AWS Whitepaper

Streaming Data Solutions on AWS

Streaming Data Solutions on AWS: AWS Whitepaper

Table of Contents

Streaming Data Solutions on AWS

Publication date: **September 1, 2021** (*Document revisions*)

Abstract

Data engineers, data analysts, and big data developers are looking to evolve their analytics from batch to real-time so their companies can learn about what their customers, applications, and products are doing right now and react promptly. This whitepaper discusses the evolution of analytics from batch to real-time. It describes how services such as Amazon Kinesis Data Streams, Amazon Data Firehose, Amazon EMR, Amazon Managed Service for Apache Flink, Amazon Managed Streaming for Apache Kafka (Amazon MSK), and other services can be used to implement real-time applications, and provides common design patterns using these services.

Introduction

Businesses today receive data at massive scale and speed due to the explosive growth of data sources that continuously generate streams of data. Whether it is log data from application servers, clickstream data from websites and mobile apps, or telemetry data from Internet of Things (IoT) devices, it all contains information that can help you learn about what your customers, applications, and products are doing right now.

Having the ability to process and analyze this data in real-time is essential to do things such as continuously monitor your applications to ensure high service uptime and personalize promotional offers and product recommendations. Real-time and near-real-time processing can also make other common use cases, such as website analytics and machine learning, more accurate and actionable by making data available to these applications in seconds or minutes instead of hours or days.

Real-time and near-real-time application scenarios

You can use streaming data services for real-time and near-real-time applications such as application monitoring, fraud detection, and live leaderboards. Real-time use cases require millisecond end-to-end latencies– from ingestion, to processing, all the way to emitting the results to target data stores and other systems. For example, Netflix uses Amazon Kinesis Data Streams to monitor the communications between all its applications so it can detect and fix issues quickly, ensuring high service uptime and availability to its customers. While the most commonly applicable use case is application performance monitoring, there are an increasing number of real-time applications in ad tech, gaming, and IoT that fall under this category.

Common near-real-time use cases include analytics on data stores for data science and machine learning (ML). You can use streaming data solutions to continuously load real-time data into your data lakes. You can then update ML models more frequently as new data becomes available, ensuring accuracy and reliability of the outputs. For example, Zillow uses Kinesis Data Streams to collect public record data and multiple listing service (MLS) listings, and then provide home buyers and sellers with the most up-to-date home value estimates in near-real-time. ZipRecruiter uses Amazon MSK for their event logging pipelines, which are critical infrastructure components that collect, store, and continually process over six billion events per day from the ZipRecruiter employment marketplace.

Difference between batch and stream processing

You need a different set of tools to collect, prepare, and process real-time streaming data than those tools that you have traditionally used for batch analytics. With traditional analytics, you gather the data, load it periodically into a database, and analyze it hours, days, or weeks later. Analyzing real-time data requires a different approach. Stream processing applications process data continuously in real-time, even before it is stored. Streaming data can come in at a blistering pace and data volumes can vary up and down at any time. Stream data processing platforms have to be able to handle the speed and variability of incoming data and process it as it arrives, often millions to hundreds of millions of events per hour.

Stream processing challenges

Processing real-time data as it arrives can enable you to make decisions much faster than is possible with traditional data analytics technologies. However, building and operating your own custom streaming data pipelines is complicated and resource-intensive:

- You have to build a system that can cost-effectively collect, prepare, and transmit data coming simultaneously from thousands of data sources.
- You need to fine-tune the storage and compute resources so that data is batched and transmitted efficiently for maximum throughput and low latency.
- You have to deploy and manage a fleet of servers to scale the system so you can handle the varying speeds of data you are going to throw at it.

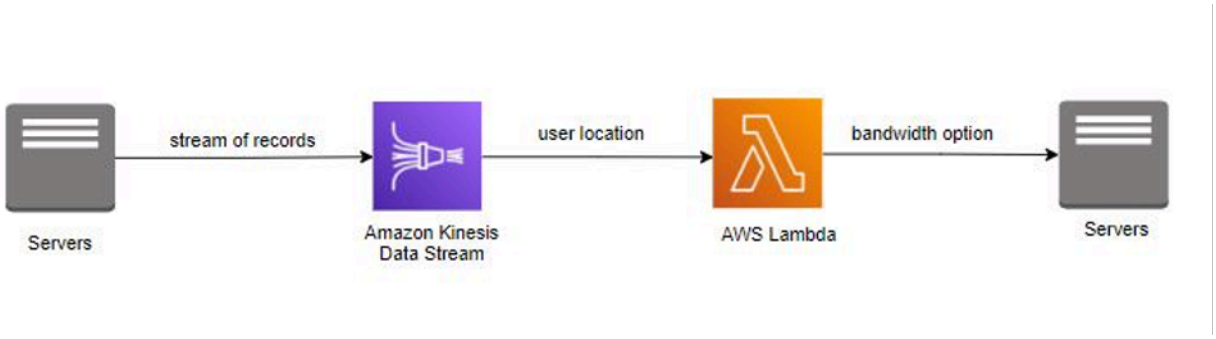
Version upgrade is a complex and costly process. After you have built this platform, you have to monitor the system and recover from any server or network failures by catching up on data processing from the appropriate point in the stream, without creating duplicate data. You also need a dedicated team for infrastructure management. All of this takes valuable time and money and, at the end of the day, most companies just never get there and must settle for the status quo and operate their business with information that is hours or days old.

Streaming data solutions: examples

To better understand how organizations are doing real-time data processing using AWS services, this whitepaper uses five examples. Each example reviews a scenario and discusses in detail how AWS real-time data streaming services are used to solve the problem.

Scenario 1: Internet offering based on location

Company *InternetProvider* provides internet services with a variety of bandwidth options to users across the world. When a user signs up for internet, company *InternetProvider* provides the user with different bandwidth options based on user's geographic location. Given these requirements, company *InternetProvider* implemented an Amazon Kinesis Data Streams to consume user details and location. The user details and location are enriched with different bandwidth options prior to publishing back to the application. AWS Lambda enables this real-time enrichment.



Processing streams of data with AWS Lambda

Amazon Kinesis Data Streams

Amazon Kinesis Data Streams enables you to build custom, real-time applications using popular stream processing frameworks and load streaming data into many different data stores. A Kinesis stream can be configured to continuously receive events from hundreds of thousands of data producers delivered from sources like website click-streams, IoT sensors, social media feeds and application logs. Within milliseconds, data is available to be read and processed by your application.

When implementing a solution with Kinesis Data Streams, you create custom data-processing applications known as Kinesis Data Streams applications. A typical Kinesis Data Streams application reads data from a Kinesis stream as data records.

Data put into Kinesis Data Streams is ensured to be highly available and elastic, and is available in milliseconds. You can continuously add various types of data such as clickstreams, application logs, and social media to a Kinesis stream from hundreds of thousands of sources. Within seconds, the data will be available for your Kinesis Applications to read and process from the stream.

Amazon Kinesis Data Streams is a fully managed streaming data service. It manages the infrastructure, storage, networking, and configuration needed to stream your data at the level of your data throughput.

Sending data to Amazon Kinesis Data Streams

There are several ways to send data to Kinesis Data Streams, providing flexibility in the designs of your solutions.

- You can write code utilizing one of the AWS SDKs that are supported by multiple popular languages.
- You can use the Amazon Kinesis Agent, a tool for sending data to Kinesis Data Streams.

