Developer Guide

Amazon Data Firehose

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon Data Firehose: Developer Guide

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Services or capabilities described in AWS documentation might vary by Region. To see the differences applicable to the AWS European Sovereign Cloud Region, see the AWS European Sovereign Cloud User Guide.

Table of Contents

x

What is Amazon Data Firehose 1

Learn key concepts 1

Understand data flow in Amazon Data Firehose 2

Working with AWS SDKs 3

Complete prerequisites to set up Firehose 4

Sign up for AWS 4

Sign up for an AWS account 4

(Optional) Download libraries and tools 4

Tutorial: Create a Firehose stream 5

Choose source and destination for your Firehose stream 5

Configure source settings 7

Configure source settings for Amazon MSK 8

Configure source settings for Amazon Kinesis Data Streams 9

(Optional) Configure record transformation and format conversion 10

Configure destination settings 12

Configure destination settings for Amazon S3 13

Configure destination settings for Apache Iceberg Tables 17

Configure destination settings for Amazon Redshift 17

Configure destination settings for OpenSearch Service 23

Configure destination settings for OpenSearch Serverless 25

Configure destination settings for HTTP Endpoint 27

Configure destination settings for Datadog 29

Configure destination settings for Grafana Cloud 31

Configure destination settings for Honeycomb 33

Configure destination settings for Coralogix 35

Configure destination settings for Dynatrace (Classic cloud platform) 37

Configure destination settings for Dynatrace (New cloud platform) 39

Configure destination settings for LogicMonitor 40

Configure destination settings for Logz.io 42

Configure destination settings for MongoDB Atlas 44

Configure destination settings for New Relic 46

Configure destination settings for Snowflake 47

Configure destination settings for Splunk 51

Configure destination settings for Splunk Observability Cloud ... 53
Configure destination settings for Sumo Logic ... 55
Configure destination settings for Elastic ... 56
Configure backup settings ... 58
Configure buffering hints ... 59
Configure advanced settings ... 62
Test your Firehose stream ... 64
Prerequisites ... 64
Test with Amazon S3 ... 64
Test with Amazon Redshift ... 65
Test with OpenSearch Service ... 65
Test with Splunk ... 66
Test with Apache Iceberg Tables ... 67
Send data to a Firehose stream ... 68
Configure Kinesis agent to send data ... 68
Prerequisites ... 69
Manage AWS credentials ... 69
Create custom credential providers ... 70
Download and install the Agent ... 70
Configure and start the Agent ... 72
Specify agent configuration settings ... 73
Configure multiple file directories and streams ... 77
Pre-process data with Agents ... 78
Use common Agent CLI commands ... 82
Troubleshoot issues when sending from Kinesis Agent ... 83
Send data with AWS SDK ... 85
Single write operations using PutRecord ... 85
Batch write operations using PutRecordBatch ... 85
Send CloudWatch Logs to Firehose ... 86
Decompress CloudWatch Logs ... 86
Extract message after decompression of CloudWatch Logs ... 87
Enable decompression on a new Firehose stream from console ... 88
Enable decompression on an existing Firehose stream ... 89
Disable decompression on Firehose stream ... 90
Troubleshoot decompression in Firehose ... 90
Send CloudWatch Events to Firehose ... 92

Configure AWS IoT to send data to Firehose ... 92
Transform source data ... 94
Understand data transformation flow ... 94
Lambda invocation duration ... 94
Required parameters for data transformation ... 95
Supported Lambda blueprints ... 96
Handle failure in data transformation ... 97
Back up source records ... 98
Partition streaming data ... 99
Enable dynamic partitioning ... 99
Understand partitioning keys ... 100
Create partitioning keys with inline parsing ... 101
Create partitioning keys with an AWS Lambda function ... 102
Use Amazon S3 bucket prefix to deliver data ... 105
Add a new line delimiter when delivering data to Amazon S3 ... 106
Apply dynamic partitioning to aggregated data ... 107
Troubleshoot dynamic partitioning errors ... 108
Buffer data for dynamic partitioning ... 108
Convert input data format ... 110
Deserializer ... 110
Schema ... 111
Serializer ... 112
Enable record format conversion ... 112
Enable record format conversion from console ... 113
Manage record format conversion from Firehose API ... 113
Handling errors for data format conversion ... 114
Understand data delivery ... 115
Understand delivery across AWS accounts and regions ... 117
Understand HTTP endpoint delivery request and response specifications ... 117
Request format ... 118
Response format ... 121
Examples ... 124
Handle data delivery failures ... 125
Amazon S3 ... 125
Amazon Redshift ... 126
Amazon OpenSearch Service and OpenSearch Serverless ... 126

Splunk ... 127
HTTP endpoint destination ... 128
Snowflake ... 129
Configure Amazon S3 object name format ... 130
Understand custom prefixes for Amazon S3 objects ... 139
Configure index rotation for OpenSearch Service ... 144
Pause and resume data delivery ... 144
Pause a Firehose stream ... 145
Resume a Firehose stream ... 146
Deliver data to Apache Iceberg Tables ... 147
Considerations and limitations ... 147
Prerequisites ... 151
Prerequisites to deliver to Iceberg Tables in Amazon S3 ... 151
Prerequisites to deliver to Amazon S3 Tables ... 152
Set up the Firehose stream ... 153
Configure source and destination ... 153
Configure data transformation ... 153
Connect data catalog ... 153
Configure JQ expressions ... 154
Configure unique keys ... 154
Specify retry duration ... 156
Handle failed delivery or processing ... 157
Handle errors ... 157
Configure buffer hints ... 157
Configure advanced settings ... 158
Route incoming records to a single Iceberg table ... 158
Route incoming records to different Iceberg tables ... 159
Provide routing information to Firehose with JSONQuery expression ... 160
Provide routing information using an AWS Lambda function ... 161
Monitor metrics ... 165
Understand supported data types ... 166
Data types examples ... 166
Resources ... 170
Tag a Firehose stream ... 171
Understand tag basics ... 171
Track costs with tagging ... 172

Know tag restrictions ... 172
Security ... 174
Data Protection ... 175
Server-side encryption with Kinesis Data Streams ... 175
Server-side encryption with Direct PUT or other data sources ... 175
Controlling access ... 177
Grant access to your Firehose resources ... 178
Grant Firehose access to your private Amazon MSK cluster ... 179
Allow Firehose to assume an IAM role ... 179
Grant Firehose access to AWS Glue for data format conversion ... 180
Grant Firehose access to an Amazon S3 destination ... 181
Grant Firehose access to Amazon S3 Tables ... 184
Grant Firehose access to an Apache Iceberg Tables destination ... 191
Grant Firehose access to an Amazon Redshift destination ... 192
Grant Firehose access to a public OpenSearch Service destination ... 197
Grant Firehose access to an OpenSearch Service destination in a VPC ... 197
Grant Firehose access to a public OpenSearch Serverless destination ... 198
Grant Firehose access to an OpenSearch Serverless destination in a VPC ... 201
Grant Firehose access to a Splunk destination ... 203
Accessing Splunk in VPC ... 205
Tutorial: Ingest VPC flow logs into Splunk using Amazon Data Firehose ... 208
Accessing Snowflake or HTTP end point ... 208
Grant Firehose access to a Snowflake destination ... 208
Accessing Snowflake in VPC ... 210
Grant Firehose access to an HTTP endpoint destination ... 214
Cross-account delivery from Amazon MSK ... 215
Cross-account delivery to an Amazon S3 destination ... 218
Cross-account delivery to an OpenSearch Service destination ... 220
Using tags to control access ... 221
Authenticate with AWS Secrets Manager ... 224
Understand secrets ... 224
Create a secret ... 225
Use the secret ... 225
Rotate the secret ... 227
Manage IAM roles through console ... 227
Choose an existing IAM role ... 228

Create a new IAM role from console ... 228
Edit IAM role from console ... 230
Compliance validation ... 231
Resilience ... 232
Disaster recovery ... 233
Understand infrastructure security ... 233
Using Firehose with AWS PrivateLink ... 233
Implement security best practices ... 238
Implement least privilege access ... 238
Use IAM roles ... 239
Implement server-side encryption in dependent resources ... 239
Use CloudTrail to monitor API calls ... 239
Monitor Amazon Data Firehose ... 240
Implement best practices with CloudWatch Alarms ... 240
Monitoring with CloudWatch Metrics ... 241
CloudWatch metrics for dynamic partitioning ... 242
CloudWatch metrics for data delivery ... 243
Data ingestion metrics ... 257
API-level CloudWatch metrics ... 266
Data Transformation CloudWatch Metrics ... 269
CloudWatch Logs Decompression Metrics ... 270
Format Conversion CloudWatch Metrics ... 271
Server-Side Encryption (SSE) CloudWatch Metrics ... 271
Dimensions for Amazon Data Firehose ... 272
Amazon Data Firehose Usage Metrics ... 272
Access CloudWatch Metrics for Amazon Data Firehose ... 274
Monitor with CloudWatch Logs ... 274
Data delivery errors ... 276
Access CloudWatch logs for Amazon Data Firehose ... 311
Monitor Agent Health ... 312
Monitor with CloudWatch ... 313
Log Firehose API calls ... 313
Firehose information in CloudTrail ... 314
Example: Firehose log file entries ... 315
Code examples ... 320
Basics ... 320

Actions ... 321
Scenarios ... 332
Put records to Firehose ... 332
Troubleshoot errors ... 346
Common issues ... 346
Firehose stream unavailable ... 346
No data at destination ... 347
Data freshness metric increasing or not emitted ... 347
Record format conversion to Apache Parquet fails ... 348
Missing fields for transformed object for Lambda ... 349
Troubleshooting Amazon S3 ... 349
Troubleshooting Amazon Redshift ... 350
Troubleshooting Amazon OpenSearch Service ... 351
Troubleshooting Splunk ... 353
Troubleshooting Snowflake ... 354
Firehose stream creation fails ... 354
Troubleshooting Firehose endpoint reachability ... 356
Troubleshooting HTTP Endpoints ... 357
CloudWatch Logs ... 357
Troubleshooting MSK As Source ... 360
Hose creation fails ... 361
Hose Suspended ... 361
Hose Backpresurred ... 361
Incorrect Data Freshness ... 361
MSK cluster connection issues ... 362
Quota ... 365
Document history ... 369

What is Amazon Data Firehose?

Amazon 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 OpenSearch Serverless, Splunk, Apache Iceberg Tables, and any custom HTTP endpoint or HTTP endpoints owned by supported third-party service providers, including Datadog, Dynatrace, LogicMonitor, MongoDB, New Relic, Coralogix, and Elastic. With Amazon Data Firehose, you don't need to write applications or manage resources. You configure your data producers to send data to Amazon Data Firehose, and it automatically delivers the data to the destination that you specified. You can also configure Amazon 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?

Learn key concepts

As you get started with Amazon Data Firehose, you can benefit from understanding the following concepts.

Firehose stream

The underlying entity of Amazon Data Firehose. You use Amazon Data Firehose by creating a Firehose stream and then sending data to it. For more information, see Tutorial: Create a Firehose stream from console and Send data to a Firehose stream.

Record

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

Data producer

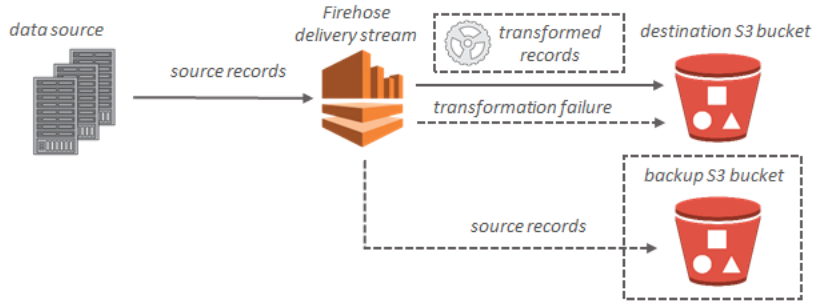
Producers send records to Firehose streams. For example, a web server that sends log data to a Firehose stream is a data producer. You can also configure your Firehose stream to automatically read data from an existing Kinesis data stream, and load it into destinations. For more information, see Send data to a Firehose stream.

Buffer size and buffer interval

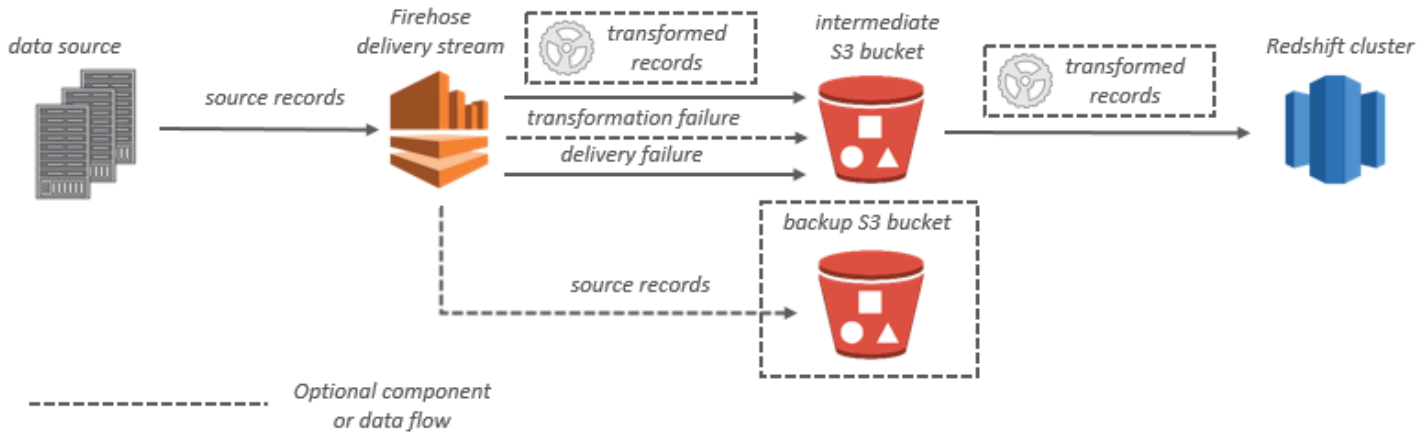
Amazon Data Firehose buffers incoming streaming data to a certain size or for a certain period of time before delivering it to destinations. **Buffer Size** is in MBs and **Buffer Interval** is in seconds.

Understand data flow in Amazon Data Firehose

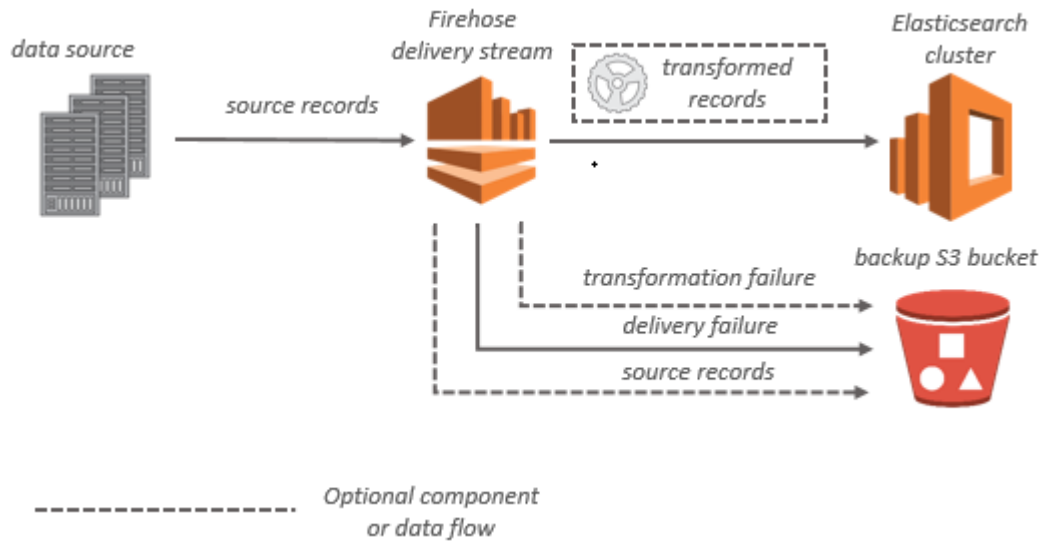
For Amazon S3 destinations, streaming data is delivered to your S3 bucket. If data transformation is enabled, you can optionally back up source data to another Amazon S3 bucket.



