International Journal of Computer Trends and Technology
ISSN: 2231–2803 / https://doi.org/10.14445/22312803/IJCTT-V72I11P102

Volume 72 Issue 11, 8-16, November 2024

Original Article

AWS Event Driven Architecture

Gaurav Prabhakar

Senior Software Engineer, Application Architect, Tx, USA.

Corresponding Author : gauravprabhakar.engg@gmail.com

Received: 16 September 2024 Revised: 19 October 2024 Accepted: 08 November 2024 Published: 28 November 2024

Abstract *- This document describes the implementation of Event-Driven Architecture (EDA) in Amazon Web Services (AWS). EDA is important in technology regarding the machine's operation, robustness, and efficiency. Given the importance of real-time data processing and timely recommendations to users, EDA increases efficiency by allowing applications to respond to events and check information in a timely manner. This model is particularly useful for microservices and serverless computing, where independent services can generate and consume events, thus improving flexibility and control. EDA also improves fault tolerance and load distribution, allowing the system to adapt to different operating and non-operating conditions. As companies aim to deliver a consistent and personalized experience, EDA's ability to quickly adapt and continuously integrate new capabilities is critical.*

Keywords *- AWS, Event-driven, EDA, Pub-Sub, SNS, SQS.*

1. Introduction

AWS Event Driven Architecture (EDA) uses events as a trigger to help applications respond to actions or changes autonomously. While traditional architectures struggle with scalability and real-time data processing, a notable gap exists in the efficient handling of unstructured data and seamless integration of microservices. This creates a substantial gap for a more scalable and loosely coupled application. The issue comes with achieving the goals within the AWS landscape, especially for modern applications utilizing microservices and serverless computing. These challenges are addressed by EDA where applications can be designed to react to the real-world data streams in a timely and agile manner.

2. AWS Event-Driven Architecture

AWS Event-Driven Architecture (EDA), found within a modern architecture Vue, is a pattern that uses events to trigger actions and enable communication between decoupled services, especially in microservices-based applications. By using this approach, flexibility and scalability are much higher as all components can respond dynamically to state changes for example, when an item is added to the shopping cart in any e-commerce application. An event in the messaging domain is a substantial change of state or update, like a new user registration or a change in inventory stock. Events may include rich details about the change, or they may include identifiers that systems can use to look up more data.An AWS event message is structured like a JSON format with metadata and details on the event. Here's a general example of an AWS event message structure:

```
{
  "version": "0",
  "id": "unique-event-id",
  "detail-type": "event-type",
  "source": "event-source",
  "account": "account-id",
  "time": "timestamp",
  "region": "region",
  "resources": ["resource-arn"],
  "detail": {
    "key1": "value1",
    "key2": "value2"
  }
}
```

Version: The version of the structure of the event.

id: A unique identifier for the event.

Detail-type: The type of event (e.g., "EC2 Instance State-change Notification").

Source: The service that generated the event (e.g., "aws.ec2").

Account: The AWS account ID.

Time: The timestamp when the event occurred.

Region: The AWS region is the source of the event.

Resources: An array/list of resource ARNs participating in the event.

Detail: A JSON object with additional information about the event.

2.1. Key Components of Event-Driven Architecture

An event-driven architecture typically includes three primary components:

2.1.1. Event Producers

These services produce and publish events to the event router.

2.1.2. Event Routers

The router filters and routes events from the producers to the various consumers. This way, the services remain decoupled without any dependency on each other. The following are some of the standard routing strategies-

1. **Message Filtering:** Events are filtered based on specific criteria before being routed to the proper consumers.
2. **Topic-Based Routing:** Events publish to topics, and consumers subscribe to the topics they are interested in.
3. **Content-Based Routing:** Events are routed based on their content or payload.
4. **Fan-Out Routing:** One event is routed to multiple consumers simultaneously.
5. **Sequence-Based Routing:** Events are routed based on their sequence or order of arrival.

2.1.3. Event Consumers

These services receive and process events generated by producers, enabling them to react to changes in the application ecosystem.

