Streaming Data Solutions on AWS with Amazon Kinesis

July 2017

© 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Notices

This document is provided for informational purposes only. It represents AWS's current product offerings and practices as of the date of issue of this document, which are subject to change without notice. Customers are responsible for making their own independent assessment of the information in this document and any use of AWS's products or services, each of which is provided "as is" without warranty of any kind, whether express or implied. This document does not create any warranties, representations, contractual commitments, conditions or assurances from AWS, its affiliates, suppliers or licensors. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

Contents

Introduction 1

Real-time Application Scenarios 1

Difference Between Batch and Stream Processing 2

Stream Processing Challenges 2

From Batch to Real-time: An Example 3

Example Scenario: Toll Road Billing and Notification 3

Requirement 1: More Recent Data in the Data Warehouse 4

Amazon Kinesis Firehose 5

Requirement 2: Billing Threshold Alerts 12

Amazon Kinesis Analytics 14

Amazon Kinesis Streams 16

Requirement 3: Other Threshold Alerts 22

Complete Architecture 23

Conclusion 24

Contributors 25

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 Streams, Amazon Kinesis Firehose, and Amazon Kinesis Analytics 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 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 Application Scenarios

There are two types of use case scenarios for streaming data applications:

- **Evolving from Batch to Streaming Analytics**

You can perform real-time analytics on data that has been traditionally analyzed using batch processing in data warehouses or using Hadoop frameworks. The most common use cases in this category include data lakes, data science, and machine learning. You can use streaming data solutions to continuously load real-time data into your data lakes. You can also update machine learning models more frequently as new data becomes available, ensuring accuracy and reliability of the outputs. For example, Zillow uses Amazon Kinesis Streams to collect public record data and MLS listings, and then provide home buyers and sellers with the most up-to-date home value estimates in near real-time. Zillow also sends the same data to its Amazon Simple Storage Service (S3) data lake using Kinesis Streams so that all the applications work with the most recent information.

- **Building Real-Time Applications**

You can use streaming data services for real-time applications such as application monitoring, fraud detection, and live leaderboards. These 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 Kinesis 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.

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. Instead of running database queries over stored data, 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. 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. 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.

From Batch to Real-time: An Example

To better understand how organizations are evolving from batch to stream processing with AWS, let's walk through an example. In this example, we'll review a scenario and discuss in detail how AWS services Amazon Kinesis Streams,[1] Amazon Kinesis Firehose,[2] and Amazon Kinesis Analytics[3] are used to solve the problem.

Batch processing is a common practice for data processing. Organizations often run regular jobs to analyze their data at a frequency applicable for their use case. For example, an organization might run a process at the end of the month to determine how much to bill each of their customers. Or, they might run an hourly job to analyze logs from their IT applications to determine what errors occurred in the past hour. While these monthly or hourly processes are valuable, what if the same data could be analyzed as it gets created? Are there additional insights that could be gleaned, or additional value that could be created?

Consider the monthly billing scenario again. By analyzing a customer's usage data as it is generated, an organization can enable valuable features, such as notifying users that they're approaching a pre-defined billing limit. If the IT application logs can be analyzed in real-time, a system administrator can be notified immediately to investigate and take corrective action.

Now let's combine these two into a single scenario and review how we can build a solution.

Example Scenario: Toll Road Billing and Notification

In this simplified example, a fictitious company, ABC Tolls, operates toll highways throughout the country. Customers that register with ABC Tolls receive a transceiver for their automobile. When the customer drives through the tolling area, a sensor receives information from the transceiver and records details of the transaction to a relational database. ABC Tolls has a traditional batch architecture. Each day, a scheduled extract-transform-load (ETL) process is executed that processes the daily transactions and transforms them so they can be loaded into their data warehouse. The next day, the ABC Tolls business analysts review the data using a reporting tool. In addition, once a month (at the

end of the billing cycle) another process aggregates all the transactions for each of the ABC Tolls customers to calculate their monthly payment.

ABC Tolls would like to make some modifications to its system. The first requirement comes from its business analyst team. They have asked for the ability to run reports from their data warehouse with data that is no older than 30 minutes.

ABC Tolls is also developing a new mobile application for its customers. While developing the application, they decided to create some new features. One feature gives customers the ability to set a spending threshold for their account. If a customer's cumulative toll bill surpasses this threshold, ABC Tolls wants to send an in-application message to the customer to notify them that the threshold has been breached within 10 minutes of the breach occurring.

Finally, the ABC Tolls operations team has some additional requirements that they'd like to introduce to the system. While monitoring their tolling stations, they want to be immediately notified when the vehicle traffic for a tolling station falls below a pre-defined threshold for each 30-minute period in a day. For example, they know from historical data that one of their tolling stations sees approximately 360 vehicles on Wednesdays between 2:00 pm and 2:30 pm. In that 30-minute window, the operations team wants to be notified if the tolling station sees fewer than 100 vehicles. Their operators can then investigate to determine if the traffic is normal, or if some other factor has contributed to the unexpected value (e.g., a defective sensor or an automobile accident on the highway).

The ABC Tolls engineering team determines that their current architecture needs some modifications to support these requirements. They decide to build a streaming data ingestion and analytics system to support the requirements. Let's review each requirement and take a look at the architecture enhancements that will support each one.

Requirement 1: More Recent Data in the Data Warehouse

Currently, the data in the ABC Tolls data warehouse can be up to 24 hours old because of their daily batch process. Their current data warehouse solution is Amazon Redshift. While reviewing the features of the Amazon Kinesis services,

they recognized that Kinesis Firehose can receive a stream of data records and insert them into Amazon Redshift. They created a Kinesis Firehose delivery stream and configured it so that it would copy data to their Amazon Redshift table every 15 minutes. Their current solution stores records to a file system as part of their batch process. As part of this new solution, they used the Amazon Kinesis Agent on their servers to forward their log data to Kinesis Firehose. Since Kinesis Firehose uses Amazon S3 to store raw streaming data before it is copied to Amazon Redshift, ABC Tolls didn't need to build another solution to archive their raw data.