The Amazon Kinesis Producer Library (KPL) simplifies the producer application development by enabling developers to achieve high write throughput to one or more Kinesis data streams.

The KPL is an easy to use, highly configurable library that you install on your hosts. It acts as an intermediary between your producer application code and the Kinesis Streams API actions. For more information about the KPL and its ability to produce events synchronously and asynchronously with code examples, refer to Writing to your Kinesis Data Streams Using the KPL

There are two different operations in the Kinesis Data Streams API that add data to a stream: `PutRecords` and `PutRecord`. The `PutRecords` operation sends multiple records to your stream per HTTP request while, `PutRecord` submits one record per HTTP request. To achieve higher throughput for most applications, use `PutRecords`.

For more information about these APIs, refer to Adding Data to a Stream. The details for each API operation can be found in the Amazon Kinesis Data Streams API Reference.

Processing data in Amazon Kinesis Data Streams

To read and process data from Kinesis streams, you need to create a consumer application. There are varied ways to create consumers for Kinesis Data Streams. Some of these approaches include

using Amazon Managed Service for Apache Flink to analyze streaming data using KCL, using AWS Lambda, AWS Glue streaming ETL jobs, and using the Kinesis Data Streams API directly.

Consumer applications for Kinesis Data Streams can be developed using the KCL, which helps you consume and process data from Kinesis Data Streams The KCL takes care of many of the complex tasks associated with distributed computing such as load balancing across multiple instances, responding to instance failures, checkpointing processed records, and reacting to resharding. The KCL enables you to focus on the writing record-processing logic. For more information on how to build your own KCL application, refer to Using the Kinesis Client Library.

You can subscribe Lambda functions to automatically read batches of records off your Kinesis stream and process them if records are detected on the stream. AWS Lambda periodically polls the stream (once per second) for new records and when it detects new records, it invokes the Lambda function passing the new records as parameters. The Lambda function is only run when new records are detected. You can map a Lambda function to a shared-throughput consumer (standard iterator)

You can build a consumer that uses a feature called enhanced fan-out when you require dedicated throughput that you do not want to contend with other consumers that are receiving data from the stream. This feature enables consumers to receive records from a stream with throughput of up to two MB of data per second per shard.

For most cases, using Managed Service for Apache Flink, KCL, AWS Glue, or AWS Lambda should be used to process data from a stream. However, if you prefer, you can create a consumer application from scratch using the Kinesis Data Streams API. The Kinesis Data Streams API provides the `GetShardIterator` and `GetRecords` methods to retrieve data from a stream.

In this pull model, your code extracts data directly from the shards of the stream. For more information about writing your own consumer application using the API, refer to Developing Custom Consumers with Shared Throughput Using the AWS SDK for Java. Details about the API can be found in the Amazon Kinesis Data Streams API Reference.

Processing streams of data with AWS Lambda

AWS Lambda enables you to run code without provisioning or managing servers. With Lambda, you can run code for virtually any type of application or backend service with zero administration. Just upload your code, and Lambda takes care of everything required to run and scale your code with high availability. You can set up your code to automatically trigger from other AWS services, or call it directly from any web or mobile app.

AWS Lambda integrates natively with Amazon Kinesis Data Streams . The polling, checkpointing, and error handling complexities are abstracted when you use this native integration. This allows the Lambda function code to focus on business logic processing.

You can map a Lambda function to a shared-throughput (standard iterator), or to a dedicated-throughput consumer with enhanced fan-out. With a standard iterator, Lambda polls each shard in your Kinesis stream for records using HTTP protocol. To minimize latency and maximize read throughput, you can create a data stream consumer with enhanced fan-out. Stream consumers in this architecture get a dedicated connection to each shard without competing with other applications reading from the same stream.Amazon Kinesis Data Streams pushes records to Lambda over HTTP/2.

By default, AWS Lambda invokes your function as soon as records are available in the stream. To buffer the records for batch scenarios, you can implement a batch window for up to five minutes at the event source. If your function returns an error, Lambda retries the batch until processing succeeds or the data expires.

Summary

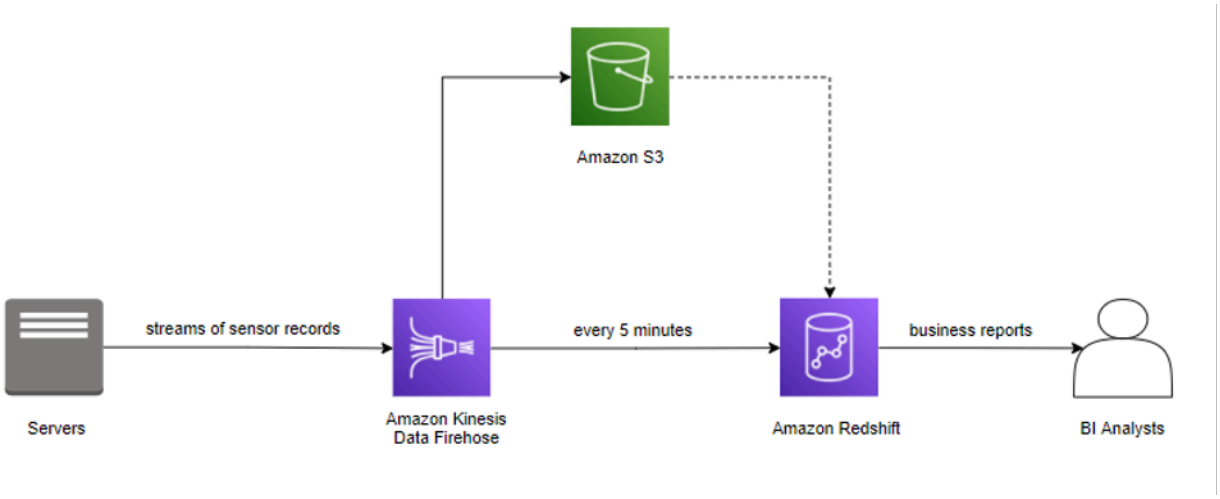
Company *InternetProvider* leveraged Amazon Kinesis Data Streams to stream user details and location. The stream of record was consumed by AWS Lambda to enrich the data with bandwidth options stored in the function's library. After the enrichment, AWS Lambda published the bandwidth options back to the application. Amazon Kinesis Data Streams and AWS Lambda handled provisioning and management of servers, enabling company *InternetProvider* to focus more on business application development.

Scenario 2: Near-real-time data for security teams

Company *ABC2Badge* provides sensors and badges for corporate or large-scale events such as AWS re:Invent. Users sign up for the event and receive unique badges that the sensors pick up across the campus. As users pass by a sensor, their anonymized information is recorded into a relational database.

In an upcoming event, due to the high volume of attendees, *ABC2Badge* has been requested by the event security team to gather data for the most concentrated areas of the campus every 15 minutes. This will give the security team enough time to react and disperse security personal proportionally to concentrated areas. Given this new requirement from the security team and the inexperience of building a streaming solution, to process date in near-real-time, *ABC2Badge* is looking for a simple yet scalable and reliable solution.

Their current data warehouse solution is Amazon Redshift. While reviewing the features of the Amazon Kinesis services, they recognized that Amazon Data Firehose can receive a stream of data records, batch the records based on buffer size and/or time interval, and insert them into Amazon Redshift. They created a Firehose delivery stream and configured it so it would copy data to their Amazon Redshift tables every five minutes. As part of this new solution, they used the Amazon Kinesis Agent on their servers. Every five minutes, Firehose loads data into Amazon Redshift, where the business intelligence (BI) team is enabled to perform its analysis and send the data to the security team every 15 minutes.