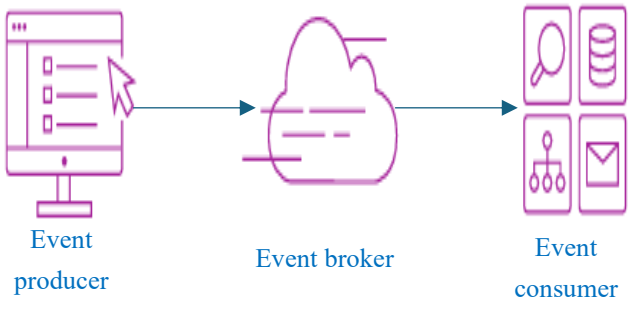

Fig. 1 Components of Event-Driven Architecture

3. Benefits of AWS Event-Driven Architecture

Using AWS Event-Driven Architecture has many advantages, including:

- **Scalability:** The services, such as AWS Lambda, can automatically be scaled with the incoming volume of events. This will let the applications handle bursts of traffic without degradation in performance.
- **Agility:** It allows developers to innovate faster because decoupled services reduce dependencies and allow independent deployments.
- **Cost Efficiency:** The pay-as-you-go pricing model reduces costs because only the time taken up in computing due to incoming events is billed.
- **Loose Coupling:** EDA decouples services; hence, it allows for scaling and updating independently. This reduces the risk of cascading failures through a system.
- **Improved Fault Tolerance:** One failing service doesn't take the others down since they are not dependent on each other.

3.1.Benefits of Event Driven Architecture (EDA) vs. Monolithic and Service Oriented Architecture (SOA)

EDA offers superior scalability, flexibility and real-time capabilities, making it ideal for modern applications,while monolithic and SOA have their own specific advantages depending on the use case.

3.1.1. EDA

- **Scalability:** Independently scalable component.
- **Flexibility:** Easy integration and replacement of components.
- **Real-Time:** Excellent for real-time data processing.
- **Decoupling:** Loosely coupled services improve fault isolation and maintenance.

3.1.2. Monolithic

- **Simplicity:** Easier initial development and deployment.
- **Performance:** Potentially better for small, simple applications.
- **Cost:** Lower initial setup costs.

3.1.3. SOA

- **Reusability:** Services can be reused across applications.
- **Integration:** Better with enterprise systems.
- **Granular Scaling:** More fine-grained than monolithic, less flexible than EDA.
- **Centralized Governance:** Simplifies compliance and management.

4. Common Use Cases of Event Driven Architecture

In general, event-driven architectures will be appropriate solutions for a wide range of needs, which may include but are not limited to the following:

- **Microservices Communication:** These enhance interservice coordination in big applications.
- **Automating Business Workflows:** These automate epetitive business processes based on events.
- **Integrating SaaS Applications:** Events connect various SaaS products, unlock data silos, and operationalize visibility.

Event-driven architecture turns out to be most useful when rapid scale-up is demanded during periods of peak demand, such as when event monitoring and alerting will be needed, thus allowing immediate responses to changes in resource states.

5. Implementing Event-Driven Architecture with AWS

5.1. AWS provides Several Services to Enable and Empower Event-Driven Architecture in Integrating and Processing the Events Seamlessly, including the following

5.1.1. Amazon EventBridge

Amazon EventBridge is a serverless event bus service that seamlessly connects application components by using events. This allows for efficient event-driven workflows since it can integrate several AWS services, SaaS applications, and even custom events.

5.1.2. Amazon Simple Notification Service (SNS)

Amazon SNS is a fully managed messaging service that allows users to publish messages to multiple subscribers in one go and is well-suited for broadcast messaging.

5.1.3. Amazon Simple Queue Service (SQS)

This is the message queuing service that enables different components to communicate in a decoupled manner, where a producer can send messages and consumers can asynchronously fetch those messages.

5.1.4. AWS Lambda

This allows running code without managing servers and computes in response to events driven by various AWS services, besides the ease of integration with EventBridge, SNS and SQS to process incoming events directly.

5.1.5. Amazon Step Functions

A serverless orchestration service that allows the coordination of multiple AWS services into workflows, enhancing the management of complex event-driven applications.

5.1.6. API Gateway

Amazon API Gateway is a fully managed service that enables developers and companies to easily create, publish, and manage APIs of applications.

5.1.7. AWS Kinesis

Amazon Kinesis Data Streams is a real-time, fully managed service provided by AWS. It ingests massive streams of data coming from thousands of sources and processes gigabytes of data per second.