Figure 1 depicts this solution.

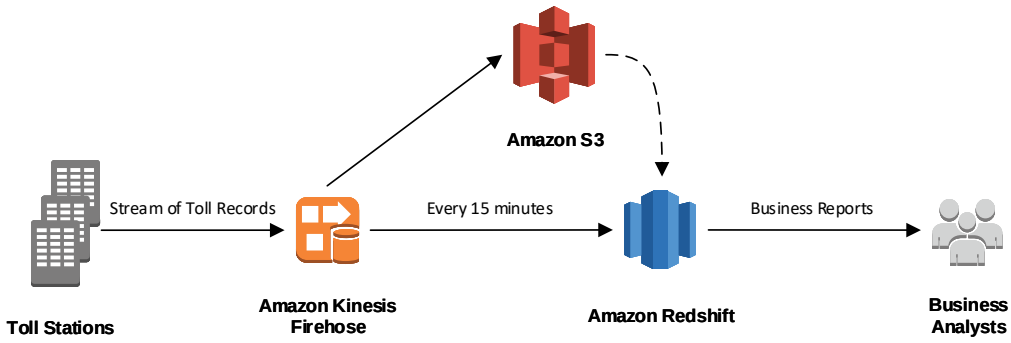

Figure 1: New solution using Amazon Kinesis Firehose

For this portion of the architecture, ABC Tolls chose Kinesis Firehose. Let's review the features of Kinesis Firehose in detail.

Amazon Kinesis Firehose

Amazon Kinesis Firehose is the easiest way to load streaming data into AWS. It can capture, transform, and load streaming data into Amazon Kinesis Analytics, Amazon S3, Amazon Redshift, and Amazon Elasticsearch Service, enabling near real-time analytics with existing business intelligence tools and dashboards that you're already using today. It's a fully managed service that automatically scales to match the throughput of your data and requires no ongoing administration. It can also batch, compress, and encrypt the data before loading it, minimizing the amount of storage used at the destination and increasing security.

Kinesis Firehose is a fully managed service. You do not need to write applications or manage resources. You configure your data producers to send

data to Kinesis Firehose, which automatically delivers the data to the destination that you specified. You can also configure Kinesis Firehose to transform your data before data delivery.

Sending Data to an Amazon Kinesis 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 Kinesis Firehose. AWS has also created a utility to help send data to your delivery stream.

Using the API

The Kinesis 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.

In each method, you must specify the name of the delivery stream and the data record, or array of data records, when using the method. Each data record consists of a data blob that can be up to 1,000 KB in size and any kind of data.

For detailed information and sample code for the Kinesis Firehose API operations, refer to Writing to a Firehose Delivery Stream Using the AWS SDK.[4]

Using the Amazon Kinesis Agent

The Amazon Kinesis Agent is a stand-alone Java software application that offers an easy way to collect and send data to Kinesis Streams and Kinesis Firehose. The agent continuously monitors a set of files and sends new data to your stream. The agent handles file rotation, checkpointing, and retry upon failures. It delivers all of your data in a reliable, timely, and simple manner. It also emits Amazon CloudWatch metrics to help you better monitor and troubleshoot the streaming process.

You can install the agent on Linux-based server environments such as web servers, log servers, and database servers. After installing the agent, configure it by specifying the files to monitor and the destination stream for the data. After the agent is configured, it durably collects data from the files and reliably sends it to the delivery stream.

The agent can monitor multiple file directories and write to multiple streams. It can also be configured to pre-process data records before they're sent to your stream or delivery stream.

If you're considering a migration from a traditional batch file system to streaming data, it's possible that your applications are already logging events to files on the file systems of your application servers. Or, if your application uses a popular logging library (such as Log4j), it is typically a straight-forward task to configure it to write to local files. Regardless of how the data is written to a log file, you should consider using the agent in this scenario. It provides a simple solution that requires little or no change to your existing system. In many cases, it can be used concurrently with your existing batch solution. In this scenario, it provides a stream of data to Kinesis Streams, using the log files as a source of data for the stream.

In our example scenario, ABC Tolls chose to use the agent to send streaming data to their delivery stream. They were already creating log files, so forwarding the log entries to Kinesis Firehose was a simple installation and configuration of the agent. No additional code was needed to start streaming their data.

Data Transformation

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 Amazon Elasticsearch Service.

To enable streaming data transformations, Kinesis Firehose uses an AWS Lambda function that you create to transform your data.[5]

Data Transformation Flow

When you enable Kinesis Firehose data transformation, Kinesis Firehose buffers incoming data up to 3 MB or the buffering size you specified for the delivery stream, whichever is smaller. Kinesis Firehose then invokes the specified Lambda function with each buffered batch asynchronously. The transformed data is sent from Lambda to Kinesis Firehose for buffering. Transformed data is delivered to the destination when the specified buffering size or buffering interval is reached, whichever happens first. Figure 2 depicts this process for a delivery stream that delivers data to Amazon S3.

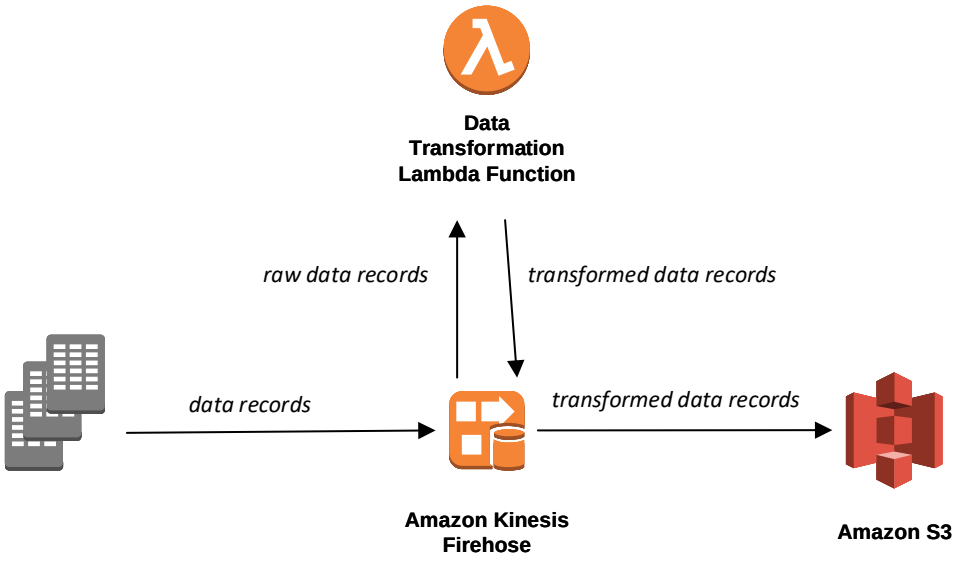

Figure 2: Buffering data using Kinesis Firehose and Lambda functions

Data Delivery

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 Kinesis Firehose delivers data to each destination, which we'll review in the following sections.

Amazon Simple Storage Service

Amazon S3 is object storage with a simple web service interface to store and retrieve any amount of data from anywhere on the web.[6] It's designed to deliver 99.999999999% durability, and scale past trillions of objects worldwide. You can use Amazon S3 as primary storage for cloud-native applications, as a bulk repository or "data lake" for analytics, and as a target for backup and recovery and disaster recovery.

Data Delivery Format

For data delivery to Amazon S3, Kinesis 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. You may want to add a record separator at the end of each record before you send it to Kinesis Firehose so that you can divide a delivered S3 object to individual records.

Data Delivery Frequency

The frequency of data delivery to Amazon S3 is determined by the S3 buffer size and buffer interval value you configured for your delivery stream. Kinesis Firehose buffers incoming data before delivering it to Amazon S3. You can configure the values for the Amazon S3 buffer size (1 MB to 128 MB) or buffer interval (60 seconds to 900 seconds). The condition satisfied first triggers data delivery to Amazon S3. Note that in circumstances where data delivery to the destination is falling behind data writing to the delivery stream, Kinesis Firehose raises the buffer size dynamically to catch up and make sure that all data is delivered to the destination.

Data Flow

Figure 3 shows the flow of data for Amazon S3 destinations.

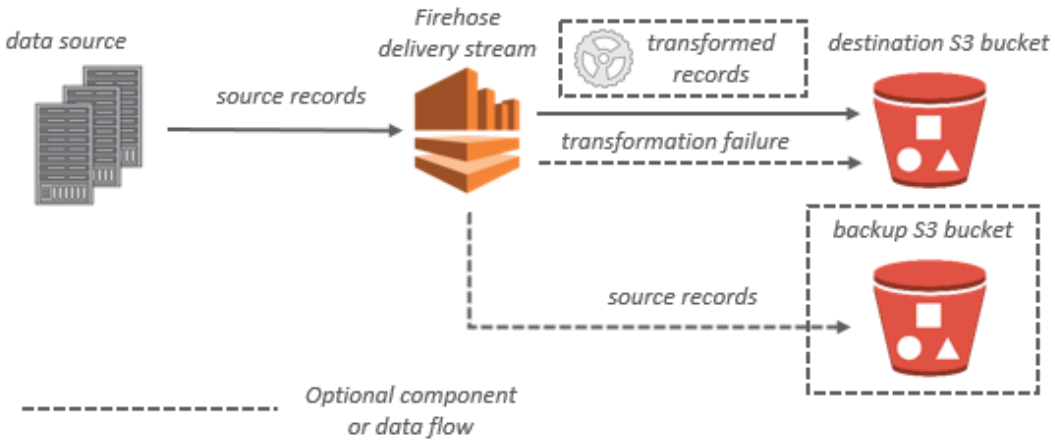

Figure 3: Data flow from Kinesis Firehose to S3 buckets

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 business intelligence (BI) tools.[7] It allows 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 execution. Most results come back in seconds.

In our example, ABC Tolls was already using Amazon Redshift as their data warehouse solution. When they implemented their streaming data solution, they configured their delivery stream to deliver their streaming data to their existing Amazon Redshift cluster.

Data Delivery Format

For data delivery to Amazon Redshift, Kinesis Firehose first delivers incoming data to your S3 bucket in the format described earlier. Kinesis Firehose then issues an Amazon Redshift COPY command to load the data from your S3 bucket to your Amazon Redshift cluster. You need to make sure that after Kinesis Firehose concatenates multiple incoming records to an S3 object, the S3 object can be copied to your Amazon Redshift cluster. For more information, see Amazon Redshift COPY Command Data Format Parameters.

Data Delivery Frequency

The frequency of data COPY operations from Amazon S3 to Amazon Redshift is determined by how fast your Amazon Redshift cluster can finish the COPY command. If there is still data to copy, Kinesis Firehose issues a new COPY command as soon as the previous COPY command is successfully finished by Amazon Redshift.

Data Flow

Figure 4 shows the flow of data for Amazon Redshift destinations.

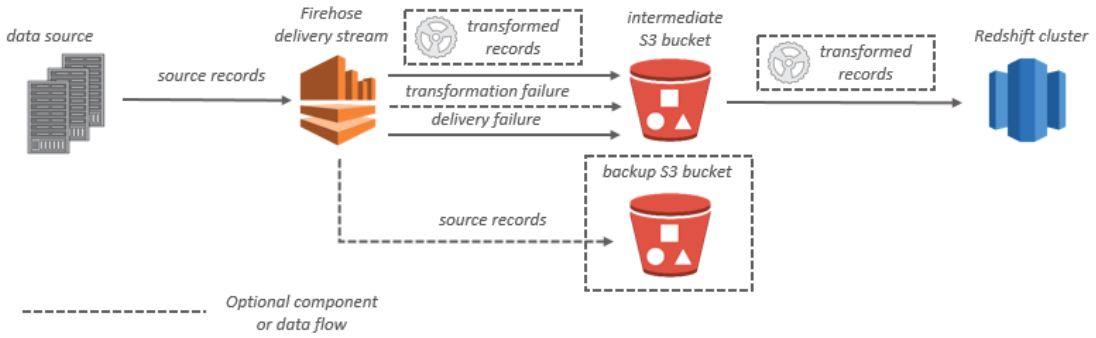

Figure 4: Data flow from Kinesis Firehose to Amazon Redshift

Amazon Elasticsearch Service

Amazon Elasticsearch Service (Amazon ES) is a fully managed service that delivers the Elasticsearch easy-to-use APIs and real-time capabilities along with the availability, scalability, and security required by production workloads.[8] Amazon ES makes it easy to deploy, operate, and scale Elasticsearch for log analytics, full text search, application monitoring, and more.