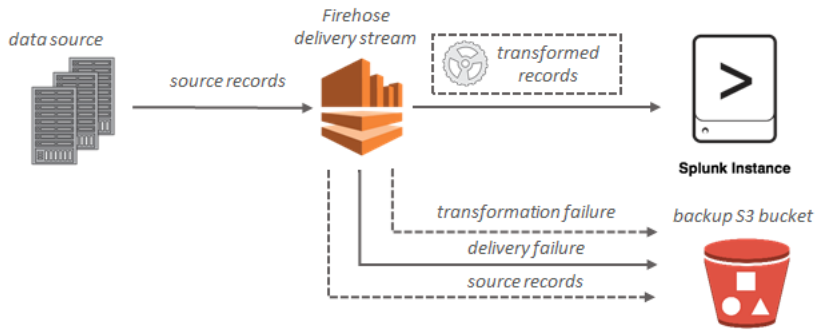
For Amazon Redshift destinations, streaming data is delivered to your S3 bucket first. Amazon Data Firehose then issues an Amazon Redshift **COPY** command to load data from your S3 bucket to your Amazon Redshift cluster. If data transformation is enabled, you can optionally back up source data to another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



For OpenSearch Service destinations, streaming data is delivered to your OpenSearch Service 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.



Using Firehose with an AWS SDK

AWS software development kits (SDKs) are available for many popular programming languages. Each SDK provides an API, code examples, and documentation that make it easier for developers to build applications in their preferred language.

Complete prerequisites to set up Amazon Data Firehose

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

Tasks

- Sign up for AWS
- (Optional) Download libraries and tools

Sign up for AWS

Sign up for an AWS account

To get started with AWS, you need an AWS account. For information about creating an AWS account, see Getting started with an AWS account in the *AWS Account Management Reference Guide*.

(Optional) Download libraries and tools

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

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

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

Tutorial: Create a Firehose stream from console

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

You can update the configuration of your Firehose stream at any time after it's created, using the Amazon Data Firehose console or UpdateDestination. Your Firehose 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 Firehose 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 Configure Amazon S3 object name format.

Perform the steps in the following topics to create a Firehose stream.

Topics

- Choose source and destination for your Firehose stream
- Configure source settings
- (Optional) Configure record transformation and format conversion
- Configure destination settings
- Configure backup settings
- Configure advanced settings

Choose source and destination for your Firehose stream

1. Open the Firehose console at https://eusc-de-east-1.console.amazonaws-eusc.eu/firehose/.
2. Choose **Create Firehose stream**.
3. On the **Create Firehose stream** page, choose a source for your Firehose stream from one of the following options.

   - **Direct PUT** – Choose this option to create a Firehose stream that producer applications write to directly. Here is a list of AWS services and agents and open source services that integrate with Direct PUT in Amazon Data Firehose. This list is not exhaustive, and there may be additional services that can be used to send data directly to Firehose.

     - AWS SDK

- AWS Lambda
- AWS CloudWatch Logs
- AWS CloudWatch Events
- AWS Cloud Metric Streams
- AWS IoT
- AWS Eventbridge
- Amazon Simple Email Service
- Amazon SNS
- AWS WAF web ACL logs
- Amazon API Gateway - Access logs
- Amazon Pinpoint
- Amazon MSK Broker Logs
- Amazon Route 53 Resolver query logs
- AWS Network Firewall Alerts Logs
- AWS Network Firewall Flow Logs
- Amazon Elasticache Redis SLOWLOG
- Kinesis Agent (linux)
- Kinesis Tap (windows)
- Fluentbit
- Fluentd
- Apache Nifi
- Snowflake

- **Amazon Kinesis Data Streams** – Choose this option to configure a Firehose stream that uses a Kinesis data stream as a data source. You can then use 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 Sending data to a Firehose stream with Kinesis Data Streams.
- **Amazon MSK** – Choose this option to configure a Firehose stream that uses Amazon MSK as a data source. You can then use Firehose to read data easily from an existing Amazon MSK clusters and load it into specified S3 buckets. For more information, see Sending data to a Firehose stream with Amazon MSK.

4. Choose a destination for your Firehose stream from one of the following destinations that Firehose supports.

   - Amazon OpenSearch Service
   - Amazon OpenSearch Serverless
   - Amazon Redshift
   - Amazon S3
   - Apache Iceberg Tables
   - Coralogix
   - Datadog
   - Dynatrace
   - Elastic
   - HTTP Endpoint
   - Honeycomb
   - Logic Monitor
   - Logz.io
   - MongoDB Cloud
   - New Relic
   - Splunk
   - Splunk Observability Cloud
   - Sumo Logic
   - Snowflake
5. For **Firehose stream name**, you can either use the name that the console generates for you or add a Firehose stream of your choice.

Configure source settings

You can configure the source settings based on the source that you choose to send information to a Firehose stream from console. You can configure source settings for Amazon MSK and Amazon Kinesis Data Streams as the source. There are no source settings available for Direct PUT as the source.

Configure source settings for Amazon MSK

When you choose Amazon MSK to send information to a Firehose stream, you can choose between MSK provisioned and MSK-Serverless clusters. You can then use Firehose to read data easily from a specific Amazon MSK cluster and topic and load it into the specified S3 destination.

In the **Source settings** section of the page, provide values for the following fields.

Amazon MSK cluster connectivity

Choose either the **Private bootstrap brokers** (recommended) or **Public bootstrap brokers** option based on your cluster configuration. Bootstrap brokers is what Apache Kafka client uses as a starting point to connect to the cluster. Public bootstrap brokers are intended for public access from outside of AWS, while private bootstrap brokers are intended for access from within AWS. For more information about Amazon MSK, see Amazon Managed Streaming for Apache Kafka.

To connect to a provisioned or serverless Amazon MSK cluster through private bootstrap brokers, the cluster must meet all of the following requirements.

- The cluster must be active.
- The cluster must have IAM as one of its access control methods.
- Multi-VPC private connectivity must be enabled for the IAM access control method.
- You must add to this cluster a resource-based policy which grants Firehose service principal the permission to invoke the Amazon MSK `CreateVpcConnection` API operation.

To connect to a provisioned Amazon MSK cluster through public bootstrap brokers, the cluster must meet all of the following requirements.

- The cluster must be active.
- The cluster must have IAM as one of its access control methods.
- The cluster must be public-accessible.

MSK cluster account

You can choose the account where the Amazon MSK cluster resides. This can be one of the following.

- **Current account** – Allows you to ingest data from an MSK cluster in the current AWS account. For this, you must specify the ARN of the Amazon MSK cluster from where your Firehose stream will read data.

- **Cross-account** – Allows you to ingest data from an MSK cluster in another AWS account. For more information, see Cross-account delivery from Amazon MSK.

Topic

Specify the Apache Kafka topic from which you want your Firehose stream to ingest data. You cannot update this topic after Firehose stream creation completes.

> **Note**
>
> Firehose automatically decompresses Apache Kafka messages.

Configure source settings for Amazon Kinesis Data Streams

Configure the source settings for Amazon Kinesis Data Streams to send information to a Firehose stream as following.

> **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 Firehose stream, Amazon Data Firehose de-aggregates the records before it delivers them to the destination. If you configure your Firehose stream to transform the data, Amazon 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.

Under the **Source settings**, choose an existing stream in the **Kinesis data stream** list, or enter a data stream ARN in the format `arn:aws:kinesis:[Region]:[AccountId]:stream/[StreamName]`.

If you do not have an existing data stream then choose **Create** to create a new one from Amazon Kinesis console. You may need an IAM role that has the necessary permission on the Kinesis stream. For more information, see ???. After you create a new stream, choose the refresh icon 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 Firehose stream, the Amazon Data Firehose `PutRecord` and `PutRecordBatch` operations are disabled. To add data to your Firehose stream in this case, use the Kinesis Data Streams `PutRecord` and `PutRecords` operations.

Amazon Data Firehose starts reading data from the `LATEST` position of your Kinesis stream. For more information about Kinesis Data Streams positions, see GetShardIterator.

Amazon Data Firehose calls the Kinesis Data Streams GetRecords operation once per second for each shard. However, when full backup is enabled, Firehose calls the Kinesis Data Streams `GetRecords` operation twice per second for each shard, one for primary delivery destination and another for full backup.

More than one Firehose stream can read from the same Kinesis stream. Other Kinesis applications (consumers) can also read from the same stream. Each call from any Firehose 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.

Proceed to the next step to configure record transformation and format conversion.

(Optional) Configure record transformation and format conversion

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

If you choose Amazon MSK as the source for your Firehose stream.

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

1. **Data transformation**

   To create a Firehose stream that doesn't transform incoming data, do not check the **Enable data transformation** checkbox.

To specify a Lambda function for Firehose to invoke and use to transform incoming data before delivering it, check the **Enable data transformation** checkbox. 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 Firehose. For more information, see Transform source data in Amazon Data Firehose.

2. In the **Convert record format** section, provide values for the following field:

   Record format conversion

   To create a Firehose 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 Firehose to use to convert your record format. For more information, see *Convert input data format*.

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

If you choose Amazon Kinesis Data Streams or Direct PUT as the source for your Firehose stream

In the Source settings section, provide the following fields.

1. Under **Transform records**, choose one of the following:

   a. If your destination is Amazon S3 or Splunk, in the **Decompress source records Amazon CloudWatch Logs** section, choose **Turn on decompression**.

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

      Data transformation

      To create a Firehose stream that doesn't transform incoming data, do not check the **Enable data transformation** checkbox.

To specify a Lambda function for Amazon Data Firehose to invoke and use to transform incoming data before delivering it, check the **Enable data transformation** checkbox. 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 Amazon Data Firehose. For more information, see Transform source data in Amazon Data Firehose.

2. In the **Convert record format** section, provide values for the following field:

   Record format conversion

   To create a Firehose 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 Amazon Data Firehose to use to convert your record format. For more information, see *Convert input data format*.

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

Configure destination settings

This section describes the settings that you must configure for your Firehose stream based on the destination you select.

Topics

- Configure destination settings for Amazon S3
- Configure destination settings for Apache Iceberg Tables
- Configure destination settings for Amazon Redshift
- Configure destination settings for OpenSearch Service
- Configure destination settings for OpenSearch Serverless
- Configure destination settings for HTTP Endpoint
- Configure destination settings for Datadog
- Configure destination settings for Grafana Cloud

- Configure destination settings for Honeycomb
- Configure destination settings for Coralogix
- Configure destination settings for Dynatrace (Classic cloud platform)
- Configure destination settings for Dynatrace (New cloud platform)
- Configure destination settings for LogicMonitor
- Configure destination settings for Logz.io
- Configure destination settings for MongoDB Atlas
- Configure destination settings for New Relic
- Configure destination settings for Snowflake
- Configure destination settings for Splunk
- Configure destination settings for Splunk Observability Cloud
- Configure destination settings for Sumo Logic
- Configure destination settings for Elastic

Configure destination settings for Amazon S3

You must specify the following settings in order to use Amazon S3 as the destination for your Firehose 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.

  New line delimiter

  You can configure your Firehose 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**. If you plan to use Athena to query S3 objects with aggregated records, enable this option.

  Dynamic partitioning

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

Multi record deaggregation

This is the process of parsing through the records in the Firehose 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, Amazon Data Firehose parses the records and looks for multiple valid JSON objects within each API call. When the Firehose 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 deaggregation 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.

Firehose stream performs the following processing steps in the following order: KPL (protobuf) deaggregation, JSON or delimiter deaggregation, Lambda processing, data partitioning, data format conversion, and Amazon S3 delivery.

Multi record deaggregation type

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

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 dynamically 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. 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 Firehose stream.

S3 bucket prefix

When you enable and configure dynamic partitioning, you must specify the S3 bucket prefixes to which Amazon 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 bucket and S3 error output prefix time zone

Choose a time zone that you want to use for date and time in custom prefixes for Amazon S3 objects. By default, Firehose adds a time prefix in UTC. You can change the time zone used in S3 prefixes if you want to use different time zone.

Buffering hints

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 compression

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 Firehose streams with Amazon Redshift as the destination.

S3 file extension format (optional)

Specify a file extension format for objects delivered to Amazon S3 destination bucket. If you enable this feature, specified file extension will override default file extensions appended by Data Format Conversion or S3 compression features such as .parquet or .gz. Make sure if you configured the right file extension when you use this feature with Data Format Conversion or S3 compression. File extension must start with a period (.) and can contain allowed characters: 0-9a-z!-_.*'(). File extension cannot exceed 128 characters.

S3 encryption

Firehose supports Amazon S3 server-side encryption with AWS Key Management Service (SSE-KMS) for encrypting delivered data in Amazon S3. You can choose to use the default encryption type specified in the destination S3 bucket or to encrypt with a key from the list of AWS KMS keys that you own. If you encrypt the data with AWS KMS keys, you can use either the default AWS managed key (aws/s3) or a customer managed key. For more information, see Protecting Data Using Server-Side Encryption with AWS KMS-Managed Keys (SSE-KMS).

Configure destination settings for Apache Iceberg Tables

Firehose supports Apache Iceberg Tables as a destination in all AWS Regions except China Regions, AWS GovCloud (US) Regions, Asia Pacific (Taipei), Asia Pacific (Malaysia), Asia Pacific (New Zealand), and Mexico (Central).

For more information on Apache Iceberg Tables as your destination, see Deliver data to Apache Iceberg Tables with Amazon Data Firehose.

Configure destination settings for Amazon Redshift

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

Choose either of the following procedures based on whether you have an Amazon Redshift provisioned cluster or an Amazon Redshift Serverless workgroup.

- Amazon Redshift Provisioned Cluster
- Configure destination settings for Amazon Redshift Serverless workgroup

> **Note**
>
> Firehose can't write to Amazon Redshift clusters that use enhanced VPC routing.

Amazon Redshift Provisioned Cluster

This section describes settings for using Amazon Redshift provisioned cluster as your Firehose 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 Amazon Data Firehose IP addresses. For more information, see Grant Firehose access to an Amazon Redshift destination.

Authentication

You can either choose to enter the username/password directly or retrieve the secret from AWS Secrets Manager to access the Amazon Redshift cluster.

- **User name**

  Specify 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**

  Specify the password for the user that has permissions to access the cluster.

- **Secret**

  Select a secret from AWS Secrets Manager that contains the credentials for the Amazon Redshift cluster. If you do not see your secret in the drop-down list, create one in AWS Secrets Manager for your Amazon Redshift credentials. For more information, see Authenticate with AWS Secrets Manager in Amazon Data Firehose.

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

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.

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. 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 Configure Amazon S3 object name format.

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

Buffering hints

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 compression

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 Firehose streams with Amazon Redshift as the destination.

S3 file extension format (optional)

S3 file extension format (optional) – Specify a file extension format for objects delivered to Amazon S3 destination bucket. If you enable this feature, specified file extension will override default file extensions appended by Data Format Conversion or S3 compression features such as .parquet or .gz. Make sure if you configured the right file extension when you use this feature with Data Format Conversion or S3 compression. File extension must start with a period (.) and can contain allowed characters: 0-9a-z!-_.*'(). File extension cannot exceed 128 characters.

S3 encryption

Firehose supports Amazon S3 server-side encryption with AWS Key Management Service (SSE-KMS) for encrypting delivered data in Amazon S3. You can choose to use the default encryption type specified in the destination S3 bucket or to encrypt with a key from the list of AWS KMS keys that you own. If you encrypt the data with AWS KMS keys, you can use either the default AWS managed key (aws/s3) or a customer managed key. For more information, see Protecting Data Using Server-Side Encryption with AWS KMS-Managed Keys (SSE-KMS).

Configure destination settings for Amazon Redshift Serverless workgroup

This section describes settings for using Amazon Redshift Serverless workgroup as your Firehose stream destination.

- Enter values for the following fields:

Workgroup name

The Amazon Redshift Serverless workgroup to which S3 bucket data is copied. Configure the Amazon Redshift Serverless workgroup to be publicly accessible and unblock the Firehose IP addresses. For more information, see the Connect to a publicly accessible Amazon Redshift Serverless instance section in Connecting to Amazon Redshift Serverless and also Grant Firehose access to an Amazon Redshift destination.

Authentication

You can either choose to enter the username/password directly or retrieve the secret from AWS Secrets Manager to access the Amazon Redshift Serverless workgroup.

- **User name**

  Specify an Amazon Redshift user with permissions to access the Amazon Redshift Serverless workgroup. This user must have the Amazon Redshift `INSERT` permission for copying data from the S3 bucket to the Amazon Redshift Serverless workgroup.

- **Password**

  Specify the password for the user that has permissions to access the Amazon Redshift Serverless workgroup.