New solution using Amazon Data Firehose

Amazon Data Firehose

Amazon Data Firehose is the easiest way to load streaming data into AWS. It can capture, transform, and load streaming data into Amazon Managed Service for Apache Flink, Amazon Simple Storage Service (Amazon S3), Amazon Redshift, Amazon OpenSearch Service (OpenSearch Service), and Splunk. Additionally, Firehose can load streaming data into any custom HTTP endpoint or HTTP endpoints owned by supported third-party service providers.

Firehose enables near-real-time analytics with existing business intelligence tools and dashboards that you're already using today. It's a fully managed serverless service that automatically scales to match the throughput of your data and requires no ongoing administration. Firehose can batch, compress, and encrypt the data before loading, minimizing the amount of storage used at the destination and increasing security. It can also transform the source data using AWS Lambda and deliver the transformed data to destinations. You configure your data producers to send data to Firehose, which automatically delivers the data to the destination that you specify.

Sending data to a Firehose delivery stream

To send data to your delivery stream, there are several options. AWS offers SDKs for many popular programming languages, each of which provides APIs for Amazon Data Firehose. AWS has a utility to help send data to your delivery stream. Firehose has been integrated with other AWS services to send data directly from those services into your delivery stream.

Using Amazon Kinesis agent

Amazon Kinesis agent is a standalone software application that continuously monitors a set of log files for new data to be sent to the delivery stream. The agent automatically handles file rotation, checkpointing, retries upon failures, and emits Amazon CloudWatch metrics for monitoring and troubleshooting of the delivery stream. Additional configurations, such data pre-processing, monitoring multiple file directories, and writing to multiple delivery streams, can be applied to the agent.

The agent can be installed on Linux or Windows-based servers such as web servers, log servers, and database servers. Once the agent is installed, you simply specify the log files it will monitor and the delivery stream it will send to. The agent will durably and reliably send new data to the delivery stream.

Using API with AWS SDK and AWS services as a source

The Firehose API offers two operations for sending data to your delivery stream. `PutRecord` sends one data record within one call. `PutRecordBatch` can send multiple data records within one call, and can achieve higher throughput per producer. In each method, you must specify the name of the delivery stream and the data record, or array of data records, when using this method. For more information and sample code for the Firehose API operations, refer to Writing to a Firehose Delivery Stream Using the AWS SDK.

Firehose also runs with Firehose, CloudWatch Logs, CloudWatch Events, Amazon Simple Notification Service (Amazon SNS), Amazon API Gateway, and AWS IoT. You can scalably and reliably send your streams of data, logs, events, and IoT data directly into a Firehose destination.

Processing data before delivery to destination

In some scenarios, you might want to transform or enhance your streaming data before it is delivered to its destination. For example, data producers might send unstructured text in each data record, and you need to transform it to JSON before delivering it to OpenSearch Service. Or

you might want to convert the JSON data into a columnar file format such as Apache Parquet or Apache ORC before storing the data in Amazon S3.

Firehose has built-in data format conversion capability. With this, you can easily convert your streams of JSON data into Apache Parquet or Apache ORC file formats.

Data transformation flow

To enable streaming data transformations, Firehose uses a Lambda function that you create to transform your data. Firehose buffers incoming data to a specified buffer size for the function and then invokes the specified Lambda function asynchronously. The transformed data is sent from Lambda to Firehose, and Firehose delivers the data to the destination.

Data format conversion

You can also enable Firehose data format conversion, which will convert your stream of JSON data to Apache Parquet or Apache ORC. This feature can only convert JSON to Apache Parquet or Apache ORC. If you have data that is in CSV, you can transform that data via a Lambda function to JSON, and then apply the data format conversion.

Data delivery

As a near-real-time delivery stream, Firehose buffers incoming data. After your delivery stream's buffering thresholds have been reached, your data is delivered to the destination you've configured. There are some differences in how Firehose delivers data to each destination, which this paper reviews in the following sections.

Amazon S3

Amazon S3 is object storage with a simple web service interface to store and retrieve any amount of data from anywhere on the web. It's designed to deliver 99.999999999% durability, and scale past trillions of objects worldwide.

Data delivery to Amazon S3

For data delivery to Amazon S3, Firehose concatenates multiple incoming records based on the buffering configuration of your delivery stream, and then delivers them to Amazon S3 as an S3 object. The frequency of data delivery to S3 is determined by the S3 buffer size (1 MB to 128 MB) or buffer interval (60 seconds to 900 seconds), whichever comes first.

Data delivery to your S3 bucket might fail for various reasons. For example, the bucket might not exist anymore, or the AWS Identity and Access Management (IAM) role that Firehose assumes might not have access to the bucket. Under these conditions, Firehose keeps retrying for up to 24 hours until the delivery succeeds. The maximum data storage time of Firehose is 24 hours. If data delivery fails for more than 24 hours, your data is lost.

Amazon Redshift

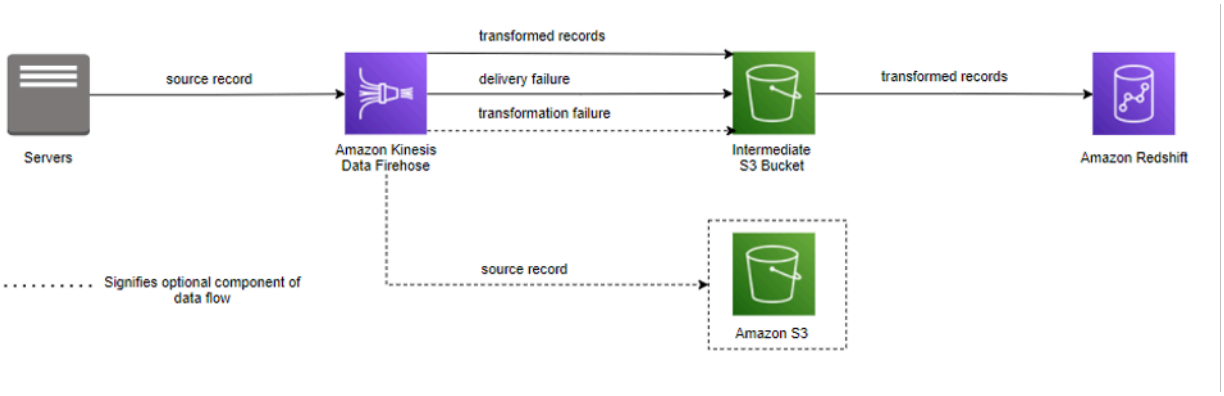
Amazon Redshift is a fast, fully managed data warehouse that makes it simple and cost-effective to analyze all your data using standard SQL and your existing BI tools. It enables you to run complex analytic queries against petabytes of structured data using sophisticated query optimization, columnar storage on high-performance local disks, and massively parallel query running.

Data delivery to Amazon Redshift

For data delivery to Amazon Redshift, Firehose first delivers incoming data to your S3 bucket in the format described earlier. Firehose then issues an Amazon Redshift COPY command to load the data from your S3 bucket to your Amazon Redshift cluster.

The frequency of data COPY operations from S3 to Amazon Redshift is determined by how fast your Amazon Redshift cluster can finish the COPY command. For an Amazon Redshift destination, you can specify a retry duration (0–7200 seconds) when creating a delivery stream to handle data delivery failures. Firehose retries for the specified time duration and skips that particular batch of S3 objects if unsuccessful. The skipped objects' information is delivered to your S3 bucket as a manifest file in the errors/ folder, which you can use for manual backfill.