The data streams seamlessly integrate with other AWS services to enable powerful data processing pipelines for applications that require real-time analytics, log and event data processing, and fraud detection. With use cases that demand instant insight, Kinesis Data Streams scales to meet demand to ensure timely, scalable and efficient data processing.

5.1.8. Amazon S3 Event Notifications

It is configurable to send notifications regarding events on an S3 bucket, such as creating or deleting an object. Events can then trigger other services like AWS Lambda.

5.1.9. Amazon DynamoDB Streams

Captures changes to DynamoDB tables, creating a time-ordered sequence of events that can trigger AWS Lambda functions for various real-time applications.

5.2. Common Patterns

This section outlines the foundational building blocks and patterns commonly encountered in event-driven architecture. These patterns allow for decoupled, asynchronous communication between producers and consumers while allowing for the unique characteristics that are increasingly necessary to fulfill various application requirements.

5.2.1. Point-to-point messaging

Messages are typically consumed by one consumer in the P2P pattern. The event broker often uses point-to-point messaging. A queue is a messaging channel that enables asynchronous communication between a sender and a receiver. Services like SQS, Amazon MQ, and Aws Lambda can commonly be used.

5.2.2. Pub/sub messaging

Whereas point-to-point messaging messages usually go to just one consumer, with publish-subscribe messaging, you have the possibility to really broadcast messages and send a copy to each consumer. The event broker in these models is frequently an event router. Unlike queues, event routers typically don’t offer the persistence of events.