- **Secret**

  Select a secret from AWS Secrets Manager that contains the credentials for the Amazon Redshift Serverless workgroup. If you do not see your secret in the drop-down list, create one in AWS Secrets Manager for your Amazon Redshift credentials. For more information, see Authenticate with AWS Secrets Manager in Amazon Data Firehose.

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

Amazon 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 Serverless workgroup. 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.

Firehose doesn't delete the data from your S3 bucket after loading it to your Amazon Redshift Serverless workgroup. 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. 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 Configure Amazon S3 object name format.

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 Serverless workgroup. 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 Firehose to retry if data **COPY** to your Amazon Redshift Serverless workgroup fails. Firehose retries every 5 minutes until the retry duration ends. If you set the retry duration to 0 (zero) seconds, Firehose does not retry upon a **COPY** command failure.

Buffering hints

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 compression

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 Firehose streams with Amazon Redshift as the destination.

S3 file extension format (optional)

S3 file extension format (optional) – Specify a file extension format for objects delivered to Amazon S3 destination bucket. If you enable this feature, specified file extension will

override default file extensions appended by Data Format Conversion or S3 compression features such as .parquet or .gz. Make sure if you configured the right file extension when you use this feature with Data Format Conversion or S3 compression. File extension must start with a period (.) and can contain allowed characters: 0-9a-z!-_.*'(). File extension cannot exceed 128 characters.

S3 encryption

Firehose supports Amazon S3 server-side encryption with AWS Key Management Service (SSE-KMS) for encrypting delivered data in Amazon S3. You can choose to use the default encryption type specified in the destination S3 bucket or to encrypt with a key from the list of AWS KMS keys that you own. If you encrypt the data with AWS KMS keys, you can use either the default AWS managed key (aws/s3) or a customer managed key. For more information, see Protecting Data Using Server-Side Encryption with AWS KMS-Managed Keys (SSE-KMS).

Configure destination settings for OpenSearch Service

Firehose supports Elasticsearch versions – 1.5, 2.3, 5.1, 5.3, 5.5, 5.6, as well as all 6.*, 7.*, and 8.* versions. Firehose supports Amazon OpenSearch Service 2.x and 3.x.

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

- Enter values for the following fields:

  OpenSearch Service domain

  The OpenSearch Service domain to which your data is delivered.

  Index

  The OpenSearch Service index name to be used when indexing data to your OpenSearch Service cluster.

  Index rotation

  Choose whether and how often the OpenSearch Service index should be rotated. If index rotation is enabled, Amazon Data Firehose appends the corresponding timestamp to the specified index name and rotates. For more information, see Configure index rotation for OpenSearch Service.

Type

The OpenSearch Service type name to be used when indexing data to your OpenSearch Service cluster. For Elasticsearch 7.x and OpenSearch 1.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, Firehose returns an error during runtime.

For Elasticsearch 7.x, leave this field empty.

Retry duration

Time duration for Firehose to retry if an index request to OpenSearch fails. For retry duration, you can set any value between 0-7200 seconds. The default retry duration is 300 seconds. Firehose will retry multiple times with exponential back off until the retry duration expires.

After the retry duration expires, Firehose delivers the data to Dead Letter Queue (DLQ), a configured S3 error bucket. For data delivered to DLQ, you have to re-drive the data back from configured S3 error bucket to OpenSearch destination.

If you want to block Firehose stream from delivering data to DLQ due to downtime or maintenance of OpenSearch clusters, you can configure the retry duration to a higher value in seconds. You can increase the retry duration value above to 7200 seconds by contacting the AWS support.

DocumentID type

Indicates the method for setting up document ID. The supported methods are Firehose-generated document ID and OpenSearch Service-generated document ID. Firehose-generated document ID is the default option when the document ID value is not set. OpenSearch Service-generated document ID is the recommended option because it supports write-heavy operations, including log analytics and observability, consuming fewer CPU resources at the OpenSearch Service domain and thus, resulting in improved performance.

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 Amazon Data Firehose to use when it sends data to your OpenSearch Service domain. You can use the same security groups that the OpenSearch Service domain is using. 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 Firehose stream. If you use the same security group for both your Firehose 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.

> **Important**
>
> When you specify subnets for delivering data to the destination in a private VPC, make sure you have enough number of free IP addresses in chosen subnets. If there is no available free IP address in a specified subnet, Firehose cannot create or add ENIs for the data delivery in the private VPC, and the delivery will be degraded or fail.

Buffer hints

Amazon 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.

Configure destination settings for OpenSearch Serverless

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

- Enter values for the following fields:

  OpenSearch Serverless collection

  The endpoint for a group of OpenSearch Serverless indexes to which your data is delivered.

  Index

  The OpenSearch Serverless index name to be used when indexing data to your OpenSearch Serverless collection.

Destination VPC connectivity

If your OpenSearch Serverless collection is in a private VPC, use this section to specify that VPC. Also specify the subnets and subgroups that you want Amazon Data Firehose to use when it sends data to your OpenSearch Serverless collection.

> **Important**
>
> When you specify subnets for delivering data to the destination in a private VPC, make sure you have enough number of free IP addresses in chosen subnets. If there is no available free IP address in a specified subnet, Firehose cannot create or add ENIs for the data delivery in the private VPC, and the delivery will be degraded or fail.

Retry duration

Time duration for Firehose to retry if an index request to OpenSearch Serverless fails. For retry duration, you can set any value between 0-7200 seconds. The default retry duration is 300 seconds. Firehose will retry multiple times with exponential back off until the retry duration expires.

After the retry duration expires, Firehose delivers the data to Dead Letter Queue (DLQ), a configured S3 error bucket. For data delivered to DLQ, you have to re-drive the data back from configured S3 error bucket to OpenSearch Serverless destination.

If you want to block Firehose stream from delivering data to DLQ due to downtime or maintenance of OpenSearch Serverless clusters, you can configure the retry duration to a higher value in seconds. You can increase the retry duration value above to 7200 seconds by contacting the AWS support.

Buffer hints

Amazon 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.

Configure destination settings for HTTP Endpoint

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 Understand HTTP endpoint delivery request and response specifications.

- 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.

  Authentication

  You can either choose to enter the access key directly or retrieve the secret from AWS Secrets Manager to access the HTTP endpoint.

  - **(Optional) Access key**

    Contact the endpoint owner if you need to obtain the access key to enable data delivery to their endpoint from Firehose.

  - **Secret**

    Select a secret from AWS Secrets Manager that contains the access key for the HTTP endpoint. If you do not see your secret in the drop-down list, create one in AWS Secrets Manager for the access key. For more information, see Authenticate with AWS Secrets Manager in Amazon Data Firehose.

Content encoding

Amazon 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 Amazon Data Firehose retries sending data to the selected HTTP endpoint.

After sending data, Amazon 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, Amazon Data Firehose starts the retry duration counter. It keeps retrying until the retry duration expires. After that, Amazon Data Firehose considers it a data delivery failure and backs up the data to your Amazon S3 bucket.

Every time that Amazon 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, Amazon Data Firehose still waits for the acknowledgment until it receives it or the acknowledgement timeout period is reached. If the acknowledgment times out, Amazon 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 Amazon Data Firehose to retry sending data, set this value to 0.

Parameters - optional

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

Buffering hints

Amazon 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 Firehose stream and try again.

Configure destination settings for Datadog

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 where you want to send data from one of the following options in the drop-down menu.

  - **Datadog logs - US1**
  - **Datadog logs - US3**
  - **Datadog logs - US5**
  - **Datadog logs - AP1**
  - **Datadog logs - EU**
  - **Datadog logs - GOV**
  - **Datadog metrics - US**
  - **Datadog metrics - US5**
  - **Datadog metrics - AP1**
  - **Datadog metrics - EU**
  - **Datadog configurations - US1**
  - **Datadog configurations - US3**
  - **Datadog configurations - US5**
  - **Datadog configurations - AP1**
  - **Datadog configurations - EU**

- **Datadog configurations - US GOV**

Authentication

You can either choose to enter the API key directly or retrieve the secret from AWS Secrets Manager to access Datadog.

- **API key**

  Contact Datadog to obtain the API key that you need to enable data delivery to this endpoint from Firehose.

- **Secret**

  Select a secret from AWS Secrets Manager that contains the API key for Datadog. If you do not see your secret in the drop-down list, create one in AWS Secrets Manager. For more information, see Authenticate with AWS Secrets Manager in Amazon Data Firehose.

Content encoding

Amazon 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 Amazon Data Firehose retries sending data to the selected HTTP endpoint.

After sending data, Amazon 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, Amazon Data Firehose starts the retry duration counter. It keeps retrying until the retry duration expires. After that, Amazon Data Firehose considers it a data delivery failure and backs up the data to your Amazon S3 bucket.

Every time that Amazon 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, Amazon Data Firehose still waits for the acknowledgment until it receives it or the acknowledgement timeout period is reached. If the acknowledgment times out, Amazon 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 Amazon Data Firehose to retry sending data, set this value to 0.

Parameters - optional

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

Buffering hints

Amazon 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.

Configure destination settings for Grafana Cloud

This section describes options for using **Grafana Cloud** for your destination.

Both the HTTP endpoint URL format and the API key scopes that are required depend on the type of data that you send. For instructions on configuring the connection, see the following documentation on the Grafana website:

- Metrics data – Amazon CloudWatch metric streams
- Logs data – Amazon Data Firehose logs

- Choose options to use Grafana Cloud as the destination for your Firehose stream.

  Ingestion type

  Choose to deliver **Metrics** or **Logs** (default) to Grafana Cloud for further analysis and processing.

  HTTP endpoint URL

  Specify the URL for the HTTP endpoint. The format of the URL depends on the type of data that you send.

  For metrics data, use the following format.

```
https://aws-metric-streams-MIMIR_CELL_ID.DOMAIN/aws-metrics/api/v1/push
```

For logs data, use the following format.

```
https://aws-LOKI_CELL_ID.grafana.net/aws-logs/api/v1/push
```

For instructions on determining the values for your Grafana Cloud account, see the Grafana connection documentation for the type of data that you send.

Authentication

You can either choose to enter the API key directly or retrieve the secret from AWS Secrets Manager to access Grafana Cloud.

- **API key**

  Generate the API key in Grafana Cloud that you need to enable data delivery to this endpoint from Firehose. The scopes that the API key requires depend on the type of data that you send. For instructions on generating the API key, see the Grafana connection documentation for the type of data that you send.

- **Secret**

  Select a secret from AWS Secrets Manager that contains the API key for Grafana Cloud. If you do not see your secret in the drop-down list, create one in AWS Secrets Manager. For more information, see Authenticate with AWS Secrets Manager in Amazon Data Firehose.

Content encoding

Choose whether you want to enable content encoding to compress the body of the request. Amazon Data Firehose uses content encoding to compress the body of a request before sending it to the destination. When enabled, the content is compressed in the **GZIP** format.

Retry duration

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

After sending 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, Firehose starts the retry duration counter. It keeps retrying until the retry duration expires. After that, Firehose considers it a data delivery failure and backs up the data to your Amazon S3 bucket.

Every time that Firehose sends data to the HTTP endpoint, either during the initial attempt or after retrying, it restarts the acknowledgement timeout counter and waits for an acknowledgement from the HTTP endpoint.

Even if the retry duration expires, Firehose still waits for the acknowledgment until it receives it or the acknowledgement timeout period is reached. If the acknowledgment times out, 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 Firehose to retry sending data, set this value to 0.

Parameters - optional

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

Buffering hints

Amazon Data Firehose buffers incoming data before delivering it to the specified destination. The buffer hints include the buffer size and interval for your streams. The recommended buffer size for the destination varies according to the service provider.

Configure destination settings for Honeycomb

This section describes options for using **Honeycomb** for your destination. For more information about Honeycomb, see https://docs.honeycomb.io/getting-data-in/metrics/aws-cloudwatch-metrics/ .

- Provide values for the following fields:

  Honeycomb Kinesis endpoint

  Specify the URL for the HTTP endpoint in the following format: https://api.honeycomb.io/1/kinesis_events/{{dataset}}

  Authentication

  You can either choose to enter the API key directly or retrieve the secret from AWS Secrets Manager to access Honeycomb.

- **API key**

  Contact Honeycomb to obtain the API key that you need to enable data delivery to this endpoint from Firehose.

- **Secret**

  Select a secret from AWS Secrets Manager that contains the API key for Honeycomb. If you do not see your secret in the drop-down list, create one in AWS Secrets Manager. For more information, see Authenticate with AWS Secrets Manager in Amazon Data Firehose.

Content encoding

Amazon Data Firehose uses content encoding to compress the body of a request before sending it to the destination. Choose **GZIP** to enable content encoding of your request. This is the recommended option for the Honeycomb destination.

Retry duration

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

After sending data, Amazon 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, Amazon Data Firehose starts the retry duration counter. It keeps retrying until the retry duration expires. After that, Amazon Data Firehose considers it a data delivery failure and backs up the data to your Amazon S3 bucket.

Every time that Amazon 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, Amazon Data Firehose still waits for the acknowledgment until it receives it or the acknowledgement timeout period is reached. If the acknowledgment times out, Amazon 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 Amazon Data Firehose to retry sending data, set this value to 0.

Parameters - optional

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

Buffering hints

Amazon 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.

Configure destination settings for Coralogix

This section describes options for using **Coralogix** for your destination. For more information about Coralogix, see Get Started with Coralogix.

- Provide values for the following fields:

  HTTP endpoint URL

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

  - **Coralogix - US**
  - **Coralogix - SINGAPORE**
  - **Coralogix - IRELAND**
  - **Coralogix - INDIA**
  - **Coralogix - STOCKHOLM**

  Authentication

  You can either choose to enter the private key directly or retrieve the secret from AWS Secrets Manager to access Coralogix.

  - **Private key**

    Contact Coralogix to obtain the private key that you need to enable data delivery to this endpoint from Firehose.

  - **Secret**

Select a secret from AWS Secrets Manager that contains the private key for Coralogix. If you do not see your secret in the drop-down list, create one in AWS Secrets Manager. For more information, see Authenticate with AWS Secrets Manager in Amazon Data Firehose.

Content encoding

Amazon Data Firehose uses content encoding to compress the body of a request before sending it to the destination. Choose **GZIP** to enable content encoding of your request. This is the recommended option for the Coralogix destination.

Retry duration

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

After sending data, Amazon 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, Amazon Data Firehose starts the retry duration counter. It keeps retrying until the retry duration expires. After that, Amazon Data Firehose considers it a data delivery failure and backs up the data to your Amazon S3 bucket.

Every time that Amazon 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, Amazon Data Firehose still waits for the acknowledgment until it receives it or the acknowledgement timeout period is reached. If the acknowledgment times out, Amazon 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 Amazon Data Firehose to retry sending data, set this value to 0.

Parameters - optional

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

- applicationName: the environment where you are running Data Firehose
- subsystemName: the name of the Data Firehose integration
- computerName: the name of the Firehose stream in use

Buffering hints

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

Configure destination settings for Dynatrace (Classic cloud platform)

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

- Choose options to use Dynatrace (Classic cloud platform) as the destination for your Firehose stream.

  Ingestion type

  Choose whether you want to deliver **Metrics** or **Logs** (default) in Dynatrace for further analysis and processing.

  HTTP endpoint URL

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

  Authentication

  You can either choose to enter the API token directly or retrieve the secret from AWS Secrets Manager to access Dynatrace.

  - **API token**

    Generate the Dynatrace API token that you need to enable data delivery to this endpoint from Firehose. For more information, see Dynatrace API - Tokens and authentication.

  - **Secret**

    Select a secret from AWS Secrets Manager that contains the API token for Dynatrace. If you do not see your secret in the drop-down list, create one in AWS Secrets Manager. For more information, see Authenticate with AWS Secrets Manager in Amazon Data Firehose.

API URL

Provide the API URL of your Dynatrace environment.

Content encoding

Choose whether you want to enable content encoding to compress body of the request. Amazon Data Firehose uses content encoding to compress the body of a request before sending it to the destination. When enabled, the content it compressed in the **GZIP** format.

Retry duration

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

After sending 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, Firehose starts the retry duration counter. It keeps retrying until the retry duration expires. After that, Firehose considers it a data delivery failure and backs up the data to your Amazon S3 bucket.

Every time that Firehose sends data to the HTTP endpoint, either during the initial attempt or after retrying, it restarts the acknowledgement timeout counter and waits for an acknowledgement from the HTTP endpoint.