Following is an architecture diagram of Firehose to Amazon Redshift data flow. Although this data flow is unique to Amazon Redshift, Firehose follows similar patterns for the other destination targets.



Data flow from Firehose to Amazon Redshift

Amazon OpenSearch Service (OpenSearch Service)

OpenSearch Service is a fully managed service that delivers the OpenSearch easy-to-use APIs and real-time capabilities along with the availability, scalability, and security required by production workloads. OpenSearch Service makes it easy to deploy, operate, and scale OpenSearch for log analytics, full text search, and application monitoring.

Data delivery to OpenSearch Service

For data delivery to OpenSearch Service, Firehose buffers incoming records based on the buffering configuration of your delivery stream, and then generates an OpenSearch bulk request to index multiple records to your OpenSearch cluster. The frequency of data delivery to OpenSearch Service is determined by the OpenSearch buffer size (1 MB to 100 MB) and buffer interval (60 seconds to 900 seconds) values, whichever comes first.

For the OpenSearch Service destination, you can specify a retry duration (0–7200 seconds) when creating a delivery stream. Firehose retries for the specified time duration, and then skips that particular index request. The skipped documents are delivered to your S3 bucket in the `elasticsearch_failed/` folder, which you can use for manual backfill.

Amazon Data Firehose can rotate your OpenSearch Service index based on a time duration. Depending on the rotation option you choose (`NoRotation`, `OneHour`, `OneDay`, `OneWeek`, or `OneMonth`), Firehose appends a portion of the Coordinated Universal Time (UTC) arrival timestamp to your specified index name.

Custom HTTP endpoint or supported third-party service provider

Firehose can send data either to Custom HTTP endpoints or supported third-party providers such as Datadog, Dynatrace, LogicMonitor, MongoDB, New Relic, Splunk, and Sumo Logic.

Custom HTTP endpoint or supported third-party service provider

For Firehose to successfully deliver data to custom HTTP endpoints, these endpoints must accept requests and send responses using certain Firehose request and response formats.

When delivering data to an HTTP endpoint owned by a supported third-party service provider, you can use the integrated AWS Lambda service to create a function to transform the incoming record(s) to the format that matches the format the service provider's integration is expecting.

For data delivery frequency, each service provider has a recommended buffer size. Work with your service provider for more information on their recommended buffer size. For data delivery failure

handling, Firehose establishes a connection with the HTTP endpoint first by waiting for a response from the destination. Firehose continues to establish connection, until the retry duration expires. After that, Firehose considers it a data delivery failure and backs up the data to your S3 bucket.

Summary

Firehose can persistently deliver your streaming data to a supported destination. It's a fully-managed solution, requiring little or no development. For Company *ABC2Badge*, using Firehose was a natural choice. They were already using Amazon Redshift as their data warehouse solution. Because their data sources continuously wrote to transaction logs, they were able to leverage the Amazon Kinesis Agent to stream that data without writing any additional code. Now that company *ABC2Badge* has created a stream of sensor records and are receiving these records via Firehose, they can use this as the basis for the security team use case.

Scenario 3: Preparing clickstream data for data insights processes

Fast Sneakers is a fashion boutique with a focus on trendy sneakers. The price of any given pair of shoes can go up or down depending on inventory and trends, such as what celebrity or sports star was spotted wearing brand name sneakers on TV last night. It is important for *Fast Sneakers* to track and analyze those trends to maximize their revenue.

Fast Sneakers does not want to introduce additional overhead into the project with new infrastructure to maintain. They want to be able to split the development to the appropriate parties, where the data engineers can focus on data transformation and their data scientists can work on their ML functionality independently.

To react quickly and automatically adjust prices according to demand, *Fast Sneakers* streams significant events (like click-interest and purchasing data), transforming and augmenting the event data and feeding it to a ML model. Their ML model is able to determine if a price adjustment is required. This allows *Fast Sneakers* to automatically modify their pricing to maximize profit on their products.

Fast Sneakers real-time price adjustments

This architecture diagram shows the real-time streaming solution *Fast Sneakers* created utilizing Kinesis Data Streams, AWS Glue, and DynamoDB Streams. By taking advantage of these services, they have a solution that is elastic and reliable without needing to spend time on setting up and maintaining the supporting infrastructure. They can spend their time on what brings value to their company by focusing on a streaming extract, transform, load (ETL) job and their machine learning model.

To better understand the architecture and technologies that are used in their workload, the following are some details of the services used.

AWS Glue and AWS Glue streaming

AWS Glue is a fully managed ETL service that you can use to catalog your data, clean it, enrich it, and move it reliably between data stores. With AWS Glue, you can significantly reduce the cost, complexity, and time spent creating ETL jobs. AWS Glue is serverless, so there is no infrastructure to set up or manage. You pay only for the resources consumed while your jobs are running.

Utilizing AWS Glue, you can create a consumer application with an AWS Glue streaming ETL job. This enables you to utilize Apache Spark and other Spark-based modules writing to consume and process your event data. The next section of this document goes into more depth about this scenario.

AWS Glue Data Catalog

The AWS Glue Data Catalog contains references to data that is used as sources and targets of your ETL jobs in AWS Glue. The AWS Glue Data Catalog is an index to the location, schema, and runtime metrics of your data. You can use information in the Data Catalog to create and monitor your ETL jobs. Information in the Data Catalog is stored as metadata tables, where each table specifies a single data store. By setting up a crawler, you can automatically assess numerous types of data stores, including DynamoDB, S3, and Java Database Connectivity (JDBC) connected stores, extract metadata and schemas, and then create table definitions in the AWS Glue Data Catalog.

To work with Amazon Kinesis Data Streams in AWS Glue streaming ETL jobs, it is best practice to define your stream in a table in an AWS Glue Data Catalog database. You define a stream-sourced table with the Kinesis stream, one of the many formats supported (CSV, JSON, ORC, Parquet, Avro or a customer format with Grok). You can manually enter a schema, or you can leave this step to your AWS Glue job to determine during runtime of the job.

AWS Glue streaming ETL job

AWS Glue runs your ETL jobs in an Apache Spark serverless environment. AWS Glue runs these jobs on virtual resources that it provisions and manages in its own service account. In addition to being able to run Apache Spark based jobs, AWS Glue provides an additional level of functionality on top of Spark with DynamicFrames.

DynamicFrames are distributed tables that support nested data such as structures and arrays. Each record is self-describing, designed for schema flexibility with semi-structured data. A record in a `DynamicFrame` contains both data and the schema describing the data. Both Apache Spark `DataFrames` and `DynamicFrames` are supported in your ETL scripts, and you can convert them back and forth. `DynamicFrames` provide a set of advanced transformations for data cleaning and ETL.

By using Spark Streaming in your AWS Glue Job, you can create streaming ETL jobs that run continuously, and consume data from streaming sources like Amazon Kinesis Data Streams, Apache Kafka, and Amazon MSK. The jobs can clean, merge, and transform the data, then load the results into stores including Amazon S3, Amazon DynamoDB, or JDBC data stores.

