNS on the other hand, is a fully managed pub/sub (publish/subscribe) service allowing a one-to-many communication model. It is suited for scenarios where a single message needs to be sent to multiple subscribers in parallel, such as in fan-out patterns. It supports high throughput and can handle many subscribers. SNS is often used when we wish to decouple different parts of our system by broadcasting a message to various services at once.

Amazon EventBridge: A relatively new service, EventBridge is used when we wish to connect AWS services or integrate with third-party SaaS applications in a scalable manner. While it also supports one-to-many communication similar to SNS, it has more limitations than SNS in terms of throughput and distribution, but excels in the ability to conveniently integrate AWS services with both external and internal SaaS products.

An infographic comparing and contrasting the three different services is given below:

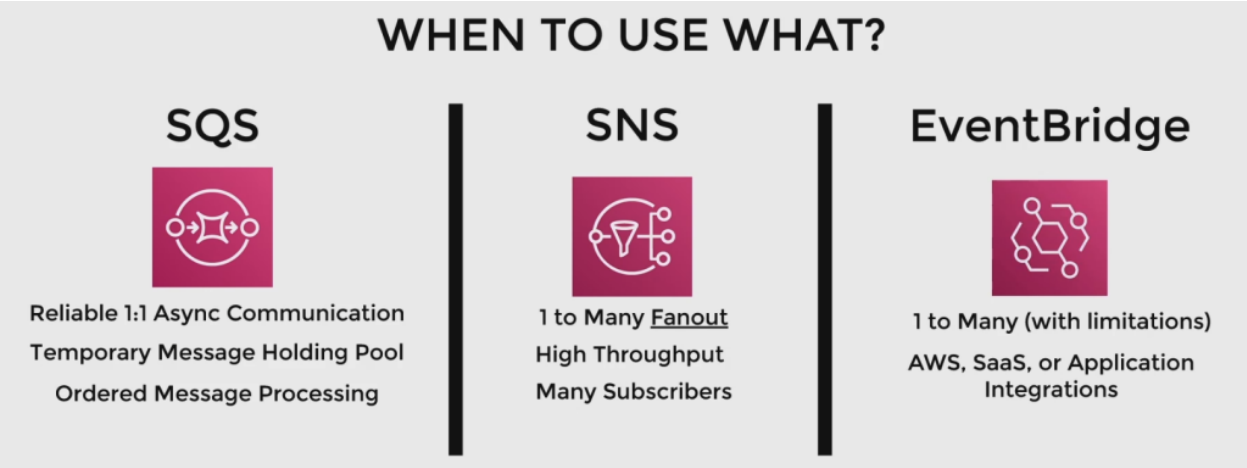


Image credit: Be a Better Dev; (Do check out his YouTube Channel)

SQS Size-based Scaling and other features

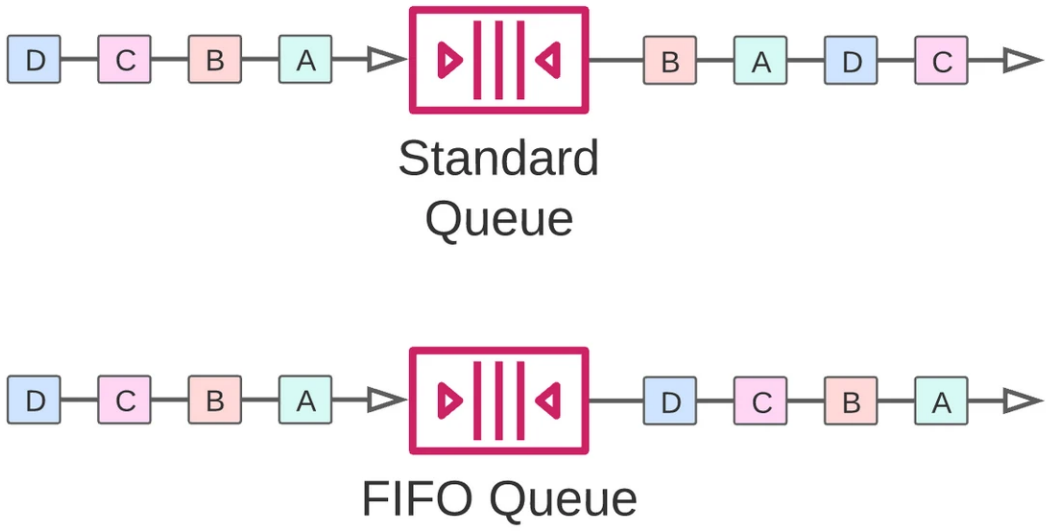
Though we gave a brief description of SQS in the last section, the service actually has many interesting sub-features and use cases which makes it a much more versatile service than was described in the last section. Though a full exploration of all the sub-features and use cases is beyond the scope of this book, there are two that I would like to point out: **SQS based scaling**, and **SQS-triggered functions**. A concise description of both is given below:

SQS based scaling: There might be cases where we wish to scale the resources within an Autoscaling Group based on the amount of orders or requests being received. This can easily be done using SQS, as AWS allows us to scale Autoscaling Groups based on the size (i.e. number of processes) of the queue, increasing or decreasing compute capacity depending on the number of orders in a queue, for example.

SQS-triggered functions: SQS can also act as an event source for AWS Lambda functions. When messages arrive in the queue, they can trigger Lambda functions to process these events, making SQS an effective mechanism for building event-driven, serverless workflows.

SQS FIFO

Amazon SQS FIFO (First-In-First-Out) queues are designed to provide strict message ordering and ensure exactly-once processing. This means that messages are processed precisely in the sequence they were sent and are never duplicated, making FIFO queues particularly suitable for scenarios where message order is critical. This is in contrast to SQS Standard, where orders are processed at random and/or in an order not decided upon or controlled by us. An image comparing how the two SQS Queue types process requests differently is given below:



Source: FourTheorem

By maintaining this strict ordering, SQS FIFO queues are ideal for use cases such as financial transactions, e-commerce order processing, or complex workflow management systems—where processing messages out of order or more than once could lead to errors, inconsistencies, or significant operational risks.

SQS Duplication

Sometimes when utilizing SQS it is possible to face the problem of messages being repeatedly processed, a phenomena termed as **SQS Duplication**. The two major reasons why SQS duplication happens are as follows: **Visibility** and **Deletion**.

Let us deal with visibility first. Say that there is an SQS queue and two instances that consume from it. If both instances consume from the same SQS queue at the same time, then the same item in the queue will be processed twice. In order to prevent this, SQS provides us with an option called the visibility timeout, the time period during which a message is invisible to other services which consume from the same SQS queue. If however, the consumer doesn't delete the message within this timeout (perhaps due to a processing delay or failure), the message becomes visible again, leading to its reprocessing by the same or another consumer.

The second reason why SQS duplication may occur be due to a failure in deleting items in the SQS queue once it has been used. Note that SQS does not automatically delete the message. Instead, the consumer must explicitly delete the message using the `DeleteMessage` action after it has been successfully processed.