Even if the retry duration expires, Firehose still waits for the acknowledgment until it receives it or the acknowledgement timeout period is reached. If the acknowledgment times out, 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 Firehose to retry sending data, set this value to 0.

Parameters - optional

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

Buffering hints

Amazon Data Firehose buffers incoming data before delivering it to the specified destination. The buffer hints include the buffer size and interval for your streams. The recommended buffer size for the destination varies according to the service provider.

Configure destination settings for Dynatrace (New cloud platform)

This section describes options for using **Dynatrace (New cloud platform)** for your destination. For more information, see https://docs.dynatrace.com/docs/shortlink/aws-monitor-hub#new-cloud-platform-monitoring.

- Choose options to use Dynatrace (New cloud platform) as the destination for your Firehose stream.

  Ingestion type

  Choose **Logs** (default) to deliver log and event data to Dynatrace for further analysis and processing.

  HTTP endpoint URL

  Enter your Dynatrace environment ID URL (for example, `https://abc12345.apps.dynatrace.com`).

  Authentication

  You can either choose to enter the ingest platform token directly or retrieve the secret from AWS Secrets Manager to access Dynatrace.

  - **Use ingest platform token**

    Generate the Dynatrace ingest platform token that you need to enable data delivery to this endpoint from Firehose. For more information, see Dynatrace API - Ingest Platform Tokens and authentication.

  - **Secret**

    Select a secret from AWS Secrets Manager that contains the ingest platform token for Dynatrace. If you do not see your secret in the drop-down list, create one in AWS Secrets Manager. For more information, see Authenticate with AWS Secrets Manager in Amazon Data Firehose.

  Content encoding

  Choose whether you want to enable content encoding to compress the body of the request. Amazon Data Firehose uses content encoding to compress the body of a request before sending it to the destination. When enabled, the content is compressed in the **GZIP** format.

Retry duration

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

After sending 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, Firehose starts the retry duration counter. It keeps retrying until the retry duration expires. After that, Firehose considers it a data delivery failure and backs up the data to your Amazon S3 bucket.

Every time that Firehose sends data to the HTTP endpoint, either during the initial attempt or after retrying, it restarts the acknowledgement timeout counter and waits for an acknowledgement from the HTTP endpoint.

Even if the retry duration expires, Firehose still waits for the acknowledgment until it receives it or the acknowledgement timeout period is reached. If the acknowledgment times out, Firehose checks the retry counter. If time remains, Firehose retries and repeats this process until it receives an acknowledgment or the retry time expires.

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

Parameters - optional

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

Buffering hints

Amazon Data Firehose buffers incoming data before delivering it to the specified destination. The buffer hints include the buffer size and interval for your streams. The recommended buffer size for the destination varies according to the service provider.

Configure destination settings for LogicMonitor

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
```

Authentication

You can either choose to enter the API key directly or retrieve the secret from AWS Secrets Manager to access LogicMonitor.

- **API key**

  Contact LogicMonitor to obtain the API key that you need to enable data delivery to this endpoint from Firehose.

- **Secret**

  Select a secret from AWS Secrets Manager that contains the API key for LogicMonitor. If you do not see your secret in the drop-down list, create one in AWS Secrets Manager. For more information, see Authenticate with AWS Secrets Manager in Amazon Data Firehose.

Content encoding

Amazon 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 Amazon Data Firehose retries sending data to the selected HTTP endpoint.

After sending data, Amazon 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, Amazon Data Firehose starts the retry duration counter. It keeps retrying until the retry duration expires. After that, Amazon Data Firehose considers it a data delivery failure and backs up the data to your Amazon S3 bucket.

Every time that Amazon 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, Amazon Data Firehose still waits for the acknowledgment until it receives it or the acknowledgement timeout period is reached. If the acknowledgment times out, Amazon 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 Amazon Data Firehose to retry sending data, set this value to 0.

Parameters - optional

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

Buffering hints

Amazon 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.

Configure destination settings for Logz.io

This section describes options for using **Logz.io** for your destination. For more information, see https://logz.io/.

> **Note**
>
> In the Europe (Milan) region, Logz.io is not supported as an Amazon Data Firehose destination.

- Provide values for the following fields:

  HTTP endpoint URL

  Specify the URL for the HTTP endpoint in the following format. The URL must be an HTTPS URL.

  ```
  https://listener-aws-metrics-stream-<region>.logz.io/
  ```

For example

```
https://listener-aws-metrics-stream-us.logz.io/
```

Authentication

You can either choose to enter the shipping token directly or retrieve the secret from AWS Secrets Manager to access Logz.io.

- **Shipping token**

  Contact Logz.io to obtain the shipping token that you need to enable data delivery to this endpoint from Firehose.

- **Secret**

  Select a secret from AWS Secrets Manager that contains the shipping token for Logz.io. If you do not see your secret in the drop-down list, create one in AWS Secrets Manager. For more information, see Authenticate with AWS Secrets Manager in Amazon Data Firehose.

Retry duration

Specify how long Amazon Data Firehose retries sending data to Logz.io.

After sending data, Amazon 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, Amazon Data Firehose starts the retry duration counter. It keeps retrying until the retry duration expires. After that, Amazon Data Firehose considers it a data delivery failure and backs up the data to your Amazon S3 bucket.

Every time that Amazon 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, Amazon Data Firehose still waits for the acknowledgment until it receives it or the acknowledgement timeout period is reached. If the acknowledgment times out, Amazon 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 Amazon Data Firehose to retry sending data, set this value to 0.

Parameters - optional

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

Buffering hints

Amazon 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.

Configure destination settings for MongoDB Atlas

This section describes options for using **MongoDB Atlas** for your destination. For more information, see MongoDB Atlas on Amazon Web Services.

- Provide values for the following fields:

  API Gateway URL

  Specify the URL for the HTTP endpoint in the following format.

  ```
  https://xxxxx.execute-api.region.amazonaws.com/stage
  ```

  The URL must be an HTTPS URL.

  Authentication

  You can either choose to enter the API key directly or retrieve the secret from AWS Secrets Manager to access MongoDB Atlas.

  - **API key**

    Follow instructions in MongoDB Atlas on Amazon Web Services to obtain the `APIKeyValue` that you need to enable data delivery to this endpoint from Firehose.

  - **Secret**

    Select a secret from AWS Secrets Manager that contains the API Key for API Gateway that is backed by Lambda interacting with MongoDB Atlas. If you do not see your secret

in the drop-down list, create one in AWS Secrets Manager. For more information, see Authenticate with AWS Secrets Manager in Amazon Data Firehose.

Content encoding

Amazon 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 Amazon Data Firehose retries sending data to the selected third-party provider.

After sending data, Amazon 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, Amazon Data Firehose starts the retry duration counter. It keeps retrying until the retry duration expires. After that, Amazon Data Firehose considers it a data delivery failure and backs up the data to your Amazon S3 bucket.

Every time that Amazon 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, Amazon Data Firehose still waits for the acknowledgment until it receives it or the acknowledgement timeout period is reached. If the acknowledgment times out, Amazon 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 Amazon Data Firehose to retry sending data, set this value to 0.

Buffering hints

Amazon 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

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

Configure destination settings for New Relic

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 list.

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

  Authentication

  You can either choose to enter the API key directly or retrieve the secret from AWS Secrets Manager to access New Relic.

  - **API key**

    Enter your License Key, which is a 40-characters hexadecimal string, from your New Relic One Account settings. You need this API key to enable data delivery to this endpoint from Firehose.

  - **Secret**

    Select a secret from AWS Secrets Manager that contains the API key for New Relic. If you do not see your secret in the drop-down list, create one in AWS Secrets Manager. For more information, see Authenticate with AWS Secrets Manager in Amazon Data Firehose.

  Content encoding

  Amazon 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 Amazon Data Firehose retries sending data to the New Relic HTTP endpoint.

After sending data, Amazon 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, Amazon Data Firehose starts the retry duration counter. It keeps retrying until the retry duration expires. After that, Amazon Data Firehose considers it a data delivery failure and backs up the data to your Amazon S3 bucket.

Every time that Amazon 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, Amazon Data Firehose still waits for the acknowledgment until it receives it or the acknowledgement timeout period is reached. If the acknowledgment times out, Amazon 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 Amazon Data Firehose to retry sending data, set this value to 0.

Parameters - optional

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

Buffering hints

Amazon 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.

Configure destination settings for Snowflake

This section describes options for using Snowflake for your destination.

> **Note**
>
> Firehose integration with Snowflake is available in the US East (N. Virginia), US West (Oregon), Europe (Ireland), US East (Ohio), Asia Pacific (Tokyo), Europe (Frankfurt), Asia Pacific (Singapore), Asia Pacific (Seoul), and Asia Pacific (Sydney), Asia Pacific (Mumbai),

Europe (London), South America (Sao Paulo), Canada (Central), Europe (Paris), Asia Pacific (Osaka), Europe (Stockholm), Asia Pacific (Jakarta) AWS Regions.

Connection settings

- Provide values for the following fields:

  Snowflake account URL

  Specify a Snowflake account URL. For example: `xy12345.us-east-1.aws.snowflakecomputing.com`. Refer to Snowflake documentation on how to determine your account URL. Note that you mustn't specify the port number, whereas protocol (https://) is optional.

  Authentication

  You can either choose to enter the userlogin, private key, and passphrase manually or retrieve the secret from AWS Secrets Manager to access Snowflake.

  - **User login**

    Specify the Snowflake user to be used for loading data. Make sure the user has access to insert data into the Snowflake table.

  - **Private key**

    Specify the private key for authentication with Snowflake in PKCS8 format. Additionally, do not include PEM header and footer as part of the private key. If the key is split across multiple lines, remove the line breaks. Following is an example of what your private key must look like.

    ```
    -----BEGIN PRIVATE KEY-----
    KEY_CONTENT
    -----END PRIVATE KEY-----
    ```

    Remove the space in `KEY_CONTENT` and provide that to Firehose. No header/footer or newline characters are required.

  - **Passphrase**

Specify the passphrase to decrypt the encrypted private key. You can leave this field empty if the private key is not encrypted. For information, see Using Key Pair Authentication & Key Rotation.

- **Secret**

  Select a secret from AWS Secrets Manager that contains the credentials for Snowflake. If you do not see your secret in the drop-down list, create one in AWS Secrets Manager. For more information, see Authenticate with AWS Secrets Manager in Amazon Data Firehose.

Role configuration

Use default Snowflake role – If this option is selected, Firehose will not pass any role to Snowflake. Default role is assumed to load data. Please make sure the default role has permission to insert data in to Snowflake table.

Use custom Snowflake role – Enter a non-default Snowflake role to be assumed by Firehose when loading data into Snowflake table.

Snowflake connectivity

Options are **Private** or **Public**.

Private VPCE ID (optional)

The VPCE ID for Firehose to privately connect with Snowflake. The ID format is com.amazonaws.vpce.[region].vpce-svc-*[id]*. For more information, see AWS PrivateLink & Snowflake.

> **Note**
>
> If your Snowflake cluster is private link enabled, use `AwsVpceIds`-based network policy to allow Amazon Data Firehose data. Firehose doesn't require you to configure an IP-based network policy in your Snowflake account. Having an IP-based network policy enabled could interfere with Firehose connectivity. If you have an edge case that requires IP-based policy, contact the Firehose team by submitting a support ticket. For a list of the VPCE IDs that you can use, refer to the Accessing Snowflake in VPC.

Database configuration

- You must specify the following settings in order to use Snowflake as the destination for your Firehose stream.
  - Snowflake database – All data in Snowflake is maintained in databases.
  - Snowflake schema – Each database consists of one or more schemas, which are logical groupings of database objects, such as tables and views
  - Snowflake table – All data in Snowflake is stored in database tables, logically structured as collections of columns and rows.

Data loading options for your Snowflake table

- Use JSON keys as column names
- Use VARIANT columns
  - Content column name – Specify a column name in the table, where the raw data has to be loaded.
  - Metadata column name (optional) – Specify a column name in the table, where the metadata information has to be loaded. When you enable this field, you will see the following column in the Snowflake table based on the source type.

    For Direct PUT as source

    ```
    {
    "firehoseDeliveryStreamName" : "streamname",
    "IngestionTime" : "timestamp"
    }
    ```

    For Kinesis Data Stream as source

    ```
    {
    "kinesisStreamName" : "streamname",
    "kinesisShardId" : "Id",
    "kinesisPartitionKey" : "key",
    "kinesisSequenceNumber" : "1234",
    "subsequenceNumber" : "2334",
    "IngestionTime" : "timestamp"
    }
    ```

Retry duration

Time duration (0–7200 seconds) for Firehose to retry if either opening channel or delivery to Snowflake fails due to Snowflake service issues. Firehose retries with exponential backoff until the retry duration ends. If you set the retry duration to 0 (zero) seconds, Firehose does not retry upon Snowflake failures and routes data to Amazon S3 error bucket.

Buffer hints

Amazon 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. For more information, see Configure buffering hints.

Configure destination settings for Splunk

This section describes options for using Splunk for your destination.

> **Note**
>
> Firehose delivers data to Splunk clusters configured with Classic Load Balancer or an Application Load Balancer.

- Provide values for the following fields:

  Splunk cluster endpoint

  To determine the endpoint, see Configure Amazon Data 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 Data Firehose to send data to the Splunk platform in the Splunk documentation.

  Authentication

  You can either choose to enter the authentication token directly or retrieve the secret from AWS Secrets Manager to access Splunk.

- **Authentication token**

  To set up a Splunk endpoint that can receive data from Amazon Data Firehose, see Installation and configuration overview for the Splunk Add-on for Amazon Data Firehose in the Splunk documentation. Save the token that you get from Splunk when you set up the endpoint for this Firehose stream and add it here.

- **Secret**

  Select a secret from AWS Secrets Manager that contains the authentication token for Splunk. If you do not see your secret in the drop-down list, create one in AWS Secrets Manager. For more information, see Authenticate with AWS Secrets Manager in Amazon Data Firehose.

HEC acknowledgement timeout

Specify how long Amazon Data Firehose waits for the index acknowledgement from Splunk. If Splunk doesn't send the acknowledgment before the timeout is reached, Amazon Data Firehose considers it a data delivery failure. Amazon 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 Amazon Data Firehose retries sending data to Splunk.

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

Every time that Amazon 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, Amazon Data Firehose still waits for the acknowledgment until it receives it or the acknowledgement timeout period is reached. If the acknowledgment times out, Amazon 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 Amazon Data Firehose to retry sending data, set this value to 0.

Buffering hints

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

Configure destination settings for Splunk Observability Cloud

This section describes options for using **Splunk Observability Cloud** for your destination. For more information, see https://docs.splunk.com/observability/en/gdi/get-data-in/connect/aws/aws-apiconfig.html#connect-to-aws-using-the-splunk-observability-cloud-api.

- Provide values for the following fields:

  Cloud Ingest Endpoint URL

  You can find your Splunk Observability Cloud's Real-time Data Ingest URL in Profile > Organizations > Real-time Data Ingest Endpoint in Splunk Observability console.

  Authentication

  You can either choose to enter the access token directly or retrieve the secret from AWS Secrets Manager to access Splunk Observability Cloud.

  - **Access Token**

    Copy your Splunk Observability access token with INGEST authorization scope from **Access Tokens** under **Settings** in Splunk Observability console.

  - **Secret**

    Select a secret from AWS Secrets Manager that contains the access token for Splunk Observability Cloud. If you do not see your secret in the drop-down list, create one in AWS Secrets Manager. For more information, see Authenticate with AWS Secrets Manager in Amazon Data Firehose.

Content Encoding

Amazon 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 Amazon Data Firehose retries sending data to the selected HTTP endpoint.

After sending data, Amazon 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, Amazon Data Firehose starts the retry duration counter. It keeps retrying until the retry duration expires. After that, Amazon Data Firehose considers it a data delivery failure and backs up the data to your Amazon S3 bucket.

Every time that Amazon 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, Amazon Data Firehose still waits for the acknowledgment until it receives it or the acknowledgement timeout period is reached. If the acknowledgment times out, Amazon 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 Amazon Data Firehose to retry sending data, set this value to 0.

Parameters - optional

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

Buffering hints

Amazon 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.

Configure destination settings for Sumo Logic

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://deploymentname.sumologic.net/receiver/v1/kinesis/dataType/access token`. The URL must be an HTTPS URL.

  Content encoding

  Amazon 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 Amazon Data Firehose retries sending data to Sumo Logic.

  After sending data, Amazon 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, Amazon Data Firehose starts the retry duration counter. It keeps retrying until the retry duration expires. After that, Amazon Data Firehose considers it a data delivery failure and backs up the data to your Amazon S3 bucket.

  Every time that Amazon 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, Amazon Data Firehose still waits for the acknowledgment until it receives it or the acknowledgement timeout period is reached. If the acknowledgment times out, Amazon 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 Amazon Data Firehose to retry sending data, set this value to 0.

Parameters - optional

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

Buffering hints

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

Configure destination settings for Elastic

This section describes options for using **Elastic** for your destination.

- Provide values for the following fields:

  Elastic endpoint URL

  Specify the URL for the HTTP endpoint in the following format: `https://<cluster-id>.es.<region>.aws.elastic-cloud.com`. The URL must be an HTTPS URL.

  Authentication

  You can either choose to enter the API key directly or retrieve the secret from AWS Secrets Manager to access Elastic.

  - **API key**

    Contact Elastic to obtain the API key that you require to enable data delivery to their service from Firehose.

  - **Secret**

    Select a secret from AWS Secrets Manager that contains the API key for Elastic. If you do not see your secret in the drop-down list, create one in AWS Secrets Manager. For more information, see Authenticate with AWS Secrets Manager in Amazon Data Firehose.

Content encoding

Amazon Data Firehose uses content encoding to compress the body of a request before sending it to the destination. Choose **GZIP** (which is what selected by default) or **Disabled** to enable/disable content encoding of your request.

Retry duration

Specify how long Amazon Data Firehose retries sending data to Elastic.

After sending data, Amazon 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, Amazon Data Firehose starts the retry duration counter. It keeps retrying until the retry duration expires. After that, Amazon Data Firehose considers it a data delivery failure and backs up the data to your Amazon S3 bucket.

Every time that Amazon 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, Amazon Data Firehose still waits for the acknowledgment until it receives it or the acknowledgement timeout period is reached. If the acknowledgment times out, Amazon 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 Amazon Data Firehose to retry sending data, set this value to 0.

Parameters - optional

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

Buffering hints

Amazon Data Firehose buffers incoming data before delivering it to the specified destination. The recommended buffer size for the Elastic destination is 1 MiB.

Configure backup settings

Amazon Data Firehose uses Amazon S3 to backup all or failed only data that it attempts to deliver to your chosen destination.

> **Important**
>
> - Backup settings are only supported if the source for your Firehose stream is Direct PUT or Kinesis Data Streams.
> - Zero buffering feature is only available for the application destinations and is not available for Amazon S3 backup destination.

You can specify the S3 backup settings for your Firehose stream if you made one of the following choices.

- If you set Amazon S3 as the destination for your Firehose 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 Firehose stream.
- If you set Amazon Redshift as the destination for your Firehose 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 Firehose stream – Amazon OpenSearch Service, Datadog, Dynatrace, HTTP Endpoint, LogicMonitor, MongoDB Cloud, New Relic, Splunk, or Sumo Logic, Snowflake, Apache Iceberg Tables.

The following are the backup settings for your Firehose 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 Amazon Data Firehose backs up your data.
- S3 backup bucket prefix - this is the prefix where Amazon 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.

- Buffering hints, compression and encryption for backup - Amazon Data Firehose uses Amazon S3 to backup all or failed only data that it attempts to deliver to your chosen destination. Amazon 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 Amazon Data Firehose to the data delivery to Amazon S3. If data delivery to the destination falls behind data writing to the Firehose stream, Amazon Data Firehose raises the buffer size dynamically to catch up. This action helps ensure that all data is delivered to the destination.
- S3 compression - 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 Firehose stream with Amazon Redshift as the destination.
- S3 file extension format (optional) – Specify a file extension format for objects delivered to Amazon S3 destination bucket. If you enable this feature, specified file extension will override default file extensions appended by Data Format Conversion or S3 compression features such as .parquet or .gz. Make sure if you configured the right file extension when you use this feature with Data Format Conversion or S3 compression. File extension must start with a period (.) and can contain allowed characters: 0-9a-z!-_.*‘(). File extension cannot exceed 128 characters.
- Firehose supports Amazon S3 server-side encryption with AWS Key Management Service (SSE-KMS) for encrypting delivered data in Amazon S3. You can choose to use the default encryption type specified in the destination S3 bucket or to encrypt with a key from the list of AWS KMS keys that you own. If you encrypt the data with AWS KMS keys, you can use either the default AWS managed key (aws/s3) or a customer managed key. For more information, see Protecting Data Using Server-Side Encryption with AWS KMS-Managed Keys (SSE-KMS).

Configure buffering hints

Amazon Data Firehose buffers incoming streaming data in memory to a certain size (buffering size) and for a certain period of time (buffering interval) before delivering it to the specified destinations. You would use buffering hints when you want to deliver optimal sized files to Amazon S3 and get better performance from data processing applications or to adjust Firehose delivery rate to match destination speed.

You can configure the buffering size and the buffer interval while creating new Firehose streams or update the buffering size and the buffering interval on your existing Firehose streams. Buffering size is measured in MBs and buffering interval is measured in seconds. However, if you specify a

value for one of them, you must also provide a value for the other. The first buffer condition that is satisfied triggers Firehose to deliver the data. If you don't configure the buffering values, then the default values are used.

You can configure Firehose buffering hints through the AWS Management Console, AWS Command Line Interface, or AWS SDKs. For existing streams, you can reconfigure buffering hints with a value that suits your use cases using the **Edit** option in the console or using the UpdateDestination API. For new streams, you can configure buffering hints as part of new stream creation using the console or using the CreateDeliveryStream API. To adjust the buffering size, set `SizeInMBs` and `IntervalInSeconds` in the destination specific `DestinationConfiguration` parameter of the CreateDeliveryStream or UpdateDestination API.

> **Note**
>
> - Buffer hints are applied on a shard or partition level, while dynamic partitioning buffer hints are applied on stream or topic level.
> - To meet lower latencies of real-time use cases, you can use zero buffering interval hint. When you configure buffering interval as zero seconds, Firehose will not buffer data and will deliver data within a few seconds. Before you change buffering hints to a lower value, check with the vendor for recommended buffering hints of Firehose for their destinations.
> - Zero buffering feature is only available for the application destinations and is not available for Amazon S3 backup destination.
> - Zero buffering feature is not available for dynamic partitioning.
> - Firehose uses multi-part upload for S3 destination when you configure a buffer time interval less than 60 seconds to offer lower latencies. Due to multi-part upload for S3 destination, you will see some increase in S3 PUT API costs if you choose a buffer time interval less than 60 seconds.

For destination specific buffering hint ranges and default values, see the following table:

Destination	Buffering size in MB (default in parenthesis)	Buffering interval in seconds (default in parenthesis)
Amazon S3	1-128 (5)	0-900 (300)
Apache Iceberg Tables	1-128 (5)	0-900 (300)
Amazon Redshift	1-128 (5)	0-900 (300)
OpenSearch Serverless	1-100 (5)	0-900 (300)
OpenSearch	1-100 (5)	0-900 (300)
Splunk	1-5 (5)	0-60 (60)
Datadog	1-4 (4)	0-900 (60)
Coralogix	1-64 (6)	0-900 (60)
Dynatrace	1-64 (5)	0-900 (60)
Elastic	1	0-900 (60)
Honeycomb	1-64 (15)	0-900 (60)
HTTP endpoint	1-64 (5)	0-900 (60)
LogicMonitor	1-64 (5)	0-900 (60)
Logzio	1-64 (5)	0-900 (60)
mongoDB	1-16 (5)	0-900 (60)
newRelic	1-64 (5)	0-900 (60)
sumoLogic	1-64 (1)	0-900 (60)

Destination	Buffering size in MB (default in parenthesis)	Buffering interval in seconds (default in parenthesis)
Splunk Observability Cloud	1-64 (1)	0-900 (60)
Snowflake	1 - 128 (1)	0 - 900 (0)

Configure advanced settings

The following section contains details about the advanced settings for your Firehose stream.

- Server-side encryption - Amazon 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 - Amazon Data Firehose logs errors related to processing and delivery. Additionally, when data transformation is enabled, it can log Lambda invocations and send data delivery errors to CloudWatch Logs. For more information, see Monitor Amazon Data Firehose Using CloudWatch Logs.

  > **Important**
  >
  > While optional, enabling Amazon Data Firehose error logging during Firehose stream creation is strongly recommended. This practice ensures that you can access error details in case of record processing or delivery failures.

- Permissions - Amazon Data Firehose uses IAM roles for all the permissions that the Firehose stream needs. You can choose to create a new role where required permissions are assigned automatically, or choose an existing role created for Amazon Data Firehose. The role is used to grant 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?.

> **Note**
>
> The IAM role (including placeholders) is created based on the configuration you choose when creating a Firehose stream. If you make any changes to the Firehose stream source or destination, you must manually update the IAM role.

- Tags - You can add tags to organize your AWS resources, track costs, and control access.

  If you specify tags in the `CreateDeliveryStream` action, Amazon Data Firehose performs an additional authorization on the `firehose:TagDeliveryStream` action to verify if users have permissions to create tags. If you do not provide this permission, requests to create new Firehose streams with IAM resource tags will fail with an `AccessDeniedException` such as following.