AWS Glue processes and writes out data in 100-second windows, by default. This allows data to be processed efficiently, and permits aggregations to be performed on data arriving later than expected. You can configure the window size by adjusting it to accommodate the speed in response vs the accuracy of your aggregation. AWS Glue streaming jobs use checkpoints to track the data

that has been read from the Kinesis Data Stream. For a walkthrough on creating a streaming ETL job in AWS Glue you can refer to Adding Streaming ETL Jobs in AWS Glue

Amazon DynamoDB

Amazon DynamoDB is a key-value and document database that delivers single-digit millisecond performance at any scale. It's a fully managed, multi-Region, multi-active, durable database with built-in security, backup and restore, and in-memory caching for internet-scale applications. DynamoDB can handle more than ten trillion requests per day, and can support peaks of more than 20 million requests per second.

Change data capture for DynamoDB streams

A DynamoDB stream is an ordered flow of information about changes to items in a DynamoDB table. When you enable a stream on a table, DynamoDB captures information about every modification to data items in the table. DynamoDB runs on AWS Lambda so that you can create triggers—pieces of code that automatically respond to events in DynamoDB streams. With triggers, you can build applications that react to data modifications in DynamoDB tables.

When a stream is enabled on a table, you can associate the stream Amazon Resource Name (ARN) with a Lambda function that you write. Immediately after an item in the table is modified, a new record appears in the table's stream. AWS Lambda polls the stream and invokes your Lambda function synchronously when it detects new stream records.

Amazon SageMaker and Amazon SageMaker service endpoints

Amazon SageMaker is a fully managed platform that enables developers and data scientists with the ability to build, train, and deploy ML models quickly and at any scale. SageMaker includes modules that can be used together or independently to build, train, and deploy your ML models. With Amazon SageMaker service endpoints, you can create managed hosted endpoint for real-time inference with a deployed model that you developed within or outside of Amazon SageMaker.

By utilizing the AWS SDK, you can invoke a SageMaker endpoint passing content type information along with content and then receive real-time predictions based on the data passed. This enables you to keep the design and development of your ML models separated from your code that performs actions on the inferred results.

This enables your data scientists to focus on ML, and the developers who are using the ML model to focus on how they use it in their code. For more information on how to invoke an endpoint in SageMaker, refer to InvokeEnpoint in the Amazon SageMaker API Reference.

Inferring data insights in real time

The previous architecture diagram shows that *Fast Sneakers'* existing web application added a Kinesis Data Stream containing click-stream events, which provides traffic and event data from the website. The product catalog, which contains information such as categorization, product attributes, and pricing, and the order table, which has data such as items ordered, billing, shipping, and so on, are separate DynamoDB tables. The data stream source and the appropriate DynamoDB tables have their metadata and schemas defined in the AWS Glue Data Catalog to be used by the AWS Glue streaming ETL job.

By utilizing Apache Spark, Spark streaming, and `DynamicFrames` in their AWS Glue streaming ETL job, *Fast Sneakers* is able to extract data from either data stream and transform it, merging data from the product and order tables. With the hydrated data from the transformation, the datasets to get inference results from are submitted to a DynamoDB table.

The DynamoDB Stream for the table triggers a Lambda function for each new record written. The Lambda function submits the previously transformed records to a SageMaker Endpoint with the AWS SDK to infer what, if any, price adjustments are necessary for a product. If the ML model identifies an adjustment to the price is required, the Lambda function writes the price change to the product in the catalog DynamoDB table.

Summary

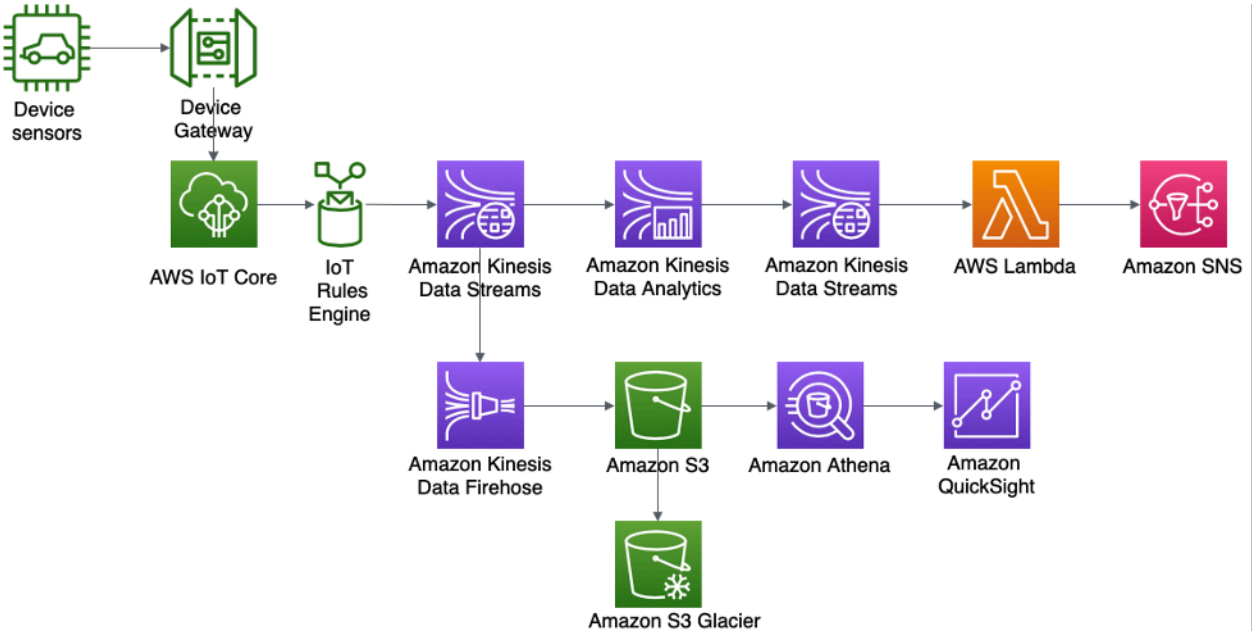
Amazon Kinesis Data Streams makes it easy to collect, process, and analyze real-time, streaming data so you can get timely insights and react quickly to new information. Combined with the AWS Glue serverless data integration service, you can create real-time event streaming applications that prepare and combine data for ML.

Because both Kinesis Data Streams and AWS Glue services are fully managed, AWS takes away the undifferentiated heavy lifting of managing infrastructure for your big data platform, letting you focus on generating data insights based on your data.

Fast Sneakers can utilize real-time event processing and ML to enable their website to make fully automated real-time price adjustments, to maximize their product stock. This brings the most value to their business while avoiding the need to create and maintain a big data platform.

Scenario 4: Device sensors real-time anomaly detection and notifications

Company *ABC4Logistics* transports highly flammable petroleum products such as gasoline, liquid propane (LPG), and naphtha from the port to various cities. There are hundreds of vehicles which have multiple sensors installed on them for monitoring things such as location, engine temperature, temperature inside the container, driving speed, parking location, road conditions, and so on. One of the requirements *ABC4Logistics* has is to monitor the temperatures of the engine and the container in real-time and alert the driver and the fleet monitoring team in case of any anomaly. To detect such conditions and generate alerts in real-time, *ABC4Logistics* implemented the following architecture on AWS.



ABC4Logistics's device sensors real-time anomaly detection and notifications architecture

Data from device sensors is ingested by AWS IoT Gateway, where the AWS IoT rules engine will make the streaming data available in Amazon Kinesis Data Streams. Using Managed Service for Apache Flink, *ABC4Logistics* can perform the real-time analytics on streaming data in Kinesis Data Streams.