Data Delivery Format

For data delivery to Amazon ES, Kinesis Firehose buffers incoming records based on the buffering configuration of your delivery stream and then generates an Elasticsearch bulk request to index multiple records to your Elasticsearch cluster. You need to make sure that your record is UTF-8 encoded and flattened to a single-line JSON object before you send it to Kinesis Firehose.

Data Delivery Frequency

The frequency of data delivery to Amazon ES is determined by the Elasticsearch buffer size and buffer interval values that you configured for your delivery stream. Kinesis Firehose buffers incoming data before delivering it to Amazon ES. You can configure the values for the Elasticsearch buffer size (1 MB to 100 MB) or buffer interval (60 seconds to 900 seconds). The condition satisfied first triggers data delivery to Amazon ES. Note that in circumstances where data delivery to the destination is falling behind data writing to the delivery stream, Kinesis Firehose raises the buffer size dynamically to catch up and make sure that all data is delivered to the destination.

Data Flow

Figure 5 shows the flow of data for Amazon ES destinations.

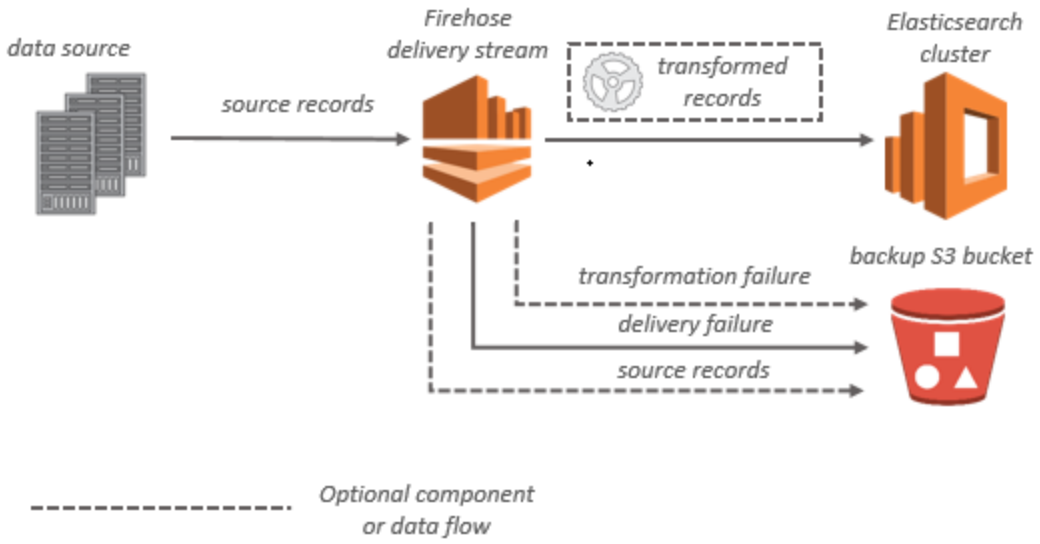

Figure 5: Data delivery from Kinesis Firehose to Amazon ES cluster

Summary

Kinesis Firehose is the easiest way to persist your streaming data to a supported destination. It's a fully-managed solution, requiring little or no development to use. For ABC Tolls, using Kinesis Firehose was a natural choice. They were already using Amazon Redshift as their data warehouse solution. And because their data sources were continuously writing to transaction logs, they were able to leverage the Amazon Kinesis Agent to stream that data without writing any additional code.

Now that ABC Tolls has created a stream of toll records and are receiving these records via Kinesis Firehose, they can use this as the basis for their other streaming data requirements.

Requirement 2: Billing Threshold Alerts

To support the feature to send a notification when a spending threshold is breached, the ABC Tolls development team has created a mobile application and an Amazon DynamoDB table.[9] The application allows customers to set their threshold, and the table stores this value for each customer. The table is also used to store the cumulative amount spent by each customer, each month. To provide timely notifications, ABC Tolls needs to update the cumulative value in

this table in a timely manner, and compare that value with the threshold to determine if a notification should be sent to the customer. Since their toll transactions are already streaming through Kinesis Firehose, they decided to use this streaming data as the source for their aggregation and alerting. And because Kinesis Analytics enabled them to use SQL to aggregate the streaming data, it is an ideal solution to the problem. In this solution, Kinesis Analytics totals the value of the transactions for each customer over a 10-minute time period (window). At the end of the window, it sends the totals to a Kinesis stream. This stream is the event source for an AWS Lambda function. The Lambda function queries the DynamoDB table to retrieve the thresholds and current total spent by each customer represented in the output from Kinesis Analytics. For each customer, the Lambda function updates the current total in DynamoDB and also compares the total with the threshold. If the threshold has been exceeded, it uses the AWS SDK to tell Amazon Simple Notification Service (SNS) to send a notification to the customers.

Figure 6 shows the architecture for this solution.

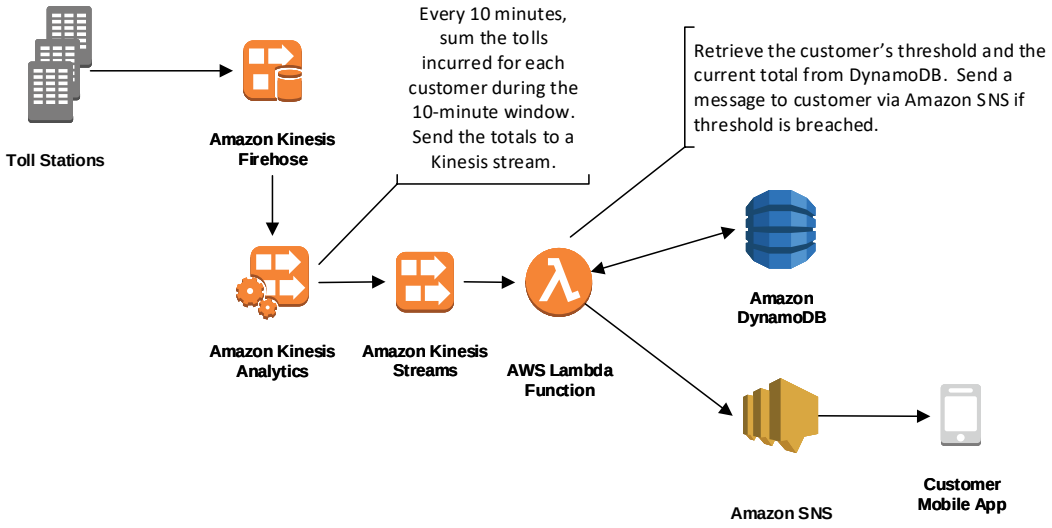

Figure 6: Architecture for billing threshold alerts and notifications

With this solution, ABC Tolls provides their customers with a timely notification when they approach spending limits.

To glean real-time insights from their streaming data, ABC Tolls chose to use Kinesis Analytics to analyze their streaming data. With Kinesis Analytics, ABC Tolls used SQL, a language they were already familiar with, to inspect their data as it streamed through their delivery stream. Let's review Kinesis Analytics in more detail.

Amazon Kinesis Analytics

With Kinesis Analytics, you can process and analyze streaming data using SQL. The service enables you to quickly author and run powerful SQL code against streaming sources to perform time series analytics, feed real-time dashboards, and create real-time metrics.

To get started with Kinesis Analytics, you create a Kinesis Analytics application that continuously reads and processes streaming data. The service supports ingesting data from Kinesis Streams and Kinesis Firehose streaming sources. You then author your SQL code using the interactive editor and test it with live streaming data. You can also configure destinations where you want Kinesis Analytics to persist the results. Kinesis Analytics supports Kinesis Firehose (Amazon S3, Amazon Redshift, and Amazon Elasticsearch Service), and Kinesis Streams as destinations.

Key Concepts

An *application* is the primary resource in Kinesis Analytics that you can create in your account. Kinesis Analytics applications continuously read and process streaming data in real-time. You write application code using SQL to process the incoming streaming data and produce output. Kinesis Analytics then writes the output to a configured destination. Figure 7 illustrates a typical application architecture.

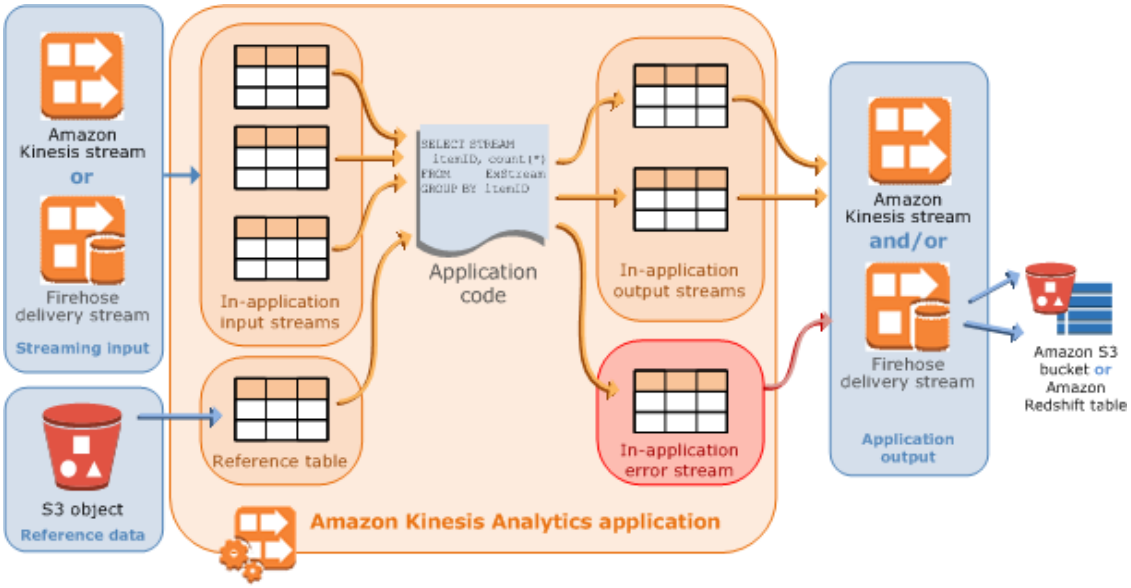

Figure 7: Architecture for a Kinesis Analytics application

Each application has a name, description, version ID, and status. When creating an application, you simply configure the input, create the application code, and configure the output.

Input

The application input is the streaming source for your application. You can select either a Kinesis stream or a delivery stream as the streaming source. You can optionally configure a reference data source to enrich your input data stream within the application. It results in an in-application reference table. You must store your reference data as an object in an S3 bucket. When the application starts, Kinesis Analytics reads the S3 object and creates an in-application table.

ABC Tolls used their delivery stream as input to their Kinesis Analytics application.

Application Code

Your application code consists of a series of SQL statements that process input and produce output. You can write SQL statements against in-application streams, reference tables, and you can write JOIN queries to combine data from both of these sources.

In its simplest form, application code can be a single SQL statement that selects from a streaming input and inserts results into a streaming output. It can also be a series of SQL statements where output of one statement feeds into the input of the next SQL statement. Further, you can write application code to split an input stream into multiple streams and then apply additional queries to process these streams.

Output

In your application code, query results go to in-application streams. In your application code, you can create one or more in-application streams to hold intermediate results. You can then optionally configure application output to persist data in the in-application streams, which hold your application output (also referred to as in-application output streams), to external destinations. External destinations can be a delivery stream or a Kinesis stream.

ABC Tolls used Kinesis Streams as the destination for their aggregated values.

Summary

Kinesis Analytics enables you to use SQL to glean insights from your data as it streams through the system. ABC Tolls wrote their SQL to perform 10-minute-long aggregations to total the tolls incurred by their customers. The output values of these 10-minute aggregations could be compared with their customers' thresholds.

As mentioned earlier, Kinesis Analytics outputs its results to either Kinesis Streams or Kinesis Firehose. In this example, ABC Tolls chose to send the output of Kinesis Analytics to a Kinesis stream because of the integration of Kinesis Streams with AWS Lambda. Let's learn more about Kinesis Streams.

Amazon Kinesis Streams

Amazon Kinesis Streams enables you to build custom, real-time applications using popular stream processing frameworks and load streaming data into any data store. You can configure hundreds of thousands of data producers to continuously put data into a Kinesis stream, for example, data from website clickstreams, application logs, IoT sensors, and social media feeds. Within less than a second, the data will be available for your application to read and process from the stream.

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

While you can use Kinesis Streams to solve a variety of streaming data problems, a common use is the real-time aggregation or analysis of data followed by loading the aggregate data into a data warehouse or map-reduce cluster.

Data is put into Kinesis Streams, which ensures durability and elasticity. The delay between the time a record is put into the stream and the time it can be retrieved (put-to-get delay) is typically less than 1 second—in other words, a Kinesis Streams application can start consuming the data from the stream almost immediately after the data is added. Because Kinesis Streams is a managed service, it relieves you of the operational burden of creating and running a data intake pipeline.

Sending Data to Amazon Kinesis Streams

There are several mechanisms to send data to your stream. AWS offers SDKs for many popular programming languages, each of which provides APIs for Kinesis Streams. AWS has also created several utilities to help send data to your stream. Let's review each of the approaches you can use and why you might choose each.

Amazon Kinesis Agent

The Amazon Kinesis Agent was discussed earlier as a tool that can be used to send data to Kinesis Firehose. The same tool can be used to send data to Kinesis Streams. For details on installing and configuring the Kinesis agent, see Writing to Amazon Kinesis Firehose Using Amazon Kinesis Agent.[10]

Amazon Kinesis Producer Library (KPL)

The KPL simplifies producer application development, allowing developers to achieve high write throughput to one or more Kinesis streams. The KPL is an easy-to-use, highly configurable library that you install on your hosts that generate the data that you wish to stream to Kinesis Streams. It acts as an intermediary between your producer application code and the Kinesis Streams API actions. The KPL performs the following primary tasks:

- Writes to one or more Kinesis streams with an automatic and configurable retry mechanism

- Collects records and uses `PutRecords` to write multiple records to multiple shards per request
- Aggregates user records to increase payload size and improve throughput
- Integrates seamlessly with the Amazon Kinesis Client Library (KCL) to de-aggregate batched records on the consumer
- Submits Amazon CloudWatch metrics on your behalf to provide visibility into producer performance

The KPL can be used in either synchronous or asynchronous use cases. We suggest using the higher performance of the asynchronous interface unless there is a specific reason to use synchronous behavior. For more information about these two use cases and example code, see Writing to your Streams Stream Using the KPL.[11]

The KPL can help build high-performance producers. Consider a situation where your Amazon Elastic Compute Cloud (EC2) instances serve as a proxy for collecting 100-byte events from hundreds or thousands of low power devices and writing records into a Kinesis stream. These EC2 instances must each write thousands of events per second to your Kinesis stream. To achieve the throughput needed, producers must implement complicated logic such as batching or multithreading, in addition to retry logic and record de-aggregation at the consumer side. The KPL performs all of these tasks for you.

Because the KPL buffers your records before they're sent to a Kinesis stream, the KPL can incur an additional processing delay, depending on the length of time you've configured the KPL to buffer records before sending them to Kinesis. Larger buffer time results in higher packing efficiencies and better performance. Applications that cannot tolerate this additional delay may need to use the AWS SDK directly.

If your application does not log records to a local file, and it creates a large number of small records per second, consider using the KPL.

For details about using the KPL to produce data, refer to Developing Amazon Kinesis Streams Producers Using the Amazon Kinesis Producer Library.[12]

Amazon Kinesis API

After a stream is created, you can add your data records to it. A record is a data structure that contains the data to be processed in the form of a data blob. After you store the data in the record, Kinesis Streams does not inspect, interpret, or change the data in any way.

There are two different operations in the Kinesis Streams API that add data to a stream: `PutRecords` and `PutRecord`. The `PutRecords` operation sends multiple records to your stream per HTTP request, and the singular `PutRecord` operation sends records to your stream one at a time (a separate HTTP request is required for each record). You should prefer using `PutRecords` for most applications because it will achieve higher throughput per data producer.

Because the APIs are exposed in all AWS SDKs, using the API to write records provides the most flexible solution to send data to a Kinesis stream. If you are unable to use the Kinesis Agent or KPL (for example, you want to write messages directly from a mobile application, or you want to minimize message end-to-end latency as much as possible), then use the APIs to write records to your Kinesis stream.

For details about these APIs, refer to Using the API in the Kinesis Streams documentation.[13] The details for each API operation can be found in the Amazon Kinesis Streams API Reference.[14]

Processing Data in Amazon Kinesis Streams

A consumer is an application that reads and processes data from Kinesis Streams. You can build consumers for Kinesis Streams in several ways. In this section, we'll discuss four of the most common approaches: using Kinesis Analytics, using the KCL, using Amazon Lambda, and using the Kinesis Streams API directly.

Using Amazon Kinesis Analytics

Earlier, we discussed how Kinesis Analytics can be used to analyze streaming data using standard SQL. Kinesis Analytics can read the data from your Kinesis stream, and process it using the SQL you provide. To learn more about processing your streaming data using Kinesis Analytics, see Configuring Application Input in the Kinesis Analytics Developer Guide.

Using the Amazon Kinesis Client Library (KCL)

You can develop a consumer application for Kinesis Streams using the KCL. Although you can use the Kinesis Streams API to get data from an Amazon Kinesis stream, we recommend using the design patterns and code for consumer applications provided by the KCL.

The KCL helps you consume and process data from a Kinesis stream. This type of application is also referred to as a consumer. 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 writing record-processing logic.

The KCL is a Java library; support for languages other than Java is provided using a multi-language interface. At run time, a KCL application instantiates a worker with configuration information, and then uses a record processor to process the data received from a Kinesis stream. You can run a KCL application on any number of instances. Multiple instances of the same application coordinate on failures and load-balance dynamically. You can also have multiple KCL applications working on the same stream, subject to throughput limits. The KCL acts as an intermediary between your record processing logic and Kinesis Streams.

For detailed information on how to build your own KCL application, refer to Developing Amazon Kinesis Streams Consumers Using the Amazon Kinesis Client Library[15].

Using AWS Lambda

AWS Lambda is a compute service that lets you run code without provisioning or managing servers.[16] AWS Lambda executes your code only when needed and scales automatically. With AWS Lambda, you can run code with zero administration. AWS Lambda runs your code on a high-availability compute infrastructure and performs all of the administration of the compute resources, including server and operating system maintenance, capacity provisioning and automatic scaling, and code monitoring and logging. All you need to do is supply your code in one of the languages that AWS Lambda supports.

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 then polls the stream periodically (once per second) for new records. When it detects new records, it invokes your Lambda function by passing the new records as a parameter. If no new records are detected, your Lambda function is not invoked.

For detailed information on using AWS Lambda to consume data from Kinesis Streams, refer to Using AWS Lambda with Amazon Kinesis.[17]

Using the API

For most use cases, you should use the KCL or AWS Lambda to retrieve and process data from a stream. However, if you prefer to write your own consumer application from scratch, there are several methods that enable this. The Kinesis Streams API provides the `GetShardIterator` and `GetRecords` methods to retrieve data from a stream. This is a pull model, where your code draws data directly from the shards of the stream. For more information about writing your own consumer application using the API, refer to Developing Amazon Kinesis Streams Consumers Using the Amazon Kinesis Streams API. Details about the API can be found in the Amazon Kinesis Streams API Reference.[18]

Choosing the Best Consumer Model for Your Application

How do you know which consumer model is best for your use case? Each approach has its own set of tradeoffs and you'll need to decide what's important to you. Here is some general guidance to help you choose the correct consumer model.

In most cases, consider starting with AWS Lambda. Its ease of use and the simple deployment model will enable you to quickly build a data consumer. The tradeoff to using AWS Lambda is that each invocation of your Lambda function should be considered stateless. That is, you can't easily use results from previous invocations of your function (e.g., earlier batches of records from your stream). Also consider that the maximum execution time for a single Lambda function is 5 minutes. If a single batch of records takes longer than 5 minutes to process, AWS Lambda might not be the best consumer for your use case.

If you decide that you can't use AWS Lambda, consider building your own processing application with the KCL. Because you deploy KCL applications to EC2 instances within your AWS account, you have a lot of flexibility and control in the local data persistence and state requirements for your data.

Your third option is to build your own application using the APIs directly. This gives you the most control and flexibility, but you also need to build your own logic to handle common consumer application features like checkpointing, scaling, and failover.

Summary

Kinesis Streams makes it easy to receive streaming data. You can scale a Kinesis stream to handle just a few records per second or millions of records per second. For ABC Tolls, the data rate streaming into their stream wasn't large. However, they benefitted from the direct integration with AWS Lambda, which enabled them to easily make API calls to Amazon SNS for user notifications.

Requirement 3: Other Threshold Alerts

The final requirement is similar to the previous one, but it introduces an additional problem. To recap this final requirement, the ABC Tolls operators want to be immediately notified when the vehicle traffic for a tolling station falls below a pre-defined threshold for each 30-minute period in a day. For example, they know from historical data that one of their tolling stations sees approximately 360 vehicles on Wednesdays between 2:00 pm and 2:30 pm. In that 30-minute window, if a tolling station sees fewer than 100 vehicles, they want to be notified.

ABC Tolls wants to compare current vehicle totals for each station with a known average rate for that station. To accomplish this, they created a file containing threshold traffic values for each 30-minute window, for each station. As described earlier, Kinesis Analytics supports the use of reference data. It will create an in-application stream (like a table) based on the data in a file stored in an S3 bucket. With this in place, ABC Tolls developers were able to write SQL in their Kinesis Analytics application to count the number of vehicles seen at each station over a 30-minute window and compare those values with the thresholds in the file. If the threshold was breached, Kinesis Analytics outputs a record to a Kinesis stream. When records arrive in the stream, a Lambda function is executed, which uses Amazon SNS to send a notification to AWS Tolls operators. Figure 8 illustrates the architecture for this scenario.

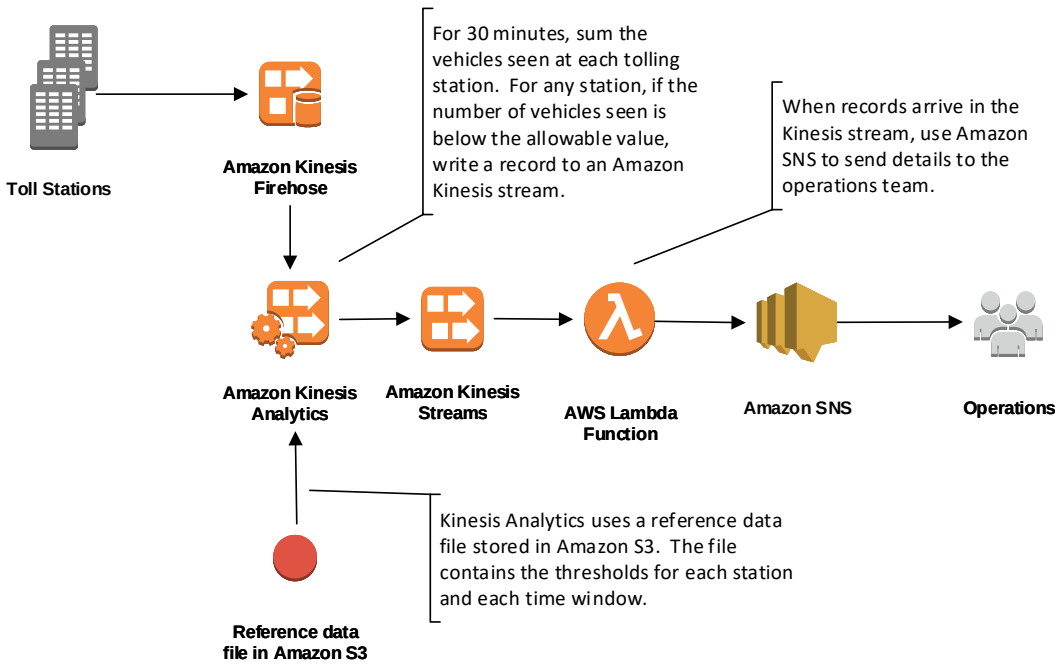

Figure 8: Architecture for alerts and notifications using 30 minute periods

Complete Architecture

With a solution to each requirement, we now have our overall streaming solution, as shown in Figure 9.

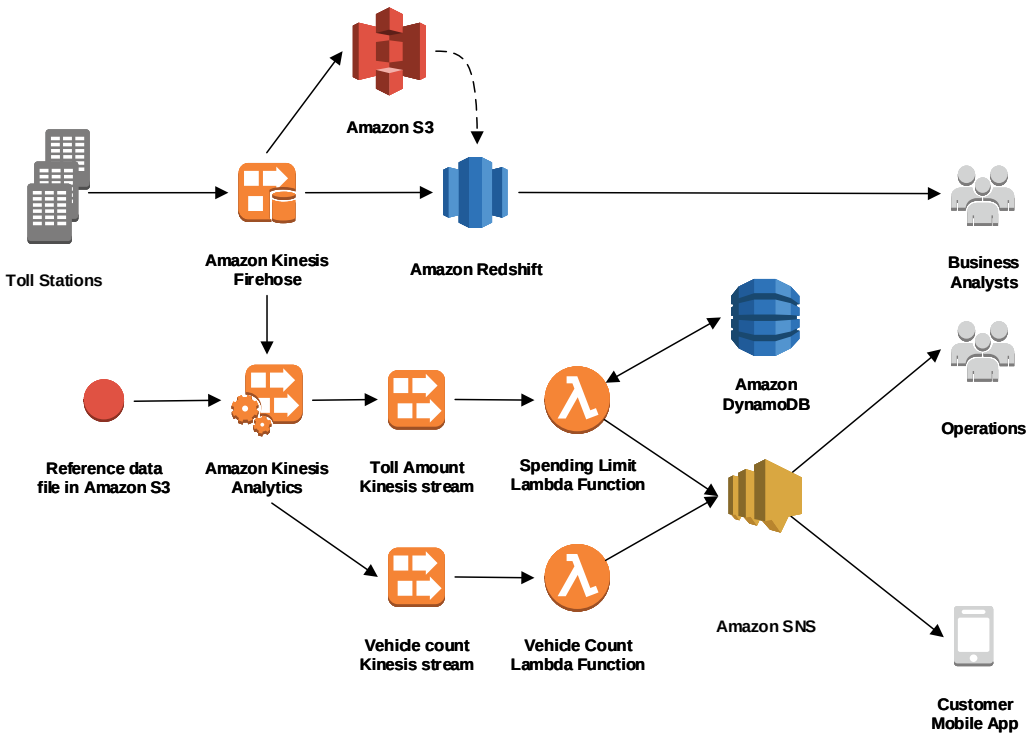

Figure 9: The architecture of the overall streaming solution

This design provides ABC Tolls with a flexible, reactive architecture. By streaming their customers' transactions in real-time, they are able to realize their requirements with very little development effort and minimal infrastructure to manage.

Conclusion

In this document, we reviewed how the fictitious company, ABC Tolls, used Amazon Kinesis services to move a traditional batch workflow to a streaming workflow. This migration provided them with the ability to add new features and functionality that weren't possible with their legacy batch solution.

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

Contributors

The following individuals and organizations contributed to this document:

- Allan MacInnis, Solutions Architect, AWS
- Chander Matrubhutam, Product Marketing Manager, AWS

Notes

[1] https://aws.amazon.com/kinesis/streams/

[2] https://aws.amazon.com/kinesis/firehose/

[3] https://aws.amazon.com/kinesis/analytics/

[4] http://docs.aws.amazon.com/firehose/latest/dev/writing-with-sdk.html

[5] https://aws.amazon.com/lambda/

[6] https://aws.amazon.com/s3/

[7] https://aws.amazon.com/redshift/

[8] https://aws.amazon.com/elasticsearch-service/

[9] https://aws.amazon.com/dynamodb/

[10] http://docs.aws.amazon.com/firehose/latest/dev/writing-with-agents.html

[11] http://docs.aws.amazon.com/streams/latest/dev/kinesis-kpl-writing.html

[12] http://docs.aws.amazon.com/streams/latest/dev/developing-producers-with-kpl.html

[13] http://docs.aws.amazon.com/streams/latest/dev/developing-producers-with-sdk.html

[14] http://docs.aws.amazon.com/kinesis/latest/APIReference/Welcome.html

[15] http://docs.aws.amazon.com/streams/latest/dev/developing-consumers-with-kcl.html

[16] https://aws.amazon.com/lambda/

[17] http://docs.aws.amazon.com/lambda/latest/dg/with-kinesis.html

[18] http://docs.aws.amazon.com/kinesis/latest/APIReference/Welcome.html