  ```
  AccessDeniedException
  User: arn:aws:sts::x:assumed-role/x/x is not authorized to perform:
   firehose:TagDeliveryStream on resource: arn:aws:firehose:us-east-1:x:deliverystream/
  x with an explicit deny in an identity-based policy.
  ```

  The following example demonstrates a policy that allows users to create a Firehose stream and apply tags.

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

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

Testing Firehose stream with 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 Firehose stream. This enables you to test the configuration of your Firehose 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 Data Firehose charges apply when your Firehose 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.

Prerequisites

Before you begin, create a Firehose stream. For more information, see Tutorial: Create a Firehose stream from console.

Test with Amazon S3

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

To test a Firehose stream using Amazon S3

1. Open the Firehose console at https://eusc-de-east-1.console.amazonaws-eusc.eu/firehose/.
2. Choose an active Firehose stream. The Firehose stream must be in **Active** status before you can start sending data.
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 with Amazon Redshift

Use the following procedure to test your Firehose stream with Amazon Redshift as the destination.

To test a Firehose stream using Amazon Redshift

1. Your Firehose 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 Firehose console at https://eusc-de-east-1.console.amazonaws-eusc.eu/firehose/.
3. Choose an active Firehose stream. The Firehose stream must be in **Active** status before you can start sending data.
4. Edit the destination details for your Firehose 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 Firehose stream to point to another table.
9. (Optional) Delete the `firehose_test_table` table.

Test with OpenSearch Service

Use the following procedure to test your Firehose stream using Amazon OpenSearch Service as the destination.

To test a Firehose stream using OpenSearch Service

1. Open the Firehose console at https://eusc-de-east-1.console.amazonaws-eusc.eu/firehose/.
2. Choose an active Firehose stream. The Firehose stream must be in **Active** status before you can start sending data.
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 with Splunk

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

To test a Firehose stream using Splunk

1. Open the Firehose console at https://eusc-de-east-1.console.amazonaws-eusc.eu/firehose/.
2. Choose an active Firehose stream. The Firehose stream must be in **Active** status before you can start sending data.
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.

Test with Apache Iceberg Tables

Use the following procedure to test your Firehose stream with Apache Iceberg Tables as the destination.

To test a Firehose stream using Apache Iceberg Tables

1. Open the Firehose console at https://eusc-de-east-1.console.amazonaws-eusc.eu/firehose/.
2. Choose an active Firehose stream. The Firehose stream must be in **Active** status before you can start sending data.
3. Under **Test with demo data**, choose **Start sending demo data** to generate sample stock ticker data.
4. Follow the instructions on screen to verify that data is being delivered to your Apache Iceberg Tables. Note that it might take a few minutes for new objects to appear in your bucket, based on its buffering configuration.
5. If the test data doesn't appear in your Apache Iceberg Tables, check your Amazon S3 bucket for failed events.
6. When you finish testing, choose **Stop sending demo data** to stop incurring usage charges.

Send data to a Firehose stream

This section describes how you can use different data sources to send data to your Firehose stream. If you are new to Amazon Data Firehose, take some time to become familiar with the concepts and terminology presented in What is Amazon Data Firehose?.

> **Note**
>
> Some AWS services can only send messages and events to a Firehose stream that is in the same Region. If your Firehose 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 Firehose stream is in the same Region as your other services. For information on service endpoints for each Region, see Amazon Data Firehose endpoints.

You can send data to your Firehose stream from the following data sources.

Topics

- Configure Kinesis agent to send data
- Send data with AWS SDK
- Send CloudWatch Logs to Firehose
- Send CloudWatch Events to Firehose
- Configure AWS IoT to send data to Firehose

Configure Kinesis agent to send data

Amazon Kinesis agent is a standalone Java software application that serves as a reference implementation to show how you can collect and send data to Firehose. The agent continuously monitors a set of files and sends new data to your Firehose stream. The agent shows how you can handle file rotation, checkpointing, and retry upon failures. It shows how you can deliver your data in a reliable, timely, and simple manner. It also shows how you can emit CloudWatch metrics to better monitor and troubleshoot the streaming process. To learn more, awslabs/amazon-kinesis-agent.

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 Specify agent configuration settings).

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

Prerequisites

Before you start using Kinesis Agent, make sure you meet the following 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 Amazon Data Firehose PutRecordBatch operation for the agent to send data to your Firehose 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 Data Firehose, Monitor Kinesis Agent health, and Authentication and Access Control for Amazon CloudWatch.

Manage AWS credentials

Manage your AWS credentials using one of the following methods:

- Create a custom credentials provider. For details, see the section called “Create custom credential providers”.
- 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 “Specify agent configuration settings”).
- 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 Amazon Data Firehose service. Specify that role when you configure the agent (see assumeRoleARN and assumeRoleExternalId). 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.

Create 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 “Specify agent configuration settings”.

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*. If you have trouble connecting, see Troubleshooting Connecting to Your Instance in the *Amazon EC2 User Guide*.

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
  ```

You can download the latest agent with the following command

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

- **To set up the agent in a Docker container**

  Kinesis Agent can be run in a container as well via the amazonlinux container base. Use the following Dockerfile and then run `docker build`.

  ```
  FROM amazonlinux

  RUN yum install -y aws-kinesis-agent which findutils
  COPY agent.json /etc/aws-kinesis/agent.json

  CMD ["start-aws-kinesis-agent"]
  ```

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 Firehose 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 Firehose 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 Specify agent configuration settings.

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 Firehose stream. Agent activity is logged in `/var/log/aws-kinesis-agent/aws-kinesis-agent.log`.

Specify 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 Amazon 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`

Configuration Setting	Description
`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 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 Firehose 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 Firehose stream. Default: 0 (no aggregation)
`dataProcessingOptions`	The list of processing options applied to each parsed record before it is sent to the Firehose stream. The processing options are performed in the specified order. For more information, see Pre-process data with Agents.
`deliveryStream`	[Required] The name of the Firehose 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

Configuration Setting	Description
	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 Firehose 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 Firehose 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 Firehose stream. Value range: 1–500 Default: 500

Configuration Setting	Description
`minTimeBetweenFilePollsMillis`	The time 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 Amazon Data Firehose record size limit. (1,000 KB) Default: `'\n'` (newline)

Configure multiple file directories and 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 Firehose stream respectively. You can specify different endpoints for Kinesis Data Streams and Amazon Data Firehose so that your data stream and Firehose 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.

Pre-process data with Agents

The agent can pre-process the records parsed from monitored files before sending them to your Firehose 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"}
```

Use common Agent CLI commands

The following table provides a set of common use cases and corresponding commands for working with the AWS Kinesis agent.

Use case	Command
Automatically start the agent on system start up	`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`

Troubleshoot issues when sending from Kinesis Agent

This table provides troubleshooting information and solutions for common issues faced when using the Amazon Kinesis Agent.

Issue	Solution
Why does Kinesis Agent not work on Windows?	Kinesis Agent for Windows is different software than Kinesis Agent for Linux platforms.
Why is Kinesis Agent slowing down and/or `RecordSendErrors` increasing?	This is usually due to throttling from Kinesis. Check the `WriteProvisionedThroughputExceeded` metric for Kinesis Data Streams or the `ThrottledRecords` metric for Firehose streams. Any increase from 0 in these metrics indicates that the stream limits need to be increased. For more information, see Kinesis Data Stream limits and Firehose streams. Once you rule out throttling, see if the Kinesis Agent is configured to tail a large amount of small files. There

Issue	Solution
	is a delay when Kinesis Agent tails a new file, so Kinesis Agent should be tailing a small amount of larger files. Try consolidating your log files into larger files.
How to resolve the `java.lang.OutOfMemoryError` exceptions?	This happends when Kinesis Agent does not have enough memory to handle its current workload. Try increasing `JAVA_START_HEAP` and `JAVA_MAX_HEAP` in `/usr/bin/start-aws-kinesis-agent` and restarting the agent.
How to resolve the `IllegalStateException : connection pool shut down` exceptions?	Kinesis Agent does not have enough connections to handle its current workload. Try increasing `maxConnections` and `maxSendingThreads` in your general agent configuration settings at `/etc/aws-kinesis/agent.json`. The default value for these fields is 12 times the runtime processors available. See AgentConfiguration.java for more about advanced agent configurations settings.
How can I debug another issue with Kinesis Agent?	`DEBUG` level logs can be enabled in `/etc/aws-kinesis/log4j.xml`.
How should I configure Kinesis Agent?	The smaller the `maxBufferSizeBytes`, the more frequently Kinesis Agent will send data. This can be good as it decreases delivery time of records, but it also increases the requests per second to Kinesis.
Why is Kinesis Agent sending duplicate records?	This occurs due to a misconfiguration in file tailing. Make sure that each `fileFlow's filePattern` is only matching one file. This can also occur if the `logrotate` mode being used is in `copytruncate` mode. Try changing the mode to the default or create mode to avoid duplication. For more information on handling duplicate records, see Handling Duplicate Records.

Send data with AWS SDK

You can use the Amazon Data Firehose API to send data to a Firehose stream using the AWS SDK for Java, .NET, Node.js, Python, or Ruby. If you are new to Amazon Data Firehose, take some time to become familiar with the concepts and terminology presented in What is Amazon Data Firehose?. 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 Amazon Data Firehose API offers two operations for sending data to your Firehose stream: PutRecord and PutRecordBatch. `PutRecord()` sends one data record within one call and `PutRecordBatch()` can send multiple data records within one call.

Single write operations using PutRecord

Putting data requires only the Firehose stream name and a byte buffer (<=1000 KB). Because Amazon 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 Firehose 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 Firehose API Operations.

Batch write operations using PutRecordBatch

Putting data requires only the Firehose stream name and a list of records. Because Amazon 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 Firehose 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 Firehose API Operations.

Send CloudWatch Logs to Firehose

CloudWatch Logs events can be sent to Firehose using CloudWatch subscription filters. For more information, see Subscription filters with Amazon Data Firehose.

CloudWatch Logs events are sent to Firehose in compressed gzip format. If you want to deliver decompressed log events to Firehose destinations, you can use the decompression feature in Firehose to automatically decompress CloudWatch Logs.

> **Important**
>
> Currently, Firehose does not support the delivery of CloudWatch Logs to Amazon OpenSearch Service destination because Amazon CloudWatch combines multiple log events into one Firehose record and Amazon OpenSearch Service cannot accept multiple log events in one record. As an alternative, you can consider Using subscription filter for Amazon OpenSearch Service in CloudWatch Logs.

Decompress CloudWatch Logs

If you are using Firehose to deliver CloudWatch Logs and want to deliver decompressed data to your Firehose stream destination, use Firehose Data Format Conversion (Parquet, ORC) or Dynamic partitioning. You must enable decompression for your Firehose stream.

You can enable decompression using the AWS Management Console, AWS Command Line Interface or AWS SDKs.

> **Note**
>
> If you enable the decompression feature on a stream, use that stream exclusively for CloudWatch Logs subscriptions filters, and not for Vended Logs. If you enable the decompression feature on a stream that is used to ingest both CloudWatch Logs and Vended Logs, the Vended Logs ingestion to Firehose fails. This decompression feature is only for CloudWatch Logs.

Extract message after decompression of CloudWatch Logs

When you enable decompression, you have the option to also enable message extraction. When using message extraction, Firehose filters out all metadata, such as owner, loggroup, logstream, and others from the decompressed CloudWatch Logs records and delivers only the content inside the message fields. If you are delivering data to a Splunk destination, you must turn on message extraction for Splunk to parse the data. Following are sample outputs after decompression with and without message extraction.

Fig 1: Sample output after decompression without message extraction:


```
{
 "owner": "111111111111",
 "logGroup": "CloudTrail/logs",
 "logStream": "111111111111_CloudTrail/logs_us-east-1",
 "subscriptionFilters": [
 "Destination"
 ],
 "messageType": "DATA_MESSAGE",
 "logEvents": [
 {
 "id": "31953106606966983378809025079804211143289615424298221568",
 "timestamp": 1432826855000,
 "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":\"Root1\"}"
 },
 {
 "id": "31953106606966983378809025079804211143289615424298221569",
 "timestamp": 1432826855000,
 "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":\"Root2\"}"
 },
 {
 "id": "31953106606966983378809025079804211143289615424298221570",
```

```
 "timestamp": 1432826855000,
 "message": "{\"eventVersion\":\"1.03\",\"userIdentity\":{\"type\":\"Root3\"}"
 }
 ]
}
```

Fig 2: Sample output after decompression with message extraction:

```
{"eventVersion":"1.03","userIdentity":{"type":"Root1"}
{"eventVersion":"1.03","userIdentity":{"type":"Root2"}
{"eventVersion":"1.03","userIdentity":{"type":"Root3"}
```

Enable decompression on a new Firehose stream from console

To enable decompression on a new Firehose stream using the AWS Management Console

1. Sign in to the AWS Management Console and open the Kinesis console at https://eusc-de-east-1.console.amazonaws-eusc.eu/kinesis.
2. Choose **Amazon Data Firehose** in the navigation pane.
3. Choose **Create Firehose stream**.
4. Under **Choose source and destination**

   Source

   The source of your Firehose stream. Choose one of the following sources:

   - **Direct PUT** – Choose this option to create a Firehose stream that producer applications write to directly. For a list of AWS services and agents and open source services that are integrated with Direct PUT in Firehose, see this section.
   - **Kinesis stream:** Choose this option to configure a Firehose stream that uses a Kinesis data stream as a data source. You can then use Firehose to read data easily from an existing Kinesis data stream and load it into destinations. For more information, see Writing to Firehose Using Kinesis Data Streams

   Destination

   The destination of your Firehose stream. Choose one of the following:

   - Amazon S3
   - Splunk

5. Under **Firehose stream name**, enter a name for your stream.
6. (Optional) Under **Transform records**:
   - In the **Decompress source records from Amazon CloudWatch Logs** section, choose **Turn on decompression**.
   - If you want to use message extraction after decompression, choose **Turn on message extraction**.

Enable decompression on an existing Firehose stream

This section provides instructions for enabling decompression on existing Firehose streams. It covers two scenarios – streams with Lambda processing disabled and streams with Lambda processing already enabled. The following sections outline step-by-step procedures for each case, including the creation or modification of Lambda functions, updating Firehose settings, and monitoring CloudWatch metrics to ensure successful implementation of the built-in Firehose decompression feature.

Enabling decompression when Lambda processing is disabled

To enable decompression on an existing Firehose stream with Lambda processing disabled, you must first enable Lambda processing. This condition is only valid for existing streams. Following steps show how to enable decompression on existing streams that do not have Lambda processing enabled.

1. Create a Lambda function. You can either create a dummy record pass through or can use this blueprint to create a new Lambda function.
2. Update your current Firehose stream to enable Lambda processing and use the Lambda function that you created for processing.
3. Once you update the stream with new Lambda function, go back to Firehose console and enable decompression.
4. Disable the Lambda processing that you enabled in step 1. You can now delete the function that you created in step 1.

Enabling decompression when Lambda processing is enabled

If you already have a Firehose stream with a Lambda function, to perform decompression you can replace it with the Firehose decompression feature. Before you proceed, review your Lambda

function code to confirm that it only performs decompression or message extraction. The output of your Lambda function should look similar to the examples shown in Fig 1 or Fig 2. If the output looks similar, you can replace the Lambda function using the following steps.

1. Replace your current Lambda function with this blueprint. The new blueprint Lambda function automatically detects whether the incoming data is compressed or decompressed. It only performs decompression if its input data is compressed.
2. Turn on decompression using the built-in Firehose option for decompression.
3. Enable CloudWatch metrics for your Firehose stream if it's not already enabled. Monitor the metric `CloudWatchProcessorLambda_IncomingCompressedData` and wait until this metric changes to zero. This confirms that all input data sent to your Lambda function is decompressed and the Lambda function is no longer required.
4. Remove the Lambda data transformation because you no longer need it to decompress your stream.

Disable decompression on Firehose stream

To disable decompression on a data stream using the AWS Management Console

1. Sign in to the AWS Management Console and open the Kinesis console at https://eusc-de-east-1.console.amazonaws-eusc.eu/kinesis.
2. Choose **Amazon Data Firehose** in the navigation pane.
3. Choose the Firehose stream you wish to edit.
4. On **Firehose stream details** page, choose the **Configuration** tab.
5. In the **Transform and convert records** section, choose **Edit**.
6. Under **Decompress source records from Amazon CloudWatch Logs**, clear **Turn on decompression** and then choose **Save changes**.

Troubleshoot decompression in Firehose

The following table shows how Firehose handles errors during data decompression and processing, including delivering records to an error S3 bucket, logging errors, and emitting metrics. It also explains the error message returned for unauthorized data put operations.

Issue	Solution
What happens to the source data in case of an error during decompression?	If Amazon Data Firehose is not able to decompress the record, the record is delivered as is (in compressed format) to error S3 bucket you specified during Firehose stream creation time. Along with the record, the delivered object also includes error code and error message and these objects will be delivered to an S3 bucket prefix called `decompression-failed` . Firehose will continue to process other records after a failed decompression of a record.
What happens to the source data in case of an error in the processing pipeline after successful decompression?	If Amazon Data Firehose errors out in the processing steps after decompression like Dynamic Partitioning and Data Format Conversion, the record is delivered in compressed format to the error S3 bucket you specified during Firehose stream creation time. Along with the record, the delivered object also includes error code and error message.
How are you informed in case of an error or an exception?	In case of an error or an exception during decompression, if you configure CloudWatch Logs, Firehose will log error messages into CloudWatch Logs. Additionally, Firehose sends metrics to CloudWatch metrics that you can monitor. You can also optionally create alarms based on metrics emitted by Firehose.
What happens when `put` operations don't come from CloudWatch Logs?	When customer `puts` do not come from CloudWatch Logs, then the following error message is returned: `Put to Firehose failed for AccountId: <accountID>, FirehoseName: <firehosename> because the request is not originating from allowed source types.`
What metrics does Firehose emit for the decompression feature?	Firehose emits metrics for decompression of every record. You should select the period (1 min), statistic (sum), date range to get the number of `Decompres`

Issue	Solution
	`sedRecords` failed or succeeded or `Decompressedbytes` failed or succeeded. For more information, see CloudWatch Logs Decompression Metrics.

Send CloudWatch Events to Firehose

You can configure Amazon CloudWatch to send events to a Firehose 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 Firehose stream

1. Sign in to the AWS Management Console and open the CloudWatch console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudwatch/.
2. Choose **Create rule**.
3. On the **Step 1: Create rule** page, for **Targets**, choose **Add target**, and then choose **Firehose stream**.
4. Choose an existing **Firehose stream**.

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

Configure AWS IoT to send data to Firehose

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

To create an action that sends events to an existing Firehose 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 Firehose 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.

Transform source data in Amazon Data Firehose

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

Understand data transformation flow

When you enable Firehose data transformation, Firehose buffers incoming data. The buffering size hint ranges between 0.2 MB and 3MB. The default Lambda buffering size hint is 1 MB for all destinations, except Splunk and Snowflake. For Splunk and Snowflake, the default buffering hint is 256 KB. The Lambda buffering interval hint ranges between 0 and 900 seconds. The default Lambda buffering interval hint is sixty seconds for all destinations except Snowflake. For Snowflake, the default buffering hint interval is 30 seconds. To adjust the buffering size, set the ProcessingConfiguration parameter of the CreateDeliveryStream or UpdateDestination API with the ProcessorParameter called `BufferSizeInMBs` and `IntervalInSeconds`. Firehose then invokes the specified Lambda function synchronously with each buffered batch using the AWS Lambda synchronous invocation mode. The transformed data is sent from Lambda to Firehose. 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.

Lambda invocation duration

Amazon 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 Amazon Data Firehose does if such an error occurs, see the section called “Handle failure in data transformation”.

Required parameters for data transformation

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

For Kinesis Data Streams and Direct PUT

The following parameters are required for all transformed records from Lambda.

- `recordId` – The record ID is passed from Amazon 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`, Amazon Data Firehose considers it successfully processed. Otherwise, Amazon Data Firehose considers it unsuccessfully processed.
- `data` – The transformed data payload, after base64-encoding.

  Following is a sample Lambda result output:

```
 {
    "recordId": "<recordId from the Lambda input>",
    "result": "Ok",
    "data": "<Base64 encoded Transformed data>"
}
```

For Amazon MSK

The following parameters are required for all transformed records from Lambda.

- `recordId` – The record ID is passed from 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`, Firehose considers it successfully processed. Otherwise, Firehose considers it unsuccessfully processed.

- `KafkaRecordValue` – The transformed data payload, after base64-encoding.

  Following is a sample Lambda result output:

```
{
    "recordId": "<recordId from the Lambda input>",
    "result": "Ok",
    "kafkaRecordValue": "<Base64 encoded Transformed data>"
}
```

Supported Lambda blueprints

These blueprints demonstrate how you can create and use AWS Lambda functions to transform data in your Amazon Data Firehose data streams.

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://eusc-de-east-1.console.amazonaws-eusc.eu/lambda/.
2. Choose **Create function**, and then choose **Use a blueprint**.
3. In the **Blueprints** field, search for the keyword `firehose` to find the Amazon Data Firehose Lambda blueprints.

List of blueprints:

- **Process records sent to Amazon Data Firehose stream (Node.js, Python)**

  This blueprint shows a basic example of how to process data in your Firehose data stream using AWS Lambda.

  Latest release date: November, 2016.

  Release notes: none.

- **Process CloudWatch Logs sent to Firehose**

This blueprint is deprecated. Do not use this blueprint. It might incur high charges when the decompressed CloudWatch Logs data is more than 6MB (Lambda limit). For information on processing CloudWatch Logs sent to Firehose, see Writing to Firehose Using CloudWatch Logs.

- **Convert Amazon Data Firehose stream records in syslog format to JSON (Node.js)**

  This blueprint shows how you can convert input records in RFC3164 Syslog format to JSON.

  Latest release date: Nov, 2016.

  Release notes: none.

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.

Handle failure in data transformation

If your Lambda function invocation fails because of a network timeout or because you've reached the Lambda invocation limit, Amazon Data Firehose retries the invocation three times by default. If the invocation does not succeed, Amazon 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 Monitor Amazon Data Firehose Using CloudWatch Logs.

If the status of the data transformation of a record is `ProcessingFailed`, Amazon 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 a 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 Amazon Data Firehose.

`errorCode`

The HTTP error code returned by Lambda.

`errorMessage`

The error message returned by Lambda.

`attemptEndingTimestamp`

The time that Amazon Data Firehose stopped attempting Lambda invocations.

`rawData`

The base64-encoded record data.

`lambdaARN`

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

Back up source records

Amazon 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 Firehose stream. You cannot disable source record backup after you enable it.

Partition streaming data in Amazon Data Firehose

Dynamic partitioning enables you to continuously partition streaming data in 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. Firehose 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, 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

- Enable dynamic partitioning in Amazon Data Firehose
- Understand partitioning keys
- Use Amazon S3 bucket prefix to deliver data
- Apply dynamic partitioning to aggregated data
- Troubleshoot dynamic partitioning errors
- Buffer data for dynamic partitioning

Enable dynamic partitioning in Amazon Data Firehose

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

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

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

Once dynamic partitioning on an active Firehose 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, Firehose starts using the new keys and the new S3 prefix expressions.

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

Understand 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.

You can create partitioning keys with the following methods.

- **Inline parsing** – this method uses Firehose built-in support mechanism, a jq parser, for extracting the keys for partitioning from data records that are in JSON format. Currently, we only support `jq 1.6` version.

- **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.

Create 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.

The following sample data record shows how you can define partitioning keys for it with inline parsing. Note that the data should be encoded in Base64 format. You can also refer to the CLI example.

```
{
   "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, Firehose uses the right column above to evaluate the parameters based on the data in each record.

Create 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 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 Firehose stream processing Lambda function in Python that replays every read record from input to output and extracts partitioning 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'),
                          "day": 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 Firehose stream processing Lambda function in Go that replays every read record from input to output and extracts partitioning 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 DataFirehoseEventRecordData struct {
 CustomerId string `json:"customerId"`
}

func handleRequest(evnt events.DataFirehoseEvent) (events.DataFirehoseResponse, 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.DataFirehoseResponse

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

  var transformedRecord events.DataFirehoseResponseRecord
  transformedRecord.RecordID = record.RecordID
  transformedRecord.Result = events.DataFirehoseTransformedStateOk
  transformedRecord.Data = record.Data

  var metaData events.DataFirehoseResponseRecordMetadata
  var recordData DataFirehoseEventRecordData
  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)
}
```

Use Amazon S3 bucket prefix to deliver data

When you create a Firehose stream that uses Amazon S3 as the destination, you must specify an Amazon S3 bucket where 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 Firehose stream is optional. However, if you choose to enable dynamic partitioning, you must specify the S3 bucket prefixes to which Firehose delivers partitioned data.

In every Firehose stream where you enable dynamic partitioning, the S3 bucket prefix value consists of expressions based on the specified partitioning keys for that Firehose 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}/"
}
```

Firehose evaluates the above expression at runtime. It groups records that match the same evaluated S3 prefix expression into a single data set. Firehose then delivers each data set to the evaluated S3 prefix. The frequency of data set delivery to S3 is determined by the Firehose 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 Firehose stream and Custom Prefixes for Amazon S3 Objects.

Add a new line delimiter when delivering data to Amazon S3

You can enable **New Line Delimiter** 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 multi-record 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.

Apply dynamic partitioning to 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 Firehose stream and separating them.

Multi record deaggregation can either be of `JSON` type, meaning that the separation of records is based on consecutive JSON objects. Deaggregation can also be of the type `Delimited`, 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 delimiter `####`, you must specify it in the base-64 encoded format, which translates it to `IyMjIw==`. Record deaggregation by JSON or by delimiter is capped at 500 per record.

> **Note**
>
> When deaggregating JSON records, make sure that your input is still presented in the supported JSON format. JSON objects must be on a single line with no delimiter or newline-delimited (JSONL) only. An array of JSON objects is not a valid input.
> These are examples of correct input: `{"a":1}{"a":2}` and `{"a":1}\n{"a":2}`
> This is an example of the incorrect input: `[{"a":1}, {"a":2}]`

With aggregated data, when you enable dynamic partitioning, 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.

> **Important**
>
> When you use Data Transformation feature in Firehose, the deaggregation will be applied before the Data Transformation. Data coming into Firehose will be processed in the following order: Deaggregation → Data Transformation via Lambda → Partitioning Keys.

Troubleshoot dynamic partitioning errors

If Amazon Data Firehose is not able to parse data records in your Firehose 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 Firehose stream where you enable dynamic partitioning. The S3 error bucket prefix contains all the records that 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 Firehose stream if you want to enable dynamic partitioning for this Firehose stream. If you don't want to enable dynamic partitioning for a Firehose stream, specifying an S3 error bucket prefix is optional.

Buffer data for dynamic partitioning

Amazon 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 Firehose streams or update the buffer size and the buffer interval on your existing Firehose streams. A buffer size is measured in MBs and a buffer interval is measured in seconds.

> **Note**
>
> Zero buffering feature is not available for dynamic partitioning.

When dynamic partitioning is enabled, Firehose internally buffers records that belong to a given partition based on the configured buffering hint (size and time) before delivering these records to your Amazon S3 bucket. In order to deliver maximum size objects, Firehose uses multi-stage

buffering internally. Therefore, end-to-end delay of a batch of records might be 1.5 times of the configured buffering hint time. This affects the data freshness of a Firehose stream.

The active partition count is the total number of active partitions within the delivery buffer. For example, if the dynamic partitioning query constructs 3 partitions per second and you have a buffer hint configuration triggering delivery every 60 seconds, then on average you would have 180 active partitions. If Firehose cannot deliver the data in a partition to a destination, this partition is counted as active in the delivery buffer until it can be delivered.

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, Firehose creates an object with the buffer data and delivers it to the specified Amazon S3 prefix. After the object is delivered, the buffer for that partition and the partition itself are deleted and removed from the active partitions count.

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 a limit of 500 per Firehose stream, the rest of the records in the Firehose stream are delivered to the specified S3 error bucket prefix (activePartitionExceeded). You can use the Amazon Data Firehose Limits form to request an increase of this quota up to 2500 active partitions per given Firehose stream. If you need more partitions, you can create more Firehose streams and distribute the active partitions across them.

Convert input data format in Amazon Data Firehose

Amazon 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 *Transform source data*.

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

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

Deserializer

Amazon Data Firehose requires a deserializer to read the JSON of your input data. You can choose one of the following two types of deserializer.

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}{"b": 1}` and this is the incorrect input: `[{"a":1}, {"a":2}]`.

- Apache Hive JSON SerDe
- OpenX JSON SerDe

Choose 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 Amazon 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, Amazon 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.

Schema

Amazon Data Firehose requires a schema to determine how to interpret that data. Use AWS Glue to create a schema in the AWS Glue Data Catalog. Amazon Data Firehose then references that schema and uses it to interpret your input data. You can use the same schema to configure both Amazon Data Firehose and your analytics software. For more information, see Populating the AWS Glue Data Catalog in the *AWS Glue Developer Guide*.

> **Note**
>
> The schema created in AWS Glue Data Catalog should match the input data structure. Otherwise, the converted data will not contain attributes that are not specified in the schema. If you use nested JSON, use a STRUCT type in the schema that mirrors the structure of your JSON data. See this example for how to handle nested JSON with a STRUCT type.

> **Important**
>
> For data types that do not specify a size limit, there is a practical limit of 32 MBs for all of the data in a single row.
> If you specify length for `CHAR` or `VARCHAR`, Firehose truncates the strings at the specified length when it reads the input data. If the underlying data string is longer, it remains unchanged.

Serializer

Firehose requires a serializer to convert the data to the target columnar storage format (Parquet or ORC) – You can choose one of the following two types of serializers.

- ORC SerDe
- Parquet SerDe

Choose 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.

Enable record format conversion

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

following section shows how to enable record format conversion from console and Firehose API operations. For an example of how to set up record format conversion with CloudFormation, see AWS::DataFirehose::DeliveryStream.

Enable record format conversion from console

You can enable data format conversion on the console when you create or update a Firehose stream. With data format conversion enabled, Amazon S3 is the only destination that you can configure for the Firehose 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 Amazon 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 Firehose stream

1. Sign in to the AWS Management Console, and open the Amazon Data Firehose console at https://eusc-de-east-1.console.amazonaws-eusc.eu/firehose/.
2. Choose a Firehose stream to update, or create a new Firehose stream by following the steps in Tutorial: Create a Firehose stream from console.
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.

Manage record format conversion from Firehose API

If you want Amazon 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 Amazon 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.

Handling errors for data format conversion

When Amazon 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, Amazon Data Firehose retries it forever, blocking further delivery. For each failed record, Amazon Data Firehose writes a JSON document with the following schema:

```
{
  "attemptsMade": long,
  "arrivalTimestamp": long,
  "ErrorCode": string,
  "ErrorMessage": string,
  "attemptEndingTimestamp": long,
  "rawData": string,
  "sequenceNumber": string,
  "subSequenceNumber": long,
  "dataCatalogTable": {
    "catalogId": string,
    "databaseName": string,
    "tableName": string,
    "region": string,
    "versionId": string,
    "catalogArn": string
  }
}
```

Understand data delivery in Amazon Data Firehose

When you send data to your Firehose stream, it's automatically delivered to your chosen destination. The following table explains data delivery to different destinations.

Destination	Details
Amazon S3	For data delivery to Amazon S3, Firehose concatenates multiple incoming records based on the buffering configuration of your Firehose stream. It then delivers the records to Amazon S3 as an Amazon S3 object. By default, Firehose concatenates data without any delimiters. If you want to have new line delimiters between records, you can add new line delimiters by enabling the feature in the Firehose console configuration or API parameter. Data delivery between Firehose and Amazon S3 destination is encrypted with TLS (HTTPS).
Amazon Redshift	For data delivery to Amazon Redshift, Firehose first delivers incoming data to your S3 bucket in the format described earlier. Firehose then issues an Amazon Redshift **COPY** command to load the data from your S3 bucket to your Amazon Redshift provisioned cluster or Amazon Redshift Serverless workgroup. Ensure that after Amazon Data Firehose concatenates multiple incoming records to an Amazon S3 object, the Amazon S3 object can be copied to your Amazon Redshift provisioned cluster or Amazon Redshift Serverles s workgroup. For more information, see Amazon Redshift COPY Command Data Format Parameters.
OpenSearch Service and OpenSearch Serverless	For data delivery to OpenSearch Service and OpenSearch Serverles s, Amazon Data Firehose buffers incoming records based on the buffering configuration of your Firehose stream. It then generates an OpenSearch Service or OpenSearch Serverless bulk request to index multiple records to your OpenSearch Service cluster or OpenSearch Serverless collection. Make sure that your record is UTF-8 encoded and flattened to a single-line JSON object before you send it to Amazon Data Firehose. Also, the `rest.acti on.multi.allow_explicit_index` option for your

Destination	Details
	OpenSearch Service cluster must be set to true (default) to take bulk requests with an explicit index that is set per record. For more information, see OpenSearch Service Configure Advanced Options in the *Amazon OpenSearch Service Developer Guide*.
Splunk	For data delivery to Splunk, Amazon 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. To redrive the data that was delivered to S3 error bucket (S3 backup) back to Splunk, follow the steps mentioned in the Splunk documentation.
HTTP endpoint	For data delivery 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 integrati on 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.
Snowflake	For data delivery to Snowflake, Amazon Data Firehose internall y buffers data for one second and uses Snowflake streaming API operations to insert data to Snowflake. By default, records that you insert are flushed and committed to the Snowflake table every second. After you make the insert call, Firehose emits a CloudWatch metric that measures how long it took for the data to be committed to Snowflake. Firehose currently supports only single JSON item as record payload and doesn’t support JSON arrays. Make sure that your input payload is a valid JSON object and is well formed without any extra double quotes, quotes, or escape characters.

Each Firehose destination has its own data delivery frequency. For more information, see Configure buffering hints.

Duplicate records

Amazon Data Firehose uses at-least-once semantics for data delivery. In some circumstances, such as when data delivery times out, delivery retries by Amazon Data Firehose might introduce duplicates if the original data-delivery request eventually goes through. This applies to all destination types that Amazon Data Firehose supports, except for Amazon S3 destinations, Apache Iceberg Tables, and Snowflake destinations.

Topics

- Understand delivery across AWS accounts and regions
- Understand HTTP endpoint delivery request and response specifications
- Handle data delivery failures
- Configure Amazon S3 object name format
- Configure index rotation for OpenSearch Service
- Pause and resume data delivery

Understand delivery across AWS accounts and regions

Amazon Data Firehose supports data delivery to HTTP endpoint destinations across AWS accounts. The Firehose stream and the HTTP endpoint that you choose as your destination can belong to different AWS accounts.

Amazon Data Firehose also supports data delivery to HTTP endpoint destinations across AWS regions. You can deliver data from a Firehose stream in one AWS region to an HTTP endpoint in another AWS region. You can also delivery data from a Firehose 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.

Understand HTTP endpoint delivery request and response specifications

For Amazon Data Firehose to successfully deliver data to custom HTTP endpoints, these endpoints must accept requests and send responses using certain Amazon Data Firehose request and response formats. This section describes the format specifications of the HTTP requests that the Amazon Data Firehose service sends to custom HTTP endpoints, as well as the format

specifications of the HTTP responses that the Amazon Data Firehose service expects. HTTP endpoints have 3 minutes to respond to a request before Amazon Data Firehose times out that request. Amazon Data Firehose treats responses that do not adhere to the proper format as delivery failures.

Request format

Path and URL Parameters

These are configured directly by you as part of a single URL field. Amazon 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 Firehose 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 Firehose 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. Amazon Data Firehose restricts the size of the access key to 4096 bytes. Amazon Data Firehose does not attempt to interpret this key in any way. The configured key is copied verbatim into the value of this header. However, if you use Secrets Manager to configure the key, then the secret must follow a specific JSON object format: `{"api_key": "..."}`.

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. Amazon Data Firehose does not modify 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 Firehose 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 Firehose 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 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 Amazon 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 Amazon 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(120, 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 Firehose stream. Amazon 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 Firehose 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": "01234567890123456789012345678901234567890123 45",
          "timestamp": 1510109208016,
          "message": "log message 1"
        },
        {
          "id": "01234567890123456789012345678901234567890123 45",
          "timestamp": 1510109208017,
          "message": "log message 2"
        }
      ]
```

```
      }
    }
  ]
}
```

Handle data delivery failures

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

When you setup a Firehose stream, for many destinations such as OpenSearch, Splunk, and HTTP endpoints, you also setup an S3 bucket where data that fails to be delivered can be backed up. For more information about how Firehose backs up data in case of failed deliveries, see the relevant destination sections on this page. For more information about how to grant access to S3 buckets where data that fails to be delivered can be backed up, see Grant Firehose Access to an Amazon S3 Destination. When Firehose (a) fails to deliver data to the stream destination, and (b) fails to write data to the backup S3 bucket for failed deliveries, it effectively pauses stream delivery until such time that data can either be delivered to the destination or written to the backup S3 location.

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 Amazon Data Firehose assumes might not have access to the bucket, the network failed, or similar events. Under these conditions, Amazon Data Firehose keeps retrying for up to 24 hours until the delivery succeeds. The maximum data storage time of Amazon Data Firehose is 24 hours. If data delivery fails for more than 24 hours, your data is lost.

Data delivery to your S3 bucket can fail for various reasons, such as:

- The bucket no longer exists.
- The IAM role assumed by Amazon Data Firehose lacks access to the bucket.
- Network issues.
- S3 errors, such as HTTP 500s or other API failures.

In these cases, Amazon Data Firehose will retry delivery:

- **DirectPut sources:** Retries continue for up to 24 hours.

- **Kinesis Data Streams or Amazon MSK sources:** Retries continue indefinitely, up to the retention policy defined on the stream.

Amazon Data Firehose delivers failed records to a S3 error bucket only when Lambda processing or parquet conversion fails. Other failure scenarios will result in continuous retry attempts to S3 until the retention period is reached. When Firehose successfully delivers records to S3, it creates an S3 object file, and in cases of partial record failures, it automatically retries delivery and updates the same S3 object file with the successfully processed records.

Amazon Redshift

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

Data delivery to your Amazon Redshift provisioned cluster or Amazon Redshift Serverless workgroup might fail for several reasons. For example, you might have an incorrect cluster configuration of your Firehose stream, a cluster or workgroup under maintenance, or a network failure. Under these conditions, Amazon 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 and OpenSearch Serverless

For the OpenSearch Service and OpenSearch Serverless destination, you can specify a retry duration (0–7200 seconds) during Firehose stream creation.

Data delivery to your OpenSearch Service cluster or OpenSearch Serverless collection might fail for several reasons. For example, you might have an incorrect OpenSearch Service cluster or OpenSearch Serverless collection configuration of your Firehose stream, an OpenSearch Service cluster or OpenSearch Serverless collection under maintenance, a network failure, or similar events. Under these conditions, Amazon 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 `AmazonOpenSearchService_failed/` folder, which you can use for manual backfill.

For OpenSearch Service, 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 OpenSearch Service)",
    "errorMessage": "(error message returned by OpenSearch Service)",
    "attemptEndingTimestamp": "(the time when Firehose stopped attempting index
 request)",
    "esDocumentId": "(intended OpenSearch Service document ID)",
    "esIndexName": "(intended OpenSearch Service index name)",
    "esTypeName": "(intended OpenSearch Service type name)",
    "rawData": "(base64-encoded document data)"
}
```

For OpenSearch Serverless, 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 OpenSearch Serverless)",
    "errorMessage": "(error message returned by OpenSearch Serverless)",
    "attemptEndingTimestamp": "(the time when Firehose stopped attempting index
 request)",
    "osDocumentId": "(intended OpenSearch Serverless document ID)",
    "osIndexName": "(intended OpenSearch Serverless index name)",
    "rawData": "(base64-encoded document data)"
}
```

Splunk

When Amazon 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, Amazon Data Firehose starts the retry duration counter. It keeps retrying until the retry duration expires. After that, Amazon Data Firehose considers it a data delivery failure and backs up the data to your Amazon S3 bucket.

Every time Amazon 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, Amazon Data Firehose still waits for the acknowledgment until it receives it or the acknowledgement timeout is reached. If the acknowledgment times out, Amazon 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. Amazon Data Firehose backs up in Amazon
 S3 data for which the acknowledgement timeout expired.",
  "attemptEndingTimestamp": 13626284715507,
  "rawData":
 "MiAyNTE2MjAyNzIyMDkgZW5pLTA1ZjMyMmQ1IDIxOC45Mi4xODguMjE0IDE3Mi4xNi4xLjE2NyAyNTIzMyAxNDMzIDYgM
  "EventId": "49577193928114147339600778471082492393164139877200035842.0"
}
```

HTTP endpoint destination

When Amazon 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, Amazon Data Firehose starts the retry duration counter. It keeps retrying until the retry duration expires. After that, Amazon Data Firehose considers it a data delivery failure and backs up the data to your Amazon S3 bucket.