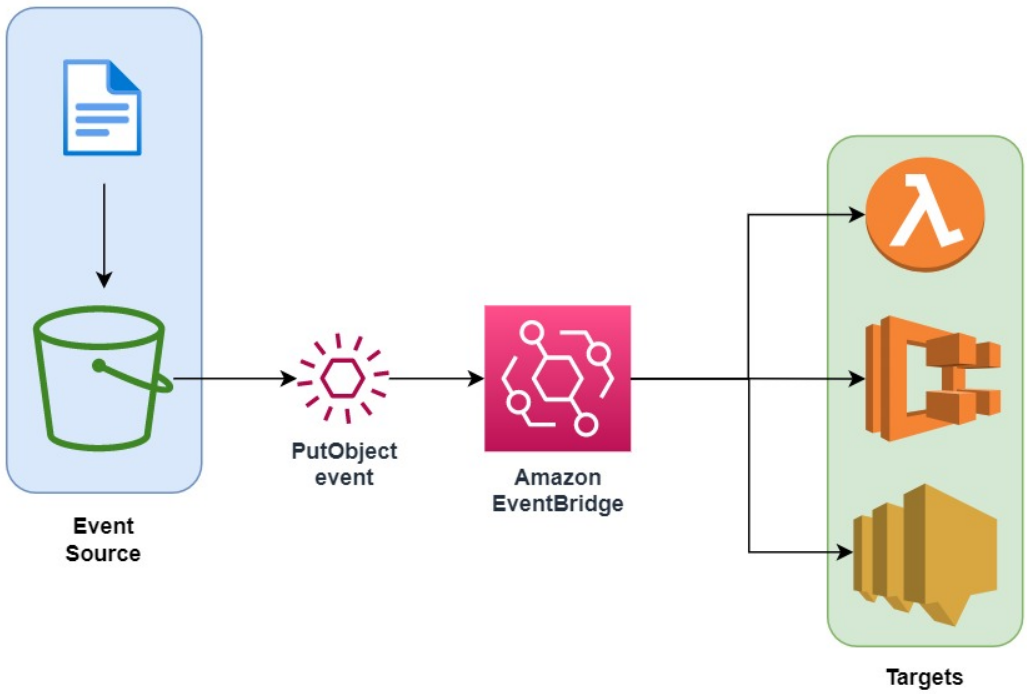
To avoid this duplication problem however is relatively easy and can be easily mitigated by:

- Setting the visibility timeout appropriately, most likely using the `ChangeMessageVisibilty` API call, allowing enough time for message processing.
- Ensuring that the services which consume the messages from the SQS queue successfully delete a message from the queue after reading it.

Amazon EventBridge

One of three aforementioned services used for the decoupling of architectures, **Amazon EventBridge (formerly called Amazon CloudWatch Events)** is a serverless event bus that makes it easy to connect applications together. It uses data from your own applications, integrated software as a service (SaaS) applications, and AWS services.

This simplifies the process of building event-driven architectures by decoupling event producers from event consumers. This allows producers and consumers to be scaled, updated, and deployed independently. Loose coupling improves developer agility in addition to application resiliency.



Source: AWS

Further, something worth noting is that we can use Amazon EventBridge (Amazon CloudWatch Events) to run Amazon ECS tasks when certain AWS events occur. This can allow us to easily integrate AWS events with tasks defined in AWS Lambda, ECS and other targeted serves.

An architecture diagram showing how EventBridge rules can be used to run an Amazon ECS task whenever a file is uploaded to a certain Amazon S3 bucket using the Amazon S3 PUT operation has been provided for the easy understanding of the reader.

Simple Email Service (SES)

Businesses since the dawn of the internet age have become increasingly reliant on emails for outreach to customers and other stakeholders. Now this has led to a great deal of services that manage the formatting, sending and monitoring of emails often called **Email Communication APIs** and broadly categorized under the more broader term of **CPaaS (Communication Platforms as a Service)**.

Simple Email Service or just SES for short, is one such Email Communication API made available and managed by AWS. SES allows us to send, process and receive emails just like most other communication APIs but its ability to take advantage of AWS vast hardware infrastructure and easy integration with other AWS services has led to multiple unique use cases and standout features, some of which are listed below:

- **Event-Driven Email Sending:** SES can be integrated with Amazon EventBridge to trigger emails based on events in your application. This is useful for automating responses to specific actions like order status changes, user sign-ups, or system alerts. For example, when a user's subscription is about to expire, SES can send an automated reminder email driven by a scheduled event or condition.

- **Transactional Email Optimization:** SES enables us to personalize transactional emails (like order confirmations or password resets) at scale by integrating with Lambda and other AWS services for dynamic content generation. For example, SES can be combined with Lambda to customize an email with personalized data such as user-specific discounts or usage reports.