Using Managed Service for Apache Flink, *ABC4Logistics* can detect if temperature readings from the sensors deviate from the normal readings over a period of ten seconds, and ingest the record onto another Kinesis Data Streams instance, identifying the anomalous records. Amazon Kinesis Data Streams then invokes Lambda functions, which can send the alerts to the driver and the fleet monitoring team through Amazon SNS.

Data in Kinesis Data Streams is also pushed down to Amazon Data Firehose. Amazon Data Firehose persists this data in Amazon S3, allowing *ABC4Logistics* to perform batch or near-real time analytics on sensor data. *ABC4Logistics* uses Amazon Athena to query data in Amazon S3, and Amazon QuickSight for visualizations. For long-term data retention, the S3 Lifecycle policy is used to archive data to Amazon S3 Glacier.

Important components of this architecture are detailed next.

Amazon Managed Service for Apache Flink

Amazon Managed Service for Apache Flink enables you to transform and analyze streaming data and respond to anomalies in real time. It is a serverless service on AWS, which means Managed Service for Apache Flink takes care of provisioning, and elastically scales the infrastructure to handle any data throughput. This takes away all the undifferentiated heavy lifting of setting up and managing the streaming infrastructure, and enables you to spend more time on writing steaming applications.

With Amazon Managed Service for Apache Flink, you can interactively query streaming data using multiple options, including Standard SQL, Apache Flink applications in Java, Python and Scala, and build Apache Beam applications using Java to analyze data streams.

These options provide you with flexibility of using a specific approach depending on the complexity level of streaming application and source/target support. The following section discusses Managed Service for Apache Flink for Flink applications.

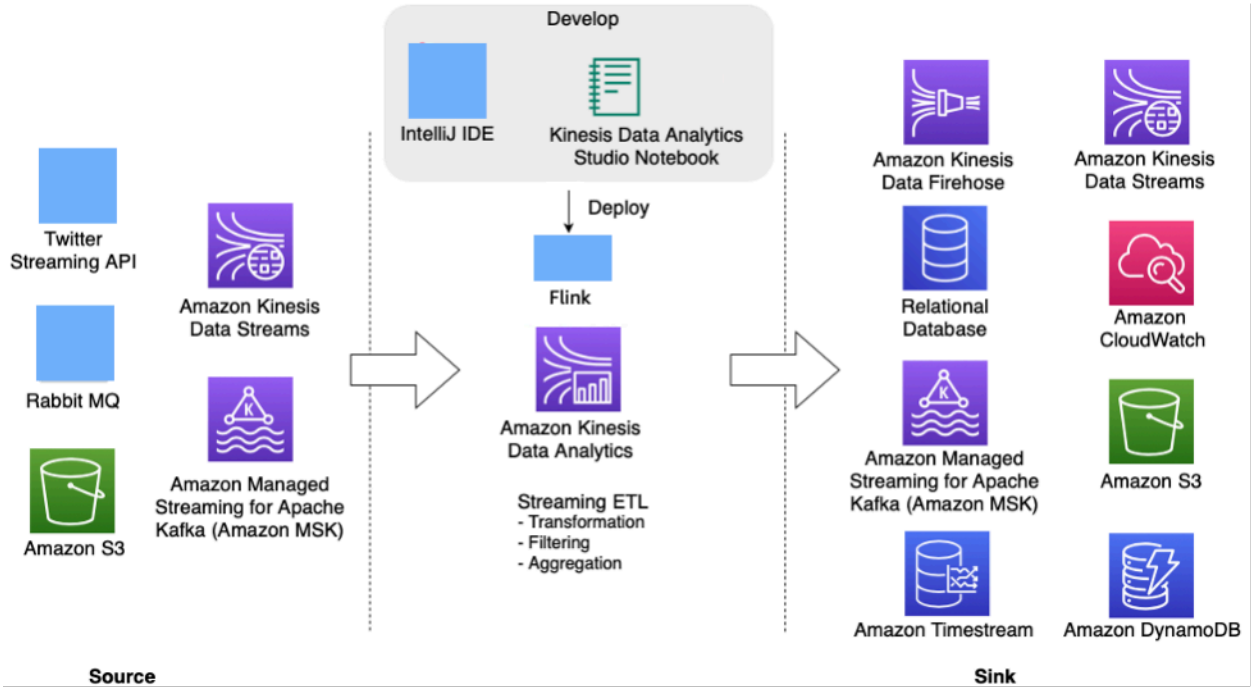
Amazon Managed Service for Apache Flink for Apache Flink applications

Apache Flink is a popular open-source framework and distributed processing engine for stateful computations over unbounded and bounded data streams. Apache Flink is designed to perform computations at in-memory speed and at scale with support for exactly-one semantics. Apache Flink-based applications help achieve low latency with high throughput in a fault tolerant manner.

With Amazon Managed Service for Apache Flink, you can author and run code against streaming sources to perform time series analytics, feed real-time dashboards, and create real-time metrics without managing the complex distributed Apache Flink environment. You can use the high-level Flink programming features in the same way that you use them when hosting the Flink infrastructure yourself.

Managed Service for Apache Flink enables you to create applications in Java, Scala, Python or SQL to process and analyze streaming data. A typical Flink application reads the data from the input stream or data location or *source*, transforms/filters or joins data using operators or functions, and stores the data on output stream or data location, or *sink*.

The following architecture diagram shows some of the supported sources and sinks for the Apache Flink application. In addition to the pre-bundled connectors for source/sink, you can also bring in custom connectors to a variety of other source/sinks for Flink Applications on Managed Service for Apache Flink.



Apache Flink application on Managed Service for Apache Flink for real-time stream processing

Developers can use their preferred IDE to develop Flink applications and deploy them on Managed Service for Apache Flink from AWS Management Console or DevOps tools.

Amazon Managed Service for Apache Flink Studio

As part of Managed Service for Apache Flink service, Managed Service for Apache Flink Studio is available for customers to interactively query data streams in real time, and easily build and run stream processing applications using SQL, Python, and Scala. Studio notebooks are powered by Apache Zeppelin.

Using Studio notebook, you have the ability to develop your Flink Application code in a notebook environment, view results of your code in real time, and visualize it within your notebook. You can

create a Studio Notebook powered by Apache Zeppelin and Apache Flink with a single click from Kinesis Data Streams and Amazon MSK console, or launch it from Managed Service for Apache Flink Console.

Once you develop the code iteratively as part of the Managed Service for Apache Flink Studio, you can deploy a notebook as a Apache Flink application, to run in streaming mode continuously, reading data from your sources, writing to your destinations, maintaining long-running application state, and scaling automatically based on the throughput of your source streams. Earlier, customers used Managed Service for Apache Flink for SQL Applications for such interactive analytics of real-time streaming data on AWS.

Managed Service for Apache Flink for SQL applications is still available, but for new projects, AWS recommends that you use the new Managed Service for Apache Flink Studio. Managed Service for Apache Flink Studio combines ease of use with advanced analytical capabilities, which makes it possible to build sophisticated stream processing applications in minutes.

For making the Apache Flink application fault-tolerant, you can make use of checkpointing and snapshots, as described in the Implementing Fault Tolerance in Managed Service for Apache Flink.

Apache Flink applications are useful for writing complex streaming analytics applications such as applications with exactly-one semantics of data processing, checkpointing capabilities, and processing data from data sources such as Kinesis Data Streams, Firehose, Amazon MSK, Rabbit MQ, and Apache Cassandra including Custom Connectors.

After processing streaming data in the Flink application, you can persist data to various sinks or destinations such as Amazon Kinesis Data Streams, Amazon Data Firehose, Amazon DynamoDB, Amazon OpenSearch Service, Amazon Timestream, Amazon S3, and so on. The Apache Flink application also provides sub-second performance guarantees.

Apache Beam applications for Managed Service for Apache Flink

Apache Beam is a programming model for processing streaming data. Apache Beam provides a portable API layer for building sophisticated data-parallel processing pipelines that may be run across a diversity of engines, or runners such as Flink, Spark Streaming, Apache Samza, and so on.

You can use the Apache Beam framework with your Apache Flink application to process streaming data. Flink applications that use Apache Beam use Apache Flink runner to run Beam pipelines.

Summary

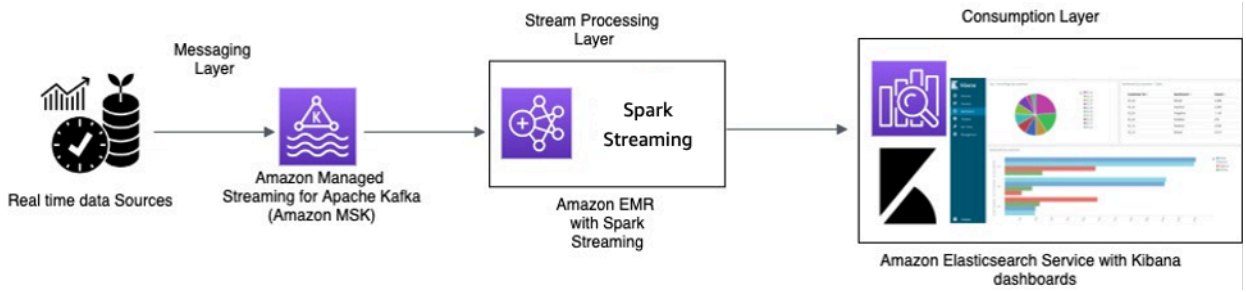
By making use of the AWS streaming services Amazon Kinesis Data Streams, Amazon Managed Service for Apache Flink, and Amazon Data Firehose,

ABC4Logistics can detect anomalous patterns in temperature readings and notify the driver and the fleet management team in real-time, preventing major accidents such as complete vehicle breakdown or fire.

Scenario 5: Real time telemetry data monitoring with Apache Kafka

ABC1Cabs is an online cab booking services company. All the cabs have IoT devices that gather telemetry data from the vehicles. Currently, *ABC1Cabs* is running Apache Kafka clusters that are designed for real-time event consumption, gathering system health metrics, activity tracking, and feeding the data into Apache Spark Streaming platform built on a Hadoop cluster on-premises.

ABC1Cabs use OpenSearch Dashboards for business metrics, debugging, alerting, and creating other dashboards. They are interested in Amazon MSK, Amazon EMR with Spark Streaming, and OpenSearch Service with OpenSearch Dashboards. Their requirement is to reduce admin overhead of maintaining Apache Kafka and Hadoop clusters, while using familiar open-source software and APIs to orchestrate their data pipeline. The following architecture diagram shows their solution on AWS.



Real-time processing with Amazon MSK and Stream processing using Apache Spark Streaming on Amazon EMR and Amazon OpenSearch Service with OpenSearch Dashboards

The cab IoT devices collect telemetry data and send to a source hub. The source hub is configured to send data in real time to Amazon MSK. Using the Apache Kafka producer library APIs, Amazon MSK is configured to stream the data into an Amazon EMR cluster. The Amazon EMR cluster has a Kafka client and Spark Streaming installed to be able to consume and process the streams of data.

Spark Streaming has sink connectors which can write data directly to defined indexes of Elasticsearch. Elasticsearch clusters with OpenSearch Dashboards can be used for metrics and dashboards. Amazon MSK, Amazon EMR with Spark Streaming, and OpenSearch Service with OpenSearch Dashboards are all managed services, where AWS manages the undifferentiated heavy lifting of infrastructure management of different clusters, which enables you to build your application using familiar open-source software with few clicks. The next section takes a closer look at these services.

Amazon Managed Streaming for Apache Kafka (Amazon MSK)

Apache Kafka is an open-source platform that enables customers to capture streaming data like click stream events, transactions, IoT events, and application and machine logs. With this information, you can develop applications that perform real-time analytics, run continuous transformations, and distribute this data to data lakes and databases in real-time.

You can use Kafka as a streaming data store to decouple applications from producer and consumers and enable reliable data transfer between the two components. While Kafka is a popular enterprise data streaming and messaging platform, it can be difficult to set up, scale, and manage in production.

Amazon MSK takes care of these managing tasks and makes it easy to set up, configure, and run Kafka, along with Apache Zookeeper, in an environment following best practices for high availability and security. You can still use Kafka's control-plane operations and data-plane operations to manage producing and consuming data.

Because Amazon MSK runs and manages open-source Apache Kafka, it makes it easy for customers to migrate and run existing Apache Kafka applications on AWS without needing to make changes to their application code.

Scaling

Amazon MSK offers scaling operations so that user can scale the cluster actively while its running. When creating an Amazon MSK cluster, you can specify the instance type of the brokers at cluster launch. You can start with a few brokers within an Amazon MSK cluster. Then, using the AWS Management Console or AWS CLI, you can scale up to hundreds of brokers per cluster.

Alternatively, you can scale your clusters by changing the size or family of your Apache Kafka brokers. Changing the size or family of your brokers gives you the flexibility to adjust your Amazon MSK cluster’s compute capacity for changes in your workloads. See Amazon MSK Pricing for assistance in determining the correct number of brokers for your Amazon MSK cluster.

After creating the Amazon MSK cluster, you can increase the amount of EBS storage per broker with exception of decreasing the storage. Storage volumes remain available during this scaling-up operation. It offers two types of scaling operations: Auto Scaling and Manual Scaling.

Amazon MSKsupports automatic expansion of your cluster's storage in response to increased usage using Application Auto Scaling policies. Your automatic scaling policy sets the target disk utilization and the maximum scaling capacity.

The storage utilization threshold helps Amazon MSK to trigger an automatic scaling operation. To increase storage using manual scaling, wait for the cluster to be in the `ACTIVE` state. Storage scaling has a cooldown period of at least six hours between events. Even though the operation makes additional storage available right away, the service performs optimizations on your cluster that can take up to 24 hours or more.

The duration of these optimizations is proportional to your storage size. Additionally, it also offers multi–Availability Zones replication within an AWS Region to provide High Availability.

Configuration

Amazon MSK provides a default configuration for brokers, topics, and Apache Zookeeper nodes. You can also create custom configurations and use them to create new Amazon MSK clusters or update existing clusters. When you create an MSK cluster without specifying a custom Amazon MSK configuration, Amazon MSK creates and uses a default configuration. For a list of default values, refer to Apache Kafka Configuration.

For monitoring purposes, Amazon MSK gathers Apache Kafka metrics and sends them to Amazon CloudWatch, where you can view them. The metrics that you configure for your MSK cluster are automatically collected and pushed to CloudWatch. Monitoring consumer lag enables you to identify slow or stuck consumers that aren't keeping up with the latest data available in a topic. When necessary, you can then take remedial actions, such as scaling or rebooting those consumers.

Migrating to Amazon MSK

Migrating from on premises to Amazon MSK can be achieved by one of the following methods.

