Amazon Kinesis Data Firehose

Developer Guide

Amazon Kinesis Data Firehose: Developer Guide

Table of Contents

What Is Amazon Kinesis Data Firehose?

Amazon Kinesis Data Firehose is a fully managed service for delivering real-time streaming data to destinations such as Amazon Simple Storage Service (Amazon S3), Amazon Redshift, Amazon OpenSearch Service (Amazon ES), Splunk, and any custom HTTP endpoint or HTTP endpoints owned by supported third-party service providers, including Datadog, Dynatrace, LogicMonitor, MongoDB, New Relic, and Sumo Logic. Kinesis Data Firehose is part of the Kinesis streaming data platform, along with Kinesis Data Streams, Kinesis Video Streams, and Amazon Kinesis Data Analytics. With Kinesis Data Firehose, you don't need to write applications or manage resources. You configure your data producers to send data to Kinesis Data Firehose, and it automatically delivers the data to the destination that you specified. You can also configure Kinesis Data Firehose to transform your data before delivering it.

For more information about AWS big data solutions, see Big Data on AWS. For more information about AWS streaming data solutions, see What is Streaming Data?

> **Note**
> Note the latest AWS Streaming Data Solution for Amazon MSK that provides AWS CloudFormation templates where data flows through producers, streaming storage, consumers, and destinations.

Key Concepts

As you get started with Kinesis Data Firehose, you can benefit from understanding the following concepts:

Kinesis Data Firehose delivery stream

The underlying entity of Kinesis Data Firehose. You use Kinesis Data Firehose by creating a Kinesis Data Firehose delivery stream and then sending data to it. For more information, see Creating an Amazon Kinesis Data Firehose Delivery Stream (p. 5) and Sending Data to an Amazon Kinesis Data Firehose Delivery Stream (p. 24).

record

The data of interest that your data producer sends to a Kinesis Data Firehose delivery stream. A record can be as large as 1,000 KB.

data producer

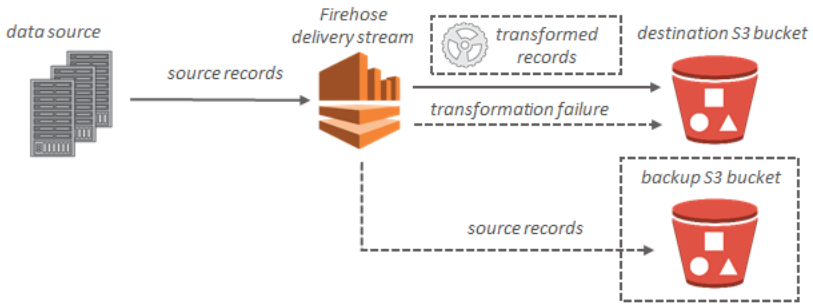
Producers send records to Kinesis Data Firehose delivery streams. For example, a web server that sends log data to a delivery stream is a data producer. You can also configure your Kinesis Data Firehose delivery stream to automatically read data from an existing Kinesis data stream, and load it into destinations. For more information, see Sending Data to an Amazon Kinesis Data Firehose Delivery Stream (p. 24).

buffer size and buffer interval

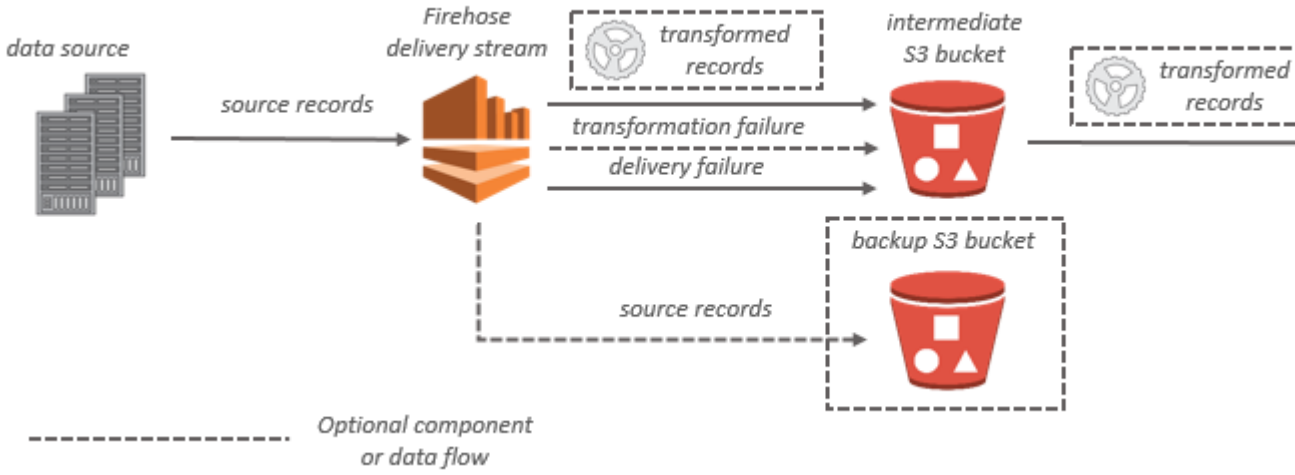
Kinesis Data Firehose buffers incoming streaming data to a certain size or for a certain period of time before delivering it to destinations. **Buffer Size** is in MBs and **Buffer Interval** is in seconds.

Data Flow

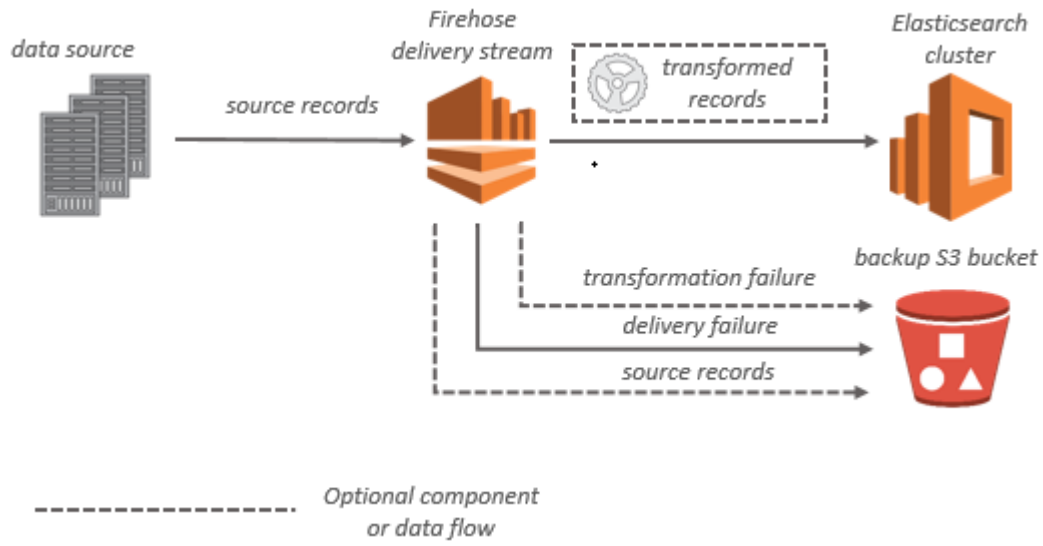
For Amazon S3 destinations, streaming data is delivered to your S3 bucket. If data transformation is enabled, you can optionally back up source data to another Amazon S3 bucket.



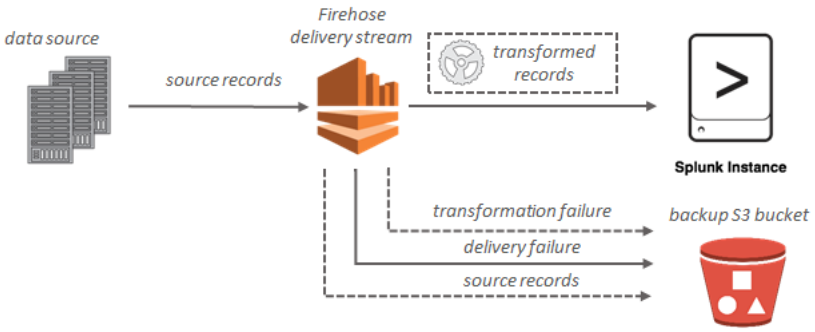
For Amazon Redshift destinations, streaming data is delivered to your S3 bucket first. Kinesis Data Firehose then issues an Amazon Redshift **COPY** command to load data from your S3 bucket to your Amazon Redshift cluster. If data transformation is enabled, you can optionally back up source data to another Amazon S3 bucket.



For Amazon ES destinations, streaming data is delivered to your Amazon ES cluster, and it can optionally be backed up to your S3 bucket concurrently.



For Splunk destinations, streaming data is delivered to Splunk, and it can optionally be backed up to your S3 bucket concurrently.



Setting Up for Amazon Kinesis Data Firehose

Before you use Kinesis Data Firehose for the first time, complete the following tasks.

Tasks

Sign Up for AWS

When you sign up for Amazon Web Services (AWS), your AWS account is automatically signed up for all services in AWS, including Kinesis Data Firehose. You are charged only for the services that you use.

If you have an AWS account already, skip to the next task. If you don't have an AWS account, use the following procedure to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

Optional: Download Libraries and Tools

The following libraries and tools will help you work with Kinesis Data Firehose programmatically and from the command line:

- The Amazon Kinesis Data Firehose API Reference is the basic set of operations that Kinesis Data Firehose supports.
- The AWS SDKs for Go, Java, .NET, Node.js, Python, and Ruby include Kinesis Data Firehose support and samples.

  If your version of the AWS SDK for Java does not include samples for Kinesis Data Firehose, you can also download the latest AWS SDK from GitHub.
- The AWS Command Line Interface supports Kinesis Data Firehose. The AWS CLI enables you to control multiple AWS services from the command line and automate them through scripts.

Creating an Amazon Kinesis Data Firehose Delivery Stream

You can use the AWS Management Console or an AWS SDK to create a Kinesis Data Firehose delivery stream to your chosen destination.

You can update the configuration of your delivery stream at any time after it's created, using the Kinesis Data Firehose console or UpdateDestination. Your Kinesis Data Firehose delivery stream remains in the `ACTIVE` state while your configuration is updated, and you can continue to send data. The updated configuration normally takes effect within a few minutes. The version number of a Kinesis Data Firehose delivery stream is increased by a value of `1` after you update the configuration. It is reflected in the delivered Amazon S3 object name. For more information, see Amazon S3 Object Name Format (p. 76).

The following topics describe how to create a Kinesis Data Firehose delivery stream:

Topics

Source, Destination, and Name

1. Sign in to the AWS Management Console and open the Kinesis console at https://console.aws.amazon.com/kinesis.
2. Choose **Data Firehose** in the navigation pane.
3. Choose **Create delivery stream**.
4. Enter values for the following fields:

   Source

   - **Direct PUT or other sources:** Choose this option to create a Kinesis Data Firehose delivery stream that producer applications write to directly.
   - **Kinesis stream:** Choose this option to configure a Kinesis Data Firehose delivery stream that uses a Kinesis data stream as a data source. You can then use Kinesis Data Firehose to read data easily from an existing Kinesis data stream and load it into destinations. For more information about using Kinesis Data Streams as your data source, see Writing to Amazon Kinesis Data Firehose Using Kinesis Data Streams.

   Delivery stream destination

   The destination of your Kinesis Data Firehose delivery stream. Kinesis Data Firehose can send data records to various destinations, including Amazon Simple Storage Service (Amazon S3), Amazon Redshift, Amazon Elastic Search Service, and any HTTP endpoint that is owned by you or any of your third-party service providers. The following are the supported destinations:

   - Amazon Elastic Search Service
   - Amazon S3
   - Datadog
   - Dynatrace
   - HTTP Endpoint

- Logic Monitor
- MongoDB Cloud
- New Relic
- Splunk
- Sumo Logic

Delivery stream name

The name of your Kinesis Data Firehose delivery stream.

Record Transformation and Record Format Conversion

Configure Kinesis Data Firehose to transform and convert your record data.

1. In the **Transform source records with AWS Lambda** section, provide values for the following field:

   Data transformation

   To create a Kinesis Data Firehose delivery stream that doesn't transform incoming data, choose **Disabled**.

   To specify a Lambda function for Kinesis Data Firehose to invoke and use to transform incoming data before delivering it, choose **Enabled**. You can configure a new Lambda function using one of the Lambda blueprints or choose an existing Lambda function. Your Lambda function must contain the status model that is required by Kinesis Data Firehose. For more information, see Amazon Kinesis Data Firehose Data Transformation (p. 58).
2. In the **Convert record format** section, provide values for the following field:

   Record format conversion

   To create a Kinesis Data Firehose delivery stream that doesn't convert the format of the incoming data records, choose **Disabled**.

   To convert the format of the incoming records, choose **Enabled**, then specify the output format you want. You need to specify an AWS Glue table that holds the schema that you want Kinesis Data Firehose to use to convert your record format. For more information, see *Record Format Conversion* (p. 68).

   For an example of how to set up record format conversion with AWS CloudFormation, see AWS::KinesisFirehose::DeliveryStream.

Destination Settings

This topic describes the destination settings for your delivery stream.

Topics

Choose Amazon S3 for Your Destination

You must specify the following settings in order to use Amazon S3 as the destination for your Kinesis Data Firehose delivery stream:

- Enter values for the following fields:

  S3 bucket

  Choose an S3 bucket that you own where the streaming data should be delivered. You can create a new S3 bucket or choose an existing one.

  S3 bucket prefix - optional

  If you don't enable dynamic partitioning, this is an optional field. If you choose to enable dynamic partitioning, you MUST specify an S3 error bucket prefix for Kinesis Data Firehose to use when delivering data to Amazon S3 in error conditions. If Kinesis Data Firehose fails to dynamically partition your incoming data, those data records are delivered to this S3 error bucket prefix. For more information, see Amazon S3 Object Name Format (p. 76) and *Custom Amazon S3 Prefixes* (p. 105)

  Dynamic partitioning

  Choose **Enabled** to enable and configure dynamic partitioning.

  Multi record deagreggation

  This is the process of parsing through the records in the delivery stream and separating them based either on valid JSON or on the specified new line delimiter.

  If you aggregate multiple events, logs, or records into a single PutRecord and PutRecordBatch API call, you can still enable and configure dynamic partitioning. With aggregated data, when you enable dynamic partitioning, Kinesis Data Firehose parses the records and looks for multiple valid JSON objects within each API call. When the delivery stream is configured with Kinesis Data Stream as a source, you can also use the built-in aggregation in the Kinesis Producer Library (KPL). Data partition functionality is executed after data is de-aggregated. Therefore, each record in each API call can be delivered to different Amazon S3 prefixes. You can also leverage the Lambda function integration to perform any other de-aggregation or any other transformation before the data partitioning functionality.

  Important
  If your data is aggregated, dynamic partitioning can be applied only after data deaggregation is performed. So if you enable dynamic partitioning to your aggregated data, you must choose **Enabled** to enable multi record deaggregation.

  Kinesis Data Firehose delivery stream preforms the following processing steps in the following order: KPL (protobuf) de-aggregation, Lambda processing, JSON de-aggregation, data partitioning, data format conversion, and Amazon S3 delivery. [Internal: Add UX in the appendix with a diagram of the processing flow]

Multi record deaggreation type

If you enabled multi recrord deaggregation, you must specify the method for Kinesis Data Firehose to deaggregate your data. Use the drop-down menu to choose either **JSON** or **Delimited**.

New line delimiter

When you enable dynamic partitioning, you can configure your delivery stream to add a new line delimiter between records in objects that are delivered to Amazon S3. To do so, choose **Enabled**. To not add a new line delimiter between records in objects that are delivered to Amazon S3, choose **Disabled**.

Inline parsing

This is one of the supported mechanisms to dynamically partition your data that is bound for Amazon S3. To use inline parsing for dynamic partitioning of your data, you must specify data record parameters to be used as partitioning keys and provide a value for each specified partitioning key. Choose **Enabled** to enable and configure inline parsing.

Important
If you specified an AWS Lambda function in the steps above for transforming your source records, you can use this function to dyncamically partition your data that is bound to S3 and you can still create your partitioning keys with inline parsing. With dynamic partitioning, you can use either inline parsing or your AWS Lambda function to create your partitioning keys. Or you can use both inline parsing and your AWS Lambda function at the same time to create your partitioning keys.

Dynamic partitioning keys

You can use the **Key** and **Value** fields to specify the data record parameters to be used as dynamic partitioning keys and jq queries to generate dynamic partitioning key values. Kinesis Data Firehose supports jq 1.6 only. You can specify up to 50 dynamic partitioning keys. You must enter valid jq expressions for your dynamic partitioning key values in order to successfully configure dynamic partitioning for your delivery stream.

S3 bucket prefix

When you enable and configure dynamic partitioning, you must specify the S3 bucket prefixes to which Kinesis Data Firehose is to deliver partitioned data.

In order for dynamic partitioning to be configured correctly, the number of the S3 bucket prefixes must be identical to the number of the specified partitioning keys.

You can partition your source data with inline parsing or with your specified AWS Lambda function. If you specified an AWS Lambda function to create partitioning keys for your source data, you must manually type in the S3 bucket prefix value(s) using the following format: "partitionKeyFromLambda:keyID". If you are using inline parsing to specify the partitioning keys for your source data, you can either manually type in the S3 bucket preview values using the following format: "partitionKeyFromQuery:keyID" or you can choose the **Apply dynamic partitioning keys** button to use your dynamic partitioning key/value pairs to auto-generate your S3 bucket prefixes. While partitioning your data with either inline parsing or AWS Lambda, you can also use the following expression forms in your S3 bucket prefix: !{namespace:value}, where namespace can be either partitionKeyFromQuery or partitionKeyFromLambda.

S3 buffer hints

Kinesis Data Firehose buffers incoming data before delivering it to the specified destination. The recommended buffer size for the destination varies from service provider to service provider.

S3 compressions and encryption

Choose GZIP, Snappy, Zip, or Hadoop-Compatible Snappy data compression, or no data compression. Snappy, Zip, and Hadoop-Compatible Snappy compression is not available for delivery streams with Amazon Redshift as the destination.

Kinesis Data Firehose supports Amazon S3 server-side encryption with AWS Key Management Service (AWS KMS) for encrypting delivered data in Amazon S3. You can choose to not encrypt the data or to encrypt with a key from the list of AWS KMS keys that you own. For more information, see Protecting Data Using Server-Side Encryption with AWS KMS–Managed Keys (SSE-KMS).

Choose Amazon Redshift for Your Destination

This section describes settings for using Amazon Redshift as your delivery stream destination.

- Enter values for the following fields:

  Cluster

  The Amazon Redshift cluster to which S3 bucket data is copied. Configure the Amazon Redshift cluster to be publicly accessible and unblock Kinesis Data Firehose IP addresses. For more information, see Grant Kinesis Data Firehose Access to an Amazon Redshift Destination (p. 43).

  User name

  An Amazon Redshift user with permissions to access the Amazon Redshift cluster. This user must have the Amazon Redshift `INSERT` permission for copying data from the S3 bucket to the Amazon Redshift cluster.

  Password

  The password for the user who has permissions to access the cluster.

  Database

  The Amazon Redshift database to where the data is copied.

  Table

  The Amazon Redshift table to where the data is copied.

  Columns

  (Optional) The specific columns of the table to which the data is copied. Use this option if the number of columns defined in your Amazon S3 objects is less than the number of columns within the Amazon Redshift table.

  Intermediate S3 destination

  Kinesis Data Firehose delivers your data to your S3 bucket first and then issues an Amazon Redshift **COPY** command to load the data into your Amazon Redshift cluster. Specify an S3 bucket that you own where the streaming data should be delivered. Create a new S3 bucket, or choose an existing bucket that you own.

  Kinesis Data Firehose doesn't delete the data from your S3 bucket after loading it to your Amazon Redshift cluster. You can manage the data in your S3 bucket using a lifecycle configuration. For more information, see Object Lifecycle Management in the *Amazon Simple Storage Service User Guide*.

Intermediate S3 prefix

(Optional) To use the default prefix for Amazon S3 objects, leave this option blank. Kinesis Data Firehose automatically uses a prefix in "`YYYY/MM/dd/HH`" UTC time format for delivered Amazon S3 objects. You can add to the start of this prefix. For more information, see Amazon S3 Object Name Format (p. 76).

COPY options

Parameters that you can specify in the Amazon Redshift **COPY** command. These might be required for your configuration. For example, "`GZIP`" is required if Amazon S3 data compression is enabled. "`REGION`" is required if your S3 bucket isn't in the same AWS Region as your Amazon Redshift cluster. For more information, see COPY in the *Amazon Redshift Database Developer Guide*.

COPY command

The Amazon Redshift **COPY** command. For more information, see COPY in the *Amazon Redshift Database Developer Guide*.

Retry duration

Time duration (0–7200 seconds) for Kinesis Data Firehose to retry if data **COPY** to your Amazon Redshift cluster fails. Kinesis Data Firehose retries every 5 minutes until the retry duration ends. If you set the retry duration to 0 (zero) seconds, Kinesis Data Firehose does not retry upon a **COPY** command failure.

S3 buffer hints

Kinesis Data Firehose buffers incoming data before delivering it to the specified destination. The recommended buffer size for the destination varies from service provider to service provider.

S3 compressions and encryption

Choose GZIP or no data compression.

Kinesis Data Firehose supports Amazon S3 server-side encryption with AWS Key Management Service (AWS KMS) for encrypting delivered data in Amazon S3. You can choose to not encrypt the data or to encrypt with a key from the list of AWS KMS keys that you own. For more information, see Protecting Data Using Server-Side Encryption with AWS KMS–Managed Keys (SSE-KMS).

Choose OpenSearch Service for Your Destination

This section describes options for using OpenSearch Service for your destination.

- Enter values for the following fields:

  Elasticsearch domain

  The Amazon ES domain to which your data is delivered.

  Index

  The Elasticsearch index name to be used when indexing data to your Amazon ES cluster.

  Index rotation

  Choose whether and how often the Elasticsearch index should be rotated. If index rotation is enabled, Kinesis Data Firehose appends the corresponding timestamp to the specified index name and rotates. For more information, see Index Rotation for the Amazon ES Destination (p. 77).

Type

The OpenSearch Service type name to be used when indexing data to your Amazon ES cluster. For Elasticsearch 6.x, there can be only one type per index. If you try to specify a new type for an existing index that already has another type, Kinesis Data Firehose returns an error during runtime.

For Elasticsearch 7.x, leave this field empty.

Retry duration

Time duration (0–7200 seconds) for Kinesis Data Firehose to retry if an index request to your Amazon ES cluster fails. Kinesis Data Firehose retries every 5 minutes until the retry duration ends. If you set the retry duration to 0 (zero) seconds, Kinesis Data Firehose does not retry upon an index request failure.

Destination VPC connectivity

If your OpenSearch Service domain is in a private VPC, use this section to specify that VPC. Also specify the subnets and subgroups that you want Kinesis Data Firehose to use when it sends data to your OpenSearch Service domain. You can use the same security group that the OpenSearch Service domain uses or different ones. If you specify different security groups, ensure that they allow outbound HTTPS traffic to the OpenSearch Service domain's security group. Also ensure that the OpenSearch Service domain's security group allows HTTPS traffic from the security groups that you specified when you configured your delivery stream. If you use the same security group for both your delivery stream and the OpenSearch Service domain, make sure the security group's inbound rule allows HTTPS traffic. For more information about security group rules, see Security group rules in the Amazon VPC documentation.

Buffer hints

Kinesis Data Firehose buffers incoming data before delivering it to the specified destination. The recommended buffer size for the destination varies from service provider to service provider.

Choose HTTP Endpoint for Your Destination

This section describes options for using **HTTP endpoint** for your destination.

Important
If you choose an HTTP endpoint as your destination, review and follow the instructions in Appendix - HTTP Endpoint Delivery Request and Response Specifications (p. 133).

- Provide values for the following fields:

  HTTP endpoint name - optional

  Specify a user friendly name for the HTTP endpoint. For example, `My HTTP Endpoint Destination`.

  HTTP endpoint URL

  Specify the URL for the HTTP endpoint in the following format: `https://xyz.httpendpoint.com`. The URL must be an HTTPS URL.

  Access key - optional

  Contact the endpoint owner to obtain the access key (if it is required) to enable data delivery to their endpoint from Kinesis Data Firehose.

Content encoding

Kinesis Data Firehose uses content encoding to compress the body of a request before sending it to the destination. Choose **GZIP** or **Disabled** to enable/disable content encoding of your request.

Retry duration

Specify how long Kinesis Data Firehose retries sending data to the selected HTTP endpoint.

After sending data, Kinesis Data Firehose first waits for an acknowledgment from the HTTP endpoint. If an error occurs or the acknowledgment doesn't arrive within the acknowledgment timeout period, Kinesis Data Firehose starts the retry duration counter. It keeps retrying until the retry duration expires. After that, Kinesis Data Firehose considers it a data delivery failure and backs up the data to your Amazon S3 bucket.

Every time that Kinesis Data Firehose sends data to the HTTP endpoint (either the initial attempt or a retry), it restarts the acknowledgement timeout counter and waits for an acknowledgement from the HTTP endpoint.

Even if the retry duration expires, Kinesis Data Firehose still waits for the acknowledgment until it receives it or the acknowledgement timeout period is reached. If the acknowledgment times out, Kinesis Data Firehose determines whether there's time left in the retry counter. If there is time left, it retries again and repeats the logic until it receives an acknowledgment or determines that the retry time has expired.

If you don't want Kinesis Data Firehose to retry sending data, set this value to 0.

Parameters - optional

Kinesis Data Firehose includes these key-value pairs in each HTTP call. These parameters can help you identify and organize your destinations.

S3 buffer hints

Kinesis Data Firehose buffers incoming data before delivering it to the specified destination. The recommended buffer size for the destination varies from service provider to service provider.

Important
For the HTTP endpoint destinations, if you are seeing 413 response codes from the destination endpoint in CloudWatch Logs, lower the buffering hint size on your delivery stream and try again.

Choose Datadog for Your Destination

This section describes options for using **Datadog** for your destination. For more information about Datadog, see https://docs.datadoghq.com/integrations/amazon_web_services/.

- Provide values for the following fields:

  HTTP endpoint URL

  Choose the HTTP endpoint URL from the following options in the drop down menu:
  - **Datadog logs - US**
  - **Datadog logs - EU**
  - **Datadog logs - GOV**
  - **Datadog metrics - US**
  - **Datadog metrics - EU**

API key

Contact Datadog to obtain the API key required to enable data delivery to this endpoint from Kinesis Data Firehose.

Content encoding

Kinesis Data Firehose uses content encoding to compress the body of a request before sending it to the destination. Choose **GZIP** or **Disabled** to enable/disable content encoding of your request.

Retry duration

Specify how long Kinesis Data Firehose retries sending data to the selected HTTP endpoint.

After sending data, Kinesis Data Firehose first waits for an acknowledgment from the HTTP endpoint. If an error occurs or the acknowledgment doesn't arrive within the acknowledgment timeout period, Kinesis Data Firehose starts the retry duration counter. It keeps retrying until the retry duration expires. After that, Kinesis Data Firehose considers it a data delivery failure and backs up the data to your Amazon S3 bucket.

Every time that Kinesis Data Firehose sends data to the HTTP endpoint (either the initial attempt or a retry), it restarts the acknowledgement timeout counter and waits for an acknowledgement from the HTTP endpoint.

Even if the retry duration expires, Kinesis Data Firehose still waits for the acknowledgment until it receives it or the acknowledgement timeout period is reached. If the acknowledgment times out, Kinesis Data Firehose determines whether there's time left in the retry counter. If there is time left, it retries again and repeats the logic until it receives an acknowledgment or determines that the retry time has expired.

If you don't want Kinesis Data Firehose to retry sending data, set this value to 0.

Parameters - optional

Kinesis Data Firehose includes these key-value pairs in each HTTP call. These parameters can help you identify and organize your destinations.

S3 buffer hints

Kinesis Data Firehose buffers incoming data before delivering it to the specified destination. The recommended buffer size for the destination varies from service provider to service provider.

Choose Dynatrace for Your Destination

This section describes options for using **Dynatrace** for your destination. For more information, see https://www.dynatrace.com/support/help/technology-support/cloud-platforms/amazon-web-services/integrations/cloudwatch-metric-streams/.

- Provide values for the following fields:

  HTTP endpoint URL

  Choose the HTTP endpoint URL (**Dynatrace US**, **Dynatrace EU**, or **Dynatrace Global**) from the drop down menu.

  API token

  Generate the Dynatrace API token required for data delivery from Kinesis Data Firehose. For more information, see https://www.dynatrace.com/support/help/dynatrace-api/basics/dynatrace-api-authentication/.

API URL

Provide the API URL of your Dynatrace environment.

Content encoding

Kinesis Data Firehose uses content encoding to compress the body of a request before sending it to the destination. Choose **GZIP** or **Disabled** to enable/disable content encoding of your request.

Retry duration

Specify how long Kinesis Data Firehose retries sending data to the selected HTTP endpoint.

After sending data, Kinesis Data Firehose first waits for an acknowledgment from the HTTP endpoint. If an error occurs or the acknowledgment doesn't arrive within the acknowledgment timeout period, Kinesis Data Firehose starts the retry duration counter. It keeps retrying until the retry duration expires. After that, Kinesis Data Firehose considers it a data delivery failure and backs up the data to your Amazon S3 bucket.

Every time that Kinesis Data Firehose sends data to the HTTP endpoint (either the initial attempt or a retry), it restarts the acknowledgement timeout counter and waits for an acknowledgement from the HTTP endpoint.

Even if the retry duration expires, Kinesis Data Firehose still waits for the acknowledgment until it receives it or the acknowledgement timeout period is reached. If the acknowledgment times out, Kinesis Data Firehose determines whether there's time left in the retry counter. If there is time left, it retries again and repeats the logic until it receives an acknowledgment or determines that the retry time has expired.

If you don't want Kinesis Data Firehose to retry sending data, set this value to 0.

Parameters - optional

Kinesis Data Firehose includes these key-value pairs in each HTTP call. These parameters can help you identify and organize your destinations.

> **Important**
> When using Dynatrace as your specified destination, you must specify at least one parameter key-value pair. You must name this key `dt-url` and set its value to the URL of your Dynatrace environment (for example, `https://xyzab123456.dynatrace.live.com`). You can then optionally specify additional parameter key-value pairs and set them to custom names and values of your choosing.

S3 buffer hints

Kinesis Data Firehose buffers incoming data before delivering it to the specified destination. The recommended buffer size for the destination varies from service provider to service provider.

Choose LogicMonitor for Your Destination

This section describes options for using **LogicMonitor** for your destination. For more information, see https://www.logicmonitor.com.

- Provide values for the following fields:

  HTTP endpoint URL

  Specify the URL for the HTTP endpoint in the following format: https://ACCOUNT.logicmonitor.com

API key

Contact LogicMonitor to obtain the API key required to enable data delivery to this endpoint from Kinesis Data Firehose.

Content encoding

Kinesis Data Firehose uses content encoding to compress the body of a request before sending it to the destination. Choose **GZIP** or **Disabled** to enable/disable content encoding of your request.

Retry duration

Specify how long Kinesis Data Firehose retries sending data to the selected HTTP endpoint.

After sending data, Kinesis Data Firehose first waits for an acknowledgment from the HTTP endpoint. If an error occurs or the acknowledgment doesn't arrive within the acknowledgment timeout period, Kinesis Data Firehose starts the retry duration counter. It keeps retrying until the retry duration expires. After that, Kinesis Data Firehose considers it a data delivery failure and backs up the data to your Amazon S3 bucket.

Every time that Kinesis Data Firehose sends data to the HTTP endpoint (either the initial attempt or a retry), it restarts the acknowledgement timeout counter and waits for an acknowledgement from the HTTP endpoint.

Even if the retry duration expires, Kinesis Data Firehose still waits for the acknowledgment until it receives it or the acknowledgement timeout period is reached. If the acknowledgment times out, Kinesis Data Firehose determines whether there's time left in the retry counter. If there is time left, it retries again and repeats the logic until it receives an acknowledgment or determines that the retry time has expired.

If you don't want Kinesis Data Firehose to retry sending data, set this value to 0.

Parameters - optional

Kinesis Data Firehose includes these key-value pairs in each HTTP call. These parameters can help you identify and organize your destinations.

S3 buffer hints

Kinesis Data Firehose buffers incoming data before delivering it to the specified destination. The recommended buffer size for the destination varies from service provider to service provider.

Choose MongoDB Cloud for Your Destination

This section describes options for using **MongoDB Cloud** for your destination. For more information, see https://www.mongodb.com.

- Provide values for the following fields:

  MongoDB Realm webhook URL

  Specify the URL for the HTTP endpoint in the following format: `https://webhooks.mongodb-realm.com`. The URL must be an HTTPS URL.

  API key

  Contact MongoDB Cloud to obtain the API key required to enable data delivery to this endpoint from Kinesis Data Firehose.

Content encoding

Kinesis Data Firehose uses content encoding to compress the body of a request before sending it to the destination. Choose **GZIP** or **Disabled** to enable/disable content encoding of your request.

Retry duration

Specify how long Kinesis Data Firehose retries sending data to the selected third-party provider.

After sending data, Kinesis Data Firehose first waits for an acknowledgment from the HTTP endpoint. If an error occurs or the acknowledgment doesn't arrive within the acknowledgment timeout period, Kinesis Data Firehose starts the retry duration counter. It keeps retrying until the retry duration expires. After that, Kinesis Data Firehose considers it a data delivery failure and backs up the data to your Amazon S3 bucket.

Every time that Kinesis Data Firehose sends data to the HTTP endpoint (either the initial attempt or a retry), it restarts the acknowledgement timeout counter and waits for an acknowledgement from the HTTP endpoint.

Even if the retry duration expires, Kinesis Data Firehose still waits for the acknowledgment until it receives it or the acknowledgement timeout period is reached. If the acknowledgment times out, Kinesis Data Firehose determines whether there's time left in the retry counter. If there is time left, it retries again and repeats the logic until it receives an acknowledgment or determines that the retry time has expired.

If you don't want Kinesis Data Firehose to retry sending data, set this value to 0.

S3 buffer hints

Kinesis Data Firehose buffers incoming data before delivering it to the specified destination. The recommended buffer size for the destination varies from service provider to service provider.

Parameters - optional

Kinesis Data Firehose includes these key-value pairs in each HTTP call. These parameters can help you identify and organize your destinations.

Choose New Relic for Your Destination

This section describes options for using **New Relic** for your destination. For more information, see https://newrelic.com.

- Provide values for the following fields:

  HTTP endpoint URL

  Choose the HTTP endpoint URL from the following options in the drop down menu:

  - **New Relic logs - US**
  - **New Relic metrics - US**
  - **New Relic metrics - EU**

  API key

  Enter your License Key (40-characters hexadecimal string) from your New Relic One Account settings. This API key is required to enable data delivery to this endpoint from Kinesis Data Firehose.

Content encoding

Kinesis Data Firehose uses content encoding to compress the body of a request before sending it to the destination. Choose **GZIP** or **Disabled** to enable/disable content encoding of your request.

Retry duration

Specify how long Kinesis Data Firehose retries sending data to the New Relic HTTP endpoint.

After sending data, Kinesis Data Firehose first waits for an acknowledgment from the HTTP endpoint. If an error occurs or the acknowledgment doesn't arrive within the acknowledgment timeout period, Kinesis Data Firehose starts the retry duration counter. It keeps retrying until the retry duration expires. After that, Kinesis Data Firehose considers it a data delivery failure and backs up the data to your Amazon S3 bucket.

Every time that Kinesis Data Firehose sends data to the HTTP endpoint (either the initial attempt or a retry), it restarts the acknowledgement timeout counter and waits for an acknowledgement from the HTTP endpoint.

Even if the retry duration expires, Kinesis Data Firehose still waits for the acknowledgment until it receives it or the acknowledgement timeout period is reached. If the acknowledgment times out, Kinesis Data Firehose determines whether there's time left in the retry counter. If there is time left, it retries again and repeats the logic until it receives an acknowledgment or determines that the retry time has expired.

If you don't want Kinesis Data Firehose to retry sending data, set this value to 0.

Parameters - optional

Kinesis Data Firehose includes these key-value pairs in each HTTP call. These parameters can help you identify and organize your destinations.

S3 buffer hints

Kinesis Data Firehose buffers incoming data before delivering it to the specified destination. The recommended buffer size for the destination varies from service provider to service provider.

Choose Splunk for Your Destination

This section describes options for using Splunk for your destination.

- Provide values for the following fields:

  Splunk cluster endpoint

  To determine the endpoint, see Configure Amazon Kinesis Firehose to Send Data to the Splunk Platform in the Splunk documentation.

  Splunk endpoint type

  Choose `Raw endpoint` in most cases. Choose `Event endpoint` if you preprocessed your data using AWS Lambda to send data to different indexes by event type. For information about what endpoint to use, see Configure Amazon Kinesis Firehose to send data to the Splunk platform in the Splunk documentation.

  Authentication token

  To set up a Splunk endpoint that can receive data from Kinesis Data Firehose, see Installation and configuration overview for the Splunk Add-on for Amazon Kinesis Firehose in the Splunk

documentation. Save the token that you get from Splunk when you set up the endpoint for this delivery stream, and add it here.

HEC acknowledgement timeout

Specify how long Kinesis Data Firehose waits for the index acknowledgement from Splunk. If Splunk doesn't send the acknowledgment before the timeout is reached, Kinesis Data Firehose considers it a data delivery failure. Kinesis Data Firehose then either retries or backs up the data to your Amazon S3 bucket, depending on the retry duration value that you set.

Retry duration

Specify how long Kinesis Data Firehose retries sending data to Splunk.

After sending data, Kinesis Data Firehose first waits for an acknowledgment from Splunk. If an error occurs or the acknowledgment doesn't arrive within the acknowledgment timeout period, Kinesis Data Firehose starts the retry duration counter. It keeps retrying until the retry duration expires. After that, Kinesis Data Firehose considers it a data delivery failure and backs up the data to your Amazon S3 bucket.

Every time that Kinesis Data Firehose sends data to Splunk (either the initial attempt or a retry), it restarts the acknowledgement timeout counter and waits for an acknowledgement from Splunk.

Even if the retry duration expires, Kinesis Data Firehose still waits for the acknowledgment until it receives it or the acknowledgement timeout period is reached. If the acknowledgment times out, Kinesis Data Firehose determines whether there's time left in the retry counter. If there is time left, it retries again and repeats the logic until it receives an acknowledgment or determines that the retry time has expired.

If you don't want Kinesis Data Firehose to retry sending data, set this value to 0.

Choose Sumo Logic for Your Destination

This section describes options for using **Sumo Logic** for your destination. For more information, see https://www.sumologic.com.

- Provide values for the following fields:

  HTTP endpoint URL

  Specify the URL for the HTTP endpoint in the following format: `https://deployment name.sumologic.net/receiver/v1/kinesis/dataType/access token`. The URL must be an HTTPS URL.

  Content encoding

  Kinesis Data Firehose uses content encoding to compress the body of a request before sending it to the destination. Choose **GZIP** or **Disabled** to enable/disable content encoding of your request.

  Retry duration

  Specify how long Kinesis Data Firehose retries sending data to New Relic.

  After sending data, Kinesis Data Firehose first waits for an acknowledgment from the HTTP endpoint. If an error occurs or the acknowledgment doesn't arrive within the acknowledgment timeout period, Kinesis Data Firehose starts the retry duration counter. It keeps retrying until the retry duration expires. After that, Kinesis Data Firehose considers it a data delivery failure and backs up the data to your Amazon S3 bucket.

Every time that Kinesis Data Firehose sends data to the HTTP endpoint (either the initial attempt or a retry), it restarts the acknowledgement timeout counter and waits for an acknowledgement from the HTTP endpoint.

Even if the retry duration expires, Kinesis Data Firehose still waits for the acknowledgment until it receives it or the acknowledgement timeout period is reached. If the acknowledgment times out, Kinesis Data Firehose determines whether there's time left in the retry counter. If there is time left, it retries again and repeats the logic until it receives an acknowledgment or determines that the retry time has expired.

If you don't want Kinesis Data Firehose to retry sending data, set this value to 0.

Parameters - optional

Kinesis Data Firehose includes these key-value pairs in each HTTP call. These parameters can help you identify and organize your destinations.

S3 buffer hints

Kinesis Data Firehose buffers incoming data before delivering it to the specified destination. The recommended buffer size for the destination varies from service provider to service provider.

Backup and Advanced Settings

This topic describes how to configure the backup and the advanced settings for your Kinesis Data Firehose delivery stream.

Backup Settings

Kinesis Data Firehose uses Amazon S3 to backup all or failed only data that it attempts to deliver to your chosen destination. You can specify the S3 backup settings for your Kinesis Data Firehose delivery stream if you made one of the following choices:

- If you set Amazon S3 as the destination for your Kinesis Data Firehose delivery stream and you choose to specify an AWS Lambda function to transform data records or if you choose to convert data record formats for your delivery stream.
- If you set Amazon Redshift as the destination for your Kinesis Data Firehose delivery stream and you choose to specify an AWS Lambda function to transform data records.
- If you set any of the following services as the destination for your Kinesis Data Firehose delivery stream: Amazon Elastic Search, Datadog, Dynatrace, HTTP Endpoint, LogicMonitor, MongoDB Cloud, New Relic, Splunk, or Sumo Logic.

The following are the backup settings for your Kinesis Data Firehose delivery stream:

- Source record backup in Amazon S3 - if S3 or Amazon Redshift is your selected destination, this setting indicates whether you want to enable source data backup or keep it disabled. If any other supported service (other than S3 or Amazon Redshift) is set as your selected destination, then this setting indicates if you want to backup all your source data or failed data only.
- S3 backup bucket - this is the S3 bucket where Kinesis Data Firehose backs up your data.
- S3 backup bucket prefix - this is the prefix where Kinesis Data Firehose backs up your data.
- S3 backup bucket error output prefix - all failed data is backed up in the this S3 bucket error output prefix.
- Buffer hints, compression and encryption for backup - Kinesis Data Firehose uses Amazon S3 to backup all or failed only data that it attempts to deliver to your chosen destination. Kinesis Data Firehose

buffers incoming data before delivering it (backing it up) to Amazon S3. You can choose a buffer size of 1–128 MiBs and a buffer interval of 60–900 seconds. The condition that is satisfied first triggers data delivery to Amazon S3. If you enable data transformation, the buffer interval applies from the time transformed data is received by Kinesis Data Firehose to the data delivery to Amazon S3. If data delivery to the destination falls behind data writing to the delivery stream, Kinesis Data Firehose raises the buffer size dynamically to catch up. This action helps ensure that all data is delivered to the destination.

- S3 compressions and encryption - choose GZIP, Snappy, Zip, or Hadoop-Compatible Snappy data compression, or no data compression. Snappy, Zip, and Hadoop-Compatible Snappy compression is not available for delivery streams with Amazon Redshift as the destination.

  Kinesis Data Firehose supports Amazon S3 server-side encryption with AWS Key Management Service (AWS KMS) for encrypting delivered data in Amazon S3. You can choose to not encrypt the data or to encrypt with a key from the list of AWS KMS keys that you own. For more information, see Protecting Data Using Server-Side Encryption with AWS KMS-Managed Keys (SSE-KMS).

Advanced Settings

The following are the advanced settings for your Kinesis Data Firehose delivery stream:

- Server-side encryption - Kinesis Data Firehose supports Amazon S3 server-side encryption with AWS Key Management Service (AWS KMS) for encrypting delivered data in Amazon S3. For more information, see Protecting Data Using Server-Side Encryption with AWS KMS–Managed Keys (SSE-KMS).
- Error logging - If data transformation is enabled, Kinesis Data Firehose can log the Lambda invocation, and send data delivery errors to CloudWatch Logs. Then you can view the specific error logs if the Lambda invocation or data delivery fails. For more information, see Monitoring Kinesis Data Firehose Using CloudWatch Logs.
- Permissions - Kinesis Data Firehose uses IAM roles for all the permissions that the delivery stream needs. You can choose to create a new role where required permissions are assigned automatically, or choose an existing role created for Kinesis Data Firehose. The role is used to grant Kinesis Data Firehose access to various services, including your S3 bucket, AWS KMS key (if data encryption is enabled), and Lambda function (if data transformation is enabled). The console might create a role with placeholders. For more information, see What is IAM?.
- Tags - You can add tags to organize your AWS resources, track costs, and control access.

Once you've chosen your backup and advanced settings, review your choices, and then choose **Create delivery stream**.

The new Kinesis Data Firehose delivery stream takes a few moments in the **Creating** state before it is available. After your Kinesis Data Firehose delivery stream is in an **Active** state, you can start sending data to it from your producer.

Testing Your Delivery Stream Using Sample Data

You can use the AWS Management Console to ingest simulated stock ticker data. The console runs a script in your browser to put sample records in your Kinesis Data Firehose delivery stream. This enables you to test the configuration of your delivery stream without having to generate your own test data.

The following is an example from the simulated data:

```
{"TICKER_SYMBOL":"QXZ","SECTOR":"HEALTHCARE","CHANGE":-0.05,"PRICE":84.51}
```

Note that standard Amazon Kinesis Data Firehose charges apply when your delivery stream transmits the data, but there is no charge when the data is generated. To stop incurring these charges, you can stop the sample stream from the console at any time.

Contents

Prerequisites

Before you begin, create a delivery stream. For more information, see Creating an Amazon Kinesis Data Firehose Delivery Stream (p. 5).

Test Using Amazon S3 as the Destination

Use the following procedure to test your delivery stream using Amazon Simple Storage Service (Amazon S3) as the destination.

To test a delivery stream using Amazon S3

1. Open the Kinesis Data Firehose console at https://console.aws.amazon.com/firehose/.
2. Choose the delivery stream.
3. Under **Test with demo data**, choose **Start sending demo data** to generate sample stock ticker data.
4. Follow the onscreen instructions to verify that data is being delivered to your S3 bucket. Note that it might take a few minutes for new objects to appear in your bucket, based on the buffering configuration of your bucket.
5. When the test is complete, choose **Stop sending demo data** to stop incurring usage charges.

Test Using Amazon Redshift as the Destination

Use the following procedure to test your delivery stream using Amazon Redshift as the destination.

To test a delivery stream using Amazon Redshift

1. Your delivery stream expects a table to be present in your Amazon Redshift cluster. Connect to Amazon Redshift through a SQL interface and run the following statement to create a table that accepts the sample data.

```
create table firehose_test_table
(
 TICKER_SYMBOL varchar(4),
 SECTOR varchar(16),
 CHANGE float,
 PRICE float
);
```

2. Open the Kinesis Data Firehose console at https://console.aws.amazon.com/firehose/.
3. Choose the delivery stream.
4. Edit the destination details for your delivery stream to point to the newly created `firehose_test_table` table.
5. Under **Test with demo data**, choose **Start sending demo data** to generate sample stock ticker data.
6. Follow the onscreen instructions to verify that data is being delivered to your table. Note that it might take a few minutes for new rows to appear in your table, based on the buffering configuration.
7. When the test is complete, choose **Stop sending demo data** to stop incurring usage charges.
8. Edit the destination details for your Kinesis Data Firehose delivery stream to point to another table.
9. (Optional) Delete the `firehose_test_table` table.

Test Using OpenSearch Service as the Destination

Use the following procedure to test your delivery stream using Amazon OpenSearch Service (OpenSearch Service) as the destination.

To test a delivery stream using OpenSearch Service

1. Open the Kinesis Data Firehose console at https://console.aws.amazon.com/firehose/.
2. Choose the delivery stream.
3. Under **Test with demo data**, choose **Start sending demo data** to generate sample stock ticker data.
4. Follow the onscreen instructions to verify that data is being delivered to your OpenSearch Service domain. For more information, see Searching Documents in an OpenSearch Service Domain in the *Amazon OpenSearch Service Developer Guide*.
5. When the test is complete, choose **Stop sending demo data** to stop incurring usage charges.

Test Using Splunk as the Destination

Use the following procedure to test your delivery stream using Splunk as the destination.

To test a delivery stream using Splunk

1. Open the Kinesis Data Firehose console at https://console.aws.amazon.com/firehose/.
2. Choose the delivery stream.
3. Under **Test with demo data**, choose **Start sending demo data** to generate sample stock ticker data.

4. Check whether the data is being delivered to your Splunk index. Example search terms in Splunk are `sourcetype="aws:firehose:json"` and `index="name-of-your-splunk-index"`. For more information about how to search for events in Splunk, see Search Manual in the Splunk documentation.

   If the test data doesn't appear in your Splunk index, check your Amazon S3 bucket for failed events. Also see Data Not Delivered to Splunk.
5. When you finish testing, choose **Stop sending demo data** to stop incurring usage charges.

Sending Data to an Amazon Kinesis Data Firehose Delivery Stream

You can send data to your Kinesis Data Firehose Delivery stream using different types of sources: You can use a Kinesis data stream, the Kinesis Agent, or the Kinesis Data Firehose API using the AWS SDK. You can also use Amazon CloudWatch Logs, CloudWatch Events, or AWS IoT as your data source. If you are new to Kinesis Data Firehose, take some time to become familiar with the concepts and terminology presented in What Is Amazon Kinesis Data Firehose? (p. 1).

> **Note**
> Some AWS services can only send messages and events to a Kinesis Data Firehose delivery stream that is in the same Region. If your delivery stream doesn't appear as an option when you're configuring a target for Amazon CloudWatch Logs, CloudWatch Events, or AWS IoT, verify that your Kinesis Data Firehose delivery stream is in the same Region as your other services.

Topics

Writing to Kinesis Data Firehose Using Kinesis Data Streams

You can configure Amazon Kinesis Data Streams to send information to a Kinesis Data Firehose delivery stream.

> **Important**
> If you use the Kinesis Producer Library (KPL) to write data to a Kinesis data stream, you can use aggregation to combine the records that you write to that Kinesis data stream. If you then use that data stream as a source for your Kinesis Data Firehose delivery stream, Kinesis Data Firehose de-aggregates the records before it delivers them to the destination. If you configure your delivery stream to transform the data, Kinesis Data Firehose de-aggregates the records before it delivers them to AWS Lambda. For more information, see Developing Amazon Kinesis Data Streams Producers Using the Kinesis Producer Library and Aggregation.

1. Sign in to the AWS Management Console and open the Kinesis Data Firehose console at https://console.aws.amazon.com/firehose/.
2. Choose **Create Delivery Stream**. On the **Name and source** page, provide values for the following fields:

   Delivery stream name

   The name of your Kinesis Data Firehose delivery stream.

Source

Choose **Kinesis stream** to configure a Kinesis Data Firehose delivery stream that uses a Kinesis data stream as a data source. You can then use Kinesis Data Firehose to read data easily from an existing data stream and load it into destinations.

To use a Kinesis data stream as a source, choose an existing stream in the **Kinesis stream** list, or choose **Create new** to create a new Kinesis data stream. After you create a new stream, choose **Refresh** to update the **Kinesis stream** list. If you have a large number of streams, filter the list using **Filter by name**.

Note
When you configure a Kinesis data stream as the source of a Kinesis Data Firehose delivery stream, the Kinesis Data Firehose `PutRecord` and `PutRecordBatch` operations are disabled. To add data to your Kinesis Data Firehose delivery stream in this case, use the Kinesis Data Streams `PutRecord` and `PutRecords` operations.

Kinesis Data Firehose starts reading data from the `LATEST` position of your Kinesis stream. For more information about Kinesis Data Streams positions, see GetShardIterator. Kinesis Data Firehose calls the Kinesis Data Streams GetRecords operation once per second for each shard.

More than one Kinesis Data Firehose delivery stream can read from the same Kinesis stream. Other Kinesis applications (consumers) can also read from the same stream. Each call from any Kinesis Data Firehose delivery stream or other consumer application counts against the overall throttling limit for the shard. To avoid getting throttled, plan your applications carefully. For more information about Kinesis Data Streams limits, see Amazon Kinesis Streams Limits.

3. Choose **Next** to advance to the Record Transformation and Record Format Conversion (p. 6) page.

Writing to Kinesis Data Firehose Using Kinesis Agent

Amazon Kinesis agent is a standalone Java software application that offers an easy way to collect and send data to Kinesis Data Firehose. The agent continuously monitors a set of files and sends new data to your Kinesis Data Firehose delivery 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.

By default, records are parsed from each file based on the newline (`'\n'`) character. However, the agent can also be configured to parse multi-line records (see Agent Configuration Settings (p. 29)).

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 delivery stream for the data. After the agent is configured, it durably collects data from the files and reliably sends it to the delivery stream.

Topics

- Use the agent to Preprocess Data (p. 32)
- agent CLI Commands (p. 35)

Prerequisites

- Your operating system must be Amazon Linux, or Red Hat Enterprise Linux version 7 or later.
- Agent version 2.0.0 or later runs using JRE version 1.8 or later. Agent version 1.1.x runs using JRE 1.7 or later.
- If you are using Amazon EC2 to run your agent, launch your EC2 instance.
- The IAM role or AWS credentials that you specify must have permission to perform the Kinesis Data Firehose PutRecordBatch operation for the agent to send data to your delivery stream. If you enable CloudWatch monitoring for the agent, permission to perform the CloudWatch PutMetricData operation is also needed. For more information, see Controlling Access with Amazon Kinesis Data Firehose (p. 39), Monitoring Kinesis Agent Health (p. 99), and Authentication and Access Control for Amazon CloudWatch.

Credentials

Manage your AWS credentials using one of the following methods:

- Create a custom credentials provider. For details, see the section called "Custom Credential Providers" (p. 26).
- Specify an IAM role when you launch your EC2 instance.
- Specify AWS credentials when you configure the agent (see the entries for `awsAccessKeyId` and `awsSecretAccessKey` in the configuration table under the section called "Agent Configuration Settings" (p. 29)).
- Edit `/etc/sysconfig/aws-kinesis-agent` to specify your AWS Region and AWS access keys.
- If your EC2 instance is in a different AWS account, create an IAM role to provide access to the Kinesis Data Firehose service. Specify that role when you configure the agent (see assumeRoleARN (p.) and assumeRoleExternalId (p.)). Use one of the previous methods to specify the AWS credentials of a user in the other account who has permission to assume this role.

Custom Credential Providers

You can create a custom credentials provider and give its class name and jar path to the Kinesis agent in the following configuration settings: `userDefinedCredentialsProvider.classname` and `userDefinedCredentialsProvider.location`. For the descriptions of these two configuration settings, see the section called "Agent Configuration Settings" (p. 29).

To create a custom credentials provider, define a class that implements the `AWSCredentialsProvider` interface, like the one in the following example.

```
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;

public class YourClassName implements AWSCredentialsProvider {
    public YourClassName() {
    }

    public AWSCredentials getCredentials() {
```

```
        return new BasicAWSCredentials("key1", "key2");
    }

    public void refresh() {
    }
}
```

Your class must have a constructor that takes no arguments.

AWS invokes the refresh method periodically to get updated credentials. If you want your credentials provider to provide different credentials throughout its lifetime, include code to refresh the credentials in this method. Alternatively, you can leave this method empty if you want a credentials provider that vends static (non-changing) credentials.

Download and Install the Agent

First, connect to your instance. For more information, see Connect to Your Instance in the *Amazon EC2 User Guide for Linux Instances*. If you have trouble connecting, see Troubleshooting Connecting to Your Instance in the *Amazon EC2 User Guide for Linux Instances*.

Next, install the agent using one of the following methods.

- **To set up the agent from the Amazon Linux repositories**

  This method works only for Amazon Linux instances. Use the following command:

  ```
  sudo yum install -y aws-kinesis-agent
  ```

  Agent v 2.0.0 or later is installed on computers with operating system Amazon Linux 2 (AL2). This agent version requires Java 1.8 or later. If required Java version is not yet present, the agent installation process installs it. For more information regarding Amazon Linux 2 see https://aws.amazon.com/amazon-linux-2/.

- **To set up the agent from the Amazon S3 repository**

  This method works for Red Hat Enterprise Linux, as well as Amazon Linux 2 instances because it installs the agent from the publicly available repository. Use the following command to download and install the latest version of the agent version 2.x.x:

  ```
  sudo yum install -y https://s3.amazonaws.com/streaming-data-agent/aws-kinesis-agent-
  latest.amzn2.noarch.rpm
  ```

  To install a specific version of the agent, specify the version number in the command. For example, the following command installs agent v 2.0.1.

  ```
  sudo yum install -y https://streaming-data-agent.s3.amazonaws.com/aws-kinesis-
  agent-2.0.1-1.amzn1.noarch.rpm
  ```

  If you have Java 1.7 and you don't want to upgrade it, you can download agent version 1.x.x, which is compatible with Java 1.7. For example, to download agent v1.1.6, you can use the following command:

  ```
  ```

```
sudo yum install -y https://s3.amazonaws.com/streaming-data-agent/aws-kinesis-
agent-1.1.6-1.amzn1.noarch.rpm
```

The latest agent v1.x.x can be downloaded using the following command:

```
sudo yum install -y https://s3.amazonaws.com/streaming-data-agent/aws-kinesis-agent-
latest.amzn1.noarch.rpm
```

- **To set up the agent from the GitHub repo**

  1. First, make sure that you have required Java version installed, depending on agent version.
  2. Download the agent from the awslabs/amazon-kinesis-agent GitHub repo.
  3. Install the agent by navigating to the download directory and running the following command:

     ```
     sudo ./setup --install
     ```

Configure and Start the Agent

To configure and start the agent

1. Open and edit the configuration file (as superuser if using default file access permissions): `/etc/aws-kinesis/agent.json`

   In this configuration file, specify the files (`"filePattern"`) from which the agent collects data, and the name of the delivery stream (`"deliveryStream"`) to which the agent sends data. The file name is a pattern, and the agent recognizes file rotations. You can rotate files or create new files no more than once per second. The agent uses the file creation time stamp to determine which files to track and tail into your delivery stream. Creating new files or rotating files more frequently than once per second does not allow the agent to differentiate properly between them.

   ```
   {
      "flows": [
           {
               "filePattern": "/tmp/app.log*",
               "deliveryStream": "yourdeliverystream"
           }
      ]
   }
   ```

   The default AWS Region is `us-east-1`. If you are using a different Region, add the `firehose.endpoint` setting to the configuration file, specifying the endpoint for your Region. For more information, see Agent Configuration Settings (p. 29).
2. Start the agent manually:

   ```
   sudo service aws-kinesis-agent start
   ```
3. (Optional) Configure the agent to start on system startup:

   ```
   sudo chkconfig aws-kinesis-agent on
   ```

The agent is now running as a system service in the background. It continuously monitors the specified files and sends data to the specified delivery stream. Agent activity is logged in `/var/log/aws-kinesis-agent/aws-kinesis-agent.log`.

Agent Configuration Settings

The agent supports two mandatory configuration settings, `filePattern` and `deliveryStream`, plus optional configuration settings for additional features. You can specify both mandatory and optional configuration settings in `/etc/aws-kinesis/agent.json`.

Whenever you change the configuration file, you must stop and start the agent, using the following commands:

```
sudo service aws-kinesis-agent stop
sudo service aws-kinesis-agent start
```

Alternatively, you could use the following command:

```
sudo service aws-kinesis-agent restart
```

The following are the general configuration settings.

Configuration Setting	Description
`assumeRoleARN`	The Amazon Resource Name (ARN) of the role to be assumed by the user. For more information, see Delegate Access Across AWS Accounts Using IAM Roles in the *IAM User Guide*.
`assumeRoleExternalId`	An optional identifier that determines who can assume the role. For more information, see How to Use an External ID in the *IAM User Guide*.
`awsAccessKeyId`	AWS access key ID that overrides the default credentials. This setting takes precedence over all other credential providers.
`awsSecretAccessKey`	AWS secret key that overrides the default credentials. This setting takes precedence over all other credential providers.
`cloudwatch.emitMetrics`	Enables the agent to emit metrics to CloudWatch if set (true). Default: true
`cloudwatch.endpoint`	The regional endpoint for CloudWatch. Default: `monitoring.us-east-1.amazonaws.com`
`firehose.endpoint`	The regional endpoint for Kinesis Data Firehose. Default: `firehose.us-east-1.amazonaws.com`
`sts.endpoint`	The regional endpoint for the AWS Security Token Service. Default: `https://sts.amazonaws.com`
`userDefinedCredentialsProvider.classname`	If you define a custom credentials provider, provide its fully-qualified class name using this setting. Don't include `.class` at the end of the class name.
`userDefinedCredentialsProvider.location`	If you define a custom credentials provider, use this setting to specify the absolute path of the jar that contains the custom credentials provider. The

Configuration Setting	Description
	agent also looks for the jar file in the following location: `/usr/share/aws-kinesis-agent/lib/`.

The following are the flow configuration settings.

Configuration Setting	Description
`aggregatedRecordSizeBytes`	To make the agent aggregate records and then put them to the delivery stream in one operation, specify this setting. Set it to the size that you want the aggregate record to have before the agent puts it to the delivery stream. Default: 0 (no aggregation)
`dataProcessingOptions`	The list of processing options applied to each parsed record before it is sent to the delivery stream. The processing options are performed in the specified order. For more information, see Use the agent to Preprocess Data (p. 32).
`deliveryStream`	[Required] The name of the delivery stream.
`filePattern`	[Required] A glob for the files that need to be monitored by the agent. Any file that matches this pattern is picked up by the agent automatically and monitored. For all files matching this pattern, grant read permission to `aws-kinesis-agent-user`. For the directory containing the files, grant read and execute permissions to `aws-kinesis-agent-user`. **Important** The agent picks up any file that matches this pattern. To ensure that the agent doesn't pick up unintended records, choose this pattern carefully.
`initialPosition`	The initial position from which the file started to be parsed. Valid values are `START_OF_FILE` and `END_OF_FILE`. Default: `END_OF_FILE`
`maxBufferAgeMillis`	The maximum time, in milliseconds, for which the agent buffers data before sending it to the delivery stream. Value range: 1,000–900,000 (1 second to 15 minutes) Default: 60,000 (1 minute)
`maxBufferSizeBytes`	The maximum size, in bytes, for which the agent buffers data before sending it to the delivery stream. Value range: 1–4,194,304 (4 MB) Default: 4,194,304 (4 MB)
`maxBufferSizeRecords`	The maximum number of records for which the agent buffers data before sending it to the delivery stream. Value range: 1–500 Default: 500

Configuration Setting	Description
`minTimeBetweenFilePolls`	The minimal interval, in milliseconds, at which the agent polls and parses the monitored files for new data. Value range: 1 or more Default: 100
`multiLineStartPattern`	The pattern for identifying the start of a record. A record is made of a line that matches the pattern and any following lines that don't match the pattern. The valid values are regular expressions. By default, each new line in the log files is parsed as one record.
`skipHeaderLines`	The number of lines for the agent to skip parsing at the beginning of monitored files. Value range: 0 or more Default: 0 (zero)
`truncatedRecordTerminator`	The string that the agent uses to truncate a parsed record when the record size exceeds the Kinesis Data Firehose record size limit. (1,000 KB) Default: `'\n'` (newline)

Monitor Multiple File Directories and Write to Multiple Streams

By specifying multiple flow configuration settings, you can configure the agent to monitor multiple file directories and send data to multiple streams. In the following configuration example, the agent monitors two file directories and sends data to a Kinesis data stream and a Kinesis Data Firehose delivery stream respectively. You can specify different endpoints for Kinesis Data Streams and Kinesis Data Firehose so that your data stream and Kinesis Data Firehose delivery stream don't need to be in the same Region.

```
{
    "cloudwatch.emitMetrics": true,
    "kinesis.endpoint": "https://your/kinesis/endpoint",
    "firehose.endpoint": "https://your/firehose/endpoint",
    "flows": [
        {
            "filePattern": "/tmp/app1.log*",
            "kinesisStream": "yourkinesisstream"
        },
        {
            "filePattern": "/tmp/app2.log*",
            "deliveryStream": "yourfirehosedeliverystream"
        }
    ]
}
```

For more detailed information about using the agent with Amazon Kinesis Data Streams, see Writing to Amazon Kinesis Data Streams with Kinesis Agent.

Use the agent to Preprocess Data

The agent can pre-process the records parsed from monitored files before sending them to your delivery stream. You can enable this feature by adding the `dataProcessingOptions` configuration setting to your file flow. One or more processing options can be added, and they are performed in the specified order.

The agent supports the following processing options. Because the agent is open source, you can further develop and extend its processing options. You can download the agent from Kinesis Agent.

Processing Options

`SINGLELINE`

Converts a multi-line record to a single-line record by removing newline characters, leading spaces, and trailing spaces.

```
{
    "optionName": "SINGLELINE"
}
```

`CSVTOJSON`

Converts a record from delimiter-separated format to JSON format.

```
{
    "optionName": "CSVTOJSON",
    "customFieldNames": [ "field1", "field2", ... ],
    "delimiter": "yourdelimiter"
}
```

`customFieldNames`

[Required] The field names used as keys in each JSON key value pair. For example, if you specify `["f1", "f2"]`, the record "v1, v2" is converted to `{"f1":"v1","f2":"v2"}`.

`delimiter`

The string used as the delimiter in the record. The default is a comma (,).

`LOGTOJSON`

Converts a record from a log format to JSON format. The supported log formats are **Apache Common Log**, **Apache Combined Log**, **Apache Error Log**, and **RFC3164 Syslog**.

```
{
    "optionName": "LOGTOJSON",
    "logFormat": "logformat",
    "matchPattern": "yourregexpattern",
    "customFieldNames": [ "field1", "field2", … ]
}
```

`logFormat`

[Required] The log entry format. The following are possible values:

- `COMMONAPACHELOG` — The Apache Common Log format. Each log entry has the following pattern by default: `"%{host} %{ident} %{authuser} [%{datetime}] \"%{request}\" %{response} %{bytes}"`.

- `COMBINEDAPACHELOG` — The Apache Combined Log format. Each log entry has the following pattern by default: `"%{host} %{ident} %{authuser} [%{datetime}] \"%{request}\" %{response} %{bytes} %{referrer} %{agent}"`.
- `APACHEERRORLOG` — The Apache Error Log format. Each log entry has the following pattern by default: `"[%{timestamp}] [%{module}:%{severity}] [pid %{processid}:tid %{threadid}] [client: %{client}] %{message}"`.
- `SYSLOG` — The RFC3164 Syslog format. Each log entry has the following pattern by default: `"%{timestamp} %{hostname} %{program}[%{processid}]: %{message}"`.

`matchPattern`

Overrides the default pattern for the specified log format. Use this setting to extract values from log entries if they use a custom format. If you specify `matchPattern`, you must also specify `customFieldNames`.

`customFieldNames`

The custom field names used as keys in each JSON key value pair. You can use this setting to define field names for values extracted from `matchPattern`, or override the default field names of predefined log formats.

Example : LOGTOJSON Configuration

Here is one example of a `LOGTOJSON` configuration for an Apache Common Log entry converted to JSON format:

```
{
    "optionName": "LOGTOJSON",
    "logFormat": "COMMONAPACHELOG"
}
```

Before conversion:

```
64.242.88.10 - - [07/Mar/2004:16:10:02 -0800] "GET /mailman/listinfo/hsdivision HTTP/1.1"
 200 6291
```

After conversion:

```
{"host":"64.242.88.10","ident":null,"authuser":null,"datetime":"07/
Mar/2004:16:10:02 -0800","request":"GET /mailman/listinfo/hsdivision
 HTTP/1.1","response":"200","bytes":"6291"}
```

Example : LOGTOJSON Configuration With Custom Fields

Here is another example `LOGTOJSON` configuration:

```
{
    "optionName": "LOGTOJSON",
    "logFormat": "COMMONAPACHELOG",
    "customFieldNames": ["f1", "f2", "f3", "f4", "f5", "f6", "f7"]
}
```

With this configuration setting, the same Apache Common Log entry from the previous example is converted to JSON format as follows:

```
{"f1":"64.242.88.10","f2":null,"f3":null,"f4":"07/Mar/2004:16:10:02 -0800","f5":"GET /
mailman/listinfo/hsdivision HTTP/1.1","f6":"200","f7":"6291"}
```

Example : Convert Apache Common Log Entry

The following flow configuration converts an Apache Common Log entry to a single-line record in JSON format:

```
{
    "flows": [
        {
            "filePattern": "/tmp/app.log*",
            "deliveryStream": "my-delivery-stream",
            "dataProcessingOptions": [
                {
                    "optionName": "LOGTOJSON",
                    "logFormat": "COMMONAPACHELOG"
                }
            ]
        }
    ]
}
```

Example : Convert Multi-Line Records

The following flow configuration parses multi-line records whose first line starts with "`[SEQUENCE=`". Each record is first converted to a single-line record. Then, values are extracted from the record based on a tab delimiter. Extracted values are mapped to specified `customFieldNames` values to form a single-line record in JSON format.

```
{
    "flows": [
        {
            "filePattern": "/tmp/app.log*",
            "deliveryStream": "my-delivery-stream",
            "multiLineStartPattern": "\\[SEQUENCE=",
            "dataProcessingOptions": [
                {
                    "optionName": "SINGLELINE"
                },
                {
                    "optionName": "CSVTOJSON",
                    "customFieldNames": [ "field1", "field2", "field3" ],
                    "delimiter": "\\t"
                }
            ]
        }
    ]
}
```

Example : LOGTOJSON Configuration with Match Pattern

Here is one example of a `LOGTOJSON` configuration for an Apache Common Log entry converted to JSON format, with the last field (bytes) omitted:

```
{
    "optionName": "LOGTOJSON",
    "logFormat": "COMMONAPACHELOG",
    "matchPattern": "^([\\d.]+) (\\S+) (\\S+) \\[([\\w:/]+\\s[+\\-]\\d{4})\\] \"(.+?)\" (\
\d{3})",
    "customFieldNames": ["host", "ident", "authuser", "datetime", "request", "response"]
}
```

Before conversion:

```
123.45.67.89 - - [27/Oct/2000:09:27:09 -0400] "GET /java/javaResources.html HTTP/1.0" 200
```

After conversion:

```
{"host":"123.45.67.89","ident":null,"authuser":null,"datetime":"27/Oct/2000:09:27:09
 -0400","request":"GET /java/javaResources.html HTTP/1.0","response":"200"}
```

agent CLI Commands

Automatically start the agent on system startup:

```
sudo chkconfig aws-kinesis-agent on
```

Check the status of the agent:

```
sudo service aws-kinesis-agent status
```

Stop the agent:

```
sudo service aws-kinesis-agent stop
```

Read the agent's log file from this location:

```
/var/log/aws-kinesis-agent/aws-kinesis-agent.log
```

Uninstall the agent:

```
sudo yum remove aws-kinesis-agent
```

Writing to Kinesis Data Firehose Using the AWS SDK

You can use the Amazon Kinesis Data Firehose API to send data to a Kinesis Data Firehose delivery stream using the AWS SDK for Java, .NET, Node.js, Python, or Ruby. If you are new to Kinesis Data Firehose, take some time to become familiar with the concepts and terminology presented in What Is Amazon Kinesis Data Firehose? (p. 1). For more information, see Start Developing with Amazon Web Services.

These examples do not represent production-ready code, in that they do not check for all possible exceptions, or account for all possible security or performance considerations.

The Kinesis Data Firehose API offers two operations for sending data to your delivery stream: PutRecord and PutRecordBatch. `PutRecord()` sends one data record within one call and `PutRecordBatch()` can send multiple data records within one call.

Topics

- Single Write Operations Using PutRecord (p. 36)

- Batch Write Operations Using PutRecordBatch (p. 36)

Single Write Operations Using PutRecord

Putting data requires only the Kinesis Data Firehose delivery stream name and a byte buffer (<=1000 KB). Because Kinesis Data Firehose batches multiple records before loading the file into Amazon S3, you may want to add a record separator. To put data one record at a time into a delivery stream, use the following code:

```
PutRecordRequest putRecordRequest = new PutRecordRequest();
putRecordRequest.setDeliveryStreamName(deliveryStreamName);

String data = line + "\n";

Record record = new Record().withData(ByteBuffer.wrap(data.getBytes()));
putRecordRequest.setRecord(record);

// Put record into the DeliveryStream
firehoseClient.putRecord(putRecordRequest);
```

For more code context, see the sample code included in the AWS SDK. For information about request and response syntax, see the relevant topic in Amazon Kinesis Data Firehose API Operations.

Batch Write Operations Using PutRecordBatch

Putting data requires only the Kinesis Data Firehose delivery stream name and a list of records. Because Kinesis Data Firehose batches multiple records before loading the file into Amazon S3, you may want to add a record separator. To put data records in batches into a delivery stream, use the following code:

```
PutRecordBatchRequest putRecordBatchRequest = new PutRecordBatchRequest();
putRecordBatchRequest.setDeliveryStreamName(deliveryStreamName);
putRecordBatchRequest.setRecords(recordList);

// Put Record Batch records. Max No.Of Records we can put in a
// single put record batch request is 500
firehoseClient.putRecordBatch(putRecordBatchRequest);

recordList.clear();
```

For more code context, see the sample code included in the AWS SDK. For information about request and response syntax, see the relevant topic in Amazon Kinesis Data Firehose API Operations.

Writing to Kinesis Data Firehose Using CloudWatch Logs

For information about how to create a CloudWatch Logs subscription that sends log events to Kinesis Data Firehose, see Subscription Filters with Amazon Kinesis Firehose.

Important
CloudWatch log events are compressed with gzip level 6. If you want to specify OpenSearch Service or Splunk as the destination for the delivery stream, use a Lambda function to uncompress the records to UTF-8.and single-line JSON. For information about using Lambda functions with a delivery stream, see *Data Transformation* (p. 58).

Writing to Kinesis Data Firehose Using CloudWatch Events

You can configure Amazon CloudWatch to send events to a Kinesis Data Firehose delivery stream by adding a target to a CloudWatch Events rule.

To create a target for a CloudWatch Events rule that sends events to an existing delivery stream

1. Sign in to the AWS Management Console and open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Choose **Create rule**.
3. On the **Step 1: Create rule** page, for **Targets**, choose **Add target**, and then choose **Firehose delivery stream**.
4. For **Delivery stream**, choose an existing Kinesis Data Firehose delivery stream.

For more information about creating CloudWatch Events rules, see Getting Started with Amazon CloudWatch Events.

Writing to Kinesis Data Firehose Using AWS IoT

You can configure AWS IoT to send information to a Amazon Kinesis Data Firehose delivery stream by adding an action.

To create an action that sends events to an existing Kinesis Data Firehose delivery stream

1. When creating a rule in the AWS IoT console, on the **Create a rule** page, under **Set one or more actions**, choose **Add action**.
2. Choose **Send messages to an Amazon Kinesis Firehose stream**.
3. Choose **Configure action**.
4. For **Stream name**, choose an existing Kinesis Data Firehose delivery stream.
5. For **Separator**, choose a separator character to be inserted between records.
6. For **IAM role name**, choose an existing IAM role or choose **Create a new role**.
7. Choose **Add action**.

For more information about creating AWS IoT rules, see AWS IoT Rule Tutorials.

Security in Amazon Kinesis Data Firehose

Cloud security at AWS is the highest priority. As an AWS customer, you will benefit from a data center and network architecture built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. The effectiveness of our security is regularly tested and verified by third-party auditors as part of the AWS compliance programs. To learn about the compliance programs that apply to Kinesis Data Firehose, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your organization's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Kinesis Data Firehose. The following topics show you how to configure Kinesis Data Firehose to meet your security and compliance objectives. You'll also learn how to use other AWS services that can help you to monitor and secure your Kinesis Data Firehose resources.

Topics

Data Protection in Amazon Kinesis Data Firehose

If you have sensitive data, you can enable server-side data encryption when you use Amazon Kinesis Data Firehose. How you do this depends on the source of your data.

> **Note**
> If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2.

Server-Side Encryption with Kinesis Data Streams as the Data Source

When you configure a Kinesis data stream as the data source of a Kinesis Data Firehose delivery stream, Kinesis Data Firehose no longer stores the data at rest. Instead, the data is stored in the data stream.

When you send data from your data producers to your data stream, Kinesis Data Streams encrypts your data using an AWS Key Management Service (AWS KMS) key before storing the data at rest. When your Kinesis Data Firehose delivery stream reads the data from your data stream, Kinesis Data Streams first decrypts the data and then sends it to Kinesis Data Firehose. Kinesis Data Firehose buffers the data in memory based on the buffering hints that you specify. It then delivers it to your destinations without storing the unencrypted data at rest.

For information about how to enable server-side encryption for Kinesis Data Streams, see Using Server-Side Encryption in the *Amazon Kinesis Data Streams Developer Guide*.

Server-Side Encryption with Direct PUT or Other Data Sources

If you send data to your delivery stream using PutRecord or PutRecordBatch, or if you send the data using AWS IoT, Amazon CloudWatch Logs, or CloudWatch Events, you can turn on server-side encryption by using the StartDeliveryStreamEncryption operation.

To stop server-side-encryption, use the StopDeliveryStreamEncryption operation.

You can also enable SSE when you create the delivery stream. To do that, specify DeliveryStreamEncryptionConfigurationInput when you invoke CreateDeliveryStream.

When the CMK is of type `CUSTOMER_MANAGED_CMK`, if the Amazon Kinesis Data Firehose service is unable to decrypt records because of a `KMSNotFoundException`, a `KMSInvalidStateException`, a `KMSDisabledException`, or a `KMSAccessDeniedException`, the service waits up to 24 hours (the retention period) for you to resolve the problem. If the problem persists beyond the retention period, the service skips those records that have passed the retention period and couldn't be decrypted, and then discards the data. Amazon Kinesis Data Firehose provides the following four CloudWatch metrics that you can use to track the four AWS KMS exceptions:

- `KMSKeyAccessDenied`
- `KMSKeyDisabled`
- `KMSKeyInvalidState`
- `KMSKeyNotFound`

For more information about these four metrics, see the section called “Monitoring with CloudWatch Metrics” (p. 79).

Important
To encrypt your delivery stream, use symmetric CMKs. Kinesis Data Firehose doesn't support asymmetric CMKs. For information about symmetric and asymmetric CMKs, see About Symmetric and Asymmetric CMKs in the AWS Key Management Service developer guide.

Controlling Access with Amazon Kinesis Data Firehose

The following sections cover how to control access to and from your Kinesis Data Firehose resources. The information they cover includes how to grant your application access so it can send data to your Kinesis Data Firehose delivery stream. They also describe how you can grant Kinesis Data Firehose access to your Amazon Simple Storage Service (Amazon S3) bucket, Amazon Redshift cluster, or Amazon OpenSearch Service cluster, as well as the access permissions you need if you use Datadog, Dynatrace, LogicMonitor, MongoDB, New Relic, Splunk, or Sumo Logic as your destination. Finally, you'll find in this topic guidance on how to configure Kinesis Data Firehose so it can deliver data to a destination that belongs to a

different AWS account. The technology for managing all these forms of access is AWS Identity and Access Management (IAM). For more information about IAM, see What is IAM?.

Contents

Grant Your Application Access to Your Kinesis Data Firehose Resources

To give your application access to your Kinesis Data Firehose delivery stream, use a policy similar to this example. You can adjust the individual API operations to which you grant access by modifying the `Action` section, or grant access to all operations with `"firehose:*"`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "firehose:DeleteDeliveryStream",
                "firehose:PutRecord",
                "firehose:PutRecordBatch",
                "firehose:UpdateDestination"
            ],
            "Resource": [
                "arn:aws:firehose:region:account-id:deliverystream/delivery-stream-name"
            ]
        }
    ]
}
```

Allow Kinesis Data Firehose to Assume an IAM Role

If you use the console to create a delivery stream, and choose the option to create a new role, AWS attaches the required trust policy to the role. Or if you want Kinesis Data Firehose to use an existing IAM role or if you create a role on your own, attach the following trust policy to that role so that Kinesis Data Firehose can assume it.

```
{
  "Version": "2012-10-17",
```

```
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "firehose.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

For information about how to modify the trust relationship of a role, see Modifying a Role.

Grant Kinesis Data Firehose Access to AWS Glue for Data Format Conversion

If your delivery stream performs data-format conversion, Kinesis Data Firehose references table definitions stored in AWS Glue. To give Kinesis Data Firehose the necessary access to AWS Glue, add the following statement to your policy. For information on how to find the ARN of the table, see Specifying AWS Glue Resource ARNs.

```
{
  "Effect": "Allow",
  "Action": [
    "glue:GetTable",
    "glue:GetTableVersion",
    "glue:GetTableVersions"
  ],
  "Resource": "table-arn"
}
```

Grant Kinesis Data Firehose Access to an Amazon S3 Destination

When you're using an Amazon S3 destination, Kinesis Data Firehose delivers data to your S3 bucket and can optionally use an AWS KMS key that you own for data encryption. If error logging is enabled, Kinesis Data Firehose also sends data delivery errors to your CloudWatch log group and streams. You are required to have an IAM role when creating a delivery stream. Kinesis Data Firehose assumes that IAM role and gains access to the specified bucket, key, and CloudWatch log group and streams.

Use the following access policy to enable Kinesis Data Firehose to access your S3 bucket and AWS KMS key. If you don't own the S3 bucket, add `s3:PutObjectAcl` to the list of Amazon S3 actions. This grants the bucket owner full access to the objects delivered by Kinesis Data Firehose. This policy also has a statement that allows access to Amazon Kinesis Data Streams. If you don't use Kinesis Data Streams as your data source, you can remove that statement.

```
{
    "Version": "2012-10-17",
    "Statement":
    [
        {
            "Effect": "Allow",
            "Action": [
                "s3:AbortMultipartUpload",
                "s3:GetBucketLocation",
                "s3:GetObject",
```

```
                "s3:ListBucket",
                "s3:ListBucketMultipartUploads",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::bucket-name",
                "arn:aws:s3:::bucket-name/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStream",
                "kinesis:GetShardIterator",
                "kinesis:GetRecords",
                "kinesis:ListShards"
            ],
            "Resource": "arn:aws:kinesis:region:account-id:stream/stream-name"
        },
        {
           "Effect": "Allow",
           "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey"
           ],
           "Resource": [
                "arn:aws:kms:region:account-id:key/key-id"
           ],
           "Condition": {
                "StringEquals": {
                    "kms:ViaService": "s3.region.amazonaws.com"
                },
                "StringLike": {
                    "kms:EncryptionContext:aws:s3:arn": "arn:aws:s3:::bucket-name/prefix*"
                }
           }
        },
        {
           "Effect": "Allow",
           "Action": [
                "logs:PutLogEvents"
           ],
           "Resource": [
                "arn:aws:logs:region:account-id:log-group:log-group-name:log-stream:log-
stream-name"
           ]
        },
        {
           "Effect": "Allow",
           "Action": [
                "lambda:InvokeFunction",
                "lambda:GetFunctionConfiguration"
           ],
           "Resource": [
                "arn:aws:lambda:region:account-id:function:function-name:function-version"
           ]
        }
    ]
}
```

For more information about allowing other AWS services to access your AWS resources, see Creating a Role to Delegate Permissions to an AWS Service in the *IAM User Guide*.

To learn how to grant Kinesis Data Firehose access to an Amazon S3 destination in another account, see the section called “Cross-Account Delivery to an Amazon S3 Destination” (p. 51).

Grant Kinesis Data Firehose Access to an Amazon Redshift Destination

Refer to the following when you are granting access to Kinesis Data Firehose when using an Amazon Redshift destination.

Topics

IAM Role and Access Policy

When you're using an Amazon Redshift destination, Kinesis Data Firehose delivers data to your S3 bucket as an intermediate location. It can optionally use an AWS KMS key you own for data encryption. Kinesis Data Firehose then loads the data from the S3 bucket to your Amazon Redshift cluster. If error logging is enabled, Kinesis Data Firehose also sends data delivery errors to your CloudWatch log group and streams. Kinesis Data Firehose uses the specified Amazon Redshift user name and password to access your cluster, and uses an IAM role to access the specified bucket, key, CloudWatch log group, and streams. You are required to have an IAM role when creating a delivery stream.

Use the following access policy to enable Kinesis Data Firehose to access your S3 bucket and AWS KMS key. If you don't own the S3 bucket, add `s3:PutObjectAcl` to the list of Amazon S3 actions, which grants the bucket owner full access to the objects delivered by Kinesis Data Firehose. This policy also has a statement that allows access to Amazon Kinesis Data Streams. If you don't use Kinesis Data Streams as your data source, you can remove that statement.

```
{
"Version": "2012-10-17",
    "Statement":
    [
        {
            "Effect": "Allow",
            "Action": [
                "s3:AbortMultipartUpload",
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:ListBucketMultipartUploads",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::bucket-name",
                "arn:aws:s3:::bucket-name/*"
            ]
        },
        {
           "Effect": "Allow",
           "Action": [
               "kms:Decrypt",
               "kms:GenerateDataKey"
           ],
           "Resource": [
               "arn:aws:kms:region:account-id:key/key-id"
           ],
           "Condition": {
               "StringEquals": {
                   "kms:ViaService": "s3.region.amazonaws.com"
               },
```

```
                "StringLike": {
                    "kms:EncryptionContext:aws:s3:arn": "arn:aws:s3:::bucket-name/prefix*"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStream",
                "kinesis:GetShardIterator",
                "kinesis:GetRecords",
                "kinesis:ListShards"
            ],
            "Resource": "arn:aws:kinesis:region:account-id:stream/stream-name"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:region:account-id:log-group:log-group-name:log-stream:log-
stream-name"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "lambda:InvokeFunction",
                "lambda:GetFunctionConfiguration"
            ],
            "Resource": [
                "arn:aws:lambda:region:account-id:function:function-name:function-version"
            ]
        }
    ]
}
```

For more information about allowing other AWS services to access your AWS resources, see Creating a Role to Delegate Permissions to an AWS Service in the *IAM User Guide*.

VPC Access to an Amazon Redshift Cluster

If your Amazon Redshift cluster is in a virtual private cloud (VPC), it must be publicly accessible with a public IP address. Also, grant Kinesis Data Firehose access to your Amazon Redshift cluster by unblocking the Kinesis Data Firehose IP addresses. Kinesis Data Firehose currently uses one CIDR block for each available Region:

- `13.58.135.96/27` for US East (Ohio)
- `52.70.63.192/27` for US East (N. Virginia)
- `13.57.135.192/27` for US West (N. California)
- `52.89.255.224/27` for US West (Oregon)
- `18.253.138.96/27` for AWS GovCloud (US-East)
- `52.61.204.160/27` for AWS GovCloud (US-West)
- `35.183.92.128/27` for Canada (Central)
- `18.162.221.32/27` for Asia Pacific (Hong Kong)
- `13.232.67.32/27` for Asia Pacific (Mumbai)
- `13.209.1.64/27` for Asia Pacific (Seoul)
- `13.228.64.192/27` for Asia Pacific (Singapore)

- `13.210.67.224/27` for Asia Pacific (Sydney)
- `13.113.196.224/27` for Asia Pacific (Tokyo)
- `52.81.151.32/27` for China (Beijing)
- `161.189.23.64/27` for China (Ningxia)
- `35.158.127.160/27` for Europe (Frankfurt)
- `52.19.239.192/27` for Europe (Ireland)
- `18.130.1.96/27` for Europe (London)
- `35.180.1.96/27` for Europe (Paris)
- `13.53.63.224/27` for Europe (Stockholm)
- `15.185.91.0/27` for Middle East (Bahrain)
- `18.228.1.128/27` for South America (São Paulo)
- `15.161.135.128/27` for Europe (Milan)
- `13.244.121.224/277` for Africa (Cape Town)
- `13.208.177.192/27` for Asia Pacific (Osaka)

For more information about how to unblock IP addresses, see the step Authorize Access to the Cluster in the *Amazon Redshift Getting Started Guide* guide.

Grant Kinesis Data Firehose Access to a Public OpenSearch Service Destination

When you're using an OpenSearch Service destination, Kinesis Data Firehose delivers data to your Amazon ES cluster, and concurrently backs up failed or all documents to your S3 bucket. If error logging is enabled, Kinesis Data Firehose also sends data delivery errors to your CloudWatch log group and streams. Kinesis Data Firehose uses an IAM role to access the specified Elasticsearch domain, S3 bucket, AWS KMS key, and CloudWatch log group and streams. You are required to have an IAM role when creating a delivery stream.

Use the following access policy to enable Kinesis Data Firehose to access your S3 bucket, OpenSearch Service domain, and AWS KMS key. If you do not own the S3 bucket, add `s3:PutObjectAcl` to the list of Amazon S3 actions, which grants the bucket owner full access to the objects delivered by Kinesis Data Firehose. This policy also has a statement that allows access to Amazon Kinesis Data Streams. If you don't use Kinesis Data Streams as your data source, you can remove that statement.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:AbortMultipartUpload",
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:ListBucketMultipartUploads",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::bucket-name",
                "arn:aws:s3:::bucket-name/*"
            ]
        },
        {
           "Effect": "Allow",
           "Action": [
```

```
                "kms:Decrypt",
                "kms:GenerateDataKey"
            ],
            "Resource": [
                "arn:aws:kms:region:account-id:key/key-id"
            ],
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "s3.region.amazonaws.com"
                },
                "StringLike": {
                    "kms:EncryptionContext:aws:s3:arn": "arn:aws:s3:::bucket-name/prefix*"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "es:DescribeElasticsearchDomain",
                "es:DescribeElasticsearchDomains",
                "es:DescribeElasticsearchDomainConfig",
                "es:ESHttpPost",
                "es:ESHttpPut"
            ],
           "Resource": [
                "arn:aws:es:region:account-id:domain/domain-name",
                "arn:aws:es:region:account-id:domain/domain-name/*"
           ]
       },
       {
            "Effect": "Allow",
            "Action": [
                "es:ESHttpGet"
            ],
            "Resource": [
                "arn:aws:es:region:account-id:domain/domain-name/_all/_settings",
                "arn:aws:es:region:account-id:domain/domain-name/_cluster/stats",
                "arn:aws:es:region:account-id:domain/domain-name/index-name*/_mapping/type-
name",
                "arn:aws:es:region:account-id:domain/domain-name/_nodes",
                "arn:aws:es:region:account-id:domain/domain-name/_nodes/stats",
                "arn:aws:es:region:account-id:domain/domain-name/_nodes/*/stats",
                "arn:aws:es:region:account-id:domain/domain-name/_stats",
                "arn:aws:es:region:account-id:domain/domain-name/index-name*/_stats"
            ]
       },
       {
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStream",
                "kinesis:GetShardIterator",
                "kinesis:GetRecords",
                "kinesis:ListShards"
            ],
            "Resource": "arn:aws:kinesis:region:account-id:stream/stream-name"
       },
       {
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:region:account-id:log-group:log-group-name:log-stream:log-
stream-name"
            ]
       },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "lambda:InvokeFunction",
                "lambda:GetFunctionConfiguration"
            ],
            "Resource": [
                "arn:aws:lambda:region:account-id:function:function-name:function-version"
            ]
        }
    ]
}
```

For more information about allowing other AWS services to access your AWS resources, see Creating a Role to Delegate Permissions to an AWS Service in the *IAM User Guide*.

To learn how to grant Kinesis Data Firehose access to an Amazon ES cluster in another account, see the section called "Cross-Account Delivery to an OpenSearch Service Destination" (p. 52).

Grant Kinesis Data Firehose Access to an OpenSearch Service Destination in a VPC

If your OpenSearch Service domain is in a VPC, make sure you give Kinesis Data Firehose the permissions that are described in the previous section. In addition, you need to give Kinesis Data Firehose the following permissions to enable it to access your OpenSearch Service domain's VPC.

- `ec2:DescribeVpcs`
- `ec2:DescribeVpcAttribute`
- `ec2:DescribeSubnets`
- `ec2:DescribeSecurityGroups`
- `ec2:DescribeNetworkInterfaces`
- `ec2:CreateNetworkInterface`
- `ec2:CreateNetworkInterfacePermission`
- `ec2:DeleteNetworkInterface`

If you revoke these permissions after you create the delivery stream, Kinesis Data Firehose can't scale out by creating more ENIs when necessary. You might therefore see a degradation in performance.

When you create or update your delivery stream, you specify a security group for Kinesis Data Firehose to use when it sends data to your OpenSearch Service domain. You can use the same security group that the OpenSearch Service domain uses or a different one. If you specify a different security group, ensure that it allows outbound HTTPS traffic to the OpenSearch Service domain's security group. Also ensure that the OpenSearch Service domain's security group allows HTTPS traffic from the security group you specified when you configured your delivery stream. If you use the same security group for both your delivery stream and the OpenSearch Service domain, make sure the security group inbound rule allows HTTPS traffic. For more information about security group rules, see Security group rules in the Amazon VPC documentation.

Grant Kinesis Data Firehose Access to a Splunk Destination

When you're using a Splunk destination, Kinesis Data Firehose delivers data to your Splunk HTTP Event Collector (HEC) endpoint. It also backs up that data to the Amazon S3 bucket that you specify, and you can optionally use an AWS KMS key that you own for Amazon S3 server-side encryption. If error logging

is enabled, Kinesis Data Firehose sends data delivery errors to your CloudWatch log streams. You can also use AWS Lambda for data transformation. If you use an AWS load balancer, make sure that it is a Classic Load Balancer. Kinesis Data Firehose supports neither Application Load Balancers nor Network Load Balancers. Also, enable duration-based sticky sessions with cookie expiration disabled. For information about how to do this, see Duration-Based Session Stickiness.

You are required to have an IAM role when creating a delivery stream. Kinesis Data Firehose assumes that IAM role and gains access to the specified bucket, key, and CloudWatch log group and streams.

Use the following access policy to enable Kinesis Data Firehose to access your S3 bucket. If you don't own the S3 bucket, add `s3:PutObjectAcl` to the list of Amazon S3 actions, which grants the bucket owner full access to the objects delivered by Kinesis Data Firehose. This policy also grants Kinesis Data Firehose access to CloudWatch for error logging and to AWS Lambda for data transformation. The policy also has a statement that allows access to Amazon Kinesis Data Streams. If you don't use Kinesis Data Streams as your data source, you can remove that statement. Kinesis Data Firehose doesn't use IAM to access Splunk. For accessing Splunk, it uses your HEC token.

```
{
    "Version": "2012-10-17",
    "Statement":
    [
        {
            "Effect": "Allow",
            "Action": [
                "s3:AbortMultipartUpload",
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:ListBucketMultipartUploads",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::bucket-name",
                "arn:aws:s3:::bucket-name/*"
            ]
        },
        {
           "Effect": "Allow",
           "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey"
           ],
           "Resource": [
                "arn:aws:kms:region:account-id:key/key-id"
           ],
           "Condition": {
                "StringEquals": {
                    "kms:ViaService": "s3.region.amazonaws.com"
                },
                "StringLike": {
                    "kms:EncryptionContext:aws:s3:arn": "arn:aws:s3:::bucket-name/prefix*"
                }
           }
        },
        {
           "Effect": "Allow",
           "Action": [
                "kinesis:DescribeStream",
                "kinesis:GetShardIterator",
                "kinesis:GetRecords",
                "kinesis:ListShards"
           ],
           "Resource": "arn:aws:kinesis:region:account-id:stream/stream-name"
        },
```

```
        {
           "Effect": "Allow",
           "Action": [
               "logs:PutLogEvents"
           ],
           "Resource": [
               "arn:aws:logs:region:account-id:log-group:log-group-name:log-stream:*"
           ]
        },
        {
           "Effect": "Allow",
           "Action": [
               "lambda:InvokeFunction",
               "lambda:GetFunctionConfiguration"
           ],
           "Resource": [
               "arn:aws:lambda:region:account-id:function:function-name:function-version"
           ]
        }
    ]
}
```

For more information about allowing other AWS services to access your AWS resources, see Creating a Role to Delegate Permissions to an AWS Service in the *IAM User Guide*.

Access to Splunk in VPC

If your Splunk platform is in a VPC, it must be publicly accessible with a public IP address. Also, grant Kinesis Data Firehose access to your Splunk platform by unblocking the Kinesis Data Firehose IP addresses. Kinesis Data Firehose currently uses the following CIDR blocks.

- `18.216.68.160/27, 18.216.170.64/27, 18.216.170.96/27` for US East (Ohio)
- `34.238.188.128/26, 34.238.188.192/26, 34.238.195.0/26` for US East (N. Virginia)
- `13.57.180.0/26` for US West (N. California)
- `34.216.24.32/27, 34.216.24.192/27, 34.216.24.224/27` for US West (Oregon)
- `18.253.138.192/26` for AWS GovCloud (US-East)
- `52.61.204.192/26` for AWS GovCloud (US-West)
- `18.162.221.64/26` for Asia Pacific (Hong Kong)
- `13.232.67.64/26` for Asia Pacific (Mumbai)
- `13.209.71.0/26` for Asia Pacific (Seoul)
- `13.229.187.128/26` for Asia Pacific (Singapore)
- `13.211.12.0/26` for Asia Pacific (Sydney)
- `13.230.21.0/27, 13.230.21.32/27` for Asia Pacific (Tokyo)
- `35.183.92.64/26` for Canada (Central)
- `18.194.95.192/27, 18.194.95.224/27, 18.195.48.0/27` for Europe (Frankfurt)
- `34.241.197.32/27, 34.241.197.64/27, 34.241.197.96/27` for Europe (Ireland)
- `18.130.91.0/26` for Europe (London)
- `35.180.112.0/26` for Europe (Paris)
- `13.53.191.0/26` for Europe (Stockholm)
- `15.185.91.64/26` for Middle East (Bahrain)
- `18.228.1.192/26` for South America (São Paulo)
- `15.161.135.192/26` for Europe (Milan)
- `13.244.165.128/26` for Africa (Cape Town)
- `13.208.217.0/26` for Asia Pacific (Osaka)

Grant Kinesis Data Firehose Access to an HTTP Endpoint Destination

You can use Kinesis Data Firehose to deliver data to any HTTP endpoint destination. Kinesis Data Firehose also backs up that data to the Amazon S3 bucket that you specify, and you can optionally use an AWS KMS key that you own for Amazon S3 server-side encryption. If error logging is enabled, Kinesis Data Firehose sends data delivery errors to your CloudWatch log streams. You can also use AWS Lambda for data transformation.

You are required to have an IAM role when creating a delivery stream. Kinesis Data Firehose assumes that IAM role and gains access to the specified bucket, key, and CloudWatch log group and streams.

Use the following access policy to enable Kinesis Data Firehose to access the S3 bucket that you specified for data backup. If you don't own the S3 bucket, add `s3:PutObjectAcl` to the list of Amazon S3 actions, which grants the bucket owner full access to the objects delivered by Kinesis Data Firehose. This policy also grants Kinesis Data Firehose access to CloudWatch for error logging and to AWS Lambda for data transformation. The policy also has a statement that allows access to Amazon Kinesis Data Streams. If you don't use Kinesis Data Streams as your data source, you can remove that statement.

Important
Kinesis Data Firehose doesn't use IAM to access HTTP endpoint destinations owned by supported third-party service providers, including Datadog, Dynatrace, LogicMonitor, MongoDB, New Relic, Splunk, or Sumo Logic. For accessing a specified HTTP endpoint destination owned by a supported third-party service provider, contact that service provider to obtain the API key or the access key that is required to enable data delivery to that service from Kinesis Data Firehose.

```
{
    "Version": "2012-10-17",
    "Statement":
    [
        {
            "Effect": "Allow",
            "Action": [
                "s3:AbortMultipartUpload",
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:ListBucketMultipartUploads",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::bucket-name",
                "arn:aws:s3:::bucket-name/*"
            ]
        },
        {
           "Effect": "Allow",
           "Action": [
               "kms:Decrypt",
               "kms:GenerateDataKey"
           ],
           "Resource": [
               "arn:aws:kms:region:account-id:key/key-id"
           ],
           "Condition": {
               "StringEquals": {
                   "kms:ViaService": "s3.region.amazonaws.com"
               },
               "StringLike": {
                   "kms:EncryptionContext:aws:s3:arn": "arn:aws:s3:::bucket-name/prefix*"
```

```
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStream",
                "kinesis:GetShardIterator",
                "kinesis:GetRecords",
                "kinesis:ListShards"
            ],
            "Resource": "arn:aws:kinesis:region:account-id:stream/stream-name"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:region:account-id:log-group:log-group-name:log-stream:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "lambda:InvokeFunction",
                "lambda:GetFunctionConfiguration"
            ],
            "Resource": [
                "arn:aws:lambda:region:account-id:function:function-name:function-version"
            ]
        }
    ]
}
```

For more information about allowing other AWS services to access your AWS resources, see Creating a Role to Delegate Permissions to an AWS Service in the *IAM User Guide*.

Important
Currently Kinesis Data Firehose does NOT support data delivery to HTTP endpoints in a VPC.

Cross-Account Delivery to an Amazon S3 Destination

You can use the AWS CLI or the Kinesis Data Firehose APIs to create a delivery stream in one AWS account with an Amazon S3 destination in a different account. The following procedure shows an example of configuring a Kinesis Data Firehose delivery stream owned by account A to deliver data to an Amazon S3 bucket owned by account B.

1. Create an IAM role under account A using steps described in Grant Kinesis Firehose Access to an Amazon S3 Destination.

   Note
   The Amazon S3 bucket specified in the access policy is owned by account B in this case. Make sure you add `s3:PutObjectAcl` to the list of Amazon S3 actions in the access policy, which grants account B full access to the objects delivered by Amazon Kinesis Data Firehose. This permission is required for cross account delivery. Kinesis Data Firehose sets the "x-amz-acl" header on the request to "bucket-owner-full-control".

2. To allow access from the IAM role previously created, create an S3 bucket policy under account B. The following code is an example of the bucket policy. For more information, see Using Bucket Policies and User Policies.

```
{
```

```
    "Version": "2012-10-17",
    "Id": "PolicyID",
    "Statement": [
        {
            "Sid": "StmtID",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::accountA-id:role/iam-role-name"
            },
            "Action": [
                "s3:AbortMultipartUpload",
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:ListBucketMultipartUploads",
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": [
                "arn:aws:s3:::bucket-name",
                "arn:aws:s3:::bucket-name/*"
            ]
        }
    ]
}
```

3. Create a Kinesis Data Firehose delivery stream under account A using the IAM role that you created in step 1.

Cross-Account Delivery to an OpenSearch Service Destination

You can use the AWS CLI or the Kinesis Data Firehose APIs to create a delivery stream in one AWS account with an OpenSearch Service destination in a different account. The following procedure shows an example of how you can create a Kinesis Data Firehose delivery stream under account A and configure it to deliver data to an OpenSearch Service destination owned by account B.

1. Create an IAM role under account A using the steps described in the section called “Grant Kinesis Data Firehose Access to a Public OpenSearch Service Destination” (p. 45).
2. To allow access from the IAM role that you created in the previous step, create an OpenSearch Service policy under account B. The following JSON is an example.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::Account-A-ID:role/firehose_delivery_role "
      },
      "Action": "es:ESHttpGet",
      "Resource": [
        "arn:aws:es:us-east-1:Account-B-ID:domain/cross-account-cluster/_all/
_settings",
        "arn:aws:es:us-east-1:Account-B-ID:domain/cross-account-cluster/_cluster/
stats",
        "arn:aws:es:us-east-1:Account-B-ID:domain/cross-account-cluster/roletest*/
_mapping/roletest",
```

```
        "arn:aws:es:us-east-1:Account-B-ID:domain/cross-account-cluster/_nodes",
        "arn:aws:es:us-east-1:Account-B-ID:domain/cross-account-cluster/_nodes/stats",
        "arn:aws:es:us-east-1:Account-B-ID:domain/cross-account-cluster/_nodes/*/
stats",
        "arn:aws:es:us-east-1:Account-B-ID:domain/cross-account-cluster/_stats",
        "arn:aws:es:us-east-1:Account-B-ID:domain/cross-account-cluster/roletest*/
_stats"
      ]
    }
  ]
}
```

3. Create a Kinesis Data Firehose delivery stream under account A using the IAM role that you created in step 1. When you create the delivery stream, use the AWS CLI or the Kinesis Data Firehose APIs and specify the `ClusterEndpoint` field instead of `DomainARN` for OpenSearch Service.

   Note
   To create a delivery stream in one AWS account with an OpenSearch Service destination in a different account, you must use the AWS CLI or the Kinesis Data Firehose APIs. You can't use the AWS Management Console to create this kind of cross-account configuration.

Using Tags to Control Access

You can use the optional `Condition` element (or `Condition` *block*) in an IAM policy to fine-tune access to Kinesis Data Firehose operations based on tag keys and values. The following subsections describe how to do this for the different Kinesis Data Firehose operations. For more on the use of the `Condition` element and the operators that you can use within it, see IAM JSON Policy Elements: Condition.

CreateDeliveryStream and TagDeliveryStream

For the `CreateDeliveryStream` and `TagDeliveryStream` operations, use the `aws:RequestTag` condition key. In the following example, `MyKey` and `MyValue` represent the key and corresponding value for a tag.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "firehose:CreateDeliveryStream",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/MyKey": "MyValue"
                }
            }
        }
    ]
}
```

UntagDeliveryStream

For the `UntagDeliveryStream` operation, use the `aws:TagKeys` condition key. In the following example, `MyKey` is an example tag key.

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": "firehose:UntagDeliveryStream",
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "aws:TagKeys": "MyKey"
                }
            }
        }
    ]
}
```

ListDeliveryStreams

You can't use tag-based access control with `ListDeliveryStreams`.

Other Kinesis Data Firehose Operations

For all Kinesis Data Firehose operations other than `CreateDeliveryStream`, `TagDeliveryStream`, `UntagDeliveryStream`, and `ListDeliveryStreams`, use the `aws:RequestTag` condition key. In the following example, `MyKey` and `MyValue` represent the key and corresponding value for a tag.

```
{
    "Version": "2012-10-17",
    "Statement": [
    {
            "Effect": "Deny",
            "Action": "firehose:DescribeDeliveryStream",
            "Resource": "*",
            "Condition": {
                "Null": {
                    "firehose:ResourceTag/MyKey": "MyValue"
            }
        }
    ]
}
```

Monitoring Amazon Kinesis Data Firehose

Kinesis Data Firehose provides monitoring functionality for your delivery streams. For more information, see *Monitoring* (p. 79).

Compliance Validation for Amazon Kinesis Data Firehose

Third-party auditors assess the security and compliance of Amazon Kinesis Data Firehose as part of multiple AWS compliance programs. These include SOC, PCI, FedRAMP, HIPAA, and others.

For a list of AWS services in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using Kinesis Data Firehose is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. If your use of Kinesis Data Firehose is subject to compliance with standards such as HIPAA, PCI, or FedRAMP, AWS provides resources to help:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- Architecting for HIPAA Security and Compliance Whitepaper – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Config – This AWS service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

Resilience in Amazon Kinesis Data Firehose

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between Availability Zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

In addition to the AWS global infrastructure, Kinesis Data Firehose offers several features to help support your data resiliency and backup needs.

Disaster Recovery

Kinesis Data Firehose runs in a serverless mode, and takes care of host degradations, Availability Zone availability, and other infrastructure related issues by performing automatic migration. When this happens, Kinesis Data Firehose ensures that the delivery stream is migrated without any loss of data.

Infrastructure Security in Kinesis Data Firehose

As a managed service, Amazon Kinesis Data Firehose is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

You use AWS published API calls to access Kinesis Data Firehose through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

VPC Endpoints (PrivateLink)

Kinesis Data Firehose provides support for VPC endpoints (PrivateLink). For more information, see Using Kinesis Data Firehose with AWS PrivateLink (p. 109).

Security Best Practices for Kinesis Data Firehose

Amazon Kinesis Data Firehose provides a number of security features to consider as you develop and implement your own security policies. The following best practices are general guidelines and don't represent a complete security solution. Because these best practices might not be appropriate or sufficient for your environment, treat them as helpful considerations rather than prescriptions.

Implement least privilege access

When granting permissions, you decide who is getting what permissions to which Kinesis Data Firehose resources. You enable specific actions that you want to allow on those resources. Therefore you should grant only the permissions that are required to perform a task. Implementing least privilege access is fundamental in reducing security risk and the impact that could result from errors or malicious intent.

Use IAM roles

Producer and client applications must have valid credentials to access Kinesis Data Firehose delivery streams, and your Kinesis Data Firehose delivery stream must have valid credentials to access destinations. You should not store AWS credentials directly in a client application or in an Amazon S3 bucket. These are long-term credentials that are not automatically rotated and could have a significant business impact if they are compromised.

Instead, you should use an IAM role to manage temporary credentials for your producer and client applications to access Kinesis Data Firehose delivery streams. When you use a role, you don't have to use long-term credentials (such as a user name and password or access keys) to access other resources.

For more information, see the following topics in the *IAM User Guide*:

- IAM Roles
- Common Scenarios for Roles: Users, Applications, and Services

Implement Server-Side Encryption in Dependent Resources

Data at rest and data in transit can be encrypted in Kinesis Data Firehose. For more information, see Data Protection in Amazon Kinesis Data Firehose.

Use CloudTrail to Monitor API Calls

Kinesis Data Firehose is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Kinesis Data Firehose.

Using the information collected by CloudTrail, you can determine the request that was made to Kinesis Data Firehose, the IP address from which the request was made, who made the request, when it was made, and additional details.

For more information, see the section called “Logging Kinesis Data Firehose API Calls with AWS CloudTrail” (p. 100).

Amazon Kinesis Data Firehose Data Transformation

Kinesis Data Firehose can invoke your Lambda function to transform incoming source data and deliver the transformed data to destinations. You can enable Kinesis Data Firehose data transformation when you create your delivery stream.

Data Transformation Flow

When you enable Kinesis Data Firehose data transformation, Kinesis Data Firehose buffers incoming data up to 3 MB by default. (To adjust the buffering size, use the `ProcessingConfiguration` API with the `ProcessorParameter` called `BufferSizeInMBs`.) Kinesis Data Firehose then invokes the specified Lambda function asynchronously with each buffered batch using the AWS Lambda synchronous invocation mode. The transformed data is sent from Lambda to Kinesis Data Firehose. Kinesis Data Firehose then sends it to the destination when the specified destination buffering size or buffering interval is reached, whichever happens first.

Important
The Lambda synchronous invocation mode has a payload size limit of 6 MB for both the request and the response. Make sure that your buffering size for sending the request to the function is less than or equal to 6 MB. Also ensure that the response that your function returns doesn't exceed 6 MB.

Data Transformation and Status Model

All transformed records from Lambda must contain the following parameters, or Kinesis Data Firehose rejects them and treats that as a data transformation failure.

recordId

The record ID is passed from Kinesis Data Firehose to Lambda during the invocation. The transformed record must contain the same record ID. Any mismatch between the ID of the original record and the ID of the transformed record is treated as a data transformation failure.

result

The status of the data transformation of the record. The possible values are: `Ok` (the record was transformed successfully), `Dropped` (the record was dropped intentionally by your processing logic), and `ProcessingFailed` (the record could not be transformed). If a record has a status of `Ok` or `Dropped`, Kinesis Data Firehose considers it successfully processed. Otherwise, Kinesis Data Firehose considers it unsuccessfully processed.

data

The transformed data payload, after base64-encoding.

Lambda Blueprints

There are blueprints that you can use to create a Lambda function for data transformation. Some of these blueprints are in the AWS Lambda console and some are in the AWS Serverless Application Repository.

To see the blueprints that are available in the AWS Lambda console

1. Sign in to the AWS Management Console and open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. Choose **Create function**, and then choose **Use a blueprint**.
3. In the **Blueprints** field, search for the keyword `firehose` to find the Kinesis Data Firehose Lambda blueprints.

To see the blueprints that are available in the AWS Serverless Application Repository

1. Go to AWS Serverless Application Repository.
2. Choose **Browse all applications**.
3. In the **Applications** field, search for the keyword `firehose`.

You can also create a Lambda function without using a blueprint. See Getting Started with AWS Lambda.

Data Transformation Failure Handling

If your Lambda function invocation fails because of a network timeout or because you've reached the Lambda invocation limit, Kinesis Data Firehose retries the invocation three times by default. If the invocation does not succeed, Kinesis Data Firehose then skips that batch of records. The skipped records are treated as unsuccessfully processed records. You can specify or override the retry options using the CreateDeliveryStream or `UpdateDestination` API. For this type of failure, you can log invocation errors to Amazon CloudWatch Logs. For more information, see Monitoring Kinesis Data Firehose Using CloudWatch Logs (p. 93).

If the status of the data transformation of a record is `ProcessingFailed`, Kinesis Data Firehose treats the record as unsuccessfully processed. For this type of failure, you can emit error logs to Amazon CloudWatch Logs from your Lambda function. For more information, see Accessing Amazon CloudWatch Logs for AWS Lambda in the *AWS Lambda Developer Guide*.

If data transformation fails, the unsuccessfully processed records are delivered to your S3 bucket in the `processing-failed` folder. The records have the following format:

```
{
    "attemptsMade": "count",
    "arrivalTimestamp": "timestamp",
    "errorCode": "code",
    "errorMessage": "message",
    "attemptEndingTimestamp": "timestamp",
    "rawData": "data",
    "lambdaArn": "arn"
}
```

`attemptsMade`

The number of invocation requests attempted.

`arrivalTimestamp`

The time that the record was received by Kinesis Data Firehose.

`errorCode`

The HTTP error code returned by Lambda.

`errorMessage`

The error message returned by Lambda.

`attemptEndingTimestamp`

The time that Kinesis Data Firehose stopped attempting Lambda invocations.

`rawData`

The base64-encoded record data.

`lambdaArn`

The Amazon Resource Name (ARN) of the Lambda function.

Duration of a Lambda Invocation

Kinesis Data Firehose supports a Lambda invocation time of up to 5 minutes. If your Lambda function takes more than 5 minutes to complete, you get the following error: Firehose encountered timeout errors when calling AWS Lambda. The maximum supported function timeout is 5 minutes.

For information about what Kinesis Data Firehose does if such an error occurs, see the section called "Data Transformation Failure Handling" (p. 59).

Source Record Backup

Kinesis Data Firehose can back up all untransformed records to your S3 bucket concurrently while delivering transformed records to the destination. You can enable source record backup when you create or update your delivery stream. You cannot disable source record backup after you enable it.

Dynamic Partitioning in Kinesis Data Firehose

Dynamic partitioning enables you to continuously partition streaming data in Kinesis Data Firehose by using keys within data (for example, `customer_id` or `transaction_id`) and then deliver the data grouped by these keys into corresponding Amazon Simple Storage Service (Amazon S3) prefixes. This makes it easier to run high performance, cost-efficient analytics on streaming data in Amazon S3 using various services such as Amazon Athena, Amazon EMR, Amazon Redshift Spectrum, and Amazon QuickSight. In addition, AWS Glue can perform more sophisticated extract, transform, and load (ETL) jobs after the dynamically partitioned streaming data is delivered to Amazon S3, in use-cases where additional processing is required.

Partitioning your data minimizes the amount of data scanned, optimizes performance, and reduces costs of your analytics queries on Amazon S3. It also increases granular access to your data. Kinesis Data Firehose delivery streams are traditionally used in order to capture and load data into Amazon S3. To partition a streaming data set for Amazon S3-based analytics, you would need to run partitioning applications between Amazon S3 buckets prior to making the data available for analysis, which could become complicated or costly.

With dynamic partitioning, Kinesis Data Firehose continuously groups in-transit data using dynamically or statically defined data keys, and delivers the data to individual Amazon S3 prefixes by key. This reduces time-to-insight by minutes or hours. It also reduces costs and simplifies architectures.

Topics

Partitioning keys

With dynamic partitioning, you create targeted data sets from the streaming S3 data by partitioning the data based on partitioning keys. Partitioning keys enable you to filter your streaming data based on specific values. For example, if you need to filter your data based on customer ID and country, you can specify the data field of `customer_id` as one partitioning key and the data field of `country` as another partitioning key. Then, you specify the expressions (using the supported formats) to define the S3 bucket prefixes to which the dynamically partitioned data records are to be delivered.

The following are the supported methods of creating partitioning keys:

- **Inline parsing** - this method uses Amazon Kinesis Data Firehose built-in support mechanism, a jq parser, for extracting the keys for partitioning from data records that are in JSON format.
- **AWS Lambda function** - this method uses a specified AWS Lambda function to extract and return the data fields needed for partitioning.

Important
When you enable dynamic partitioning, you must configure at least one of these methods to partition your data. You can configure either of these methods to specify your partitioning keys or both of them at the same time.

Creating partitioning keys with inline parsing

To configure inline parsing as the dynamic partitioning method for your streaming data, you must choose data record parameters to be used as partitioning keys and provide a value for each specified partitioning key.

Let's look at the following sample data record and and see how you can define partitioning keys for it with inline parsing:

```
{
   "type": {
    "device": "mobile",
    "event": "user_clicked_submit_button"
  },
  "customer_id": "1234567890",
  "event_timestamp": 1565382027,   #epoch timestamp
  "region": "sample_region"
}
```

For example, you can choose to partition your data based on the `customer_id` parameter or the `event_timestamp` parameter. This means that you want the value of the `customer_id` parameter or the `event_timestamp` parameter in each record to be used in determining the S3 prefix to which the record is to be delivered. You can also choose a nested parameter, like `device` with an expression `.type.device`. Your dynamic partitioning logic can depend on multiple parameters.

After selecting data parameters for your partitioning keys, you then map each parameter to a valid jq expression. The following table shows such a mapping of parameters to jq expressions:

Parameter	jq expression
`customer_id`	.customer_id
`device`	.type.device
`year`	.event_timestamp| strftime("%Y")
`month`	.event_timestamp| strftime("%m")
`day`	.event_timestamp| strftime("%d")
`hour`	.event_timestamp| strftime("%H")

At runtime, Kinesis Data Firehose uses the right column above to evaluate the parameters based on the data in each record.

Creating partitioning keys with an AWS Lambda function

For compressed or encrypted data records, or data that is in any file format other than JSON, you can use the integrated AWS Lambda function with your own custom code to decompress, decrypt, or transform the records in order to extract and return the data fields needed for partitioning. This is an expansion of the existing transform Lambda function that is available today with Kinesis Data Firehose. You can

transform, parse and return the data fields that you can then use for dynamic partitioning using the same Lambda function.

The following is an example Amazon Kinesis Firehose delivery stream processing Lambda function in Python that replays every read record from input to output and extracts partioninig keys from the records.

```
from __future__ import print_function
import base64
import json
import datetime

# Signature for all Lambda functions that user must implement
def lambda_handler(firehose_records_input, context):
    print("Received records for processing from DeliveryStream: " +
 firehose_records_input['deliveryStreamArn']
          + ", Region: " + firehose_records_input['region']
          + ", and InvocationId: " + firehose_records_input['invocationId'])

    # Create return value.
    firehose_records_output = {'records': []}

    # Create result object.
    # Go through records and process them

    for firehose_record_input in firehose_records_input['records']:
        # Get user payload
        payload = base64.b64decode(firehose_record_input['data'])
        json_value = json.loads(payload)

        print("Record that was received")
        print(json_value)
        print("\n")
        # Create output Firehose record and add modified payload and record ID to it.
        firehose_record_output = {}
        event_timestamp = datetime.datetime.fromtimestamp(json_value['eventTimestamp'])
        partition_keys = {"customerId": json_value['customerId'],
                          "year": event_timestamp.strftime('%Y'),
                          "month": event_timestamp.strftime('%m'),
                          "date": event_timestamp.strftime('%d'),
                          "hour": event_timestamp.strftime('%H'),
                          "minute": event_timestamp.strftime('%M')
                          }

        # Create output Firehose record and add modified payload and record ID to it.
        firehose_record_output = {'recordId': firehose_record_input['recordId'],
                                  'data': firehose_record_input['data'],
                                  'result': 'Ok',
                                  'metadata': { 'partitionKeys': partition_keys }}

        # Must set proper record ID
        # Add the record to the list of output records.

        firehose_records_output['records'].append(firehose_record_output)

    # At the end return processed records
    return firehose_records_output
```

The following is an example Amazon Kinesis Firehose delivery stream processing Lambda function in Go that replays every read record from input to output and extracts partioninig keys from the records.

```
package main
```

```
import (
 "fmt"
 "encoding/json"
 "time"
 "strconv"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

type KinesisFirehoseEventRecordData struct {
 CustomerId string `json:"customerId"`
}

func handleRequest(evnt events.KinesisFirehoseEvent) (events.KinesisFirehoseResponse,
 error) {

 fmt.Printf("InvocationID: %s\n", evnt.InvocationID)
 fmt.Printf("DeliveryStreamArn: %s\n", evnt.DeliveryStreamArn)
 fmt.Printf("Region: %s\n", evnt.Region)

 var response events.KinesisFirehoseResponse

 for _, record := range evnt.Records {
  fmt.Printf("RecordID: %s\n", record.RecordID)
  fmt.Printf("ApproximateArrivalTimestamp: %s\n", record.ApproximateArrivalTimestamp)

  var transformedRecord events.KinesisFirehoseResponseRecord
  transformedRecord.RecordID = record.RecordID
  transformedRecord.Result = events.KinesisFirehoseTransformedStateOk
  transformedRecord.Data = record.Data

  var metaData events.KinesisFirehoseResponseRecordMetadata
  var recordData KinesisFirehoseEventRecordData
  partitionKeys := make(map[string]string)

  currentTime := time.Now()
  json.Unmarshal(record.Data, &recordData)
  partitionKeys["customerId"] = recordData.CustomerId
  partitionKeys["year"] = strconv.Itoa(currentTime.Year())
  partitionKeys["month"] = strconv.Itoa(int(currentTime.Month()))
  partitionKeys["date"] = strconv.Itoa(currentTime.Day())
  partitionKeys["hour"] = strconv.Itoa(currentTime.Hour())
  partitionKeys["minute"] = strconv.Itoa(currentTime.Minute())
  metaData.PartitionKeys = partitionKeys
  transformedRecord.Metadata = metaData

  response.Records = append(response.Records, transformedRecord)
 }

 return response, nil
}

func main() {
 lambda.Start(handleRequest)
}
```

Amazon S3 Bucket Prefix for Dynamic Partitioning

When you create a delivery stream that uses Amazon S3 as the destination, you must specify an Amazon S3 bucket where Kinesis Data Firehose is to deliver your data. Amazon S3 bucket prefixes are used to organize the data that you store in your S3 buckets. An Amazon S3 bucket prefix is similar to a directory that enables you to group similar objects together.

With dynamic partitioning, your partitioned data is delivered into the specified Amazon S3 prefixes. If you don't enable dynamic partitioning, specifying an S3 bucket prefix for your delivery stream is optional. However, if you choose to enable dynamic partitioning, you MUST specify the S3 bucket prefixes to which Kinesis Data Firehose is to deliver partitioned data.

In every delivery stream where you enable dynamic partitioning, the S3 bucket prefix value consists of expressions based on the specified partitioning keys for that delivery stream. Using the above data record example again, you can build the following S3 prefix value that consists of expressions based on the partitioning keys defined above:

```
"ExtendedS3DestinationConfiguration": {
"BucketARN": "arn:aws:s3:::my-logs-prod",
"Prefix": "customer_id=!{partitionKeyFromQuery:customer_id}/
    device=!{partitionKeyFromQuery:device}/
    year=!{partitionKeyFromQuery:year}/
    month=!{partitionKeyFromQuery:month}/
    day=!{partitionKeyFromQuery:day}/
    hour=!{partitionKeyFromQuery:hour}/"
}
```

Kinesis Data firehose evaluates the above expression at runtime. It groups records that match the same evaluated S3 prefix expression into a single data set. Kinesis Data Firehose then delivers each data set to the evaluated S3 prefix. The frequency of data set delivery to S3 is determined by the delivery stream buffer setting. As a result, the record in this example is delivered to the following S3 object key:

```
s3://my-logs-prod/customer_id=1234567890/device=mobile/year=2019/month=08/day=09/hour=20/
my-delivery-stream-2019-08-09-23-55-09-a9fa96af-e4e4-409f-bac3-1f804714faaa
```

For dynamic partitioning, you must use the following expression format in your S3 bucket prefix: `!{namespace:value}`, where namespace can be either `partitionKeyFromQuery` or `partitionKeyFromLambda`, or both. If you are using inline parsing to create the partitioning keys for your source data, you must specify an S3 bucket prefix value that consists of expressions specified in the following format: `"partitionKeyFromQuery:keyID"`. If you are using an AWS Lambda function to create partitioning keys for your source data, you must specify an S3 bucket prefix value that consists of expressions specified in the following format: `"partitionKeyFromLambda:keyID"`.

> **Note**
> You can also specify the S3 bucket prefix value using the hive style format, for example customer_id=!{partitionKeyFromQuery:customer_id}.

For more information, see the "Choose Amazon S3 for Your Destination" in Creating an Amazon Kinesis Data Firehose Delivery Stream and Custom Prefixes for Amazon S3 Objects.

Dynamic partitioning of aggregated data

You can apply dynamic partitioning to aggregated data (for example, multiple events, logs, or records aggregated into a single `PutRecord` and `PutRecordBatch` API call) but this data must first be deaggregated. You can deaggregate your data by enabling multi record deaggregation - the process of parsing through the records in the delivery stream and separating them. Multi record deaggregation can either be of `JSON` type, meaning that the separation of records is performed based on valid JSON. Or it can be of the `Delimited` type, meaning that the separation of records is performed based on a specified custom delimiter. This custom delimiter must be a base-64 encoded string. For example, if you want to use the following string as your custom delimeter ####, you must specify it in the base-64 ecoded format, which translates it to `IyMjIw==`.

With aggregated data, when you enable dynamic partitioning, Kinesis Data Firehose parses the records and looks for either valid JSON objects or delimited records within each API call based on the specified multi record deaggregation type.

Important
If your data is aggregated, dynamic partitioning can be only be applied if your data is first deaggregated.

Adding a new line delimiter when delivering data to S3

When you enable dynamic partitioning, you can configure your delivery stream to add a new line delimiter between records in objects that are delivered to Amazon S3. This can be helpful for parsing objects in Amazon S3. This is also particularly useful when dynamic partitioning is applied to aggregated data because multirecord deaggregation (which must be applied to aggregated data before it can be dynamically partitioned) removes new lines from records as part of the parsing process.

How to enable dynamic partitioning

You can configure dynamic partitioning for your delivery streams through the Kinesis Data Firehose Management Console, CLI, or the APIs.

Important
You can enable dynamic partitioning only when you create a new delivery stream. You cannot enable dynamic partitioning for an existing delivery stream that does not have dynamic partitioning already enabled.

For detailed steps on how to enable and configure dynamic partitioning through the Amazon Kinesis Data Firehose management console while creating a new delivery stream, see Creating an Amazon Kinesis Data Firehose Delivery Stream. When you get to the task of specifying the destination for your delivery stream, make sure to follow the steps in the Choose Amazon S3 for Your Destination section, since currently, dynamic partitioning is only supported for delivery streams that use Amazon S3 as the destination.

Once dynamic partitioning on an active delivery stream is enabled, you can update the configuration by adding new or removing or updating existing partitioning keys and the S3 prefix expressions. Once updated, Amazon Kinesis Data Firehose starts using the new keys and the new S3 prefix expressions.

Important
Once you enable dynamic partitioning on a delivery stream, it cannot be disabled on this delivery stream.

Dynamic Partitioning Error Handling

If Kinesis Data Firehose is not able to parse data records in your delivery stream or it fails to extract the specified partitioning keys, or to evaluate the expressions included in the S3 prefix value, these data records are delivered to the S3 error bucket prefix that you must specify when you create the delivery stream where you enable dynamic partitioning. The S3 error bucket prefix contains all the records that Kinesis Data Firehose is not able to deliver to the specified S3 destination. These records are organized based on the error type. Along with the record, the delivered object also includes information about the error to help understand and resolve the error.

You MUST specify an S3 error bucket prefix for a delivery stream if you want to enable dynamic partitioning for this delivery stream. If you don't want to enable dynamic partitioning for a delivery stream, specifying an S3 error bucket prefix is optional.

Data buffering and dynamic partitioning

Amazon Kinesis Data Firehose buffers incoming streaming data to a certain size and for a certain period of time before delivering it to the specified destinations. You can configure the buffer size and the buffer interval while creating new delivery streams or update the buffer size and the buffer interval on your existing delivery streams. A buffer size is measured in MBs and a buffer interval is measured in seconds.

For a delivery stream where data partitioning is enabled, Kinesis Data Firehose creates one buffer per each partition in run time based on the record payload. For a delivery stream where data partitioning is enabled, the buffer size ranges from 64MB to 128MB, with the default set to 128MB, and the buffer interval ranges from 60 seconds to 900 seconds. A max throughput of 25 MB per second is supported for each active partition.

When dynamic partitioning on a delivery stream is enabled, there is a limit of 500 active partitions that can be created for that delivery stream. You can use the Amazon Kinesis Data Firehose Limits form to request an increase of this quota. A new partition is created when an S3 prefix is evaluated to a new value based on the record data fields and the S3 prefix expressions. A new buffer is created for each active partition. Every subsequent record with the same evaluated S3 prefix is delivered to that buffer. Once the buffer meets the buffer size limit or the buffer time interval, Amazon Kinesis Data Firehose creates an object with the buffer data and delivers it to the specified Amazon S3 prefix. Once the object is delivered, the buffer for that partition and the partition itself are deleted and removed from the active partitions count. Amazon Kinesis Data Firehose delivers each buffer data as a single object once the buffer size or interval are met for each partition separately. Once the number of active partitions reaches the limit of 500 per deliver stream, the rest of the records in the delivery stream are delivered to the specified S3 error bucket prefix.

Converting Your Input Record Format in Kinesis Data Firehose

Amazon Kinesis Data Firehose can convert the format of your input data from JSON to Apache Parquet or Apache ORC before storing the data in Amazon S3. Parquet and ORC are columnar data formats that save space and enable faster queries compared to row-oriented formats like JSON. If you want to convert an input format other than JSON, such as comma-separated values (CSV) or structured text, you can use AWS Lambda to transform it to JSON first. For more information, see *Data Transformation* (p. 58).

Topics

Record Format Conversion Requirements

Kinesis Data Firehose requires the following three elements to convert the format of your record data:

- **A deserializer to read the JSON of your input data** – You can choose one of two types of deserializers: Apache Hive JSON SerDe or OpenX JSON SerDe.

  Note
  When combining multiple JSON documents into the same record, make sure that your input is still presented in the supported JSON format. An array of JSON documents is NOT a valid input.
  For example, this is the correct input: `{"a":1}{"a":2}`
  And this is the INCORRECT input: `[{"a":1}, {"a":2}]`

- **A schema to determine how to interpret that data** – Use AWS Glue to create a schema in the AWS Glue Data Catalog. Kinesis Data Firehose then references that schema and uses it to interpret your input data. You can use the same schema to configure both Kinesis Data Firehose and your analytics software. For more information, see Populating the AWS Glue Data Catalog in the *AWS Glue Developer Guide*.
- **A serializer to convert the data to the target columnar storage format (Parquet or ORC)** – You can choose one of two types of serializers: ORC SerDe or Parquet SerDe.

  Important
  If you enable record format conversion, you can't set your Kinesis Data Firehose destination to be Amazon OpenSearch Service (OpenSearch Service), Amazon Redshift, or Splunk. With format conversion enabled, Amazon S3 is the only destination that you can use for your Kinesis Data Firehose delivery stream.

You can convert the format of your data even if you aggregate your records before sending them to Kinesis Data Firehose.

Choosing the JSON Deserializer

Choose the OpenX JSON SerDe if your input JSON contains time stamps in the following formats:

- yyyy-MM-dd'T'HH:mm:ss[.S]'Z', where the fraction can have up to 9 digits – For example, `2017-02-07T15:13:01.39256Z`.
- yyyy-[M]M-[d]d HH:mm:ss[.S], where the fraction can have up to 9 digits – For example, `2017-02-07 15:13:01.14`.
- Epoch seconds – For example, `1518033528`.
- Epoch milliseconds – For example, `1518033528123`.
- Floating point epoch seconds – For example, `1518033528.123`.

The OpenX JSON SerDe can convert periods (`.`) to underscores (_). It can also convert JSON keys to lowercase before deserializing them. For more information about the options that are available with this deserializer through Kinesis Data Firehose, see OpenXJsonSerDe.

If you're not sure which deserializer to choose, use the OpenX JSON SerDe, unless you have time stamps that it doesn't support.

If you have time stamps in formats other than those listed previously, use the Apache Hive JSON SerDe. When you choose this deserializer, you can specify the time stamp formats to use. To do this, follow the pattern syntax of the Joda-Time `DateTimeFormat` format strings. For more information, see Class DateTimeFormat.

You can also use the special value `millis` to parse time stamps in epoch milliseconds. If you don't specify a format, Kinesis Data Firehose uses `java.sql.Timestamp::valueOf` by default.

The Hive JSON SerDe doesn't allow the following:

- Periods (`.`) in column names.
- Fields whose type is `uniontype`.
- Fields that have numerical types in the schema, but that are strings in the JSON. For example, if the schema is (an int), and the JSON is `{"a":"123"}`, the Hive SerDe gives an error.

The Hive SerDe doesn't convert nested JSON into strings. For example, if you have `{"a": {"inner":1}}`, it doesn't treat `{"inner":1}` as a string.

Choosing the Serializer

The serializer that you choose depends on your business needs. To learn more about the two serializer options, see ORC SerDe and Parquet SerDe.

Converting Input Record Format (Console)

You can enable data format conversion on the console when you create or update a Kinesis delivery stream. With data format conversion enabled, Amazon S3 is the only destination that you can configure for the delivery stream. Also, Amazon S3 compression gets disabled when you enable format conversion. However, Snappy compression happens automatically as part of the conversion process. The framing format for Snappy that Kinesis Data Firehose uses in this case is compatible with Hadoop. This means that you can use the results of the Snappy compression and run queries on this data in Athena. For the Snappy framing format that Hadoop relies on, see BlockCompressorStream.java.

To enable data format conversion for a data delivery stream

1. Sign in to the AWS Management Console, and open the Kinesis Data Firehose console at https://console.aws.amazon.com/firehose/.
2. Choose a Kinesis Data Firehose delivery stream to update, or create a new delivery stream by following the steps in Creating an Amazon Kinesis Data Firehose Delivery Stream (p. 5).
3. Under **Convert record format**, set **Record format conversion** to **Enabled**.
4. Choose the output format that you want. For more information about the two options, see Apache Parquet and Apache ORC.
5. Choose an AWS Glue table to specify a schema for your source records. Set the Region, database, table, and table version.

Converting Input Record Format (API)

If you want Kinesis Data Firehose to convert the format of your input data from JSON to Parquet or ORC, specify the optional DataFormatConversionConfiguration element in ExtendedS3DestinationConfiguration or in ExtendedS3DestinationUpdate. If you specify DataFormatConversionConfiguration, the following restrictions apply:

- In BufferingHints, you can't set `SizeInMBs` to a value less than 64 if you enable record format conversion. Also, when format conversion isn't enabled, the default value is 5. The value becomes 128 when you enable it.
- You must set `CompressionFormat` in ExtendedS3DestinationConfiguration or in ExtendedS3DestinationUpdate to `UNCOMPRESSED`. The default value for `CompressionFormat` is `UNCOMPRESSED`. Therefore, you can also leave it unspecified in ExtendedS3DestinationConfiguration. The data still gets compressed as part of the serialization process, using Snappy compression by default. The framing format for Snappy that Kinesis Data Firehose uses in this case is compatible with Hadoop. This means that you can use the results of the Snappy compression and run queries on this data in Athena. For the Snappy framing format that Hadoop relies on, see BlockCompressorStream.java. When you configure the serializer, you can choose other types of compression.

Record Format Conversion Error Handling

When Kinesis Data Firehose can't parse or deserialize a record (for example, when the data doesn't match the schema), it writes it to Amazon S3 with an error prefix. If this write fails, Kinesis Data Firehose retries it forever, blocking further delivery. For each failed record, Kinesis Data Firehose writes a JSON document with the following schema:

```
{
  "attemptsMade": long,
  "arrivalTimestamp": long,
  "lastErrorCode": string,
  "lastErrorMessage": string,
  "attemptEndingTimestamp": long,
  "rawData": string,
  "sequenceNumber": string,
  "subSequenceNumber": long,
  "dataCatalogTable": {
    "catalogId": string,
    "databaseName": string,
    "tableName": string,
    "region": string,
    "versionId": string,
```

```
    "catalogArn": string
  }
}
```

Record Format Conversion Example

For an example of how to set up record format conversion with AWS CloudFormation, see AWS::KinesisFirehose::DeliveryStream.

Using Amazon Kinesis Data Analytics

Create a Kinesis Data Analytics Application That Reads from a Delivery Stream

1. Sign in to the AWS Management Console and open the Kinesis Data Analytics console at https://console.aws.amazon.com/kinesisanalytics.
2. Choose **Create application**
3. Specify a name for the application and an optional description. Then choose **Create application**.
4. Choose **Connect streaming data**.
5. For **Source**, choose **Kinesis Firehose delivery stream**.
6. In the list labeled **Kinesis Firehose delivery stream**, choose the delivery stream that you want your Kinesis Data Analytics to process. Alternatively, choose **Create new** to set up a new delivery stream.
7. To finish setting up your Kinesis Data Analytics application, see Getting Started with Amazon Kinesis Data Analytics for SQL Applications.

Write Data from a Kinesis Data Analytics Application to a Delivery Stream

1. To create a Kinesis Data Analytics application, follow the instructions under Getting Started with Amazon Kinesis Data Analytics for SQL Applications.
2. Open the Kinesis Data Analytics console at https://console.aws.amazon.com/kinesisanalytics.
3. In the list of application, choose the application that you want to configure to write to a delivery stream.
4. Choose **Application details**.
5. At the bottom of the page, choose **Connect to a destination**.
6. Choose **Kinesis Firehose delivery stream**, and then choose an existing delivery stream in the list or choose **Create new** to create a new delivery stream.

Amazon Kinesis Data Firehose Data Delivery

After data is sent to your delivery stream, it is automatically delivered to the destination you choose.

Important
If you use the Kinesis Producer Library (KPL) to write data to a Kinesis data stream, you can use aggregation to combine the records that you write to that Kinesis data stream. If you then use that data stream as a source for your Kinesis Data Firehose delivery stream, Kinesis Data Firehose de-aggregates the records before it delivers them to the destination. If you configure your delivery stream to transform the data, Kinesis Data Firehose de-aggregates the records before it delivers them to AWS Lambda. For more information, see Developing Amazon Kinesis Data Streams Producers Using the Kinesis Producer Library and Aggregation in the *Amazon Kinesis Data Streams Developer Guide*.

Topics

Data Delivery Format

For data delivery to Amazon Simple Storage Service (Amazon S3), Kinesis Data Firehose concatenates multiple incoming records based on the buffering configuration of your delivery stream. It then delivers the records to Amazon S3 as an Amazon S3 object. You might want to add a record separator at the end of each record before you send it to Kinesis Data Firehose. Then you can divide a delivered Amazon S3 object to individual records.

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

For data delivery to Amazon ES, Kinesis Data Firehose buffers incoming records based on the buffering configuration of your delivery stream. It then generates an Elasticsearch bulk request to index multiple records to your Elasticsearch cluster. Make sure that your record is UTF-8 encoded and flattened to a single-line JSON object before you send it to Kinesis Data Firehose. Also, the `rest.action.multi.allow_explicit_index` option for your Elasticsearch cluster must be set to true (default) to take bulk requests with an explicit index that is set per record. For more information, see Amazon ES Configure Advanced Options in the *Amazon OpenSearch Service Developer Guide*.

For data delivery to Splunk, Kinesis Data Firehose concatenates the bytes that you send. If you want delimiters in your data, such as a new line character, you must insert them yourself. Make sure that Splunk is configured to parse any such delimiters.

When delivering data to an HTTP endpoint owned by a supported third-party service provider, you can use the integrated Amazon 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. Contact the third-party service provider whose HTTP endpoint you've chosen for your destination to learn more about their accepted record format.

Data Delivery Frequency

Each Kinesis Data Firehose destination has its own data delivery frequency.

Amazon S3

The frequency of data delivery to Amazon S3 is determined by the Amazon S3 **Buffer size** and **Buffer interval** value that you configured for your delivery stream. Kinesis Data Firehose buffers incoming data before it delivers it to Amazon S3. You can configure the values for Amazon S3 **Buffer size** (1–128 MB) or **Buffer interval** (60–900 seconds). The condition satisfied first triggers data delivery to Amazon S3. When data delivery to the destination falls behind data writing to the delivery stream, Kinesis Data Firehose raises the buffer size dynamically. It can then catch up and ensure that all data is delivered to the destination.

Amazon Redshift

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 Data Firehose issues a new **COPY** command as soon as the previous **COPY** command is successfully finished by Amazon Redshift.

Amazon OpenSearch Service

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 Data Firehose buffers incoming data before delivering it to Amazon ES. You can configure the values for Elasticsearch **Buffer size** (1–100 MB) or **Buffer interval** (60–900 seconds), and the condition satisfied first triggers data delivery to Amazon ES.

Splunk

Kinesis Data Firehose buffers incoming data before delivering it to Splunk. The buffer size is 5 MB, and the buffer interval is 60 seconds. The condition satisfied first triggers data delivery to Splunk. The buffer size and interval aren't configurable. These numbers are optimal.

HTTP endpoint destination

Kinesis Data Firehose buffers incoming data before delivering it to the specified HTTP endpoint destination. The recommended buffer size for the destination varies from service provider to service provider. For example, the recommended buffer size for Datadog is 4 MiBs and the recommended buffer size for New Relic and Sumo Logic is 1 MiB. Contact the third-party service provider whose endpoint you've chosen as your data destination for more information about their recommended buffer size.

Data Delivery Failure Handling

Each Kinesis Data Firehose destination has its own data delivery failure handling.

Amazon S3

Data delivery to your S3 bucket might fail for various reasons. For example, the bucket might not exist anymore, the IAM role that Kinesis Data Firehose assumes might not have access to the bucket,

the network failed, or similar events. Under these conditions, Kinesis Data Firehose keeps retrying for up to 24 hours until the delivery succeeds. The maximum data storage time of Kinesis Data Firehose is 24 hours. If data delivery fails for more than 24 hours, your data is lost.

Amazon Redshift

For an Amazon Redshift destination, you can specify a retry duration (0–7200 seconds) when creating a delivery stream.

Data delivery to your Amazon Redshift cluster might fail for several reasons. For example, you might have an incorrect cluster configuration of your delivery stream, a cluster under maintenance, or a network failure. Under these conditions, Kinesis Data Firehose retries for the specified time duration and skips that particular batch of Amazon S3 objects. 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. For information about how to COPY data manually with manifest files, see Using a Manifest to Specify Data Files.

Amazon OpenSearch Service

For the Amazon ES destination, you can specify a retry duration (0–7200 seconds) when creating a delivery stream.

Data delivery to your Amazon ES cluster might fail for several reasons. For example, you might have an incorrect Amazon ES cluster configuration of your delivery stream, an Amazon ES cluster under maintenance, a network failure, or similar events. Under these conditions, Kinesis Data 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. Each document has the following JSON format:

```
{
    "attemptsMade": "(number of index requests attempted)",
    "arrivalTimestamp": "(the time when the document was received by Firehose)",
    "errorCode": "(http error code returned by Elasticsearch)",
    "errorMessage": "(error message returned by Elasticsearch)",
    "attemptEndingTimestamp": "(the time when Firehose stopped attempting index
 request)",
    "esDocumentId": "(intended Elasticsearch document ID)",
    "esIndexName": "(intended Elasticsearch index name)",
    "esTypeName": "(intended Elasticsearch type name)",
    "rawData": "(base64-encoded document data)"
}
```

Splunk

When Kinesis Data Firehose sends data to Splunk, it waits for an acknowledgment from Splunk. If an error occurs, or the acknowledgment doesn't arrive within the acknowledgment timeout period, Kinesis Data Firehose starts the retry duration counter. It keeps retrying until the retry duration expires. After that, Kinesis Data Firehose considers it a data delivery failure and backs up the data to your Amazon S3 bucket.

Every time Kinesis Data Firehose sends data to Splunk, whether it's the initial attempt or a retry, it restarts the acknowledgement timeout counter. It then waits for an acknowledgement to arrive from Splunk. Even if the retry duration expires, Kinesis Data Firehose still waits for the acknowledgment until it receives it or the acknowledgement timeout is reached. If the acknowledgment times out, Kinesis Data Firehose checks to determine whether there's time left in the retry counter. If there is time left, it retries again and repeats the logic until it receives an acknowledgment or determines that the retry time has expired.

A failure to receive an acknowledgement isn't the only type of data delivery error that can occur. For information about the other types of data delivery errors, see Splunk Data Delivery Errors. Any data delivery error triggers the retry logic if your retry duration is greater than 0.

The following is an example error record.

```
{
  "attemptsMade": 0,
  "arrivalTimestamp": 1506035354675,
  "errorCode": "Splunk.AckTimeout",
  "errorMessage": "Did not receive an acknowledgement from HEC before the HEC
 acknowledgement timeout expired. Despite the acknowledgement timeout, it's possible
 the data was indexed successfully in Splunk. Kinesis Firehose backs up in Amazon S3
 data for which the acknowledgement timeout expired.",
  "attemptEndingTimestamp": 13626284715507,
  "rawData":
 "MiAyNTE2MjAyNzIyMDkgZW5pLTA1ZjMyMmQ1IDIxOC45Mi4xODguMjE0IDE3Mi4xNi4xLjE2NyAyNTIzMyAxNDMzIDYgMSA0M
  "EventId": "49577193928114147339600778471082492393164139877200035842.0"
}
```

HTTP endpoint destination

When Kinesis Data Firehose sends data to an HTTP endpoint destination, it waits for a response from this destination. If an error occurs, or the response doesn't arrive within the response timeout period, Kinesis Data Firehose starts the retry duration counter. It keeps retrying until the retry duration expires. After that, Kinesis Data Firehose considers it a data delivery failure and backs up the data to your Amazon S3 bucket.

Every time Kinesis Data Firehose sends data to an HTTP endpoint destination, whether it's the initial attempt or a retry, it restarts the response timeout counter. It then waits for a response to arrive from the HTTP endpoint destination. Even if the retry duration expires, Kinesis Data Firehose still waits for the response until it receives it or the response timeout is reached. If the response times out, Kinesis Data Firehose checks to determine whether there's time left in the retry counter. If there is time left, it retries again and repeats the logic until it receives a response or determines that the retry time has expired.

A failure to receive a response isn't the only type of data delivery error that can occur. For information about the other types of data delivery errors, see HTTP Endpoint Data Delivery Errors

The following is an example error record.

```
{
 "attemptsMade":5,
 "arrivalTimestamp":1594265943615,
 "errorCode":"HttpEndpoint.DestinationException",
 "errorMessage":"Received the following response from the endpoint destination.
 {"requestId": "109777ac-8f9b-4082-8e8d-b4f12b5fc17b", "timestamp": 1594266081268,
 "errorMessage": "Unauthorized"}",
 "attemptEndingTimestamp":1594266081318,
 "rawData":"c2FtcGxlIHJhdyBkYXRh",
 "subsequenceNumber":0,
 "dataId":"49607357361271740811418664280693044274821622880012337186.0"
}
```

Amazon S3 Object Name Format

Kinesis Data Firehose adds a UTC time prefix in the format `YYYY/MM/dd/HH` before writing objects to Amazon S3. This prefix creates a logical hierarchy in the bucket, where each forward slash (/) creates a level in the hierarchy. You can modify this structure by specifying a custom prefix. For information about how to specify a custom prefix, see *Custom Amazon S3 Prefixes* (p. 105).

The Amazon S3 object name follows the pattern `DeliveryStreamName-DeliveryStreamVersion-YYYY-MM-dd-HH-MM-SS-RandomString`, where `DeliveryStreamVersion` begins with `1` and increases by 1 for every configuration change of the Kinesis Data Firehose delivery stream. You can change delivery stream configurations (for example, the name of the S3 bucket, buffering hints, compression, and encryption). You can do so by using the Kinesis Data Firehose console or the UpdateDestination API operation.

Index Rotation for the Amazon ES Destination

For the Amazon ES destination, you can specify a time-based index rotation option from one of the following five options: **NoRotation**, **OneHour**, **OneDay**, **OneWeek**, or **OneMonth**.

Depending on the rotation option you choose, Kinesis Data Firehose appends a portion of the UTC arrival timestamp to your specified index name. It rotates the appended timestamp accordingly. The following example shows the resulting index name in Amazon ES for each index rotation option, where the specified index name is **myindex** and the arrival timestamp is `2016-02-25T13:00:00Z`.

RotationPeriod	IndexName
`NoRotation`	`myindex`
`OneHour`	`myindex-2016-02-25-13`
`OneDay`	`myindex-2016-02-25`
`OneWeek`	`myindex-2016-w08`
`OneMonth`	`myindex-2016-02`

Note
With the `OneWeek` option, Data Firehose auto-create indexes using the format of <YEAR>-w<WEEK NUMBER> (for example, `2020-w33`), where the week number is calculated using UTC time and according to the following US conventions:

- A week starts on Sunday
- The first week of the year is the first week that contains a Saturday in this year

Delivery Across AWS Accounts and Across AWS Regions for HTTP Endpoint Destinations

Kinesis Data Firehose supports data delivery to HTTP endpoint destinations across AWS accounts. Kinesis Data Firehose delivery stream and the HTTP endpoint that you've chosen as your destination can be in different AWS accounts.

Kinesis Data Firehose also supports data delivery to HTTP endpoint destinations across AWS regions. You can deliver data from a delivery stream in one AWS region to an HTTP endpoint in another AWS region. You can also delivery data from a delivery stream to an HTTP endpoint destination outside of AWS regions, for example to your own on-premises server by setting the HTTP endpoint URL to your desired destination. For these scenarios, additional data transfer charges are added to your delivery costs. For more information, see the Data Transfer section in the "On-Demand Pricing" page.

Duplicated Records

Kinesis Data Firehose uses at-least-once semantics for data delivery. In some circumstances, such as when data delivery times out, delivery retries by Kinesis Data Firehose might introduce duplicates if the original data-delivery request eventually goes through. This applies to all destination types that Kinesis Data Firehose supports.

Monitoring Amazon Kinesis Data Firehose

You can monitor Amazon Kinesis Data Firehose using the following features:

- Amazon CloudWatch metrics (p. 79)— Kinesis Data Firehose sends Amazon CloudWatch custom metrics with detailed monitoring for each delivery stream.
- Amazon CloudWatch Logs (p. 93)— Kinesis Data Firehose sends CloudWatch custom logs with detailed monitoring for each delivery stream.
- Kinesis Agent (p. 99)— Kinesis Agent publishes custom CloudWatch metrics to help assess whether the agent is working as expected.
- API logging and history (p. 100)— Kinesis Data Firehose uses AWS CloudTrail to log API calls and store the data in an Amazon S3 bucket, and to maintain API call history.

Monitoring Kinesis Data Firehose Using CloudWatch Metrics

Kinesis Data Firehose integrates with Amazon CloudWatch metrics so that you can collect, view, and analyze CloudWatch metrics for your Kinesis Data Firehose delivery streams. For example, you can monitor the `IncomingBytes` and `IncomingRecords` metrics to keep track of data ingested into Kinesis Data Firehose from data producers.

The metrics that you configure for your Kinesis Data Firehose delivery streams and agents are automatically collected and pushed to CloudWatch every five minutes. Metrics are archived for two weeks; after that period, the data is discarded.

The metrics collected for Kinesis Data Firehose delivery streams are free of charge. For information about Kinesis agent metrics, see Monitoring Kinesis Agent Health (p. 99).

Topics

- Logging Kinesis Data Firehose API Calls with AWS CloudTrail (p. 100)

Dynamic Partitioning CloudWatch Metrics

If dynamic partitioning is enabled, the AWS/Firehose namespace includes the following metrics.

Metric	Description
`PartitionCount`	The number of partitions that are being processed, in other words, the active partition count. This number varies between 1 and the partition count limit of 500 (default). Units: Count
`PartitionCountExceeded`	This metric indicates if you are exceeding the partition count limit. It emits 1 or 0 based on whether limit is breached or not.
`JQProcessing.Duration`	Returns the amount of time it took to execute JQ expression in the JQ Lambda function. Units: Milliseconds
`PerPartitionThroughput`	Indicates the throughtput that is being processed per partition. This metric enables you to monitor the per partition throughput. A max throughput of 25 MB per second is supported for each active partition. Units: StandardUnit.BytesSecond
`DeliveryToS3.ObjectCount`	Indicates the number of objects that are being delivered to your S3 bucket. Units: Count

Data Delivery CloudWatch Metrics

The `AWS/Firehose` namespace includes the following service-level metrics. If you see small drops in the average for `BackupToS3.Success`, `DeliveryToS3.Success`, `DeliveryToSplunk.Success`, `DeliveryToElasticsearch.Success`, or `DeliveryToRedshift.Success`, that doesn't indicate that there's data loss. Kinesis Data Firehose retries delivery errors and doesn't move forward until the records are successfully delivered either to the configured destination or to the backup S3 bucket.

Delivery to OpenSearch Service

Metric	Description
`DeliveryToElasticsearch.Bytes`	The number of bytes indexed to Amazon ES over the specified time period. Units: Bytes
`DeliveryToElasticsearch.DataFreshness`	The age (from getting into Kinesis Data Firehose to now) of the oldest record in Kinesis Data Firehose. Any record older than this age has been delivered to Amazon ES.

Metric	Description
	Units: Seconds
DeliveryToElasticsearch.Records	The number of records indexed to Amazon ES over the specified time period. Units: Count
DeliveryToElasticsearch.Success	The sum of the successfully indexed records over the sum of records that were attempted.
DeliveryToS3.Bytes	The number of bytes delivered to Amazon S3 over the specified time period. Kinesis Data Firehose emits this metric only when you enable backup for all documents. Units: Count
DeliveryToS3.DataFreshness	The age (from getting into Kinesis Data Firehose to now) of the oldest record in Kinesis Data Firehose. Any record older than this age has been delivered to the S3 bucket. Kinesis Data Firehose emits this metric only when you enable backup for all documents. Units: Seconds
DeliveryToS3.Records	The number of records delivered to Amazon S3 over the specified time period. Kinesis Data Firehose emits this metric only when you enable backup for all documents. Units: Count
DeliveryToS3.Success	The sum of successful Amazon S3 put commands over the sum of all Amazon S3 put commands. Kinesis Data Firehose always emits this metric regardless of whether backup is enabled for failed documents only or for all documents.

Delivery to Amazon Redshift

Metric	Description
DeliveryToRedshift.Bytes	The number of bytes copied to Amazon Redshift over the specified time period. Units: Count
DeliveryToRedshift.Records	The number of records copied to Amazon Redshift over the specified time period. Units: Count
DeliveryToRedshift.Success	The sum of successful Amazon Redshift COPY commands over the sum of all Amazon Redshift COPY commands.
DeliveryToS3.Bytes	The number of bytes delivered to Amazon S3 over the specified time period. Units: Bytes

Metric	Description
DeliveryToS3.DataFreshness	The age (from getting into Kinesis Data Firehose to now) of the oldest record in Kinesis Data Firehose. Any record older than this age has been delivered to the S3 bucket. Units: Seconds
DeliveryToS3.Records	The number of records delivered to Amazon S3 over the specified time period. Units: Count
DeliveryToS3.Success	The sum of successful Amazon S3 put commands over the sum of all Amazon S3 put commands.
BackupToS3.Bytes	The number of bytes delivered to Amazon S3 for backup over the specified time period. Kinesis Data Firehose emits this metric when backup to Amazon S3 is enabled. Units: Count
BackupToS3.DataFreshness	Age (from getting into Kinesis Data Firehose to now) of the oldest record in Kinesis Data Firehose. Any record older than this age has been delivered to the Amazon S3 bucket for backup. Kinesis Data Firehose emits this metric when backup to Amazon S3 is enabled. Units: Seconds
BackupToS3.Records	The number of records delivered to Amazon S3 for backup over the specified time period. Kinesis Data Firehose emits this metric when backup to Amazon S3 is enabled. Units: Count
BackupToS3.Success	Sum of successful Amazon S3 put commands for backup over sum of all Amazon S3 backup put commands. Kinesis Data Firehose emits this metric when backup to Amazon S3 is enabled.

Delivery to Amazon S3

The metrics in the following table are related to delivery to Amazon S3 when it is the main destination of the delivery stream.

Metric	Description
DeliveryToS3.Bytes	The number of bytes delivered to Amazon S3 over the specified time period. Units: Bytes
DeliveryToS3.DataFreshness	The age (from getting into Kinesis Data Firehose to now) of the oldest record in Kinesis Data Firehose. Any record older than this age has been delivered to the S3 bucket. Units: Seconds

Metric	Description
DeliveryToS3.Records	The number of records delivered to Amazon S3 over the specified time period. Units: Count
DeliveryToS3.Success	The sum of successful Amazon S3 put commands over the sum of all Amazon S3 put commands.
BackupToS3.Bytes	The number of bytes delivered to Amazon S3 for backup over the specified time period. Kinesis Data Firehose emits this metric when backup is enabled (which is only possible when data transformation is also enabled). Units: Count
BackupToS3.DataFreshness	Age (from getting into Kinesis Data Firehose to now) of the oldest record in Kinesis Data Firehose. Any record older than this age has been delivered to the Amazon S3 bucket for backup. Kinesis Data Firehose emits this metric when backup is enabled (which is only possible when data transformation is also enabled). Units: Seconds
BackupToS3.Records	The number of records delivered to Amazon S3 for backup over the specified time period. Kinesis Data Firehose emits this metric when backup is enabled (which is only possible when data transformation is also enabled). Units: Count
BackupToS3.Success	Sum of successful Amazon S3 put commands for backup over sum of all Amazon S3 backup put commands. Kinesis Data Firehose emits this metric when backup is enabled (which is only possible when data transformation is also enabled).

Delivery to Splunk

Metric	Description
DeliveryToSplunk.Bytes	The number of bytes delivered to Splunk over the specified time period. Units: Bytes
DeliveryToSplunk.DataAckLatency	The approximate duration it takes to receive an acknowledgement from Splunk after Kinesis Data Firehose sends it data. The increasing or decreasing trend for this metric is more useful than the absolute approximate value. Increasing trends can indicate slower indexing and acknowledgement rates from Splunk indexers. Units: Seconds

Metric	Description
DeliveryToSplunk.DataFreshness	Age (from getting into Kinesis Data Firehose to now) of the oldest record in Kinesis Data Firehose. Any record older than this age has been delivered to Splunk. Units: Seconds
DeliveryToSplunk.Records	The number of records delivered to Splunk over the specified time period. Units: Count
DeliveryToSplunk.Success	The sum of the successfully indexed records over the sum of records that were attempted.
DeliveryToS3.Success	The sum of successful Amazon S3 put commands over the sum of all Amazon S3 put commands. This metric is emitted when backup to Amazon S3 is enabled.
BackupToS3.Bytes	The number of bytes delivered to Amazon S3 for backup over the specified time period. Kinesis Data Firehose emits this metric when the delivery stream is configured to back up all documents. Units: Count
BackupToS3.DataFreshness	Age (from getting into Kinesis Data Firehose to now) of the oldest record in Kinesis Data Firehose. Any record older than this age has been delivered to the Amazon S3 bucket for backup. Kinesis Data Firehose emits this metric when the delivery stream is configured to back up all documents. Units: Seconds
BackupToS3.Records	The number of records delivered to Amazon S3 for backup over the specified time period. Kinesis Data Firehose emits this metric when the delivery stream is configured to back up all documents. Units: Count
BackupToS3.Success	Sum of successful Amazon S3 put commands for backup over sum of all Amazon S3 backup put commands. Kinesis Data Firehose emits this metric when the delivery stream is configured to back up all documents.

Delivery to HTTP Endpoints

Metric	Description
DeliveryToHttpEndpoint.Bytes	The number of bytes delivered successfully to the HTTP endpoint. Units: Bytes
DeliveryToHttpEndpoint.Records	The number of records delivered successfully to the HTTP endpoint.

Metric	Description
	Units: Counts
`DeliveryToHttpEndpoint.DataFreshness`	Age of the oldest record in Kinesis Data Firehose. Units: Seconds
`DeliveryToHttpEndpoint.Success`	The sum of all successful data delivery requests to the HTTP endpoint Units: Count
`DeliveryToHttpEndpoint.ProcessedBytes`	The number of attempted processed bytes, including retries.
`DeliveryToHttpEndpoint.ProcessedRecords`	The number of attempted records including retries.

Data Ingestion Metrics

Data Ingestion Through Kinesis Data Streams

Metric	Description
`DataReadFromKinesisStream.Bytes`	When the data source is a Kinesis data stream, this metric indicates the number of bytes read from that data stream. This number includes rereads due to failovers. Units: Bytes
`DataReadFromKinesisStream.Records`	When the data source is a Kinesis data stream, this metric indicates the number of records read from that data stream. This number includes rereads due to failovers. Units: Count
`ThrottledDescribeStream`	The total number of times the `DescribeStream` operation is throttled when the data source is a Kinesis data stream. Units: Count
`ThrottledGetRecords`	The total number of times the `GetRecords` operation is throttled when the data source is a Kinesis data stream. Units: Count
`ThrottledGetShardIterator`	The total number of times the `GetShardIterator` operation is throttled when the data source is a Kinesis data stream. Units: Count

Data Ingestion Through Direct PUT

Metric	Description
BackupToS3.Bytes	The number of bytes delivered to Amazon S3 for backup over the specified time period. Kinesis Data Firehose emits this metric when data transformation is enabled for Amazon S3 or Amazon Redshift destinations. Units: Bytes
BackupToS3.DataFreshness	Age (from getting into Kinesis Data Firehose to now) of the oldest record in Kinesis Data Firehose. Any record older than this age has been delivered to the Amazon S3 bucket for backup. Kinesis Data Firehose emits this metric when data transformation is enabled for Amazon S3 or Amazon Redshift destinations. Units: Seconds
BackupToS3.Records	The number of records delivered to Amazon S3 for backup over the specified time period. Kinesis Data Firehose emits this metric when data transformation is enabled for Amazon S3 or Amazon Redshift destinations. Units: Count
BackupToS3.Success	Sum of successful Amazon S3 put commands for backup over sum of all Amazon S3 backup put commands. Kinesis Data Firehose emits this metric when data transformation is enabled for Amazon S3 or Amazon Redshift destinations.
BytesPerSecondLimit	The current maximum number of bytes per second that a delivery stream can ingest before throttling. To request an increase to this limit, go to the AWS Support Center and choose **Create case**, then choose **Service limit increase**.
DataReadFromKinesisStream.Bytes	When the data source is a Kinesis data stream, this metric indicates the number of bytes read from that data stream. This number includes rereads due to failovers. Units: Bytes
DataReadFromKinesisStream.Records	When the data source is a Kinesis data stream, this metric indicates the number of records read from that data stream. This number includes rereads due to failovers. Units: Count
DeliveryToElasticsearch.Bytes	The number of bytes indexed to Amazon ES over the specified time period. Units: Bytes
DeliveryToElasticsearch.DataFreshness	The age (from getting into Kinesis Data Firehose to now) of the oldest record in Kinesis Data Firehose. Any record older than this age has been delivered to Amazon ES. Units: Seconds

Metric	Description
DeliveryToElasticsearch.Records	The number of records indexed to Amazon ES over the specified time period. Units: Count
DeliveryToElasticsearch.Success	The sum of the successfully indexed records over the sum of records that were attempted.
DeliveryToRedshift.Bytes	The number of bytes copied to Amazon Redshift over the specified time period. Units: Bytes
DeliveryToRedshift.Records	The number of records copied to Amazon Redshift over the specified time period. Units: Count
DeliveryToRedshift.Success	The sum of successful Amazon Redshift COPY commands over the sum of all Amazon Redshift COPY commands.
DeliveryToS3.Bytes	The number of bytes delivered to Amazon S3 over the specified time period. Units: Bytes
DeliveryToS3.DataFreshness	The age (from getting into Kinesis Data Firehose to now) of the oldest record in Kinesis Data Firehose. Any record older than this age has been delivered to the S3 bucket. Units: Seconds
DeliveryToS3.Records	The number of records delivered to Amazon S3 over the specified time period. Units: Count
DeliveryToS3.Success	The sum of successful Amazon S3 put commands over the sum of all Amazon S3 put commands.
DeliveryToSplunk.Bytes	The number of bytes delivered to Splunk over the specified time period. Units: Bytes
DeliveryToSplunk.DataAckLatency	The approximate duration it takes to receive an acknowledgement from Splunk after Kinesis Data Firehose sends it data. The increasing or decreasing trend for this metric is more useful than the absolute approximate value. Increasing trends can indicate slower indexing and acknowledgement rates from Splunk indexers. Units: Seconds

Metric	Description
DeliveryToSplunk.DataFreshness	Age (from getting into Kinesis Data Firehose to now) of the oldest record in Kinesis Data Firehose. Any record older than this age has been delivered to Splunk. Units: Seconds
DeliveryToSplunk.Records	The number of records delivered to Splunk over the specified time period. Units: Count
DeliveryToSplunk.Success	The sum of the successfully indexed records over the sum of records that were attempted.
IncomingBytes	The number of bytes ingested successfully into the delivery stream over the specified time period after throttling. Units: Bytes
IncomingPutRequests	The number of successful PutRecord and PutRecordBatch requests over the specified period of time after throttling. Units: Count
IncomingRecords	The number of records ingested successfully into the delivery stream over the specified time period after throttling. Units: Count
KinesisMillisBehindLatest	When the data source is a Kinesis data stream, this metric indicates the number of milliseconds that the last read record is behind the newest record in the Kinesis data stream. Units: Millisecond
RecordsPerSecondLimit	The current maximum number of records per second that a delivery stream can ingest before throttling. Units: Count
ThrottledRecords	The number of records that were throttled because data ingestion exceeded one of the delivery stream limits. Units: Count

API-Level CloudWatch Metrics

The AWS/Firehose namespace includes the following API-level metrics.

Metric	Description
DescribeDeliveryStream.Latency	The time taken per DescribeDeliveryStream operation, measured over the specified time period.

Metric	Description
	Units: Milliseconds
DescribeDeliveryStream.Requests	The total number of DescribeDeliveryStream requests. Units: Count
ListDeliveryStreams.Latency	The time taken per ListDeliveryStream operation, measured over the specified time period. Units: Milliseconds
ListDeliveryStreams.Requests	The total number of ListFirehose requests. Units: Count
PutRecord.Bytes	The number of bytes put to the Kinesis Data Firehose delivery stream using PutRecord over the specified time period. Units: Bytes
PutRecord.Latency	The time taken per PutRecord operation, measured over the specified time period. Units: Milliseconds
PutRecord.Requests	The total number of PutRecord requests, which is equal to total number of records from PutRecord operations. Units: Count
PutRecordBatch.Bytes	The number of bytes put to the Kinesis Data Firehose delivery stream using PutRecordBatch over the specified time period. Units: Bytes
PutRecordBatch.Latency	The time taken per PutRecordBatch operation, measured over the specified time period. Units: Milliseconds
PutRecordBatch.Records	The total number of records from PutRecordBatch operations. Units: Count
PutRecordBatch.Requests	The total number of PutRecordBatch requests. Units: Count
PutRequestsPerSecondLimit	The maximum number of put requests per second that a delivery stream can handle before throttling. This number includes PutRecord and PutRecordBatch requests. Units: Count

Metric	Description
ThrottledDescribeStream	The total number of times the DescribeStream operation is throttled when the data source is a Kinesis data stream. Units: Count
ThrottledGetRecords	The total number of times the GetRecords operation is throttled when the data source is a Kinesis data stream. Units: Count
ThrottledGetShardIterator	The total number of times the GetShardIterator operation is throttled when the data source is a Kinesis data stream. Units: Count
UpdateDeliveryStream.Latency	The time taken per UpdateDeliveryStream operation, measured over the specified time period. Units: Milliseconds
UpdateDeliveryStream.Requests	The total number of UpdateDeliveryStream requests. Units: Count

Data Transformation CloudWatch Metrics

If data transformation with Lambda is enabled, the AWS/Firehose namespace includes the following metrics.

Metric	Description
ExecuteProcessing.Duration	The time it takes for each Lambda function invocation performed by Kinesis Data Firehose. Units: Milliseconds
ExecuteProcessing.Success	The sum of the successful Lambda function invocations over the sum of the total Lambda function invocations.
SucceedProcessing.Records	The number of successfully processed records over the specified time period. Units: Count
SucceedProcessing.Bytes	The number of successfully processed bytes over the specified time period. Units: Bytes

Format Conversion CloudWatch Metrics

If format conversion is enabled, the AWS/Firehose namespace includes the following metrics.

Metric	Description
`SucceedConversion.Records`	The number of successfully converted records. Units: Count
`SucceedConversion.Bytes`	The size of the successfully converted records. Units: Bytes
`FailedConversion.Records`	The number of records that could not be converted. Units: Count
`FailedConversion.Bytes`	The size of the records that could not be converted. Units: Bytes

Server-Side Encryption (SSE) CloudWatch Metrics

The `AWS/Firehose` namespace includes the following metrics that are related to SSE.

Metric	Description
`KMSKeyAccessDenied`	The number of times the service encounters a `KMSAccessDeniedException` for the delivery stream. Units: Count
`KMSKeyDisabled`	The number of times the service encounters a `KMSDisabledException` for the delivery stream. Units: Count
`KMSKeyInvalidState`	The number of times the service encounters a `KMSInvalidStateException` for the delivery stream. Units: Count
`KMSKeyNotFound`	The number of times the service encounters a `KMSNotFoundException` for the delivery stream. Units: Count

Dimensions for Kinesis Data Firehose

To filter metrics by delivery stream, use the `DeliveryStreamName` dimension.

Kinesis Data Firehose Usage Metrics

You can use CloudWatch usage metrics to provide visibility into your account's usage of resources. Use these metrics to visualize your current service usage on CloudWatch graphs and dashboards.

Service quota usage metrics are in the AWS/Usage namespace and are collected every minute.

Currently, the only metric name in this namespace that CloudWatch publishes is `ResourceCount`. This metric is published with the dimensions `Service`, `Class`, `Type`, and `Resource`.

Metric	Description
`ResourceCount`	The number of the specified resources running in your account. The resources are defined by the dimensions associated with the metric. The most useful statistic for this metric is MAXIMUM, which represents the maximum number of resources used during the 1-minute period.

The following dimensions are used to refine the usage metrics that are published by Kinesis Data Firehose.

Dimension	Description
`Service`	The name of the AWS service containing the resource. For Kinesis Data Firehose usage metrics, the value for this dimension is `Firehose`.
`Class`	The class of resource being tracked. Kinesis Data Firehose API usage metrics use this dimension with a value of `None`.
`Type`	The type of resource being tracked. Currently, when the Service dimension is `Firehose`, the only valid value for Type is `Resource`.
`Resource`	The name of the AWS resource. Currently, when the Service dimension is `Firehose`, the only valid value for Resource is `DeliveryStreams`.

Accessing CloudWatch Metrics for Kinesis Data Firehose

You can monitor metrics for Kinesis Data Firehose using the CloudWatch console, command line, or CloudWatch API. The following procedures show you how to access metrics using these different methods.

To access metrics using the CloudWatch console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. On the navigation bar, choose a region.
3. In the navigation pane, choose **Metrics**.
4. Choose the **Firehose** namespace.
5. Choose **Delivery Stream Metrics** or **Firehose Metrics**.
6. Select a metric to add to the graph.

To access metrics using the AWS CLI

Use the list-metrics and get-metric-statistics commands.

```
aws cloudwatch list-metrics --namespace "AWS/Firehose"
```

```
aws cloudwatch get-metric-statistics --namespace "AWS/Firehose" \
--metric-name DescribeDeliveryStream.Latency --statistics Average --period 3600 \
--start-time 2017-06-01T00:00:00Z --end-time 2017-06-30T00:00:00Z
```

Best Practices with CloudWatch Alarms

Add CloudWatch alarms for when the following metrics exceed the buffering limit (a maximum of 15 minutes):

- `DeliveryToS3.DataFreshness`
- `DeliveryToSplunk.DataFreshness`
- `DeliveryToElasticsearch.DataFreshness`

Also, create alarms based on the following metric math expressions.

- `IncomingBytes (Sum per 5 Minutes) / 300` approaches a percentage of `BytesPerSecondLimit`.
- `IncomingRecords (Sum per 5 Minutes) / 300` approaches a percentage of `RecordsPerSecondLimit`.
- `IncomingPutRequests (Sum per 5 Minutes) / 300` approaches a percentage of `PutRequestsPerSecondLimit`.

Another metric for which we recommend an alarm is `ThrottledRecords`.

For information about troubleshooting when alarms go to the `ALARM` state, see *Troubleshooting* (p. 123).

Monitoring Kinesis Data Firehose Using CloudWatch Logs

Kinesis Data Firehose integrates with Amazon CloudWatch Logs so that you can view the specific error logs when the Lambda invocation for data transformation or data delivery fails. You can enable Kinesis Data Firehose error logging when you create your delivery stream.

If you enable Kinesis Data Firehose error logging in the Kinesis Data Firehose console, a log group and corresponding log streams are created for the delivery stream on your behalf. The format of the log group name is `/aws/kinesisfirehose/`*`delivery-stream-name`*, where *`delivery-stream-name`* is the name of the corresponding delivery stream. The log stream name is **S3Delivery**, **RedshiftDelivery**, or **ElasticsearchDelivery**, depending on the delivery destination. Lambda invocation errors for data transformation are also logged to the log stream used for data delivery errors.

For example, if you create a delivery stream "MyStream" with Amazon Redshift as the destination and enable Kinesis Data Firehose error logging, the following are created on your behalf: a log group named `aws/kinesisfirehose/MyStream` and two log streams named **S3Delivery** and **RedshiftDelivery**. In this example, the **S3Delivery** log stream is used for logging errors related to delivery failure to the intermediate S3 bucket. The **RedshiftDelivery** log stream is used for logging errors related to Lambda invocation failure and delivery failure to your Amazon Redshift cluster.

You can enable Kinesis Data Firehose error logging through the AWS CLI, the API, or AWS CloudFormation using the `CloudWatchLoggingOptions` configuration. To do so, create a log group and a log stream in advance. We recommend reserving that log group and log stream for

Kinesis Data Firehose error logging exclusively. Also ensure that the associated IAM policy has `"logs:putLogEvents"` permission. For more information, see Controlling Access with Amazon Kinesis Data Firehose (p. 39).

Note that Kinesis Data Firehose does not guarantee that all delivery error logs are sent to CloudWatch Logs. In circumstances where delivery failure rate is high, Kinesis Data Firehose samples delivery error logs before sending them to CloudWatch Logs.

There is a nominal charge for error logs sent to CloudWatch Logs. For more information, see Amazon CloudWatch Pricing.

Contents

Data Delivery Errors

The following is a list of data delivery error codes and messages for each Kinesis Data Firehose destination. Each error message also describes the proper action to take to fix the issue.

Errors

Amazon S3 Data Delivery Errors

Kinesis Data Firehose can send the following Amazon S3-related errors to CloudWatch Logs.

Error Code	Error Message and Information
`S3.KMS.NotFoundException`	"The provided AWS KMS key was not found. If you are using what you believe to be a valid AWS KMS key with the correct role, check if there is a problem with the account to which the AWS KMS key is attached."
`S3.KMS.RequestLimitExceeded`	"The KMS request per second limit was exceeded while attempting to encrypt S3 objects. Increase the request per second limit." For more information, see Limits in the *AWS Key Management Service Developer Guide*.
`S3.AccessDenied`	"Access was denied. Ensure that the trust policy for the provided IAM role allows Kinesis Data Firehose to assume the role, and the access policy allows access to the S3 bucket."
`S3.AccountProblem`	"There is a problem with your AWS account that prevents the operation from completing successfully. Contact AWS Support."
`S3.AllAccessDisabled`	"Access to the account provided has been disabled. Contact AWS Support."
`S3.InvalidPayer`	"Access to the account provided has been disabled. Contact AWS Support."

Error Code	Error Message and Information
`S3.NotSignedUp`	"The account is not signed up for Amazon S3. Sign the account up or use a different account."
`S3.NoSuchBucket`	"The specified bucket does not exist. Create the bucket or use a different bucket that does exist."
`S3.MethodNotAllowed`	"The specified method is not allowed against this resource. Modify the bucket's policy to allow the correct Amazon S3 operation permissions."
`InternalError`	"An internal error occurred while attempting to deliver data. Delivery will be retried; if the error persists, then it will be reported to AWS for resolution."

Amazon Redshift Data Delivery Errors

Kinesis Data Firehose can send the following Amazon Redshift-related errors to CloudWatch Logs.

Error Code	Error Message and Information
`Redshift.TableNotFound`	"The table to which to load data was not found. Ensure that the specified table exists." The destination table in Amazon Redshift to which data should be copied from S3 was not found. Note that Kinesis Data Firehose does not create the Amazon Redshift table if it does not exist.
`Redshift.SyntaxError`	"The COPY command contains a syntax error. Retry the command."
`Redshift.AuthenticationFailed`	"The provided user name and password failed authentication. Provide a valid user name and password."
`Redshift.AccessDenied`	"Access was denied. Ensure that the trust policy for the provided IAM role allows Kinesis Data Firehose to assume the role."
`Redshift.S3BucketAccessDenied`	"The COPY command was unable to access the S3 bucket. Ensure that the access policy for the provided IAM role allows access to the S3 bucket."
`Redshift.DataLoadFailed`	"Loading data into the table failed. Check STL_LOAD_ERRORS system table for details."
`Redshift.ColumnNotFound`	"A column in the COPY command does not exist in the table. Specify a valid column name."
`Redshift.DatabaseNotFound`	"The database specified in the Amazon Redshift destination configuration or JDBC URL was not found. Specify a valid database name."
`Redshift.IncorrectCopyOptions`	"Conflicting or redundant COPY options were provided. Some options are not compatible in certain combinations. Check the COPY command reference for more info." For more information, see the Amazon Redshift COPY command in the *Amazon Redshift Database Developer Guide*.
`Redshift.MissingColumn`	"There is a column defined in the table schema as NOT NULL without a DEFAULT value and not included in the column list. Exclude this column, ensure that the loaded data always provides a value for this column, or add a default value to the Amazon Redshift schema for this table."

Error Code	Error Message and Information
`Redshift.ConnectionFailed`	"The connection to the specified Amazon Redshift cluster failed. Ensure that security settings allow Kinesis Data Firehose connections, that the cluster or database specified in the Amazon Redshift destination configuration or JDBC URL is correct, and that the cluster is available."
`Redshift.ColumnMismatch`	"The number of jsonpaths in the COPY command and the number of columns in the destination table should match. Retry the command."
`Redshift.IncorrectOrMissingRegion`	"Amazon Redshift attempted to use the wrong region endpoint for accessing the S3 bucket. Either specify a correct region value in the COPY command options or ensure that the S3 bucket is in the same region as the Amazon Redshift database."
`Redshift.IncorrectJsonPathsFile`	"The provided jsonpaths file is not in a supported JSON format. Retry the command."
`Redshift.MissingS3File`	"One or more S3 files required by Amazon Redshift have been removed from the S3 bucket. Check the S3 bucket policies to remove any automatic deletion of S3 files."
`Redshift.InsufficientPrivilege`	"The user does not have permissions to load data into the table. Check the Amazon Redshift user permissions for the INSERT privilege."
`Redshift.ReadOnlyCluster`	"The query cannot be executed because the system is in resize mode. Try the query again later."
`Redshift.DiskFull`	"Data could not be loaded because the disk is full. Increase the capacity of the Amazon Redshift cluster or delete unused data to free disk space."
`InternalError`	"An internal error occurred while attempting to deliver data. Delivery will be retried; if the error persists, then it will be reported to AWS for resolution."

Splunk Data Delivery Errors

Kinesis Data Firehose can send the following Splunk-related errors to CloudWatch Logs.

Error Code	Error Message and Information
`Splunk.ProxyWithoutStickySessions`	"If you have a proxy (ELB or other) between Kinesis Data Firehose and the HEC node, you must enable sticky sessions to support HEC ACKs."
`Splunk.DisabledToken`	"The HEC token is disabled. Enable the token to allow data delivery to Splunk."
`Splunk.InvalidToken`	"The HEC token is invalid. Update Kinesis Data Firehose with a valid HEC token."
`Splunk.InvalidDataFormat`	"The data is not formatted correctly. To see how to properly format data for Raw or Event HEC endpoints, see Splunk Event Data."
`Splunk.InvalidIndex`	"The HEC token or input is configured with an invalid index. Check your index configuration and try again."
`Splunk.ServerError`	"Data delivery to Splunk failed due to a server error from the HEC node. Kinesis Data Firehose will retry sending the data if the retry duration in your

Error Code	Error Message and Information
	Kinesis Data Firehose is greater than 0. If all the retries fail, Kinesis Data Firehose backs up the data to Amazon S3."
`Splunk.DisabledAck`	"Indexer acknowledgement is disabled for the HEC token. Enable indexer acknowledgement and try again. For more info, see Enable indexer acknowledgement."
`Splunk.AckTimeout`	"Did not receive an acknowledgement from HEC before the HEC acknowledgement timeout expired. Despite the acknowledgement timeout, it's possible the data was indexed successfully in Splunk. Kinesis Data Firehose backs up in Amazon S3 data for which the acknowledgement timeout expired."
`Splunk.MaxRetriesFailed`	"Failed to deliver data to Splunk or to receive acknowledgment. Check your HEC health and try again."
`Splunk.ConnectionTimeout`	"The connection to Splunk timed out. This might be a transient error and the request will be retried. Kinesis Data Firehose backs up the data to Amazon S3 if all retries fail."
`Splunk.InvalidEndpoint`	"Could not connect to the HEC endpoint. Make sure that the HEC endpoint URL is valid and reachable from Kinesis Data Firehose."
`Splunk.ConnectionClosed`	"Unable to send data to Splunk due to a connection failure. This might be a transient error. Increasing the retry duration in your Kinesis Data Firehose configuration might guard against such transient failures."
`Splunk.SSLUnverified`	"Could not connect to the HEC endpoint. The host does not match the certificate provided by the peer. Make sure that the certificate and the host are valid."
`Splunk.SSLHandshake`	"Could not connect to the HEC endpoint. Make sure that the certificate and the host are valid."

HTTPS Endpoint Data Delivery Errors

Kinesis Data Firehose can send the following HTTP Endpoint-related errors to CloudWatch Logs. If none of these errors are a match to the problem that you're experiencing, the default error is the following: "An internal error occurred while attempting to deliver data. Delivery will be retried; if the error persists, then it will be reported to AWS for resolution."

Error Code	Error Message and Information
`HttpEndpoint.RequestTimeout`	"The delivery timed out before a response was received and will be retried. If this error persists, contact the AWS Firehose service team.
`HttpEndpoint.ResponseTooLarge`	"The response received from the endpoint is too large. Contact the owner of the endpoint to resolve this issue."
`HttpEndpoint.InvalidResponseFromDestination`	"The response received from the specified endpoint is invalid. Contact the owner of the endpoint to resolve the issue."
`HttpEndpoint.DestinationException`	"The following response was received from the endpoint destination."
`HttpEndpoint.ConnectionFailed`	"Unable to connect to the destination endpoint. Contact the owner of the endpoint to resolve this issue."

Error Code	Error Message and Information
`HttpEndpoint.ConnectionError`	"Unable to maintain connection with the endpoint. Contact the owner of the endpoint to resolve this issue."
`HttpEndpoint.ConnectionLost`	"Trouble maintaining connection with the endpoint. Please reach out to the owner of the endpoint."

Amazon OpenSearch Service Data Delivery Errors

For the Amazon ES destination, Kinesis Data Firehose sends errors to CloudWatch Logs as they are returned by Elasticsearch.

Lambda Invocation Errors

Kinesis Data Firehose can send the following Lambda invocation errors to CloudWatch Logs.

Error Code	Error Message and Information
`Lambda.AssumeRoleAccessDenied`	"Access was denied. Ensure that the trust policy for the provided IAM role allows Kinesis Data Firehose to assume the role."
`Lambda.InvokeAccessDenied`	"Access was denied. Ensure that the access policy allows access to the Lambda function."
`Lambda.JsonProcessingException`	"There was an error parsing returned records from the Lambda function. Ensure that the returned records follow the status model required by Kinesis Data Firehose." For more information, see Data Transformation and Status Model (p. 58).
`Lambda.InvokeLimitExceeded`	"The Lambda concurrent execution limit is exceeded. Increase the concurrent execution limit." For more information, see AWS Lambda Limits in the *AWS Lambda Developer Guide*.
`Lambda.DuplicatedRecordId`	"Multiple records were returned with the same record ID. Ensure that the Lambda function returns unique record IDs for each record." For more information, see Data Transformation and Status Model (p. 58).
`Lambda.MissingRecordId`	"One or more record IDs were not returned. Ensure that the Lambda function returns all received record IDs." For more information, see Data Transformation and Status Model (p. 58).
`Lambda.ResourceNotFound`	"The specified Lambda function does not exist. Use a different function that does exist."
`Lambda.InvalidSubnetIDException`	"The specified subnet ID in the Lambda function VPC configuration is invalid. Ensure that the subnet ID is valid."
`Lambda.InvalidSecurityGroupIDException`	"The specified security group ID in the Lambda function VPC configuration is invalid. Ensure that the security group ID is valid."

Error Code	Error Message and Information
`Lambda.SubnetIPAddressLimitReachedException`	"AWS Lambda was not able to set up the VPC access for the Lambda function because one or more configured subnets have no available IP addresses. Increase the IP address limit." For more information, see Amazon VPC Limits - VPC and Subnets in the *Amazon VPC User Guide*.
`Lambda.ENILimitReachedException`	"AWS Lambda was not able to create an Elastic Network Interface (ENI) in the VPC, specified as part of the Lambda function configuration, because the limit for network interfaces has been reached. Increase the network interface limit." For more information, see Amazon VPC Limits - Network Interfaces in the *Amazon VPC User Guide*.

Accessing CloudWatch Logs for Kinesis Data Firehose

You can view the error logs related to Kinesis Data Firehose data delivery failure using the Kinesis Data Firehose console or the CloudWatch console. The following procedures show you how to access error logs using these two methods.

To access error logs using the Kinesis Data Firehose console

1. Sign in to the AWS Management Console and open the Kinesis console at https://console.aws.amazon.com/kinesis.
2. Choose **Data Firehose** in the navigation pane.
3. On the navigation bar, choose an AWS Region.
4. Choose a delivery stream name to go to the delivery stream details page.
5. Choose **Error Log** to view a list of error logs related to data delivery failure.

To access error logs using the CloudWatch console

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. On the navigation bar, choose a Region.
3. In the navigation pane, choose **Logs**.
4. Choose a log group and log stream to view a list of error logs related to data delivery failure.

Monitoring Kinesis Agent Health

Kinesis Agent publishes custom CloudWatch metrics with a namespace of **AWSKinesisAgent**. It helps assess whether the agent is healthy, submitting data into Kinesis Data Firehose as specified, and consuming the appropriate amount of CPU and memory resources on the data producer.

Metrics such as number of records and bytes sent are useful to understand the rate at which the agent is submitting data to the Kinesis Data Firehose delivery stream. When these metrics fall below expected thresholds by some percentage or drop to zero, it could indicate configuration issues, network errors, or agent health issues. Metrics such as on-host CPU and memory consumption and agent error counters indicate data producer resource usage, and provide insights into potential configuration or host errors. Finally, the agent also logs service exceptions to help investigate agent issues.

The agent metrics are reported in the region specified in the agent configuration setting `cloudwatch.endpoint`. For more information, see Agent Configuration Settings (p. 29).

Cloudwatch metrics published from multiple Kinesis Agents are aggregated or combined.

There is a nominal charge for metrics emitted from Kinesis Agent, which are enabled by default. For more information, see Amazon CloudWatch Pricing.

Monitoring with CloudWatch

Kinesis Agent sends the following metrics to CloudWatch.

Metric	Description
`BytesSent`	The number of bytes sent to the Kinesis Data Firehose delivery stream over the specified time period. Units: Bytes
`RecordSendAttempts`	The number of records attempted (either first time, or as a retry) in a call to `PutRecordBatch` over the specified time period. Units: Count
`RecordSendErrors`	The number of records that returned failure status in a call to `PutRecordBatch`, including retries, over the specified time period. Units: Count
`ServiceErrors`	The number of calls to `PutRecordBatch` that resulted in a service error (other than a throttling error) over the specified time period. Units: Count

Logging Kinesis Data Firehose API Calls with AWS CloudTrail

Amazon Kinesis Data Firehose is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Kinesis Data Firehose. CloudTrail captures all API calls for Kinesis Data Firehose as events. The calls captured include calls from the Kinesis Data Firehose console and code calls to the Kinesis Data Firehose API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Kinesis Data Firehose. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to Kinesis Data Firehose, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, including how to configure and enable it, see the AWS CloudTrail User Guide.

Kinesis Data Firehose Information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When supported event activity occurs in Kinesis Data Firehose, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for Kinesis Data Firehose, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

Kinesis Data Firehose supports logging the following actions as events in CloudTrail log files:

- CreateDeliveryStream
- DeleteDeliveryStream
- DescribeDeliveryStream
- ListDeliveryStreams
- ListTagsForDeliveryStream
- TagDeliveryStream
- StartDeliveryStreamEncryption
- StopDeliveryStreamEncryption
- UntagDeliveryStream
- UpdateDestination

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity Element.

Example: Kinesis Data Firehose Log File Entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `CreateDeliveryStream`, `DescribeDeliveryStream`, `ListDeliveryStreams`, `UpdateDestination`, and `DeleteDeliveryStream` actions.

```
{
  "Records":[
        {
            "eventVersion":"1.02",
```

```
            "userIdentity":{
                "type":"IAMUser",
                "principalId":"AKIAIOSFODNN7EXAMPLE",
                "arn":"arn:aws:iam::111122223333:user/CloudTrail_Test_User",
                "accountId":"111122223333",
                "accessKeyId":"AKIAI44QH8DHBEXAMPLE",
                "userName":"CloudTrail_Test_User"
            },
            "eventTime":"2016-02-24T18:08:22Z",
            "eventSource":"firehose.amazonaws.com",
            "eventName":"CreateDeliveryStream",
            "awsRegion":"us-east-1",
            "sourceIPAddress":"127.0.0.1",
            "userAgent":"aws-internal/3",
            "requestParameters":{
                "deliveryStreamName":"TestRedshiftStream",
                "redshiftDestinationConfiguration":{
                "s3Configuration":{
                    "compressionFormat":"GZIP",
                    "prefix":"prefix",
                    "bucketARN":"arn:aws:s3:::firehose-cloudtrail-test-bucket",
                    "roleARN":"arn:aws:iam::111122223333:role/Firehose",
                    "bufferingHints":{
                        "sizeInMBs":3,
                        "intervalInSeconds":900
                    },
                    "encryptionConfiguration":{
                        "kMSEncryptionConfig":{
                            "aWSKMSKeyARN":"arn:aws:kms:us-east-1:key"
                        }
                    }
                },
                "clusterJDBCURL":"jdbc:redshift://example.abc123.us-
west-2.redshift.amazonaws.com:5439/dev",
                "copyCommand":{
                    "copyOptions":"copyOptions",
                    "dataTableName":"dataTable"
                },
                "password":"",
                "username":"",
                "roleARN":"arn:aws:iam::111122223333:role/Firehose"
            }
        },
        "responseElements":{
            "deliveryStreamARN":"arn:aws:firehose:us-east-1:111122223333:deliverystream/
TestRedshiftStream"
        },
        "requestID":"958abf6a-db21-11e5-bb88-91ae9617edf5",
        "eventID":"875d2d68-476c-4ad5-bbc6-d02872cfc884",
        "eventType":"AwsApiCall",
        "recipientAccountId":"111122223333"
    },
    {
        "eventVersion":"1.02",
        "userIdentity":{
            "type":"IAMUser",
            "principalId":"AKIAIOSFODNN7EXAMPLE",
            "arn":"arn:aws:iam::111122223333:user/CloudTrail_Test_User",
            "accountId":"111122223333",
            "accessKeyId":"AKIAI44QH8DHBEXAMPLE",
            "userName":"CloudTrail_Test_User"
        },
        "eventTime":"2016-02-24T18:08:54Z",
        "eventSource":"firehose.amazonaws.com",
        "eventName":"DescribeDeliveryStream",
        "awsRegion":"us-east-1",
```

```
        "sourceIPAddress":"127.0.0.1",
        "userAgent":"aws-internal/3",
        "requestParameters":{
            "deliveryStreamName":"TestRedshiftStream"
        },
        "responseElements":null,
        "requestID":"aa6ea5ed-db21-11e5-bb88-91ae9617edf5",
        "eventID":"d9b285d8-d690-4d5c-b9fe-d1ad5ab03f14",
        "eventType":"AwsApiCall",
        "recipientAccountId":"111122223333"
    },
    {
        "eventVersion":"1.02",
        "userIdentity":{
            "type":"IAMUser",
            "principalId":"AKIAIOSFODNN7EXAMPLE",
            "arn":"arn:aws:iam::111122223333:user/CloudTrail_Test_User",
            "accountId":"111122223333",
            "accessKeyId":"AKIAI44QH8DHBEXAMPLE",
            "userName":"CloudTrail_Test_User"
        },
        "eventTime":"2016-02-24T18:10:00Z",
        "eventSource":"firehose.amazonaws.com",
        "eventName":"ListDeliveryStreams",
        "awsRegion":"us-east-1",
        "sourceIPAddress":"127.0.0.1",
        "userAgent":"aws-internal/3",
        "requestParameters":{
            "limit":10
        },
        "responseElements":null,
        "requestID":"d1bf7f86-db21-11e5-bb88-91ae9617edf5",
        "eventID":"67f63c74-4335-48c0-9004-4ba35ce00128",
        "eventType":"AwsApiCall",
        "recipientAccountId":"111122223333"
    },
    {
        "eventVersion":"1.02",
        "userIdentity":{
            "type":"IAMUser",
            "principalId":"AKIAIOSFODNN7EXAMPLE",
            "arn":"arn:aws:iam::111122223333:user/CloudTrail_Test_User",
            "accountId":"111122223333",
            "accessKeyId":"AKIAI44QH8DHBEXAMPLE",
            "userName":"CloudTrail_Test_User"
        },
        "eventTime":"2016-02-24T18:10:09Z",
        "eventSource":"firehose.amazonaws.com",
        "eventName":"UpdateDestination",
        "awsRegion":"us-east-1",
        "sourceIPAddress":"127.0.0.1",
        "userAgent":"aws-internal/3",
        "requestParameters":{
            "destinationId":"destinationId-000000000001",
            "deliveryStreamName":"TestRedshiftStream",
            "currentDeliveryStreamVersionId":"1",
            "redshiftDestinationUpdate":{
                "roleARN":"arn:aws:iam::111122223333:role/Firehose",
                "clusterJDBCURL":"jdbc:redshift://example.abc123.us-
west-2.redshift.amazonaws.com:5439/dev",
                "password":"",
                "username":"",
                "copyCommand":{
                    "copyOptions":"copyOptions",
                    "dataTableName":"dataTable"
                },
```

```
                "s3Update":{
                    "bucketARN":"arn:aws:s3:::firehose-cloudtrail-test-bucket-update",
                    "roleARN":"arn:aws:iam::111122223333:role/Firehose",
                    "compressionFormat":"GZIP",
                    "bufferingHints":{
                        "sizeInMBs":3,
                        "intervalInSeconds":900
                    },
                    "encryptionConfiguration":{
                        "kMSEncryptionConfig":{
                            "aWSKMSKeyARN":"arn:aws:kms:us-east-1:key"
                        }
                    },
                    "prefix":"arn:aws:s3:::firehose-cloudtrail-test-bucket"
                }
            }
        },
        "responseElements":null,
        "requestID":"d549428d-db21-11e5-bb88-91ae9617edf5",
        "eventID":"1cb21e0b-416a-415d-bbf9-769b152a6585",
        "eventType":"AwsApiCall",
        "recipientAccountId":"111122223333"
    },
    {
        "eventVersion":"1.02",
        "userIdentity":{
            "type":"IAMUser",
            "principalId":"AKIAIOSFODNN7EXAMPLE",
            "arn":"arn:aws:iam::111122223333:user/CloudTrail_Test_User",
            "accountId":"111122223333",
            "accessKeyId":"AKIAI44QH8DHBEXAMPLE",
            "userName":"CloudTrail_Test_User"
        },
        "eventTime":"2016-02-24T18:10:12Z",
        "eventSource":"firehose.amazonaws.com",
        "eventName":"DeleteDeliveryStream",
        "awsRegion":"us-east-1",
        "sourceIPAddress":"127.0.0.1",
        "userAgent":"aws-internal/3",
        "requestParameters":{
            "deliveryStreamName":"TestRedshiftStream"
        },
        "responseElements":null,
        "requestID":"d85968c1-db21-11e5-bb88-91ae9617edf5",
        "eventID":"dd46bb98-b4e9-42ff-a6af-32d57e636ad1",
        "eventType":"AwsApiCall",
        "recipientAccountId":"111122223333"
    }
  ]
}
```

Custom Prefixes for Amazon S3 Objects

If your destination is Amazon S3, Amazon OpenSearch Service, or Splunk, you can configure the Amazon S3 object keys used for delivering data from Kinesis Data Firehose. To do this, you specify expressions that Kinesis Data Firehose evaluates at delivery time. The final object keys have the format `<evaluated prefix><suffix>`, where the suffix has the format `<delivery stream name>-<delivery stream version>-<year>-<month>-<day>-<hour>-<minute>-<second>-<uuid><file extension>`. You can't change the suffix field.

You can use expressions of the following forms in your custom prefix: `!{namespace:value}`, where `namespace` can be one of the following, as explained in the following sections.

- `firehose`
- `timestamp`
- `partitionKeyFromQuery`
- `partitionKeyFromLambda`

If a prefix ends with a slash, it appears as a folder in the Amazon S3 bucket. For more information, see Amazon S3 Object Name Format in the *Amazon Kinesis Data Firehose Developer Guide*.

The `timestamp` namespace

Valid values for this namespace are strings that are valid Java DateTimeFormatter strings. As an example, in the year 2018, the expression `!{timestamp:yyyy}` evaluates to `2018`.

When evaluating timestamps, Kinesis Data Firehose uses the approximate arrival timestamp of the oldest record that's contained in the Amazon S3 object being written.

The timestamp is always in UTC.

If you use the `timestamp` namespace more than once in the same prefix expression, every instance evaluates to the same instant in time.

The `firehose` namespace

There are two values that you can use with this namespace: `error-output-type` and `random-string`. The following table explains how to use them.

The `firehose` namespace values

Conversion	Description	Example input	Example output	Notes
`error-output-type`	Evaluates to one of the following strings, depending on the configuration of your delivery stream, and the	`myPrefix/result=!{firehose:error-output-type}/!{timestamp:yyyy/MM/dd}`	`myPrefix/result=processing-failed/2018/08/03`	The error-output-type value can only be used in the ErrorOutputPrefix field.

Conversion	Description	Example input	Example output	Notes
	reason of failure: {processing-failed, elasticsearch-failed, splunk-failed, format-conversion-failed, http-endpoint-failed}. If you use it more than once in the same expression, every instance evaluates to the same error string..			
`random-string`	Evaluates to a random string of 11 characters. If you use it more than once in the same expression, every instance evaluates to a new random string.	`myPrefix/!{firehose:random-string}/`	`myPrefix/046b6c7f-0b/`	You can use it with both prefix types. You can place it at the beginning of the format string to get a randomized prefix, which is sometimes necessary for attaining extremely high throughput with Amazon S3.

partitionKeyFromLambda and partitionKeyFromQuery namespaces

For dynamic partitioning, you must use the following expression format in your S3 bucket prefix: `!{namespace:value}`, where namespace can be either `partitionKeyFromQuery` or `partitionKeyFromLambda`, or both. If you are using inline parsing to create the partitioning keys for your source data, you must specify an S3 bucket prefix value that consists of expressions specified in the following format: `"partitionKeyFromQuery:keyID"`. If you are using an AWS Lambda function to create partitioning keys for your source data, you must specify an S3 bucket prefix value that consists of expressions specified in the following format: `"partitionKeyFromLambda:keyID"`. For more information, see the "Choose Amazon S3 for Your Destination" in Creating an Amazon Kinesis Data Firehose Delivery Stream.

Semantic rules

The following rules apply to `Prefix` and `ErrorOutputPrefix` expressions.

- For the `timestamp` namespace, any character that isn't in single quotes is evaluated. In other words, any string escaped with single quotes in the value field is taken literally.

- If you specify a prefix that doesn't contain a timestamp namespace expression, Kinesis Data Firehose appends the expression `!{timestamp:yyyy/MM/dd/HH/}`to the value in the `Prefix` field.
- The sequence `!{` can only appear in `!{namespace:value}` expressions.
- `ErrorOutputPrefix` can be null only if `Prefix` contains no expressions. In this case, `Prefix` evaluates to `<specified-prefix>yyyy/MM/DDD/HH/` and `ErrorOutputPrefix` evaluates to `<specified-prefix><error-output-type>YYYY/MM/DDD/HH/`. `DDD` represents the day of the year.
- If you specify an expression for `ErrorOutputPrefix`, you must include at least one instance of `!{firehose:error-output-type}`.
- `Prefix` can't contain `!{firehose:error-output-type}`.
- Neither `Prefix` nor `ErrorOutputPrefix` can be greater than 512 characters after they're evaluated.
- If the destination is Amazon Redshift, `Prefix` must not contain expressions and `ErrorOutputPrefix` must be null.
- When the destination is Amazon OpenSearch Service or Splunk, and no `ErrorOutputPrefix` is specified, Kinesis Data Firehose uses the `Prefix` field for failed records.
- When the destination is Amazon S3, the `Prefix` and `ErrorOutputPrefix` in the Amazon S3 destination configuration are used for successful records and failed records, respectively. If you use the AWS CLI or the API, you can use `ExtendedS3DestinationConfiguration` to specify an Amazon S3 *backup* configuration with its own `Prefix` and `ErrorOutputPrefix`.
- When you use the AWS Management Console and set the destination to Amazon S3, Kinesis Data Firehose uses the `Prefix` and `ErrorOutputPrefix` in the destination configuration for successful records and failed records, respectively. If you specify a prefix but no error prefix, Kinesis Data Firehose automatically sets the error prefix to `!{firehose:error-output-type}/`.
- When you use `ExtendedS3DestinationConfiguration` with the AWS CLI, the API, or AWS CloudFormation, if you specify a `S3BackupConfiguration`, Kinesis Data Firehose doesn't provide a default `ErrorOutputPrefix`.
- You cannot use `partitionKeyFromLambda` and `partitionKeyFromQuery` namespaces when creating ErrorOutputPrefix expressions.

Example prefixes

`Prefix` and `ErrorOutputPrefix` examples

Input	Evaluated prefix (at 10:30 AM UTC on Aug 27, 2018)

`Prefix`: Unspecified
 	
`ErrorOutputPrefix`: `myFirehoseFailures/!{firehose:error-output-type}/`
	`Prefix`: `2018/08/27/10`
	
	`ErrorOutputPrefix`: `myFirehoseFailures/processing-failed/`

`Prefix`: `!{timestamp:yyyy/MM/dd}` `ErrorOutputPrefix`: Unspecified	Invalid input: `ErrorOutputPrefix` can't be null when Prefix contains expressions

`Prefix`: `myFirehose/DeliveredYear=!{timestamp:yyyy}/anyMonth/rand=!{firehose:random-string}`
 	
`ErrorOutputPrefix`: `myFirehoseFailures/!{firehose:error-output-type}/!{timestamp:yyyy}/anyMonth/!{timestamp:dd}`
	`Prefix`: `myFirehose/DeliveredYear=2018/anyMonth/rand=5abf82daaa5`
	
	`ErrorOutputPrefix`: `myFirehoseFailures/processing-failed/2018/anyMonth/10`

Input	Evaluated prefix (at 10:30 AM UTC on Aug 27, 2018)

`Prefix`: `myPrefix/year=!{timestamp:yyyy}/month=!{timestamp:MM}/day=!{timestamp:dd}/hour=!{timestamp:HH}/`
 	
`ErrorOutputPrefix`: `myErrorPrefix/year=!{timestamp:yyyy}/month=!{timestamp:MM}/day=!{timestamp:dd}/hour=!{timestamp:HH}/!{firehose:error-output-type}`
	`Prefix`: `myPrefix/year=2018/month=07/day=06/hour=23/`
	
	`ErrorOutputPrefix`: `myErrorPrefix/year=2018/month=07/day=06/hour=23/processing-failed`

`Prefix`: `myFirehosePrefix`
 	
`ErrorOutputPrefix`: Unspecified
	`Prefix`: `myFirehosePrefix/2018/08/27/`
	
	`ErrorOutputPrefix`: `myFirehosePrefix/processing-failed/2018/08/27/`

Using Amazon Kinesis Data Firehose with AWS PrivateLink

Interface VPC endpoints (AWS PrivateLink) for Kinesis Data Firehose

You can use an interface VPC endpoint to keep traffic between your Amazon VPC and Kinesis Data Firehose from leaving the Amazon network. Interface VPC endpoints don't require an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. Interface VPC endpoints are powered by AWS PrivateLink, an AWS technology that enables private communication between AWS services using an elastic network interface with private IPs in your Amazon VPC. For more information, see Amazon Virtual Private Cloud.

Using interface VPC endpoints (AWS PrivateLink) for Kinesis Data Firehose

To get started, create an interface VPC endpoint in order for your Kinesis Data Firehose traffic from your Amazon VPC resources to start flowing through the interface VPC endpoint. When you create an endpoint, you can attach an endpoint policy to it that controls access to Kinesis Data Firehose. For more about using policies to control access from a VPC endpoint to Kinesis Data Firehose, see Controlling Access to Services with VPC Endpoints.

The following example shows how you can set up an AWS Lambda function in a VPC and create a VPC endpoint to allow the function to communicate securely with the Kinesis Data Firehose service. In this example, you use a policy that allows the Lambda function to list the delivery streams in the current Region but not to describe any delivery stream.

Create a VPC endpoint

1. Sign in to the AWS Management Console and open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the VPC Dashboard choose **Endpoints**.
3. Choose **Create Endpoint**.
4. In the list of service names, choose `com.amazonaws.`*`your_region`*`.kinesis-firehose`.
5. Choose the VPC and one or more subnets in which to create the endpoint.
6. Choose one or more security groups to associate with the endpoint.
7. For **Policy**, choose **Custom** and paste the following policy:

```
{
    "Statement": [
        {
            "Sid": "Allow-only-specific-PrivateAPIs",
            "Principal": "*",
            "Action": [
                "firehose:ListDeliveryStreams"
```

```
            ],
            "Effect": "Allow",
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "Allow-only-specific-PrivateAPIs",
            "Principal": "*",
            "Action": [
                "firehose:DescribeDeliveryStream"
            ],
            "Effect": "Deny",
            "Resource": [
                "*"
            ]
        }
    ]
}
```

8. Choose **Create endpoint**.

Create an IAM role to use with the Lambda function

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. In the left pane, chose **Roles** and then choose **Create role**.
3. Under **Select type of trusted entity**, leave the default selection **AWS service**.
4. Under **Choose the service that will use this role**, choose **Lambda**.
5. Choose **Next: Permissions**.
6. In the list of policies, search for and add the two policies named `AWSLambdaVPCAccessExecutionRole` and `AmazonKinesisFirehoseReadOnlyAccess`.

   Important
   This is an example. You might need stricter policies for your production environment.
7. Choose **Next: Tags**. You don't need to add tags for the purpose of this exercise. Choose **Next: Review**.
8. Enter a name for the role, then choose **Create role**.

Create a Lambda function inside the VPC

1. Open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. Choose **Create function**.
3. Choose **Author from scratch**.
4. Enter a name for the function, then set **Runtime** to Python 3.6.
5. Under **Permissions**, expand **Choose or create an execution role**.
6. In the **Execution role** list, choose **Use an existing role**.
7. In the **Existing role** list, choose the role you created above.
8. Choose **Create function**.
9. Under **Function code**, paste the following code.

```
    import json
    import boto3
    import os
    from botocore.exceptions import ClientError

    def lambda_handler(event, context):
```

```
        REGION = os.environ['AWS_REGION']
        client = boto3.client(
            'firehose',
            REGION

        )
        print("Calling list_delivery_streams with ListDeliveryStreams allowed policy.")
        delivery_stream_request = client.list_delivery_streams()
        print("Successfully returned list_delivery_streams request %s." % (
            delivery_stream_request
        ))
        describe_access_denied = False
        try:
            print("Calling describe_delivery_stream with DescribeDeliveryStream denied
policy.")
            delivery_stream_info =
client.describe_delivery_stream(DeliveryStreamName='test-describe-denied')
        except ClientError as e:
            error_code = e.response['Error']['Code']
            print ("Caught %s." % (error_code))
            if error_code == 'AccessDeniedException':
                describe_access_denied = True

        if not describe_access_denied:
            raise
        else:
            print("Access denied test succeeded.")
```

10. Under **Basic settings**, set the timeout to 1 minute.
11. Under **Network**, choose the VPC where you created the endpoint above, then choose the subnets and security group that you associated with the endpoint when you created it.
12. Near the top of the page, choose **Save**.
13. Choose **Test**.
14. Enter an event name, then choose **Create**.
15. Choose **Test** again. This causes the function to run. After the execution result appears, expand **Details** and compare the log output to the function code. Successful results show a list of the delivery streams in the Region, as well as the following output:

    ```
    Calling describe_delivery_stream.

    AccessDeniedException

    Access denied test succeeded.
    ```

Availability

Interface VPC endpoints are currently supported within the following Regions:

- US East (Ohio)
- US East (N. Virginia)
- US West (N. California)
- US West (Oregon)
- Asia Pacific (Mumbai)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)

- Asia Pacific (Tokyo)
- Asia Pacific (Hong Kong)
- Canada (Central)
- China (Beijing)
- China (Ningxia)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (London)
- Europe (Paris)
- South America (São Paulo)
- AWS GovCloud (US-East)
- AWS GovCloud (US-West)

Tagging Your Delivery Streams in Amazon Kinesis Data Firehose

You can assign your own metadata to delivery streams that you create in Amazon Kinesis Data Firehose in the form of *tags*. A tag is a key-value pair that you define for a stream. Using tags is a simple yet powerful way to manage AWS resources and organize data, including billing data.

Topics

Tag Basics

You can use the Amazon Kinesis Data Firehose API to complete the following tasks:

- Add tags to a delivery stream.
- List the tags for your delivery streams.
- Remove tags from a delivery stream.

You can use tags to categorize your Kinesis Data Firehose delivery streams. For example, you can categorize delivery streams by purpose, owner, or environment. Because you define the key and value for each tag, you can create a custom set of categories to meet your specific needs. For example, you might define a set of tags that helps you track delivery streams by owner and associated application.

The following are several examples of tags:

- `Project: Project name`
- `Owner: Name`
- `Purpose: Load testing`
- `Application: Application name`
- `Environment: Production`

Tracking Costs Using Tagging

You can use tags to categorize and track your AWS costs. When you apply tags to your AWS resources, including Kinesis Data Firehose delivery streams, your AWS cost allocation report includes usage and costs aggregated by tags. You can organize your costs across multiple services by applying tags that represent business categories (such as cost centers, application names, or owners). For more information, see Use Cost Allocation Tags for Custom Billing Reports in the *AWS Billing and Cost Management User Guide*.

Tag Restrictions

The following restrictions apply to tags in Kinesis Data Firehose.

Basic restrictions

- The maximum number of tags per resource (stream) is 50.
- Tag keys and values are case-sensitive.
- You can't change or edit tags for a deleted stream.

Tag key restrictions

- Each tag key must be unique. If you add a tag with a key that's already in use, your new tag overwrites the existing key-value pair.
- You can't start a tag key with `aws:` because this prefix is reserved for use by AWS. AWS creates tags that begin with this prefix on your behalf, but you can't edit or delete them.
- Tag keys must be between 1 and 128 Unicode characters in length.
- Tag keys must consist of the following characters: Unicode letters, digits, white space, and the following special characters: `_ . / = + - @`.

Tag value restrictions

- Tag values must be between 0 and 255 Unicode characters in length.
- Tag values can be blank. Otherwise, they must consist of the following characters: Unicode letters, digits, white space, and any of the following special characters: `_ . / = + - @`.

Tagging Delivery Streams Using the Amazon Kinesis Data Firehose API

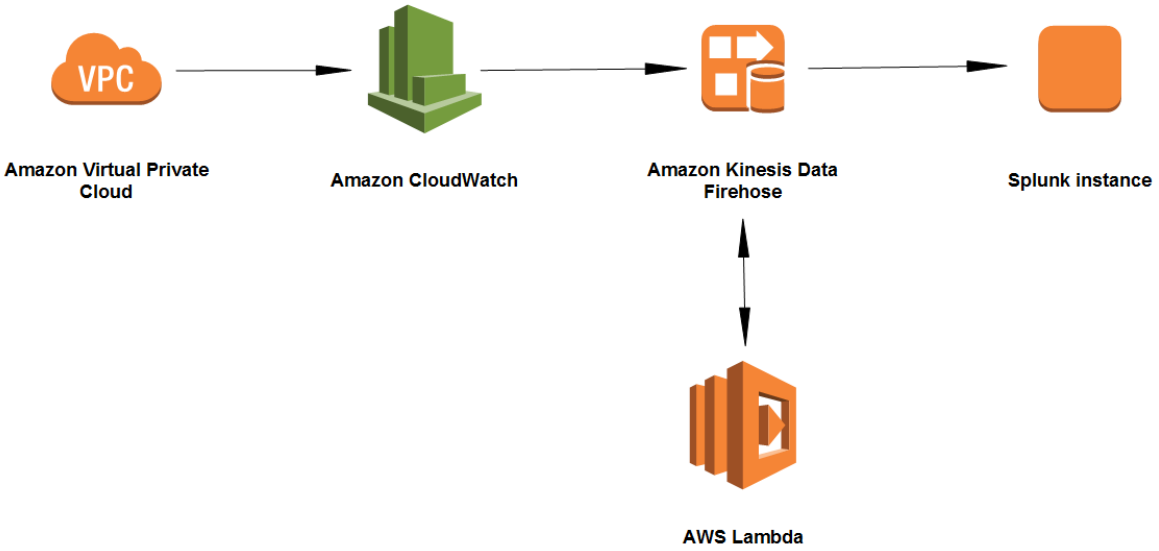
You can specify tags when you invoke CreateDeliveryStream to create a new delivery stream. For existing delivery streams, you can add, list, and remove tags using the following three operations:

- TagDeliveryStream
- ListTagsForDeliveryStream
- UntagDeliveryStream

Tutorial: Sending VPC Flow Logs to Splunk Using Amazon Kinesis Data Firehose

In this tutorial, you learn how to capture information about the IP traffic going to and from network interfaces in an Amazon Virtual Private Cloud (Amazon VPC). You then use Amazon Kinesis Data Firehose to send that information to Splunk. For more information about VPC network traffic, see VPC Flow Logs in the *Amazon VPC User Guide*.

The following diagram shows the flow of data that is demonstrated in this tutorial.



As the diagram shows, first you send the Amazon VPC flow logs to Amazon CloudWatch. Then from CloudWatch, the data goes to a Kinesis Data Firehose delivery stream. Kinesis Data Firehose then invokes an AWS Lambda function to decompress the data, and sends the decompressed log data to Splunk.

Prerequisites

Before you begin, ensure that you have the following prerequisites:

- **AWS account** — If you don't have an AWS account, create one at http://aws.amazon.com. For more information, see Setting Up for Amazon Kinesis Data Firehose (p. 4).
- **AWS CLI** — Parts of this tutorial require that you use the AWS Command Line Interface (AWS CLI). To install the AWS CLI, see Installing the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.
- **HEC token** — In your Splunk deployment, set up an HTTP Event Collector (HEC) token with the source type `aws:cloudwatchlogs:vpcflow`. For more information, see Installation and configuration overview for the Splunk Add-on for Amazon Kinesis Firehose in the Splunk documentation.

Topics

Step 1: Send Log Data from Amazon VPC to Amazon CloudWatch

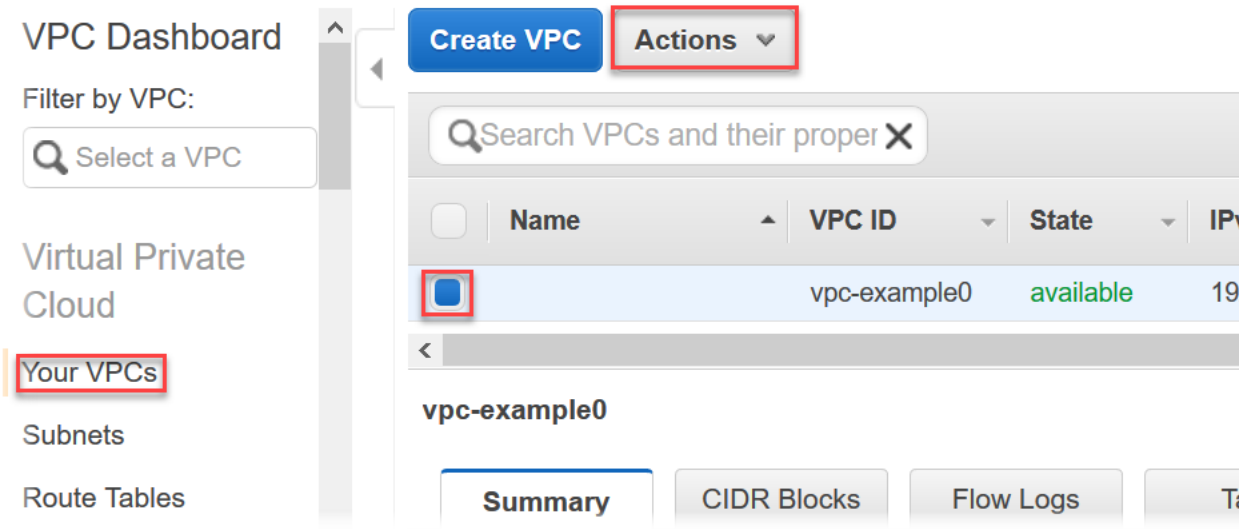
In the first part of this Kinesis Data Firehose tutorial (p. 115), you create an Amazon CloudWatch log group to receive your Amazon VPC flow logs. Then, you create flow logs for your Amazon VPC and send them to the CloudWatch log group that you created.

To create a CloudWatch log group to receive your Amazon VPC flow logs

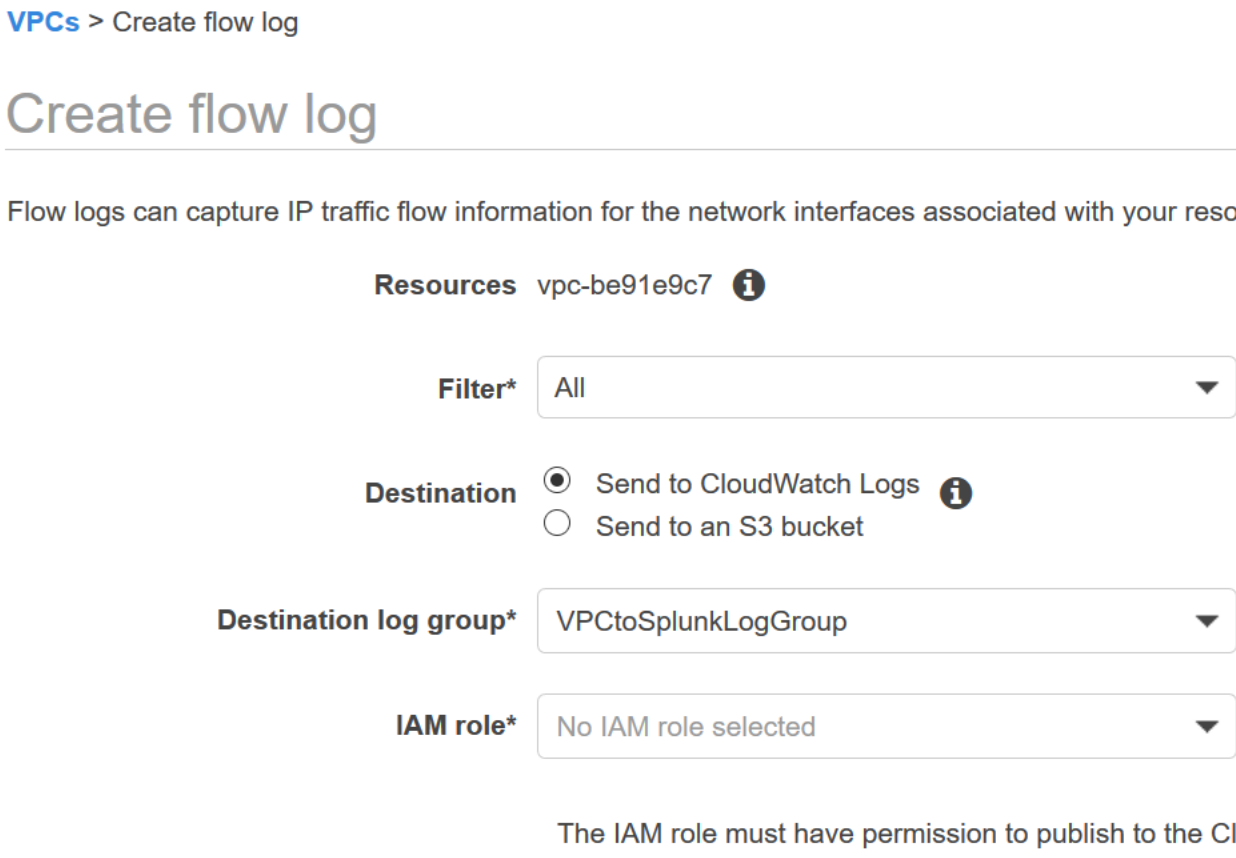
1. Sign in to the AWS Management Console and open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose **Log groups**.
3. Choose **Actions**, and then choose **Create log group**.
4. Enter the name **`VPCtoSplunkLogGroup`**, and choose **Create log group**.

To create a VPC flow log

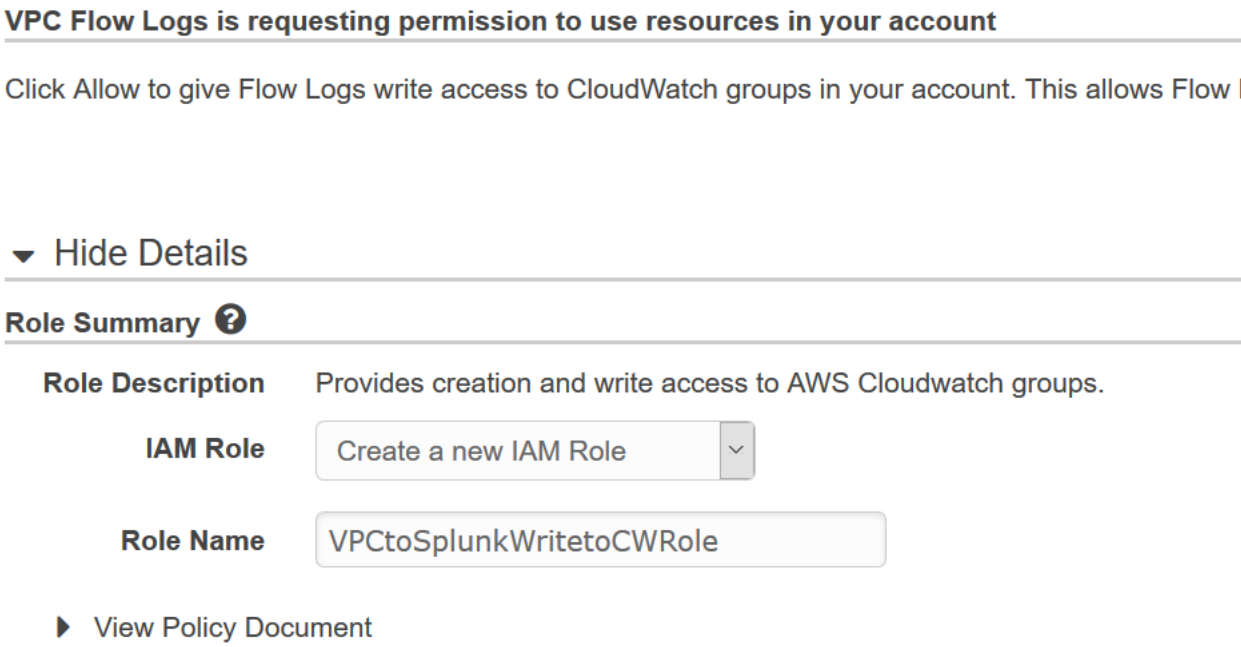
1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. In the navigation pane, choose **Your VPCs**. Then choose your VPC from the list by selecting the check box next to it.



3. Choose **Actions**, and then choose **Create flow log**.
4. In the **Filter*** list, choose **All**.
5. Keep the destination set to **Send to CloudWatch Logs**.
6. For **Destination log group***, choose **VPCtoSplunkLogGroup**, which is the log group that you created in the previous procedure.
7. To set up an IAM role, choose **Set Up Permissions**.



8. In the new window that appears, keep **IAM Role** set to **Create a new IAM Role**. In the **Role Name** box, enter **`VPCtoSplunkWritetoCWRole`**. Then choose **Allow**.



9. Return to the **Create flow log** browser tab, and refresh the **IAM role*** box. Then choose **VPCtoSplunkWritetoCWRole** in the list.
10. Choose **Create**, and then choose **Close**.
11. Back on the Amazon VPC dashboard, choose **Your VPCs** in the navigation pane. Then select the check box next to your VPC.
12. Scroll down and choose the **Flow Logs** tab, and look for the flow log that you created in the preceding steps. Ensure that its status is **Active**. If it is not, review the previous steps.

Proceed to Step 2: Create a Kinesis Data Firehose Delivery Stream with Splunk as a Destination (p. 118).

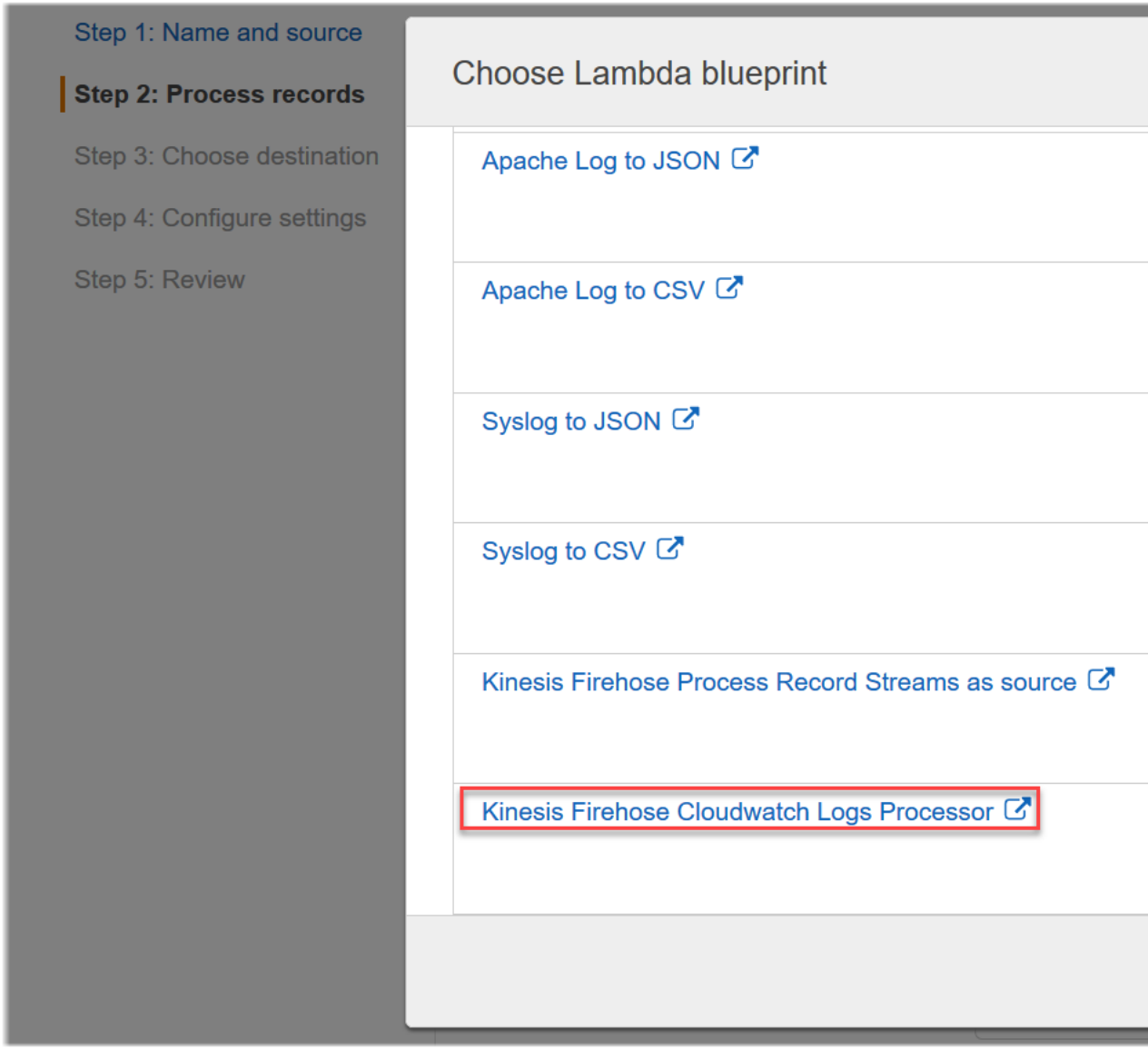
Step 2: Create a Kinesis Data Firehose Delivery Stream with Splunk as a Destination

In this part of the Kinesis Data Firehose tutorial (p. 115), you create an Amazon Kinesis Data Firehose delivery stream to receive the log data from Amazon CloudWatch and deliver that data to Splunk.

The logs that CloudWatch sends to the delivery stream are in a compressed format. However, Kinesis Data Firehose can't send compressed logs to Splunk. Therefore, when you create the delivery stream in the following procedure, you enable data transformation and configure an AWS Lambda function to uncompress the log data. Kinesis Data Firehose then sends the uncompressed data to Splunk.

To create a Kinesis Data Firehose delivery stream with Splunk as a destination

1. Open the Kinesis Data Firehose console at https://console.aws.amazon.com/firehose/.
2. Choose **Create delivery stream**.
3. For the name of the delivery stream, enter **`VPCtoSplunkStream`**. Then scroll to the bottom, and choose **Next**.
4. For **Data transformation***, choose **Enabled**.
5. For **Lambda function***, choose **Create new**.
6. In the **Choose Lambda blueprint** pane, scroll down and choose **Kinesis Firehose Cloudwatch Logs Processor**. This opens the AWS Lambda console.



7. On the AWS Lambda console, for the function name, enter **`VPCtoSplunkLambda`**.
8. In the description text under **Execution role**, choose the **IAM console** link to create a custom role. This opens the AWS Identity and Access Management (IAM) console.
9. In the IAM console, choose **Lambda**.
10. Choose **Next: Permissions**.
11. Choose **Create policy**.
12. Choose the **JSON** tab and replace the existing JSON with the following. Be sure to replace the *`your-region`* and *`your-aws-account-id`* placeholders with your AWS Region code and account ID. Don't include any hyphens or dashes in the account ID. For a list of AWS Region codes, see AWS Regions and Endpoints.

```
{
    "Version": "2012-10-17",
    "Statement": [

        {
```

```
            "Effect": "Allow",
            "Action": [
                "firehose:PutRecordBatch"
            ],
            "Resource": [
                "arn:aws:firehose:your-region:your-aws-account-id:deliverystream/
VPCtoSplunkStream"
            ]
        }
    ]
}
```

This policy allows the Lambda function to put data back into the delivery stream by invoking the `PutRecordBatch` operation. This step is needed because a Lambda function can only return up to 6 MiB of data every time Kinesis Data Firehose invokes it. If the size of the uncompressed data exceeds 6 MiB, the function invokes `PutRecordBatch` to put some of the data back into the delivery stream for future processing.

13. Back in the **Create role** window, refresh the list of policies, then choose **VPCtoSplunkLambdaPolicy** by selecting the box to its left.
14. Choose **Next: Tags**.
15. Choose **Next: Review**.
16. For **Role Name**, enter **`VPCtoSplunkLambdaRole`**, then choose **Create role**.
17. Back in the Lambda console, refresh the list of existing roles, then select **`VPCtoSplunkLambdaRole`**.
18. Scroll down and choose **Create function**.
19. In the Lambda function pane, scroll down to the **Basic settings** section, and increase the timeout to **`3`** minutes.
20. Scroll up and choose **Save**.
21. Back in the **Choose Lambda blueprint** dialog box, choose **Close**.
22. On the delivery stream creation page, under the **Transform source records with AWS Lambda** section, choose the **refresh** button. Then choose **VPCtoSplunkLambda** in the list of functions.
23. Scroll down and choose **Next**.
24. For **Destination***, choose **Splunk**.
25. For **Splunk cluster endpoint**, see the information at Configure Amazon Kinesis Firehose to send data to the Splunk platform in the Splunk documentation.
26. Keep **Splunk endpoint type** set to **Raw endpoint**.
27. Enter the value (and not the name) of your Splunk HTTP Event Collector (HEC) token.
28. For **S3 backup mode***, choose **Backup all events**.
29. Choose an existing Amazon S3 bucket (or create a new one if you want), and choose **Next**.
30. On the **Configure settings** page, scroll down to the **IAM role** section, and choose **Create new or choose**.
31. In the **IAM role** list, choose **Create a new IAM role**. For **Role Name**, enter **`VPCtoSplunkLambdaFirehoseRole`**, and then choose **Allow**.
32. Choose **Next**, and review the configuration that you chose for the delivery stream. Then choose **Create delivery stream**.

Proceed to Step 3: Send the Data from Amazon CloudWatch to Kinesis Data Firehose (p. 121).

Step 3: Send the Data from Amazon CloudWatch to Kinesis Data Firehose

In this step of this Kinesis Data Firehose tutorial (p. 115), you subscribe the delivery stream to the Amazon CloudWatch log group. This step causes the log data to flow from the log group to the delivery stream.

To send log data from CloudWatch Logs to your delivery stream

In this procedure, you use the AWS Command Line Interface (AWS CLI) to create a CloudWatch Logs subscription that sends log events to your delivery stream.

1. Save the following trust policy to a local file, and name the file `VPCtoSplunkCWtoFHTrustPolicy.json`. Be sure to replace the *your-region* placeholder with your AWS Region code.

```
{
  "Statement": {
    "Effect": "Allow",
    "Principal": { "Service": "logs.your-region.amazonaws.com" },
    "Action": "sts:AssumeRole"
  }
}
```

2. In a command window, go to the directory where you saved `VPCtoSplunkCWtoFHPolicy.json`, and run the following AWS CLI command.

```
aws iam create-role --role-name VPCtoSplunkCWtoFHRole --assume-role-policy-document
 file://VPCtoSplunkCWtoFHTrustPolicy.json
```

3. Save the following access policy to a local file, and name the file `VPCtoSplunkCWtoFHAccessPolicy.json`. Be sure to replace the *your-region* and *your-aws-account-id* placeholders with your AWS Region code and account ID.

```
{
    "Statement":[
      {
        "Effect":"Allow",
        "Action":["firehose:*"],
        "Resource":["arn:aws:firehose:your-region:your-aws-account-id:deliverystream/
VPCtoSplunkStream"]
      },
      {
        "Effect":"Allow",
        "Action":["iam:PassRole"],
        "Resource":["arn:aws:iam::your-aws-account-id:role/VPCtoSplunkCWtoFHRole"]
      }
    ]
}
```

4. In a command window, go to the directory where you saved `VPCtoSplunkCWtoFHAccessPolicy.json`, and run the following AWS CLI command.

```
aws iam put-role-policy --role-name VPCtoSplunkCWtoFHRole --
policy-name VPCtoSplunkCWtoFHAccessPolicy --policy-document file://
VPCtoSplunkCWtoFHAccessPolicy.json
```

5. Replace the *your-region* and *your-aws-account-id* placeholders in the following AWS CLI command with your AWS Region code and account ID, and then run the command.

```
aws logs put-subscription-filter --log-group-name "VPCtoSplunkLogGroup" --filter-
name "Destination" --filter-pattern "" --destination-arn "arn:aws:firehose:your-
region:your-aws-account-id:deliverystream/VPCtoSplunkStream" --role-arn
 "arn:aws:iam::your-aws-account-id:role/VPCtoSplunkCWtoFHRole"
```

Proceed to Step 4: Check the Results in Splunk and in Kinesis Data Firehose (p. 122).

Step 4: Check the Results in Splunk and in Kinesis Data Firehose

You can monitor the flow of data at several points in this example. In this step of the Kinesis Data Firehose tutorial (p. 115), you check the data in Splunk, the final destination, and you also monitor its flow through Kinesis Data Firehose.

To check the results in AWS and in Splunk

1. Open the Kinesis Data Firehose console at https://console.aws.amazon.com/firehose/.
2. In the list of delivery streams, choose **VPCtoSplunkStream**.
3. Choose the **Monitoring** tab, and view the graphs. Be sure to adjust the time range and to use the **refresh** button periodically.
4. If you don't see your data in Splunk, see Data Not Delivered to Splunk.

Important
After you verify your results, delete any AWS resources that you don't need to keep, so as not to incur ongoing charges.

Troubleshooting Amazon Kinesis Data Firehose

If Kinesis Data Firehose encounters errors while delivering or processing data, it retries until the configured retry duration expires. If the retry duration ends before the data is delivered successfully, Kinesis Data Firehose backs up the data to the configured S3 backup bucket. If the destination is Amazon S3 and delivery fails or if delivery to the backup S3 bucket fails, Kinesis Data Firehose keeps retrying until the retention period ends. For `DirectPut` delivery streams, Kinesis Data Firehose retains the records for 24 hours. For a delivery stream whose data source is a Kinesis data stream, you can change the retention period as described in Changing the Data Retention Period.

If the data source is a Kinesis data stream, Kinesis Data Firehose retries the following operations indefinitely: `DescribeStream`, `GetRecords`, and `GetShardIterator`.

If the delivery stream uses `DirectPut`, check the `IncomingBytes` and `IncomingRecords` metrics to see if there's incoming traffic. If you are using the `PutRecord` or `PutRecordBatch`, make sure you catch exceptions and retry. We recommend a retry policy with exponential back-off with jitter and several retries. Also, if you use the `PutRecordBatch` API, make sure your code checks the value of FailedPutCount in the response even when the API call succeeds.

If the delivery stream uses a Kinesis data stream as its source, check the `IncomingBytes` and `IncomingRecords` metrics for the source data stream. Additionally, ensure that the `DataReadFromKinesisStream.Bytes` and `DataReadFromKinesisStream.Records` metrics are being emitted for the delivery stream.

For information about tracking delivery errors using CloudWatch, see the section called “Monitoring with CloudWatch Logs” (p. 93).

Issues

Data Not Delivered to Amazon S3

Check the following if data is not delivered to your Amazon Simple Storage Service (Amazon S3) bucket.

- Check the Kinesis Data Firehose `IncomingBytes` and `IncomingRecords` metrics to make sure that data is sent to your Kinesis Data Firehose delivery stream successfully. For more information, see Monitoring Kinesis Data Firehose Using CloudWatch Metrics (p. 79).
- If data transformation with Lambda is enabled, check the Kinesis Data Firehose `ExecuteProcessingSuccess` metric to make sure that Kinesis Data Firehose has tried to invoke your Lambda function. For more information, see Monitoring Kinesis Data Firehose Using CloudWatch Metrics (p. 79).
- Check the Kinesis Data Firehose `DeliveryToS3.Success` metric to make sure that Kinesis Data Firehose has tried putting data to your Amazon S3 bucket. For more information, see Monitoring Kinesis Data Firehose Using CloudWatch Metrics (p. 79).
- Enable error logging if it is not already enabled, and check error logs for delivery failure. For more information, see Monitoring Kinesis Data Firehose Using CloudWatch Logs (p. 93).
- Make sure that the Amazon S3 bucket that is specified in your Kinesis Data Firehose delivery stream still exists.
- If data transformation with Lambda is enabled, make sure that the Lambda function that is specified in your delivery stream still exists.
- Make sure that the IAM role that is specified in your Kinesis Data Firehose delivery stream has access to your S3 bucket and your Lambda function (if data transformation is enabled). For more information, see Grant Kinesis Data Firehose Access to an Amazon S3 Destination (p. 41).
- If you're using data transformation, make sure that your Lambda function never returns responses whose payload size exceeds 6 MB. For more information, see Amazon Kinesis Data Firehose Data Transformation.

Data Not Delivered to Amazon Redshift

Check the following if data is not delivered to your Amazon Redshift cluster.

Data is delivered to your S3 bucket before loading into Amazon Redshift. If the data was not delivered to your S3 bucket, see Data Not Delivered to Amazon S3 (p. 123).

- Check the Kinesis Data Firehose `DeliveryToRedshift.Success` metric to make sure that Kinesis Data Firehose has tried to copy data from your S3 bucket to the Amazon Redshift cluster. For more information, see Monitoring Kinesis Data Firehose Using CloudWatch Metrics (p. 79).
- Enable error logging if it is not already enabled, and check error logs for delivery failure. For more information, see Monitoring Kinesis Data Firehose Using CloudWatch Logs (p. 93).
- Check the Amazon Redshift `STL_CONNECTION_LOG` table to see if Kinesis Data Firehose can make successful connections. In this table, you should be able to see connections and their status based on a user name. For more information, see `STL_CONNECTION_LOG` in the *Amazon Redshift Database Developer Guide*.
- If the previous check shows that connections are being established, check the Amazon Redshift `STL_LOAD_ERRORS` table to verify the reason for the COPY failure. For more information, see `STL_LOAD_ERRORS` in the *Amazon Redshift Database Developer Guide*.
- Make sure that the Amazon Redshift configuration in your Kinesis Data Firehose delivery stream is accurate and valid.
- Make sure that the IAM role that is specified in your Kinesis Data Firehose delivery stream can access the S3 bucket that Amazon Redshift copies data from, and also the Lambda function for data transformation (if data transformation is enabled). For more information, see Grant Kinesis Data Firehose Access to an Amazon S3 Destination (p. 41).
- If your Amazon Redshift cluster is in a virtual private cloud (VPC), make sure that the cluster allows access from Kinesis Data Firehose IP addresses. For more information, see Grant Kinesis Data Firehose Access to an Amazon Redshift Destination (p. 43).

- Make sure that the Amazon Redshift cluster is publicly available.
- If you're using data transformation, make sure that your Lambda function never returns responses whose payload size exceeds 6 MB. For more information, see Amazon Kinesis Data Firehose Data Transformation.

Data Not Delivered to Amazon OpenSearch Service

Check the following if data is not delivered to your Elasticsearch domain.

Data can be backed up to your Amazon S3 bucket concurrently. If data was not delivered to your S3 bucket, see Data Not Delivered to Amazon S3 (p. 123).

- Check the Kinesis Data Firehose `IncomingBytes` and `IncomingRecords` metrics to make sure that data is sent to your Kinesis Data Firehose delivery stream successfully. For more information, see Monitoring Kinesis Data Firehose Using CloudWatch Metrics (p. 79).
- If data transformation with Lambda is enabled, check the Kinesis Data Firehose `ExecuteProcessingSuccess` metric to make sure that Kinesis Data Firehose has tried to invoke your Lambda function. For more information, see Monitoring Kinesis Data Firehose Using CloudWatch Metrics (p. 79).
- Check the Kinesis Data Firehose `DeliveryToElasticsearch.Success` metric to make sure that Kinesis Data Firehose has tried to index data to the Amazon ES cluster. For more information, see Monitoring Kinesis Data Firehose Using CloudWatch Metrics (p. 79).
- Enable error logging if it is not already enabled, and check error logs for delivery failure. For more information, see Monitoring Kinesis Data Firehose Using CloudWatch Logs (p. 93).
- Make sure that the Amazon ES configuration in your delivery stream is accurate and valid.
- If data transformation with Lambda is enabled, make sure that the Lambda function that is specified in your delivery stream still exists.
- Make sure that the IAM role that is specified in your delivery stream can access your Amazon ES cluster and Lambda function (if data transformation is enabled). For more information, see Grant Kinesis Data Firehose Access to a Public OpenSearch Service Destination (p. 45).
- If you're using data transformation, make sure that your Lambda function never returns responses whose payload size exceeds 6 MB. For more information, see Amazon Kinesis Data Firehose Data Transformation.

Data Not Delivered to Splunk

Check the following if data is not delivered to your Splunk endpoint.

- If your Splunk platform is in a VPC, make sure that Kinesis Data Firehose can access it. For more information, see Access to Splunk in VPC.
- If you use an AWS load balancer, make sure that it is a Classic Load Balancer. Kinesis Data Firehose does not support Application Load Balancers or Network Load Balancers. Also, enable duration-based sticky sessions with cookie expiration disabled. For information about how to do this, see Duration-Based Session Stickiness.
- Review the Splunk platform requirements. The Splunk add-on for Kinesis Data Firehose requires Splunk platform version 6.6.X or later. For more information, see Splunk Add-on for Amazon Kinesis Firehose.
- If you have a proxy (Elastic Load Balancing or other) between Kinesis Data Firehose and the HTTP Event Collector (HEC) node, enable sticky sessions to support HEC acknowledgements (ACKs).
- Make sure that you are using a valid HEC token.

- Ensure that the HEC token is enabled. See Enable and disable Event Collector tokens.
- Check whether the data that you're sending to Splunk is formatted correctly. For more information, see Format events for HTTP Event Collector.
- Make sure that the HEC token and input event are configured with a valid index.
- When an upload to Splunk fails due to a server error from the HEC node, the request is automatically retried. If all retries fail, the data gets backed up to Amazon S3. Check if your data appears in Amazon S3, which is an indication of such a failure.
- Make sure that you enabled indexer acknowledgment on your HEC token. For more information, see Enable indexer acknowledgement.
- Increase the value of `HECAcknowledgmentTimeoutInSeconds` in the Splunk destination configuration of your Kinesis Data Firehose delivery stream.
- Increase the value of `DurationInSeconds` under `RetryOptions` in the Splunk destination configuration of your Kinesis Data Firehose delivery stream.
- Check your HEC health.
- If you're using data transformation, make sure that your Lambda function never returns responses whose payload size exceeds 6 MB. For more information, see Amazon Kinesis Data Firehose Data Transformation.
- Make sure that the Splunk parameter named `ackIdleCleanup` is set to `true`. It is false by default. To set this parameter to `true`, do the following:
  - For a managed Splunk Cloud deployment, submit a case using the Splunk support portal. In this case, ask Splunk support to enable the HTTP event collector, set `ackIdleCleanup` to `true` in `inputs.conf`, and create or modify a load balancer to use with this add-on.
  - For a distributed Splunk Enterprise deployment, set the `ackIdleCleanup` parameter to true in the `inputs.conf` file. For *nix users, this file is located under `$SPLUNK_HOME/etc/apps/splunk_httpinput/local/`. For Windows users, it is under `%SPLUNK_HOME%\etc\apps\splunk_httpinput\local\`.
  - For a single-instance Splunk Enterprise deployment, set the `ackIdleCleanup` parameter to `true` in the `inputs.conf` file. For *nix users, this file is located under `$SPLUNK_HOME/etc/apps/splunk_httpinput/local/`. For Windows users, it is under `%SPLUNK_HOME%\etc\apps\splunk_httpinput\local\`.
- See Troubleshoot the Splunk Add-on for Amazon Kinesis Firehose.

Delivery Stream Not Available as a Target for CloudWatch Logs, CloudWatch Events, or AWS IoT Action

Some AWS services can only send messages and events to a Kinesis Data Firehose delivery stream that is in the same AWS Region. Verify that your Kinesis Data Firehose delivery stream is located in the same Region as your other services.

Data Freshness Metric Increasing or Not Emitted

Data freshness is a measure of how current your data is within your delivery stream. It is the age of the oldest data record in the delivery stream, measured from the time that Kinesis Data Firehose ingested the data to the present time. Kinesis Data Firehose provides metrics that you can use to monitor data freshness. To identify the data-freshness metric for a given destination, see the section called "Monitoring with CloudWatch Metrics" (p. 79).

If you enable backup for all events or all documents, monitor two separate data-freshness metrics: one for the main destination and one for the backup.

If the data-freshness metric isn't being emitted, this means that there is no active delivery for the delivery stream. This happens when data delivery is completely blocked or when there's no incoming data.

If the data-freshness metric is constantly increasing, this means that data delivery is falling behind. This can happen for one of the following reasons.

- The destination can't handle the rate of delivery. If Kinesis Data Firehose encounters transient errors due to high traffic, then the delivery might fall behind. This can happen for destinations other than Amazon S3 (it can happen for Amazon OpenSearch Service, Amazon Redshift, or Splunk). Ensure that your destination has enough capacity to handle the incoming traffic.
- The destination is slow. Data delivery might fall behind if Kinesis Data Firehose encounters high latency. Monitor the destination's latency metric.
- The Lambda function is slow. This might lead to a data delivery rate that is less than the data ingestion rate for the delivery stream. If possible, improve the efficiency of the Lambda function. For instance, if the function does network IO, use multiple threads or asynchronous IO to increase parallelism. Also, consider increasing the memory size of the Lambda function so that the CPU allocation can increase accordingly. This might lead to faster Lambda invocations. For information about configuring Lambda functions, see Configuring AWS Lambda Functions.
- There are failures during data delivery. For information about how to monitor errors using Amazon CloudWatch Logs, see the section called “Monitoring with CloudWatch Logs” (p. 93).
- If the data source of the delivery stream is a Kinesis data stream, throttling might be happening. Check the `ThrottledGetRecords`, `ThrottledGetShardIterator`, and `ThrottledDescribeStream` metrics. If there are multiple consumers attached to the Kinesis data stream, consider the following:
  - If the `ThrottledGetRecords` and `ThrottledGetShardIterator` metrics are high, we recommend you increase the number of shards provisioned for the data stream.
  - If the `ThrottledDescribeStream` is high, we recommend you add the `kinesis:listshards` permission to the role configured in KinesisStreamSourceConfiguration.
- Low buffering hints for the destination. This might increase the number of round trips that Kinesis Data Firehose needs to make to the destination, which might cause delivery to fall behind. Consider increasing the value of the buffering hints. For more information, see BufferingHints.
- A high retry duration might cause delivery to fall behind when the errors are frequent. Consider reducing the retry duration. Also, monitor the errors and try to reduce them. For information about how to monitor errors using Amazon CloudWatch Logs, see the section called “Monitoring with CloudWatch Logs” (p. 93).
- If the destination is Splunk and `DeliveryToSplunk.DataFreshness` is high but `DeliveryToSplunk.Success` looks good, the Splunk cluster might be busy. Free the Splunk cluster if possible. Alternatively, contact AWS Support and request an increase in the number of channels that Kinesis Data Firehose is using to communicate with the Splunk cluster.

Record Format Conversion to Apache Parquet Fails

This happens if you take DynamoDB data that includes the `Set` type, stream it through Lambda to a delivery stream, and use an AWS Glue Data Catalog to convert the record format to Apache Parquet.

When the AWS Glue crawler indexes the DynamoDB set data types (`StringSet`, `NumberSet`, and `BinarySet`), it stores them in the data catalog as `SET<STRING>`, `SET<BIGINT>`, and `SET<BINARY>`, respectively. However, for Kinesis Data Firehose to convert the data records to the Apache Parquet format, it requires Apache Hive data types. Because the set types aren't valid Apache Hive data types, conversion fails. To get conversion to work, update the data catalog with Apache Hive data types. You can do that by changing `set` to `array` in the data catalog.

To change one or more data types from `set` to `array` in an AWS Glue data catalog

1. Sign in to the AWS Management Console and open the AWS Glue console at https://console.aws.amazon.com/glue/.
2. In the left pane, under the **Data catalog** heading, choose **Tables**.
3. In the list of tables, choose the name of the table where you need to modify one or more data types. This takes you to the details page for the table.
4. Choose the **Edit schema** button in the top right corner of the details page.
5. In the **Data type** column choose the first `set` data type.
6. In the **Column type** drop-down list, change the type from `set` to `array`.
7. In the **ArraySchema** field, enter `array<string>`, `array<int>`, or `array<binary>`, depending on the appropriate type of data for your scenario.
8. Choose **Update**.
9. Repeat the previous steps to convert other `set` types to `array` types.
10. Choose **Save**.

No Data at Destination Despite Good Metrics

If there are no data ingestion problems and the metrics emitted for the delivery stream look good, but you don't see the data at the destination, check the reader logic. Make sure your reader is correctly parsing out all the data.

Troubleshooting HTTP Endpoints

This section describes common troubleshooting steps when dealing with Kinesis Data Firehose delivering data to generic HTTP Endpoints destinations and to partner destinations, including Datadog, Dynatrace, LogicMonitor, MongoDB, New Relic, Splunk, or Sumo Logic. For the purposes of this section, all applicable destinations are referred to as HTTP endpoints.

Note
The information in this section does not apply to the following destinations: Splunk, ElasticSearch, S3, and Redshift.

CloudWatch Logs

It is highly recommended that you enable CloudWatch Logging for Firehose. Logs are only published when there are errors delivering to your destination.

Destination Exceptions

ErrorCode: HttpEndpoint.DestinationException

```
{
    "deliveryStreamARN": "arn:aws:firehose:us-east-1:123456789012:deliverystream/ronald-
test",
    "destination": "custom.firehose.endpoint.com...",
    "deliveryStreamVersionId": 1,
    "message": "The following response was received from the endpoint destination. 413:
 {\"requestId\": \"43b8e724-dbac-4510-adb7-ef211c6044b9\", \"timestamp\": 1598556019164,
 \"errorMessage\": \"Payload too large\"}",
```

```
    "errorCode": "HttpEndpoint.DestinationException",
    "processor": "arn:aws:lambda:us-east-1:379522611494:function:httpLambdaProcessing"
}
```

Destination exceptions indicate that Firehose **is** able to establish a connection to your endpoint and make an HTTP request, but **did not** receive a 200 response code. 2xx responses that are not 200s will also result in a destination exception. Kinesis Data Firehose logs the response code and a truncated response payload received from the configured endpoint to CloudWatch Logs. Because Kinesis Data Firehose logs the response code and payload without modification or interpretation, it is up to the endpoint to provide the exact reason why it rejected Kinesis Data Firehose's HTTP delivery request. The following are the most common troubleshooting recommendations for these exceptions:

- **400**: Indicates that you are sending a bad request due to a misconfiguration of your Kinesis Data Firehose. Make sure that you have the correct url, common attributes, content encoding, access key, and buffering hints for your destination. See the destination specific documentation on the required configuration.
- **401**: Indicates that the access key you configured for your delivery stream is incorrect or missing.
- **403**: Indicates that the access key you configured for your delivery stream does not have permissions to deliver data to the configured endpoint.
- **413**: Indicates that the request payload that Kinesis Data Firehose sends to the endpoint is too large for the endpoint to handle. Try lowering the buffering hint to the recommended size for your destination.
- **429**: Indicates that Kinesis Data Firehose is sending requests at a greater rate than the destination can handle. Fine tune your buffering hint by increasing your buffering time and/or increasing your buffering size (but still within the limit of your destination).
- **5xx**: Indicates that there is a problem with the destination. The Kinesis Data Firehose service is still working properly.

> **Important**
> Important: While these are the common troubleshooting recommendations, specific endpoints may have different reasons for providing the response codes and the endpoint specific recommendations should be followed first.

Invalid Response

ErrorCode: HttpEndpoint.InvalidResponseFromDestination

```
{
    "deliveryStreamARN": "arn:aws:firehose:us-east-1:123456789012:deliverystream/ronald-
test",
    "destination": "custom.firehose.endpoint.com...",
    "deliveryStreamVersionId": 1,
    "message": "The response received from the specified endpoint is invalid. Contact
 the owner of the endpoint to resolve the issue. Response for request 2de9e8e9-7296-47b0-
bea6-9f17b133d847 is not recognized as valid JSON or has unexpected fields. Raw response
 received: 200 {\"requestId\": null}",
    "errorCode": "HttpEndpoint.InvalidResponseFromDestination",
    "processor": "arn:aws:lambda:us-east-1:379522611494:function:httpLambdaProcessing"
}
```

Invalid response exceptions indicate that Kinesis Data Firehose received an invalid response from the endpoint destination. The response must conform to the response specifications or Kinesis Data Firehose will consider the delivery attempt a failure and will redeliver the same data until the configured retry duration is exceeded. Kinesis Data Firehose treats responses that do not follow the response

specifications as failures even if the response has a 200 status. If you are developing a Kinesis Data Firehose compatible endpoint, follow the response specifications to ensure data is successfully delivered.

Below are some of the common types of invalid responses and how to fix them:

- **Invalid JSON or Unexpected Fields**: Indicates that the response can not be properly deserialized as JSON or has unexpected fields. Ensure that the response is not content-encoded.
- **Missing RequestId**: Indicates that the response does not contain a requestId.
- **RequestId does not match**: Indicates that the requestId in the response does not match the outgoing requestId.
- **Missing Timestamp**: Indicates that the response does not contain a timestamp field. The timestamp field must be a number and not a string.
- **Missing Content-Type Header**: Indicates that the response does not contain a “content-type: application/json” header. No other content-type is accepted.

> **Important**
> Important: Kinesis Data Firehose can only deliver data to endpoints that follow the Firehose request and response specifications. If you are configuring your destination to a third party service, ensure that you are using the correct Kinesis Data Firehose compatible endpoint which will likely be different than the public ingestion endpoint. For example Datadog’s Kinesis Data Firehose endpoint is https://aws-kinesis-http-intake.logs.datadoghq.com/ while its public endpoint is https://api.datadoghq.com/.

Other Common Errors

Additional error codes and definitions are listed below.

- **Error Code: HttpEndpoint.RequestTimeout** - Indicates that the endpoint took longer than 3 minutes to respond. If you are the owner of the destination, decrease the response time of the destination endpoint. If you are not the owner of the destination, contact the owner and ask if anything can be done to lower the response time (i.e. decrease the buffering hint so there is less data being processed per request).
- **Error Code: HttpEndpoint.ResponseTooLarge** - Indicates that the response is too large. The response must be less than 1 MiB including headers.
- **Error Code: HttpEndpoint.ConnectionFailed** - Indicates a connection could not be established with the configured endpoint. This could be due to a typo in the configured url, the endpoint not being accessible to Kinesis Data Firehose, or the endpoint taking too long to respond to the connection request.
- **Error Code: HttpEndpoint.ConnectionReset** - Indicates a connection was made but reset or prematurely closed by the endpoint.
- **Error Code: HttpEndpoint.SSLHandshakeFailure** - Indicates an SSL handshake could not be successfully completed with the configured endpoint.

Amazon Kinesis Data Firehose Quota

Amazon Kinesis Data Firehose has the following quota.

- When dynamic partitioning on a delivery stream is enabled, there is a limit of 500 active partitions that can be created for that delivery stream. You can use the Amazon Kinesis Data Firehose Limits form to request an increase of this quota.
- When dynamic partitioning on a delivery stream is enabled, a max throughput of 25 MB per second is supported for each active partition. This is a hard limit.
- By default, each account can have up to 50 Kinesis Data Firehose delivery streams per Region. If you exceed this number, a call to `CreateDeliveryStream` results in a `LimitExceededException` exception. To increase this quota, you can use Service Quotas if it's available in your Region. For information about using Service Quotas, see Requesting a Quota Increase. If Service Quotas isn't available in your region, you can use the Amazon Kinesis Data Firehose Limits form to request an increase.
- When **Direct PUT** is configured as the data source, each Kinesis Data Firehose delivery stream provides the following combined quota for PutRecord and PutRecordBatch requests:
  - For US East (N. Virginia), US West (Oregon), and Europe (Ireland): 500,000 records/second, 2,000 requests/second, and 5 MiB/second.
  - For US East (Ohio), US West (N. California), AWS GovCloud (US-East), AWS GovCloud (US-West), Asia Pacific (Hong Kong), Asia Pacific (Mumbai), Asia Pacific (Seoul), Asia Pacific (Singapore), Asia Pacific (Sydney), Asia Pacific (Tokyo), Canada (Central), Europe (Frankfurt), Europe (London), Europe (Paris), Europe (Stockholm), Middle East (Bahrain), South America (São Paulo), Africa (Cape Town), and Europe (Milan): 100,000 records/second, 1,000 requests/second, and 1 MiB/second.

  To request an increase in quota, use the Amazon Kinesis Data Firehose Limits form. The three quota scale proportionally. For example, if you increase the throughput quota in US East (N. Virginia), US West (Oregon), or Europe (Ireland) to 10 MiB/second, the other two quota increase to 4,000 requests/second and 1,000,000 records/second.

  Important
  If the increased quota is much higher than the running traffic, it causes small delivery batches to destinations. This is inefficient and can result in higher costs at the destination services. Be sure to increase the quota only to match current running traffic, and increase the quota further if traffic increases.

  Important
  Note that smaller data records can lead to higher costs. Kinesis Data Firehose ingestion pricing is based on the number of data records you send to the service, times the size of each record rounded up to the nearest 5KB (5120 bytes). So, for the same volume of incoming data (bytes), if there is a greater number of incoming records, the cost incurred would be higher. For example, if the total incoming data volume is 5MiB, sending 5MiB of data over 5,000 records costs more compared to sending the same amount of data using 1,000 records. For more information, see Kinesis Data Firehose in the AWS Calculator.

  Note
  When Kinesis Data Streams is configured as the data source, this quota doesn't apply, and Kinesis Data Firehose scales up and down with no limit.
- Each Kinesis Data Firehose delivery stream stores data records for up to 24 hours in case the delivery destination is unavailable and if the source is DirectPut. If the source is Kinesis Data Streams (KDS) and the destination is unavailable, then the data will be retained based on your KDS configuration.
- The maximum size of a record sent to Kinesis Data Firehose, before base64-encoding, is 1,000 KiB.
- The PutRecordBatch operation can take up to 500 records per call or 4 MiB per call, whichever is smaller. This quota cannot be changed.

- The following operations can provide up to five invocations per second (this is a hard limit): `CreateDeliveryStream`, `DeleteDeliveryStream`, `DescribeDeliveryStream`, `ListDeliveryStreams`, `UpdateDestination`, `TagDeliveryStream`, `UntagDeliveryStream`, `ListTagsForDeliveryStream`, `StartDeliveryStreamEncryption`, `StopDeliveryStreamEncryption`.
- The buffer sizes hints range from 1 MiB to 128 MiB for Amazon S3 delivery. For Amazon OpenSearch Service (OpenSearch Service) delivery, they range from 1 MiB to 100 MiB. For AWS Lambda processing, you can set a buffering hint between 1 MiB and 3 MiB using the `BufferSizeInMBs` processor parameter. The size threshold is applied to the buffer before compression. These options are treated as hints. Kinesis Data Firehose might choose to use different values when it is optimal.
- The buffer interval hints range from 60 seconds to 900 seconds.
- For delivery from Kinesis Data Firehose to Amazon Redshift, only publicly accessible Amazon Redshift clusters are supported.
- The retry duration range is from 0 seconds to 7,200 seconds for Amazon Redshift and Amazon ES delivery.
- Kinesis Data Firehose supports Elasticsearch versions 1.5, 2.3, 5.1, 5.3, 5.5, 5.6, as well as all 6.* and 7.* versions.
- When the destination is Amazon S3, Amazon Redshift, or OpenSearch Service, Kinesis Data Firehose allows up to 5 outstanding Lambda invocations per shard. For Splunk, the quota is 10 outstanding Lambda invocations per shard.
- You can use a CMK of type CUSTOMER_MANAGED_CMK to encrypt up to 500 delivery streams.

Appendix - HTTP Endpoint Delivery Request and Response Specifications

For Kinesis Data Firehose to successfully deliver data to custom HTTP endpoints, these endpoints must accept requests and send responses using certain Kinesis Data Firehose request and response formats. This section describes the format specifications of the HTTP requests that the Kinesis Data Firehose service sends to custom HTTP endpoints, as well as the format specifications of the HTTP responses that the Kinesis Data Firehose service expects. HTTP endpoints have 3 minutes to respond to a request before Kinesis Data Firehose times out that request. Kinesis Data Firehose treats responses that do not adhere to the proper format as delivery failures.

Topics

Request Format

Path and URL Parameters

These are configured directly by you as part of a single URL field. Kinesis Data Firehose sends them as configured without modification. Only https destinations are supported. URL restrictions are applied during delivery-stream configuration.

> **Note**
> Currently, only port 443 is supported for HTTP endpoint data delivery.

HTTP Headers - X-Amz-Firehose-Protocol-Version

This header is used to indicate the version of the request/response formats. Currently the only version is 1.0.

HTTP Headers - X-Amz-Firehose-Request-Id

The value of this header is an opaque GUID that can be used for debugging and deduplication purposes. Endpoint implementations should log the value of this header if possible, for both successful and unsuccessful requests. The request ID is kept the same between multiple attempts of the same request.

HTTP Headers - Content-Type

The value of the Content-Type header is always `application/json`.

HTTP Headers - Content-Encoding

A Kinesis Data Firehose delivery stream can be configured to use GZIP to compress the body when sending requests. When this compression is enabled, the value of the Content-Encoding header is set to gzip, as per standard practice. If compression is not enabled, the Content-Encoding header is absent altogether.

HTTP Headers - Content-Length

This is used in the standard way.

HTTP Headers - X-Amz-Firehose-Source-Arn:

The ARN of the Kinesis Data Firehose delivery stream represented in ASCII string format. The ARN encodes region, AWS account ID and the stream name. For example, `arn:aws:firehose:us-east-1:123456789:deliverystream/testStream`.

HTTP Headers - X-Amz-Firehose-Access-Key

This header carries an API key or other credentials. You have the ability to create or update the API-key (aka authorization token) when creating or updating your delivery-stream. Kinesis Data Firehose restricts the size of the access key to 4096 bytes. Kinesis Data Firehose does not attempt to interpret this key in any way. The configured key is copied verbatim into the value of this header.

The contents can be arbitrary and can potentially represent a JWT token or an ACCESS_KEY. If an endpoint requires multi-field credentials (for example, username and password), the values of all of the fields should be stored together within a single access-key in a format that the endpoint understands (JSON or CSV). This field can be base-64 encoded if the original contents are binary. Kinesis Data Firehose does not modifiy and/or encode the configured value and uses the contents as is.

HTTP Headers - X-Amz-Firehose-Common-Attributes

This header carries the common attributes (metadata) that pertain to the entire request, and/or to all records within the request. These are configured directly by you when creating a delivery stream. The value of this attribute is encoded as a JSON object with the following schema:

```
"$schema": http://json-schema.org/draft-07/schema#

properties:
  commonAttributes:
    type: object
    minProperties: 0
    maxProperties: 50
    patternProperties:
      "^.{1,256}$":
        type: string
        minLength: 0
        maxLength: 1024
```

Here's an example:

```
"commonAttributes": {
    "deployment -context": "pre-prod-gamma",
    "device-types": ""
  }
```

Body - Max Size

The maximum body size is configured by you, and can be up to a maximum of 64 MiB, before compression.

Body - Schema

The body carries a single JSON document with the following JSON Schema (written in YAML):

```
"$schema": http://json-schema.org/draft-07/schema#

title: FirehoseCustomHttpsEndpointRequest
description: >
  The request body that the Firehose service sends to
  custom HTTPS endpoints.
type: object
properties:
  requestId:
    description: >
      Same as the value in the X-Amz-Firehose-Request-Id header,
      duplicated here for convenience.
    type: string
  timestamp:
    description: >
      The timestamp (milliseconds since epoch) at which the Firehose
      server generated this request.
    type: integer
  records:
    description: >
      The actual records of the Delivery Stream, carrying
      the customer data.
    type: array
    minItems: 1
    maxItems: 10000
    items:
      type: object
      properties:
        data:
          description: >
            The data of this record, in Base64. Note that empty
            records are permitted in Firehose. The maximum allowed
            size of the data, before Base64 encoding, is 1024000
            bytes; the maximum length of this field is therefore
            1365336 chars.
          type: string
          minLength: 0
          maxLength: 1365336

required:
  - requestId
  - records
```

Here's an example:

```
{
  "requestId": "ed4acda5-034f-9f42-bba1-f29aea6d7d8f",
  "timestamp": 1578090901599
  "records": [
    {
      "data": "aGVsbG8="
    },
    {
      "data": "aGVsbG8gd29ybGQ="
    }
  ]
}
```

Response Format

Default Behavior on Error

If a response fails to conform to the requirements below, the Kinesis Firehose server treats it as though it had a 500 status code with no body.

Status Code

The HTTP status code MUST be in the 2XX, 4XX or 5XX range.

The Kinesis Data Firehose server does NOT follow redirects (3XX status codes). Only response code 200 is considered as a successful delivery of the records to HTTP/EP. Response code 413 (size exceeded) is considered as a permanent failure and the record batch is not sent to error bucket if configured. All other response codes are considered as retriable errors and are subjected to back-off retry algorithm explained later.

Headers - Content Type

The only acceptable content type is application/json.

HTTP Headers - Content-Encoding

Content-Encoding MUST NOT be used. The body MUST be uncompressed.

HTTP Headers - Content-Length

The Content-Length header MUST be present if the response has a body.

Body - Max Size

The response body must be 1 MiB or less in size.

```
"$schema": http://json-schema.org/draft-07/schema#

title: FirehoseCustomHttpsEndpointResponse

description: >
  The response body that the Firehose service sends to
  custom HTTPS endpoints.
type: object
properties:
  requestId:
    description: >
      Must match the requestId in the request.
    type: string

  timestamp:
    description: >
      The timestamp (milliseconds since epoch) at which the
      server processed this request.
    type: integer

  errorMessage:
    description: >
      For failed requests, a message explaining the failure.
      If a request fails after exhausting all retries, the last
      Instance of the error message is copied to error output
      S3 bucket if configured.
    type: string
    minLength: 0
    maxLength: 8192
required:
  - requestId
```

```
  - timestamp
```

Here's an example:

```
Failure Case (HTTP Response Code 4xx or 5xx)
{
  "requestId": "ed4acda5-034f-9f42-bba1-f29aea6d7d8f",
  "timestamp": "1578090903599",
  "errorMessage": "Unable to deliver records due to unknown error."
}
Success case (HTTP Response Code 200)
{
  "requestId": "ed4acda5-034f-9f42-bba1-f29aea6d7d8f",
  "timestamp": 1578090903599
}
```

Error Response Handling

In all error cases the Kinesis Data Firehose server reattempts delivery of the same batch of records using an exponential back-off algorithm. The retries are backed off using an initial back-off time (1 second) with a jitter factor of (15%) and each subsequent retry is backed off using the formula (initial-backoff-time * (multiplier(2) ^ retry_count)) with added jitter. The backoff time is capped by a maximum interval of 2 minutes. For example on the 'n'-th retry the back off time is = MAX(120sec, (1 * (2^n)) * random(0.85, 1,15).

The parameters specified in the previous equation are subject to change. Refer to the AWS Firehose documentation for exact initial back off time, max backoff time, multiplier and jitter percentages used in exponential back off algorithm.

In each subsequent retry attempt the access key and/or destination to which records are delivered might change based on updated configuration of the delivery stream. Kinesis Data Firehose service uses the same request-id across retries in a best-effort manner. This last feature can be used for deduplication purpose by the HTTP end point server. If the request is still not delivered after the maximum time allowed (based on delivery stream configuration) the batch of records can optionally be delivered to an error bucket based on stream configuration.

Examples

Example of a CWLog sourced request:

```
{
  "requestId": "ed4acda5-034f-9f42-bba1-f29aea6d7d8f",
  "timestamp": 1578090901599,
  "records": [
   {
    "data": {
      "messageType": "DATA_MESSAGE",
      "owner": "123456789012",
      "logGroup": "log_group_name",
      "logStream": "log_stream_name",
      "subscriptionFilters": [
        "subscription_filter_name"
      ],
      "logEvents": [
```

```
        {
          "id": "01234567890123456789012345678901234567890123456789012345",
          "timestamp": 1510109208016,
          "message": "log message 1"
        },
        {
          "id": "01234567890123456789012345678901234567890123456789012345",
          "timestamp": 1510109208017,
          "message": "log message 2"
        }
      ]
    }
   }
  ]
}
```

Document History

The following table describes the important changes to the Amazon Kinesis Data Firehose documentation.

Change	Description	Date Changed
Added a topic on custom prefixes.	Added a topic about the expressions that you can use when building a custom prefix for data that is delivered to Amazon S3. See *Custom Amazon S3 Prefixes* (p. 105).	December 20, 2018
Added New Kinesis Data Firehose Tutorial	Added a tutorial that demonstrates how to send Amazon VPC flow logs to Splunk through Kinesis Data Firehose. See Tutorial: Sending VPC Flow Logs to Splunk Using Amazon Kinesis Data Firehose (p. 115).	October 30, 2018
Added Four New Kinesis Data Firehose Regions	Added Paris, Mumbai, Sao Paulo, and London. For more information, see Amazon Kinesis Data Firehose Quota (p. 131).	June 27, 2018
Added Two New Kinesis Data Firehose Regions	Added Seoul and Montreal. For more information, see Amazon Kinesis Data Firehose Quota (p. 131).	June 13, 2018
New Kinesis Streams as Source feature	Added Kinesis Streams as a potential source for records for a Firehose Delivery Stream. For more information, see Source, Destination, and Name (p. 5).	August 18, 2017
Update to console documentation	The delivery stream creation wizard was updated. For more information, see Creating an Amazon Kinesis Data Firehose Delivery Stream (p. 5).	July 19, 2017
New data transformation	You can configure Kinesis Data Firehose to transform your data before data delivery. For more information, see Amazon Kinesis Data Firehose Data Transformation (p. 58).	December 19, 2016
New Amazon Redshift COPY retry	You can configure Kinesis Data Firehose to retry a COPY command to your Amazon Redshift cluster if it fails. For more information, see Creating an Amazon Kinesis Data Firehose Delivery Stream (p. 5), Amazon Kinesis Data Firehose Data Delivery (p. 73), and Amazon Kinesis Data Firehose Quota (p. 131).	May 18, 2016
New Kinesis Data Firehose destination, Amazon OpenSearch Service	You can create a delivery stream with Amazon OpenSearch Service as the destination. For more information, see Creating an Amazon Kinesis Data Firehose Delivery Stream (p. 5), Amazon Kinesis Data Firehose Data Delivery (p. 73), and Grant Kinesis Data Firehose Access to a Public OpenSearch Service Destination (p. 45).	April 19, 2016
New enhanced CloudWatch metrics and troubleshooting features	Updated Monitoring Amazon Kinesis Data Firehose (p. 79) and Troubleshooting Amazon Kinesis Data Firehose (p. 123).	April 19, 2016

Change	Description	Date Changed
New enhanced Kinesis agent	Updated Writing to Kinesis Data Firehose Using Kinesis Agent (p. 25).	April 11, 2016
New Kinesis agents	Added Writing to Kinesis Data Firehose Using Kinesis Agent (p. 25).	October 2, 2015
Initial release	Initial release of the Amazon Kinesis Data Firehose *Developer Guide*.	October 4, 2015

AWS glossary

For the latest AWS terminology, see the AWS glossary in the *AWS General Reference*.