- **Personalized Bulk Emails:** SES can send bulk emails while personalizing each email's content. This is especially useful when sending tailored marketing campaigns where each recipient receives customized content. For example, using configuration sets, we can track individual user interactions (opens, clicks) in a bulk email campaign and take specific actions for users who haven't engaged, such as follow-up reminders.

- **Email Receiving and Processing:** Once again, SES is not just for sending emails—it can also receive and process incoming emails. We can also configure SES to automatically trigger workflows in response to received emails, such as forwarding, archiving, or triggering Lambda functions. For example, if we are running a customer support service, SES can route incoming customer inquiries to specific teams based on say, specific keywords in the email content or automatically trigger follow-up workflows.

These capabilities make the service something beyond just simple email delivery and make SES a highly adaptable tool for any business that need to manage email communications effectively especially if its already familiar with leveraging the other AWS services.

Infographic on Amazon SES; Source: AWS

Kinesis

The **Kinesis** family provides us with Amazon's set of offerings related to real-time data streaming, allowing us to collect, process, and analyze data continuously at any scale. It's ideal for applications that require real-time analytics, such as monitoring logs, processing IoT data, and running machine learning models. The Kinesis services enable us to ingest large streams of data and process it with quite low latency, providing immediate insights and actions. There are four main services within the Kinesis family, which are shown below:

Amazon Kinesis Data Streams (KDS): Used to build custom pipelines and applications that process or analyze streaming data, it provides us with the ability to continuously capture gigabytes of data per second from hundreds of sources like website clickstreams for example.

Amazon Kinesis Firehose: Responsible for simplifying the process of loading streaming data into data lakes, warehouses and analytics services into AWS data stores at a **near real-time** pace, it can automatically scale to match the throughput of our data streams. However, it should be mentioned that Kinesis Firehose is a more limited service compared to KDS in terms of supported destinations as Firehose mostly being used to send data to S3, Redshift or Elasticsearch.

Amazon Kinesis Video Streams: Similar to KDS but used specifically for securely streaming media streams (like video or audio) from connected devices to AWS data sources.

Amazon Kinesis Data Analytics: Sometimes also referred to as Amazon Managed Service for Apache Flink, it processes data streams in real time with SQL or Apache Flink, two popular languages for data analysis and processing.

Quicksight

Amazon QuickSight is a cloud-powered business intelligence (BI) service and reporting solution that allows us to easily create and publish interactive dashboards and visualizations, as well as share them with IAM Users and Groups. These visualizations can take the form of charts, graphs, maps, and many more.

QuickSight's integration with AWS services ensures seamless data connectivity and scalability, making it ideal for organizations leveraging data lakes to efficiently derive actionable business intelligence, providing insights into large datasets stored in the data lake.

The purpose of QuickSight is to democratize access to data analytics by enabling users of all skill levels, from data scientists to business managers, to create insightful reports and dashboards without relying heavily on IT teams. Its development was driven by the need for businesses to gain real-time insights from their data without the complex infrastructure management and costs associated with traditional BI tools. As data volumes grew, services like QuickSight were created to simplify the process of connecting, analyzing, and visualizing large datasets in a scalable and cost-effective manner within the AWS ecosystem.

Amazon AppFlow

As businesses digitized, they sought for ways to use software technology to streamline various processes and increase their efficiency at certain tasks. This is usually where **Software as a Service (SaaS)** products like Salesforce, Google Analytics and ServiceNow enter the picture, providing businesses with CRM (Customer Relationship Management), HR management and other enterprise solutions. In response to the widespread adoption of SaaS solutions, Amazon launched **AppFlow**, a fully managed integration service for the secure transfer of data between AWS services and SaaS applications like the ones mentioned before. It simplifies data transfer processes, enabling us to automate and control data flows without needing to write custom code or create data pipelines from scratch.

It is often used in conjunction with EventBridge, a service that we have already established as a helpful tool when working with third-party software. An example architecture showcasing how a Salesforce event can be used as the invocation source for a Lambda function is given below: