An introduction to event-driven architectures

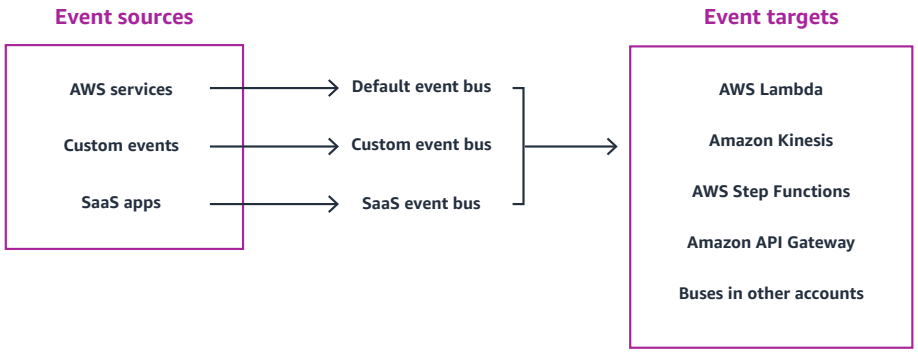
SECTION 1

Introduction

An event-driven architecture (EDA) is an architecture pattern designed to connect service components and enable complex systems to communicate. Event-driven architectures are comprised of three key components:

1. **Event producers** (e.g., software as a service (SaaS) app, mobile app, ecommerce site, point-of-sale)
2. **Event brokers** (e.g., event router, event store)
3. **Event consumers** (e.g., database, microservice, SaaS application)

Events indicate a change in state (order placed, user created) and are passed between components as messages with specific semantics. A producer publishes an event to the broker, which abstracts producers and consumers from one another by allowing them to communicate asynchronously. EDAs use events to coordinate communication between loosely coupled services.



SECTION 2

Why organizations are adopting EDA to build modern applications

An event-driven architecture enables you to provide responsive customer experiences in a scalable and reliable manner. Increasingly, organizations are using EDAs to build modern microservices applications for several reasons:

- ***Agility across teams***
  Modern applications are growing in complexity, often comprised of several custom cloud-based services, SaaS services, and on-premises services. Development teams that manage these services need to operate independently and with agility in updating, managing, and building new services.
- ***Resiliency at scale***
  Modern applications need to be resilient and reliable, with services that scale on demand and fail independently.
- ***Customer expectations***
  Modern applications need to be responsive to the needs of customers, who expect systems to respond to events as they happen.
- ***Cost-efficiency***
  Modern applications need to be cost-effective, allowing customers to pay only for the services they need when they need them, thus lowering the application's total cost of ownership (TCO).

Prior to event-driven architecture, development teams were required to coordinate closely when updating or adding new services to an application. Since changes to one service could unintentionally impact other services in the application, teams needed to understand how services outside their purview worked. This limited the ability of developers to operate independently in an agile way. Because these services were tightly coupled, each component's scalability and availability impacted the entire system, and failure in one system created failures in other systems. If there was a delay or failure in a system, the entire application could experience failures, impacting customer experiences.

Watch the video: **Getting started with event-driven architectures**

SECTION 3

Advantages of event-driven architectures

Here are the four key benefits of event-driven architecture and how it solves the limitations of the traditional request-response pattern:

1. **Increased agility**
   Event-driven architectures use event brokers to remove the need for tight coordination between developer teams that build and maintain producer and consumer services. Removing the need for close coordination between microservices lets your teams move quickly and independently. This increases developer agility and speed of feature development. Since EDAs use managed application integration services, the amount of custom code that developers are required to write to manage events and event services is reduced significantly, with the management burden passed on to cloud providers.
2. **Scalability & fault tolerance**
   Event-driven architectures are loosely coupled. This means they can scale and fail independently, which increases the resiliency of an application and becomes increasingly important as the number of integrations (microservices, SaaS, on-premises services) grows. Scaling on demand enables EDAs to handle unpredictable traffic, adding flexibility to the overall application architecture while removing single points of failure.
3. **On-demand resources**
   Event-driven architectures are push-based, meaning that everything happens on demand as the event is sent to the broker and downstream systems. This removes the need for telling dependent services about an event. Customers do not need to pay for continuous polling infrastructure, and resources can scale up and down with event volume, which reduces costs—a factor in lowering the TCO for applications that use EDAs.
4. **Real-time responsiveness**
   Event-driven architectures allow applications to respond to events in real time. Because services are responding to an event, delays or failures do not impact how or if other services respond. This ensures that customers have a better experience regardless of what is happening with other services.

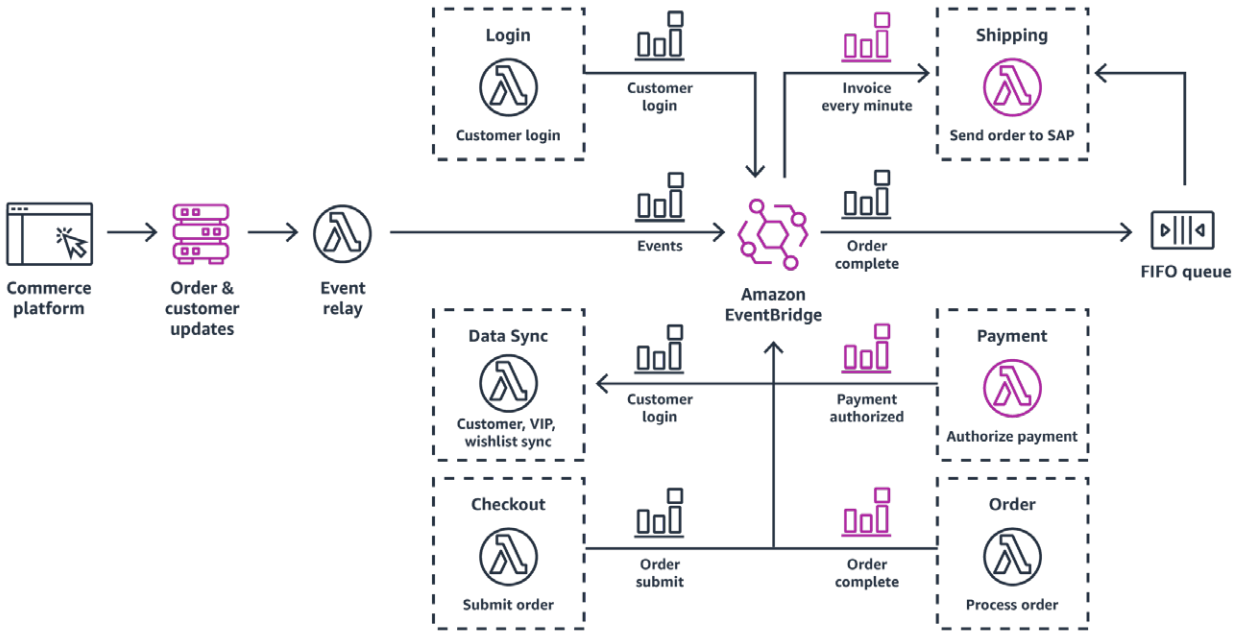
Watch the video: **Benefits of migrating to event-driven architecture**

SECTION 4

Common use cases of event-driven architectures

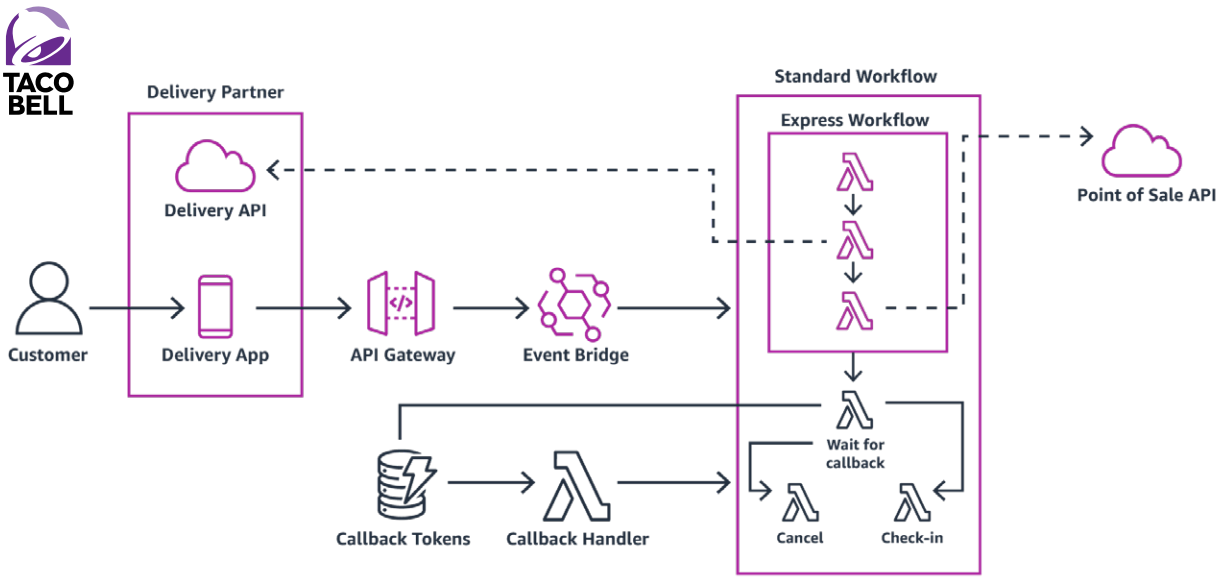
Microservices communication for web and mobile backends

This use case is often seen in retail or media and entertainment websites that need to scale up to handle unpredictable traffic. A customer visits an ecommerce website and places an order. The order event is sent to an event router, and all the downstream microservices can pick up the order event for processing—for example, submitting the order, authorizing payment, and sending the order details to a shipping provider. Because each microservice can scale and fail independently, there are no single points of failure. This pattern has helped **LEGO** scale its ecommerce website to meet peak Black Friday traffic.



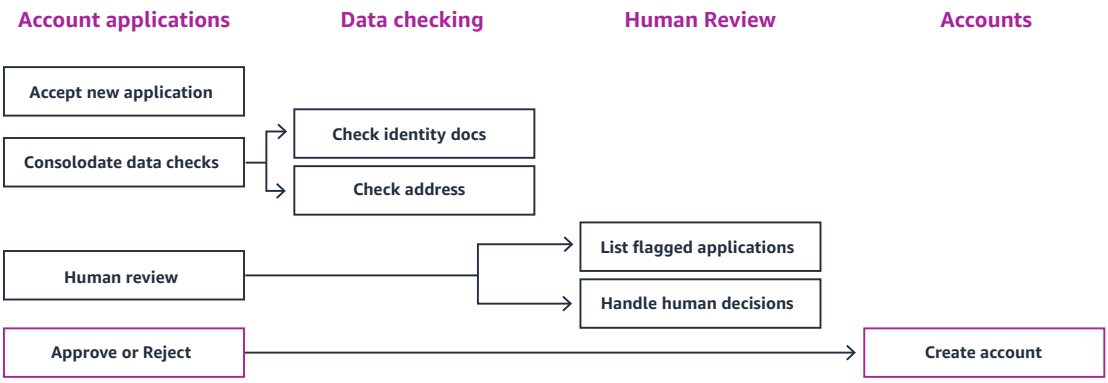
SaaS application integration

According to **BetterCloud**, organizations used on average 110 SaaS applications in 2021. Over half of respondents say the top challenge of their SaaS environment is a lack of visibility into user activity and data. To unlock siloed data, customers build event-driven architectures that ingest SaaS application events or send events to their SaaS applications. **Taco Bell** built an order middleware solution to ingest incoming delivery partner app orders and send them directly to their in-store point-of-sale application.

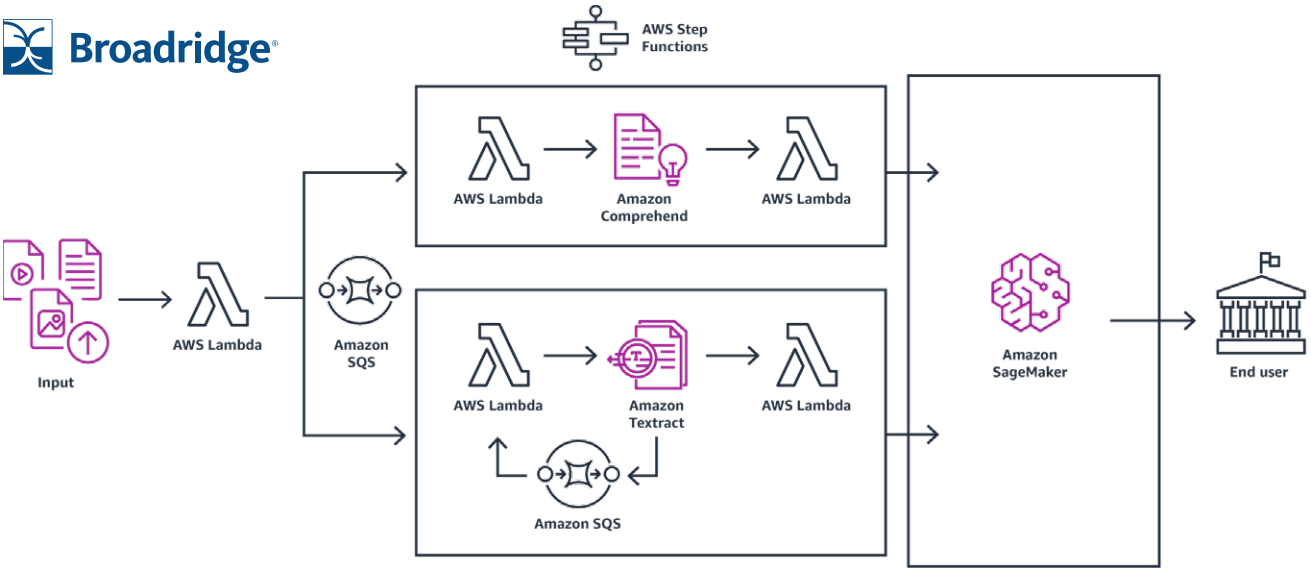


Automating business workflows

This use case is commonly seen in a financial services transaction or business process automation. Many business workflows require repeating the same steps, and those steps can be automated and executed in an event-driven way. For example, when a customer creates a new account application with a bank, the bank needs to run a few data checks (identity documentation, address, etc.), and some accounts will require a human approval stage. All these steps can be orchestrated through a workflow service, with the workflow automatically triggered anytime a new account application is submitted.



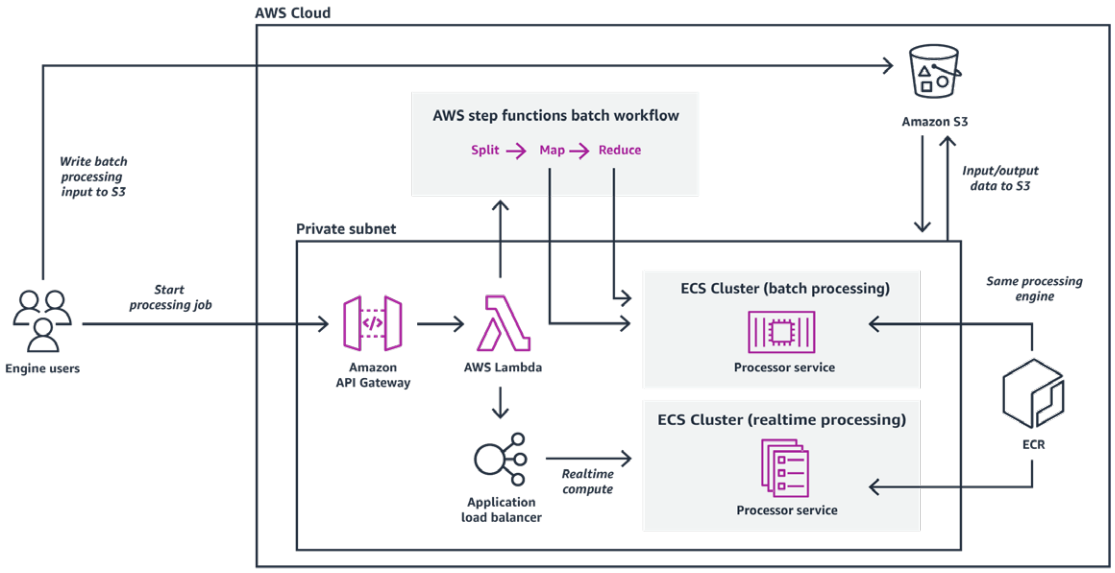
Broadridge built a workflow to extract data from proxy voting forms using machine learning, saving thousands of hours of manual data extraction and validation.



Infrastructure automation

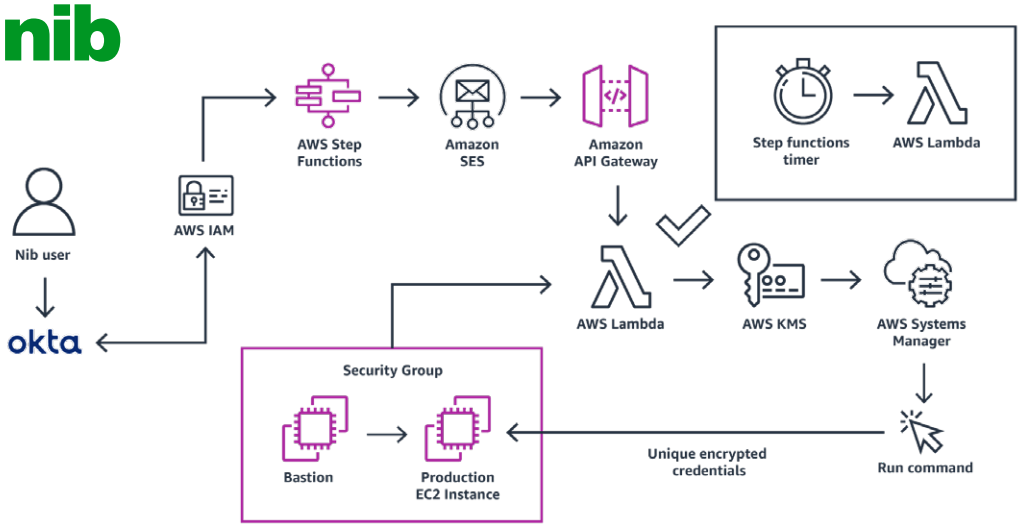
Customers can use event-driven architecture to automate their infrastructure in a variety of ways. For example, customers running compute-intensive workloads, such as financial analyses, genomic research, or media transcoding, can trigger compute resources to scale up for highly parallel processing and then scale down those resources once the job is complete. **Societe Generale** automatically scales resources up and down for credit risk analyses.

SOCIETE GENERALE



Customers in highly regulated industries (such as healthcare and finance) can use event-driven architecture to spin up security posture in response to an incident or take remediation action when a security policy sends an alert. **nib Health Funds** spins up a time-bound, auditable security posture for secure access to resources.

nib



SECTION 5

Building event-driven architectures with AWS

Event-driven architecture often combines multiple AWS services to meet your unique use case. These services include Amazon EventBridge and AWS Step Functions.

Any of the 200+ AWS services that emit events can be an event producer. Additionally, customers can ingest events from 40+ EventBridge SaaS integrations. Examples of events are a new file created in an Amazon S3 bucket, a step completed in a AWS Step Functions workflow, or a new sales opportunity is created in Salesforce.

Customers have a number of service choices for an event broker, each with different characteristics for different use cases. You can use an event bus like Amazon EventBridge, a topic from Amazon Simple Notification Service (SNS), a queue from Amazon Simple Queue Service (SQS), or a stream from Amazon Kinesis Data Streams or Amazon Managed Streaming for Apache Kafka (MSK). Often these services are combined for a customer's use case, such as using an event bus to route events to a queue for persistence and a topic for fan-out to many event consumers.

For event consumers, common patterns include invoking a Lambda function to process the event with a serverless microservice or triggering an AWS Step Functions workflow.

SECTION 6

Conclusion

Better outcomes for your business and your customers with AWS

Event-driven architectures often combine multiple benefits for a multitude of use cases. EDAs have become a key component of modern microservices-based applications. Loosely coupled applications increase developers' agility and feature release velocity. EDAs enable you to build scalable, fault tolerant applications quickly and delight your customers. Increase your development agility, save costs, and improve customer experiences with an event-driven architecture built on Amazon Web Services (AWS).

Learn more about AWS services for event-driven architecture

Read the whitepaper *Building event-driven architecture with AWS* to learn more about event-driven architecture patterns.