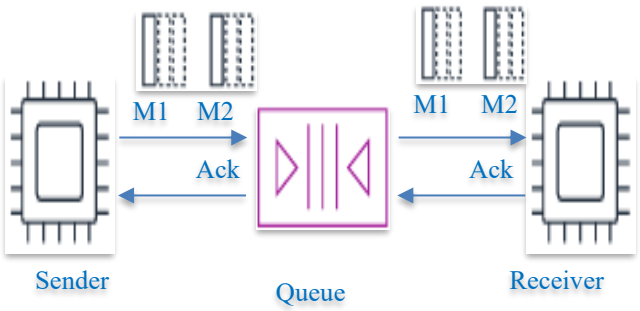

Fig. 2 Illustration of P2P Pattern

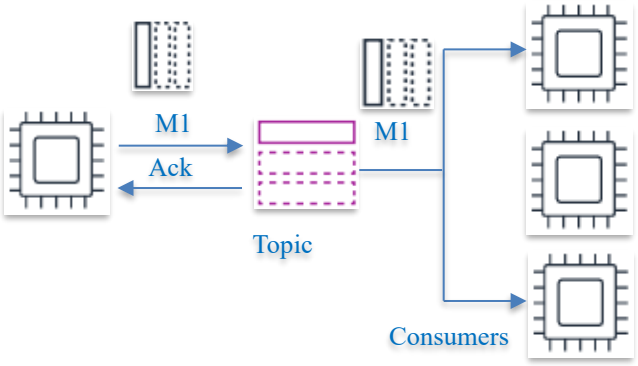

Fig. 3 Illustration of Publisher Subscriber Pattern

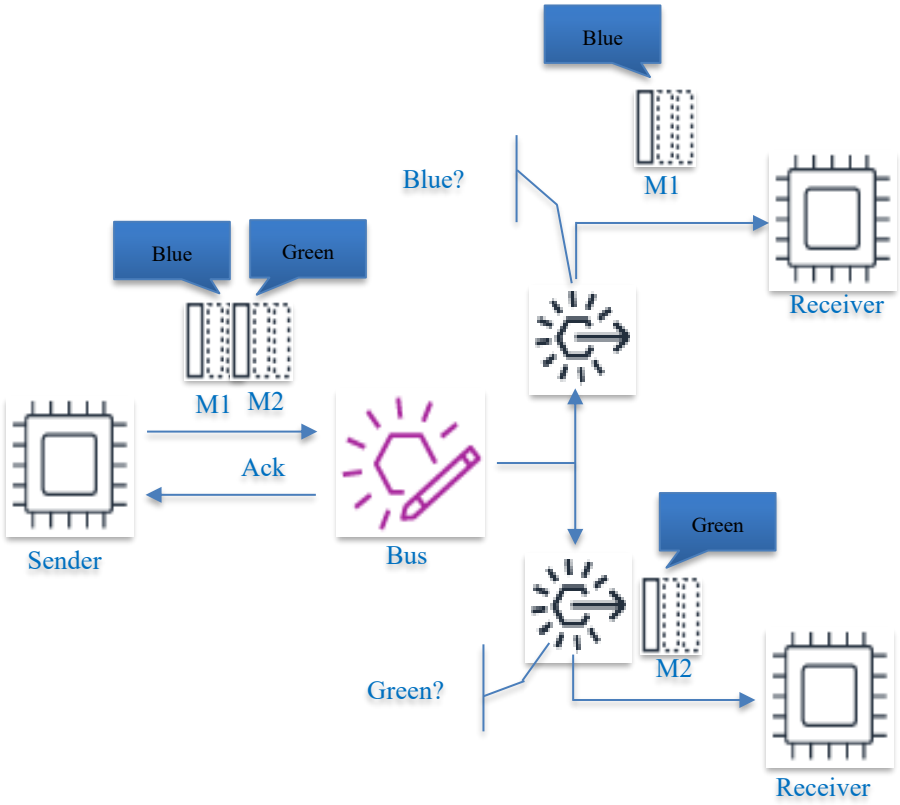

Fig. 4 Illustration of Event Bus Pattern

5.2.3.Event Bus

Yet another breed of event router is an event bus, which provides for sophisticated routing logic. Whereas topics broadcast all the messages received to subscribers, event buses can filter the incoming flow of messages and forward them to various consumers based on event attributes. Services like Amazon Simple Notification Service are used to create topics, and Amazon EventBridge is used to create event buses. Services such as Amazon EventBridge support persisting events via archive functionality. Amazon MQ also supports topics and routing.

5.2.4. Event streaming

Another abstraction for producers and consumers is through streams or continuous flows of events or data. Unlike event routers, like queues, streams almost always involve the consumer polling for new events. Consumers retain their own filtering logic to decide what events to consume while maintaining knowledge of their position in the stream. Event streams are continuous flows of events that can be processed individually or together over time. Event streaming is represented by things such as the rideshare application, which streams the changing locations of a customer as events. Each "Location Updated" event exists as a meaningful data point used to visually update a customer's location on a map. It could also analyze location events over time to provide insights, such as the driver's speed. Data streams differ from event streams because they always interpret data over time. In this model, individual data points or records aren't independently useful. Data streaming applications often persist in the data after an optional enrichment or processing of the data over a preset time to derive real-time analytics. IoT device sensor data can be considered as one such example of streaming. Individual sensor reading records may not be valuable without context, but the records collected over a period can tell a richer story. This includes Amazon Kinesis Data Streams and Amazon Managed Streaming for Apache Kafka for the event and data-streaming use cases.

5.2.5. Choreography and orchestration

There are two distinct models for how distributed services can communicate with one another, namely choreography versus orchestration. Orchestration controls communication more tightly. A central service coordinates the interaction and order in which services are invoked. Choreography achieves communication without tight control; events flow between services without centralized coordination. Many applications will use both choreography and orchestration for different use cases. Communication between bounded contexts is often how choreography is used most effectively. With choreography, producers don't expect how and when the event will be processed. Producers are only responsible for sending events to an event ingestion service and adhering to the schema. This reduces dependencies between the two bounded contexts. In a bounded context, there is often the need to control the sequence of service integrations, maintain the state, and handle errors with retries. These use cases are better suited for orchestration.

Event buses like Amazon EventBridge can help with choreography, and workflow orchestration services such as AWS Step Functions or Amazon Managed Workflows for Apache Airflow (Amazon MWAA) can help you build orchestration. Examples of how you might use choreography and orchestration together could include sending an event to trigger an AWS Step Functions workflow emitting events at different steps.