- **MirrorMaker2.0** — MirrorMaker2.0 (MM2) MM2 is a multi-cluster, data replication engine based on Apache Kafka Connect framework. MM2 is a combination of an Apache Kafka source connector and a sink connector. You can use a single MM2 cluster to migrate data between multiple clusters. MM2 automatically detects new topics and partitions, while also ensuring

the topic configurations are synced between clusters. MM2 supports migrations ACLs, topics configurations, and offset translation. For more details related to migration, refer to Migrating Clusters Using Apache Kafka's MirrorMaker. MM2 is used for use cases related to replication of topics configurations and offset translation automatically.

- **Apache Flink** — MM2 supports at least once semantics. Records can be duplicated to the destination and the consumers are expected to be idempotent to handle duplicate records. In exactly-once scenarios, semantics are required customers can use Apache Flink. It provides an alternative to achieve exactly once semantics.

  Apache Flink can also be used for scenarios where data requires mapping or transformation actions before submission to the destination cluster. Apache Flink provides connectors for Apache Kafka with sources and sinks that can read data from one Apache Kafka cluster and write to another. Apache Flink can be run on AWS by launching an Amazon EMR cluster or by running Apache Flink as an application using Amazon Managed Service for Apache Flink.
- **AWS Lambda** — With support for Apache Kafka as an event source for AWS Lambda, customers can now consume messages from a topic via a Lambda function. The AWS Lambda service internally polls for new records or messages from the event source, and then synchronously invokes the target Lambda function to consume these messages. Lambda reads the messages in batches and provides the message batches to your function in the event payload for processing. Consumed messages can then be transformed and/or written directly to your destination Amazon MSK cluster.

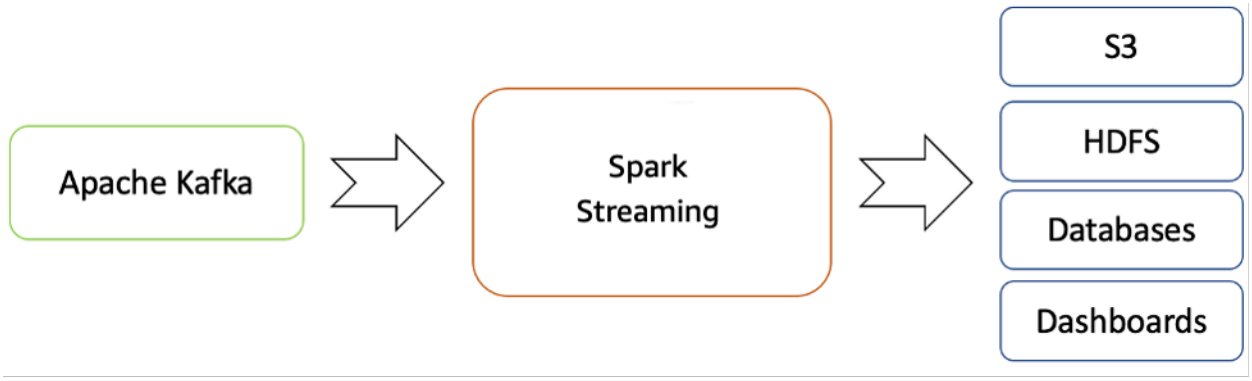
Amazon EMR with Spark streaming

Amazon EMR is a managed cluster platform that simplifies running big data frameworks, such as Apache Hadoop and Apache Spark on AWS, to process and analyze vast amounts of data.

Amazon EMR provides the capabilities of Spark and can be used to start Spark streaming to consume data from Kafka. Spark Streaming is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams.

You can create an Amazon EMR cluster using the AWS Command Line Interface (AWS CLI) or on the AWS Management Console and select Spark and Zeppelin in advanced configurations while creating the cluster. As shown in the following architecture diagram, data can be ingested from many sources such as Apache Kafka and Kinesis Data Streams, and can be processed using complex algorithms expressed with high-level functions such as map, reduce, join and window. For more information, refer to Transformations on DStreams.

Processed data can be pushed out to filesystems, databases, and live dashboards.



Real-time streaming flow from Apache Kafka to Hadoop ecosystem

By default, Apache Spark Streaming has a micro-batch run model. However, since Spark 2.3 came out, Apache has introduced a new low-latency processing mode called Continuous Processing, which can achieve end-to-end latencies as low as one millisecond with at-least-once guarantees.

Without changing the Dataset/DataFrames operations in your queries, you can choose the mode based on your application requirements. Some of the benefits of Spark Streaming are:

- It brings Apache Spark's language-integrated API to stream processing, letting you write streaming jobs the same way you write batch jobs.
- It supports Java, Scala and Python.
- It can recover both lost work and operator state (such as sliding windows) out of the box, without any extra code on your part.
- By running on Spark, Spark Streaming lets you reuse the same code for batch processing, join streams against historical data, or run ad hoc queries on the stream state and build powerful interactive applications, not just analytics.
- After the data stream is processed with Spark Streaming, OpenSearch Sink Connector can be used to write data to the OpenSearch Service cluster, and in turn, OpenSearch Service with OpenSearch Dashboards can be used as consumption layer.

Amazon OpenSearch Service with OpenSearch Dashboards

OpenSearch Service is a managed service that makes it easy to deploy, operate, and scale OpenSearch clusters in the AWS Cloud. OpenSearch is a popular open-source search and analytics engine for use cases such as log analytics, real-time application monitoring, and clickstream analysis.

OpenSearch Dashboards is an open-source data visualization and exploration tool used for log and time-series analytics, application monitoring, and operational intelligence use cases. It offers powerful and easy-to-use features such as histograms, line graphs, pie charts, heat maps, and built-in geospatial support.

OpenSearch Dashboards provides tight integration with OpenSearch, a popular analytics and search engine, which makes OpenSearch Dashboards the default choice for visualizing data stored in OpenSearch. OpenSearch Service provides an installation of OpenSearch Dashboards with every OpenSearch Service domain. You can find a link to OpenSearch Dashboards on your domain dashboard on the OpenSearch Service console.

Summary

With Apache Kafka offered as a managed service on AWS, you can focus on consumption rather than on managing the coordination between the brokers, which usually requires a detailed understanding of Apache Kafka. Features such as high availability, broker scalability, and granular access control are managed by the Amazon MSK platform.

ABC1Cabs utilized these services to build production application without needing infrastructure management expertise. They could focus on the processing layer to consume data from Amazon MSK and further propagate to visualization layer.

Spark Streaming on Amazon EMR can help real-time analytics of streaming data, and publishing on OpenSearch Dashboards on Amazon OpenSearch Service for the visualization layer.

Conclusion and contributors

Conclusion

This document reviewed several scenarios for streaming workflows. In these scenarios, streaming data processing provided the example companies with the ability to add new features and functionality.

By analyzing data as it gets created, you will gain insights into what your business is doing right now. AWS streaming services enable you to focus on your application to make time-sensitive business decisions, rather than deploying and managing the infrastructure

Contributors

- Amalia Rabinovitch, Sr. Solutions Architect, AWS
- Priyanka Chaudhary, Data Lake, Data Architect, AWS
- Zohair Nasimi, Solutions Architect, AWS
- Rob Kuhr, Solutions Architect, AWS
- Ejaz Sayyed, Sr. Partner Solutions Architect, AWS
- Allan MacInnis, Solutions Architect, AWS
- Chander Matrubhutam, Product Marketing Manager, AWS

Document revisions

To be notified about updates to this whitepaper, subscribe to the RSS feed.

Change	Description	Date
Updated	Updated for technical accuracy	September 1, 2021
Initial publication	Whitepaper first published	July 1, 2017