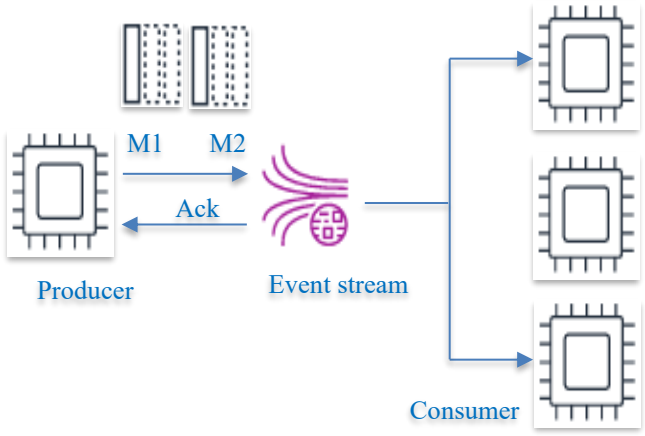

Fig. 5 Event Stream Communication

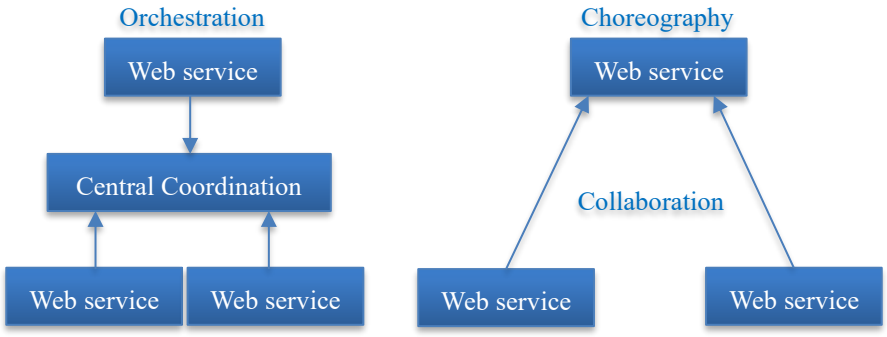

Fig. 6 Choreography and Orchestration

5.2.6. Connecting Event Sources

Most applications receive events from external sources. These might be SaaS applications, such as business applications that run payroll, store records, or handle ticketing. You can also ingest events from an existing application or database running on-premises. Events from all these sources can be used by event-driven architectures.

At the point when applications emit business events, one common way to propagate the events is with the use of a connector or message broker. Such connectors bridge SaaS applications or on-premises sources, sending these events to either a stream or router where consumers process them. Amazon EventBridge partner event sources can send events from integrated SaaS applications to your AWS applications.

Combining Patterns

While any single pattern may fulfill your needs, event-driven architectures will frequently compose a set of patterns that:

• Fan out to deliver the same message to multiple subscribers of a single topic.

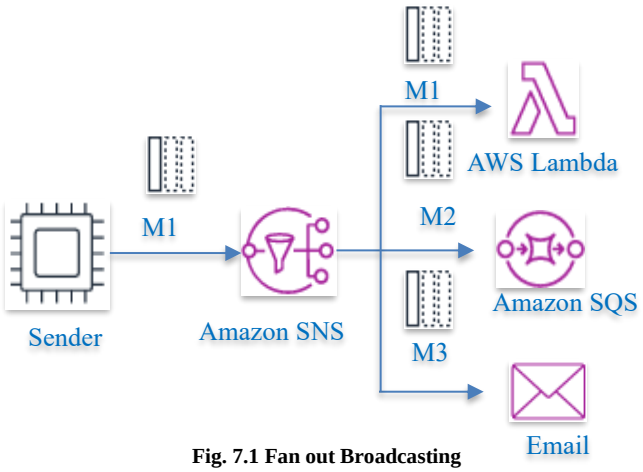

Fig. 7.1 Fan out Broadcasting

• Filter and route specific events to be forwarded to different targets.

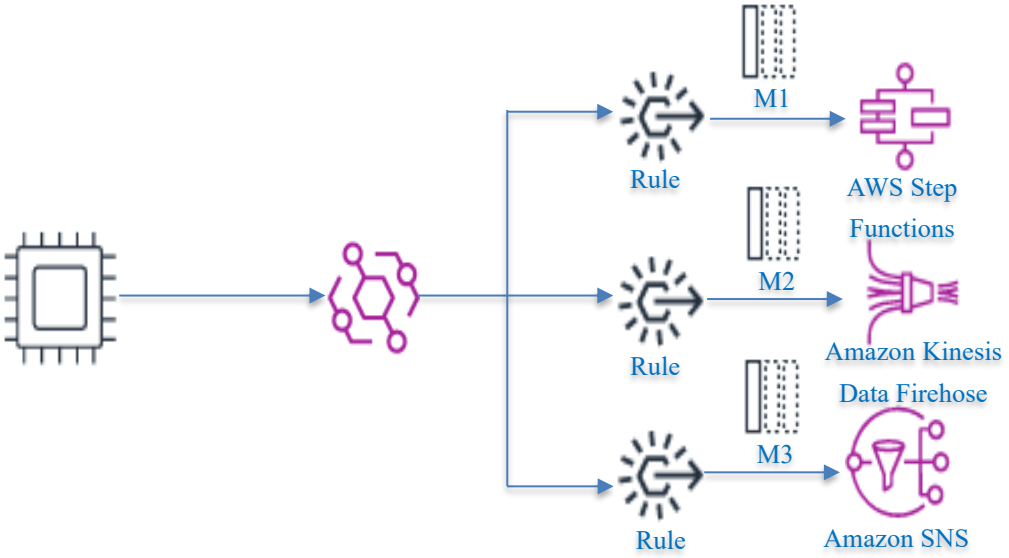

Fig. 7.2 Event-based Broadcasting

• Filter events and route them into a queue for persistence.

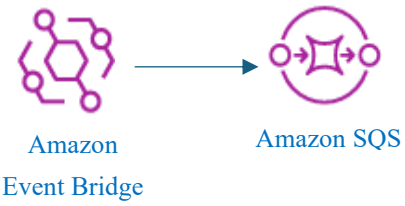

Fig. 7.3 Routing to Queue

• Orchestrate workflows and emit events at steps within the workflow.

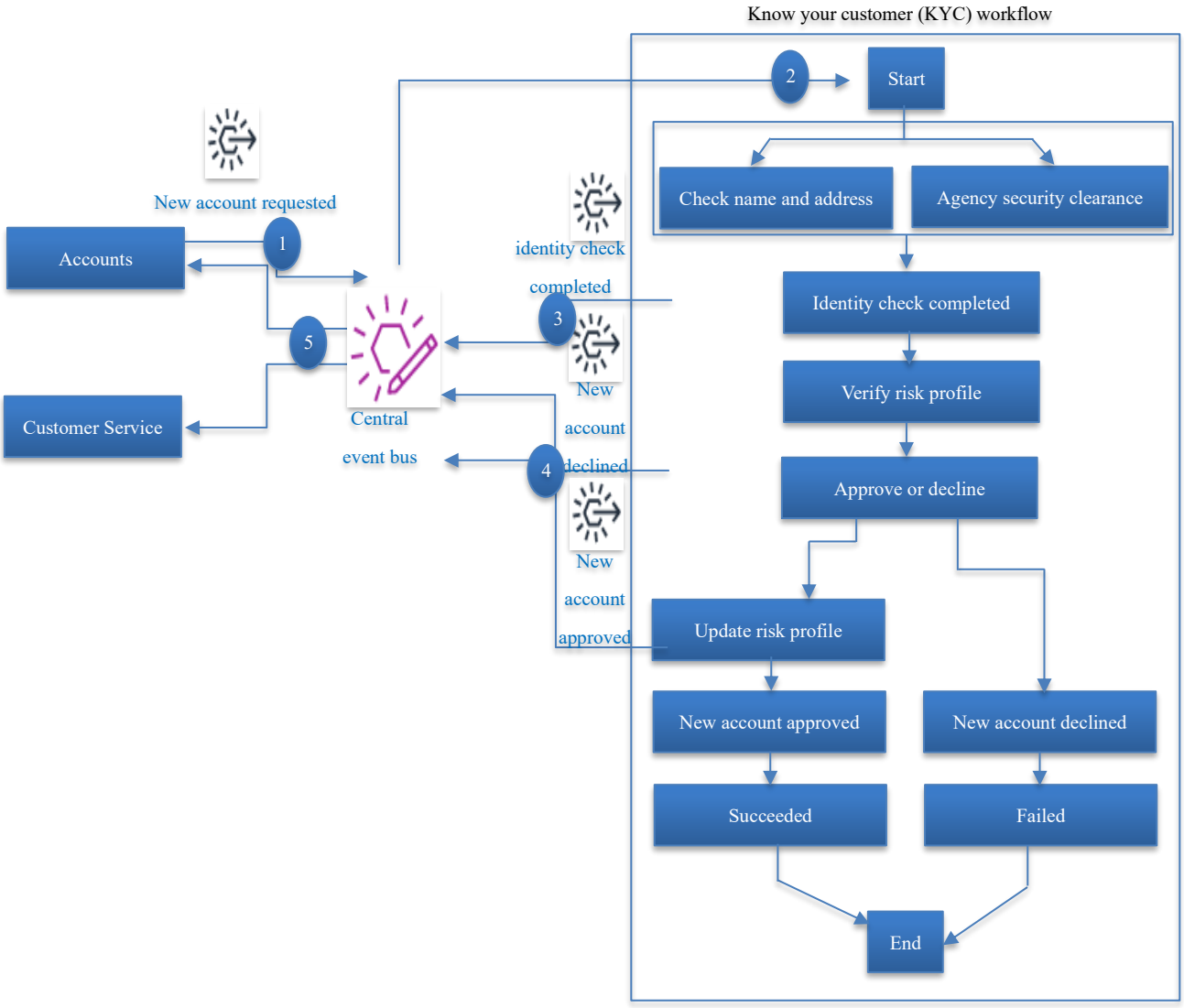

Fig. 7.4 Workflow Orchestration

5.3. Monitoring AWS Event-Driven Architecture

Measuring and monitoring the performance of AWS Event-Driven Architecture (EDA) systems is crucial for ensuring they operate efficiently and effectively. Here are some key practices and tools to consider:

5.3.1. Key Metrics

Latency: Time from event creation to processing.

Throughput: Events processed per time unit.

Error Rates: Failed event processing attempts.

Resource Utilization: CPU, memory, network usage.

Characteristics of Achmad Noe'man's Mosque Period 1964-2010 Events waiting to be processed.

5.3.2. AWS Tools

Amazon CloudWatch: Collects and tracks metrics and se ts alarms.

AWS X-Ray: Traces and analyzesrequests through services.

CloudWatch Logs: Collects and analyzes log files.

Lambda Metrics: Monitors serverless function metrics.

Best Practices

Set Up Alerts: Notifications for performance issues.

Regular Metric Reviews: Identify trends and improvement.

Use Dashboards: Visualize key metrics for a quick overview.

Implement Logging: Comprehensive event and error logs.

Optimize Resources: Ensure efficient resource allocation.

This ensures your AWS EDA systems run smoothly and efficiently.

5.4 Error Handling and Retry Mechanisms

Effective error handling and retry mechanisms are crucial for managing event failures and ensuring data integrity in an AWS Event-Driven Architecture (EDA).

5.4.1. Error Handling

1. **Dead Letter Queues (DLQs):** Capture failed messages for later review.
2. **Error Logging**: Use CloudWatch Logs to capture and a nalyze errors.
3. **Error Notifications**:Use SNS to send alerts when errors occur.

5.4.2. Retry Mechanisms

1. **Automatic Retries:** Built-in retries in services like Lambda and Step Functions.
2. **Exponential Backoff:** Gradually increase wait time bet weenretries to reduce system load.
3. **Idempotency:** Ensure multiple identical requests have t he same effect as a single request.

5.4.3. Data Integrity

1. **Transaction Management:** Use DynamoDB Transactio ns to ensure all operations succeed or fail together.
2. **Data Validation:** Validate data before processing.
3. **Versioning:** Use S3 Versioning to maintain multiple ver sions of objects.

6. Security Best Practices

Implementing robust security measures is crucial for ensuring the integrity and confidentiality of your AWS EDA. Here are some best practices to consider:

Authentication and Authorization

- Use IAM roles and policies to control access.
- Implement Multi-Factor Authentication (MFA).

Data Protection

- Encrypt data at rest and in transit.
- Regularly back up data and ensure robust recovery plan.

Network Security

- Use VPC and subnets for network isolation.
- Implement security groups and NACLs for traffic contro l.

Monitoring and Logging

- Use CloudTrail for logging AWS API calls.
- Monitor and alert with CloudWatch.

Regular Updates and Patching

- Keep systems up to date with patches.
- Conduct regular security assessments.

7. Case Study

A logistics company faced significant challenges managing millions of daily logistics events, such as transporting parcels, making deliveries, and collecting signatures. Traditional methods struggled with the volume and complexity of these events, leading to inefficiencies and high operational costs. By implementing an event-driven architecture on AWS, the company overcame these issues by building a centralized integration solution using AWS Lambda, Amazon Simple Queue Service (SQS), and Amazon EventBridge.

7.1. Problem Definition

The company's existing architecture was heavily dependent on manual interventions and a batch-processing approach, causing delays and data silos. This made it difficult to scale operations, respond to real-time events, and integrate new services swiftly.

7.2. Implementation

The company transitioned to an event-driven approach by identifying key events in its logistics process, such as parcel pickup, delivery status updates, and signature collections. They then created event producers for each event type. These events were captured and routed through Amazon EventBridge, which acted as the central hub for event processing.

AWS Lambda functions were used to handle these events in real time, performing tasks such as updating the delivery status, sending notifications to customers, and triggering further actions in other parts of the system. Amazon SQS was employed to buffer events, ensuring they were processed even if the downstream services experienced temporary failures.

7.3. Results

This transition to an event-driven architecture brought several significant improvements.

7.4. Scalability

The system could now handle peak loads effortlessly, dynamically scaling up and down based on the event volume.

7.5. Real-time Processing

Events were processed as they occurred, reducing delays and enhancing operational efficiency.

7.6. Cost Savings

By utilizing serverless services like AWS Lambda, the company reduced its infrastructure costs significantly, only paying for what they used.

7.7. Flexibility

New services and features could be added easily without disrupting the existing workflow, thanks to the decoupled nature of the architecture.

7.8. Enhanced Customer Experience

Customers received real-time delivery updates, improving satisfaction and trust in the service. The successful implementation of an event-driven architecture allowed the logistics company to improve its operations significantly, demonstrating the powerful capabilities of AWS EDA in a practical, real-world scenario.

8. Costing

Here's a table summarizing the estimated costs for a gen eral AWS event-driven architecture:

These costs are approximate and can vary based on your specific usage and configuration. For a more detailed estimat e, you can use the AWS Pricing Calculator.

9. Upcoming Challenges and Trends in Event-Driven Architecture

9.1. Challenges in EDA

- **Complexity**: Managing events and tracing their flow can be difficult.
- **Consistency**: Ensuring correct event order and system integrity.
- **Suitability**: It may be overkill for simple, non-real-time tasks.
- **Transition**: Moving to EDA requires new skills and tools.
- **Debugging Complexity**: Errors can be difficult to trace in event-driven systems due to the distributed nature of service interactions.
- **Variable Latency**: Unlike traditional systems, event-driven architectures may experience latency due to network communication, which can complicate real-time processing needs.

9.2. Trends in EDA

- **Real-Time Data:** Growing need for immediate data processing.
- **API-Driven**: APIs central to communication between services.
- **Microservices & Serverless**: Preferred for decoupling services.
- **Total Experience**: Integrating multiple experience layers to improve business outcomes.

9.3. AI/ML Integration

- **Real-Time**: AI models react to events in real time.
- **Scalability**: Modular design supports scalable AI components.
- **Flexibility**: Seamless integration of various AI models.
- **Decision Making**: Triggering AI/ML for real-time analysis.

10. Summary

Event-driven architecture facilitates the development of scalable, flexible, and decoupled applications that can respond dynamically to changes, although implementing EDA requires consideration of its inherent complexities and challenges.

Table 1. Cost Estimates

Service	Description	Estimated Cost
Amazon S3	Storage for event data and logs	$0.023 per GB-month
Amazon SQS	Message queue for decoupling components	$0.50 per million messages sent
Amazon SNS	Publish/subscribe messaging service	$0.50 per million notifications
Amazon Lambda	Serverless compute for processing events	$0.20 per million requests
Amazon DynamoDB	NoSQL database for storing event data	$0.25 per GB-month
Amazon Kinesis	Real-time datastreaming service	$1.00 per GB of data processed
Amazon EventBridge	Event routing and processing	$0.50 per million rules
Amazon CloudWatch	Monitoring and logging service	$0.10 per million metric data points

References

[1] What is an Event-Driven Architecture?, AWS Amazon. [Online]. Available: https://aws.amazon.com/event-driven-architecture/

[2] Sudarkodi Muthiah, Event-Driven Architecture Solutions on AWS, Medium. [Online]. Available: https://medium.com/@sudarkodimuthiah22/event-driven-architecture-solutions-on-aws-bc90ccc55dc8

[3] Building Event-Driven Architectures with AWS, Amazon Web Service, pp. 1-21, 2022. [Online]. Available: https://d1.awsstatic.com/SMB/building-event-driven-architectures-aws-guide-2022-smb-build-websites-and-apps-resource.pdf