Every time Amazon 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, Amazon Data Firehose still waits for the response until it receives it or the response timeout is reached. If the response times out, Amazon 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"
}
```

Snowflake

For Snowflake destination, when you create a Firehose stream, you can specify an optional retry duration (0-7200 seconds). The default value for retry duration is 60 seconds.

Data delivery to your Snowflake table might fail for several reasons like an incorrect Snowflake destination configuration, Snowflake outage, a network failure, etc. The retry policy doesn’t apply to non-retriable errors. For example, if Snowflake rejects your JSON payload because it had an extra column that's missing in the table, Firehose doesn’t attempt to deliver it again. Instead, it creates a back up for all the insert failures due to JSON payload issues to your S3 error bucket.

Similarly, if delivery fails due to an incorrect role, table, or database, Firehose doesn’t retry and writes the data to your S3 bucket. Retry duration only applies to failure due to a Snowflake service issue, transient network glitches, etc. Under these conditions, Firehose retries for the specified time duration before delivering them to S3. The failed records are delivered in snowflake-failed/ folder, which you can use for manual backfill.

The following is an example JSON for each record that you deliver to S3.

```
{
    "attemptsMade": 3,
    "arrivalTimestamp": 1594265943615,
    "errorCode": "Snowflake.InvalidColumns",
    "errorMessage": "Snowpipe Streaming does not support columns of type AUTOINCREMENT,
 IDENTITY, GEO, or columns with a default value or collation",
    "attemptEndingTimestamp": 1712937865543,
```

```
    "rawData": "c2FtcGxlIHJhdyBkYXRh"
}
```

Configure Amazon S3 object name format

When Firehose delivers data to Amazon S3, S3 object key name follows the format *<evaluated prefix><suffix>*, where the suffix has the format *<Firehose stream name>-<Firehose stream version>-<year>-<month>-<day>-<hour>-<minute>-<second>-<uuid><file extension> <Firehose stream version>* begins with 1 and increases by 1 for every configuration change of Firehose stream. You can change Firehose stream configurations (for example, the name of the S3 bucket, buffering hints, compression, and encryption). You can do so by using the Firehose console or the UpdateDestination API operation.

For *<evaluated prefix>*, Firehose adds a default time prefix in the format `YYYY/MM/dd/HH`. 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 that includes expressions that are evaluated at runtime. For information about how to specify a custom prefix, see Custom Prefixes for Amazon Simple Storage Service Objects.

By default, the time zone used for time prefix and suffix is in UTC, but you can change it to a time zone that you prefer. For example, to use Japan Standard Time instead of UTC, you can configure the time zone to Asia/Tokyo in the AWS Management Console or in API parameter setting (CustomTimeZone). The following list contains time zones that Firehose supports for S3 prefix configuration.

Supported time zones

Following is a list of time zones that Firehose supports for S3 prefix configuration.

Africa

```
Africa/Abidjan
Africa/Accra
Africa/Addis_Ababa
Africa/Algiers
Africa/Asmera
Africa/Bangui
Africa/Banjul
Africa/Bissau
Africa/Blantyre
```

```
Africa/Bujumbura
Africa/Cairo
Africa/Casablanca
Africa/Conakry
Africa/Dakar
Africa/Dar_es_Salaam
Africa/Djibouti
Africa/Douala
Africa/Freetown
Africa/Gaborone
Africa/Harare
Africa/Johannesburg
Africa/Kampala
Africa/Khartoum
Africa/Kigali
Africa/Kinshasa
Africa/Lagos
Africa/Libreville
Africa/Lome
Africa/Luanda
Africa/Lubumbashi
Africa/Lusaka
Africa/Malabo
Africa/Maputo
Africa/Maseru
Africa/Mbabane
Africa/Mogadishu
Africa/Monrovia
Africa/Nairobi
Africa/Ndjamena
Africa/Niamey
Africa/Nouakchott
Africa/Ouagadougou
Africa/Porto-Novo
Africa/Sao_Tome
Africa/Timbuktu
Africa/Tripoli
Africa/Tunis
Africa/Windhoek
```

America

```
America/Adak
```

```
America/Anchorage
America/Anguilla
America/Antigua
America/Aruba
America/Asuncion
America/Barbados
America/Belize
America/Bogota
America/Buenos_Aires
America/Caracas
America/Cayenne
America/Cayman
America/Chicago
America/Costa_Rica
America/Cuiaba
America/Curacao
America/Dawson_Creek
America/Denver
America/Dominica
America/Edmonton
America/El_Salvador
America/Fortaleza
America/Godthab
America/Grand_Turk
America/Grenada
America/Guadeloupe
America/Guatemala
America/Guayaquil
America/Guyana
America/Halifax
America/Havana
America/Indianapolis
America/Jamaica
America/La_Paz
America/Lima
America/Los_Angeles
America/Managua
America/Manaus
America/Martinique
America/Mazatlan
America/Mexico_City
America/Miquelon
America/Montevideo
America/Montreal
```

```
America/Montserrat
America/Nassau
America/New_York
America/Noronha
America/Panama
America/Paramaribo
America/Phoenix
America/Port_of_Spain
America/Port-au-Prince
America/Porto_Acre
America/Puerto_Rico
America/Regina
America/Rio_Branco
America/Santiago
America/Santo_Domingo
America/Sao_Paulo
America/Scoresbysund
America/St_Johns
America/St_Kitts
America/St_Lucia
America/St_Thomas
America/St_Vincent
America/Tegucigalpa
America/Thule
America/Tijuana
America/Tortola
America/Vancouver
America/Winnipeg
```

Antarctica

```
Antarctica/Casey
Antarctica/DumontDUrville
Antarctica/Mawson
Antarctica/McMurdo
Antarctica/Palmer
```

Asia

```
Asia/Aden
Asia/Almaty
Asia/Amman
Asia/Anadyr
```

```
Asia/Aqtau
Asia/Aqtobe
Asia/Ashgabat
Asia/Ashkhabad
Asia/Baghdad
Asia/Bahrain
Asia/Baku
Asia/Bangkok
Asia/Beirut
Asia/Bishkek
Asia/Brunei
Asia/Calcutta
Asia/Colombo
Asia/Dacca
Asia/Damascus
Asia/Dhaka
Asia/Dubai
Asia/Dushanbe
Asia/Hong_Kong
Asia/Irkutsk
Asia/Jakarta
Asia/Jayapura
Asia/Jerusalem
Asia/Kabul
Asia/Kamchatka
Asia/Karachi
Asia/Katmandu
Asia/Krasnoyarsk
Asia/Kuala_Lumpur
Asia/Kuwait
Asia/Macao
Asia/Magadan
Asia/Manila
Asia/Muscat
Asia/Nicosia
Asia/Novosibirsk
Asia/Phnom_Penh
Asia/Pyongyang
Asia/Qatar
Asia/Rangoon
Asia/Riyadh
Asia/Saigon
Asia/Seoul
Asia/Shanghai
```

```
Asia/Singapore
Asia/Taipei
Asia/Tashkent
Asia/Tbilisi
Asia/Tehran
Asia/Thimbu
Asia/Thimphu
Asia/Tokyo
Asia/Ujung_Pandang
Asia/Ulaanbaatar
Asia/Ulan_Bator
Asia/Vientiane
Asia/Vladivostok
Asia/Yakutsk
Asia/Yekaterinburg
Asia/Yerevan
```

Atlantic

```
Atlantic/Azores
Atlantic/Bermuda
Atlantic/Canary
Atlantic/Cape_Verde
Atlantic/Faeroe
Atlantic/Jan_Mayen
Atlantic/Reykjavik
Atlantic/South_Georgia
Atlantic/St_Helena
Atlantic/Stanley
```

Australia

```
Australia/Adelaide
Australia/Brisbane
Australia/Broken_Hill
Australia/Darwin
Australia/Hobart
Australia/Lord_Howe
Australia/Perth
Australia/Sydney
```

Europe

```
Europe/Amsterdam
Europe/Andorra
Europe/Athens
Europe/Belgrade
Europe/Berlin
Europe/Brussels
Europe/Bucharest
Europe/Budapest
Europe/Chisinau
Europe/Copenhagen
Europe/Dublin
Europe/Gibraltar
Europe/Helsinki
Europe/Istanbul
Europe/Kaliningrad
Europe/Kiev
Europe/Lisbon
Europe/London
Europe/Luxembourg
Europe/Madrid
Europe/Malta
Europe/Minsk
Europe/Monaco
Europe/Moscow
Europe/Oslo
Europe/Paris
Europe/Prague
Europe/Riga
Europe/Rome
Europe/Samara
Europe/Simferopol
Europe/Sofia
Europe/Stockholm
Europe/Tallinn
Europe/Tirane
Europe/Vaduz
Europe/Vienna
Europe/Vilnius
Europe/Warsaw
Europe/Zurich
```

Indian

```
Indian/Antananarivo
Indian/Chagos
Indian/Christmas
Indian/Cocos
Indian/Comoro
Indian/Kerguelen
Indian/Mahe
Indian/Maldives
Indian/Mauritius
Indian/Mayotte
Indian/Reunion
```

Pacific

```
Pacific/Apia
Pacific/Auckland
Pacific/Chatham
Pacific/Easter
Pacific/Efate
Pacific/Enderbury
Pacific/Fakaofo
Pacific/Fiji
Pacific/Funafuti
Pacific/Galapagos
Pacific/Gambier
Pacific/Guadalcanal
Pacific/Guam
Pacific/Honolulu
Pacific/Kiritimati
Pacific/Kosrae
Pacific/Majuro
Pacific/Marquesas
Pacific/Nauru
Pacific/Niue
Pacific/Norfolk
Pacific/Noumea
Pacific/Pago_Pago
Pacific/Palau
Pacific/Pitcairn
Pacific/Ponape
Pacific/Port_Moresby
```

```
Pacific/Rarotonga
Pacific/Saipan
Pacific/Tahiti
Pacific/Tarawa
Pacific/Tongatapu
Pacific/Truk
Pacific/Wake
Pacific/Wallis
```

You cannot change the suffix field except *<file extension>*. When you enable data format conversion or compression, Firehose will append a file extension based on the configuration. The following table explains the default file extension appended by Firehose:

Configuration	File extension
Data Format Conversion: Parquet	.parquet
Data Format Conversion: ORC	.orc
Compression: Gzip	.gz
Compression: Zip	.zip
Compression: Snappy	.snappy
Compression: Hadoop-Snappy	.hsnappy

You can also specify a file extension that you prefer in the Firehose console or API. File extension must start with a period (.) and can contain allowed characters: 0-9a-z!-_.*'(). File extension cannot exceed 128 characters.

> **Note**
>
> When you specify a file extension, it will override the default file extension that Firehose adds when data format conversion or compression is enabled.

Understand custom prefixes for Amazon S3 objects

Objects delivered to Amazon S3 follow the name format of <evaluated prefix><suffix>. You can specify your custom prefix that includes expressions that are evaluated at runtime. Custom prefix you specify will override the default prefix of `yyyy/MM/dd/HH`.

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 Data FirehoseDeveloper Guide*.

`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, Firehose uses the approximate arrival timestamp of the oldest record that's contained in the Amazon S3 object being written.

By default, timestamp is in UTC. But, you can specify a time zone that you prefer. For example, you can configure the time zone to Asia/Tokyo in the AWS Management Console or in API parameter setting (CustomTimeZone) if you want to use Japan Standard Time instead of UTC. To see the list of supported time zones, see Amazon S3 Object Name Format.

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

`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 Firehose stream, and the reason of failure: {processing-failed, AmazonOpenSearchService-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..	`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.
`random-string`	Evaluates to a random string of 11 characters. If you use it more than once in the same expressio	`myPrefix/!{firehose:random-string}/`	`myPrefix/046b6c7f-0b/`	You can use it with both prefix types. You can place it at the beginning

Conversion	Description	Example input	Example output	Notes
	n, every instance evaluates to a new random string.			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 Firehose 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, 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, 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, Firehose uses the `Prefix` and `ErrorOutputPrefix` in the destination configuration for successful records and failed records, respectively. If you specify a prefix using expressions, you must specify the error prefix including `!{firehose:error-output-type}`.
- When you use `ExtendedS3DestinationConfiguration` with the AWS CLI, the API, or CloudFormation, if you specify a `S3BackupConfiguration`, 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	`Prefix`: `2018/08/27/10`

Input	Evaluated prefix (at 10:30 AM UTC on Aug 27, 2018)
`ErrorOutputPrefix : myFirehoseFailures/!{firehose:error-output-type}/`	`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`

`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/`

Configure index rotation for OpenSearch Service

For the OpenSearch Service 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, Amazon 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 OpenSearch Service 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

Pause and resume data delivery

After you setup a Firehose stream, data available in the stream source is continuously delivered to the destination. If you encounter situations where your stream destination is temporarily

unavailable (for example, during planned maintenance operations), you may want to temporarily pause data delivery, and resume when the destination becomes available again.

> **Important**
>
> When you use the approach described below to pause and resume a stream, after you resume the stream, you will see that few records get delivered to the error bucket in Amazon S3 while the rest of the stream continues to get delivered to the destination. This is a known limitation of the approach, and it occurs because a small number of records that could not be previously delivered to the destination after multiple retries are tracked as failed.

Pause a Firehose stream

To pause stream delivery in Firehose, first remove permissions for Firehose to write to the S3 backup location for failed deliveries. For example, if you want to pause the Firehose stream with an OpenSearch destination, you can do this by updating permissions. For more information, see Grant Firehose Access to a Public OpenSearch Service Destination.

Remove the `"Effect": "Allow"` permission for the action `s3:PutObject`, and explicitly add a statement that applies `Effect": "Deny"` permission on the action `s3:PutObject` for the S3 bucket used for backing up failed deliveries. Next, turn off the stream destination (for example, turning off the destination OpenSearch domain), or remove permissions for Firehose to write to the destination. To update permissions for other destinations, check the section for your destination in Controlling Access with Amazon Data Firehose. After you complete these two actions, Firehose will stop delivering streams, and you can monitor this using CloudWatch metrics for Firehose.

> **Important**
>
> When you pause stream delivery in Firehose, you need to ensure that the source of the stream (for example, in Kinesis Data Streams or in Managed Service for Kafka) is configured to retain data until stream delivery is resumed and the data gets delivered to the destination. If the source is DirectPUT, Firehose will retain data for 24 hours. Data loss could happen if you do not resume the stream and deliver the data before the expiration of data retention period.

Resume a Firehose stream

To resume delivery, first revert the change made earlier to the stream destination by turning on the destination and ensuring that Firehose has permissions to deliver the stream to the destination. Next, revert the changes made earlier to permissions applied to the S3 bucket for backing up failed deliveries. That is, apply `"Effect": "Allow"` permission for the action `s3:PutObject`, and remove `"Effect": "Deny"` permission on the action `s3:PutObject` for the S3 bucket used for backing up failed deliveries. Finally, monitor using CloudWatch metrics for Firehose to confirm that the stream is being delivered to the destination. To view and troubleshoot errors, use Amazon CloudWatch Logs monitoring for Firehose.

Deliver data to Apache Iceberg Tables with Amazon Data Firehose

Apache Iceberg is a high-performance open-source table format for performing big data analytics. Apache Iceberg brings the reliability and simplicity of SQL tables to Amazon S3 data lakes, and makes it possible for open-source analytics engines like Spark, Flink, Trino, Hive, and Impala to work with the same data concurrently. For more information, see Apache Iceberg and Considerations and limitations.

You can use Firehose to deliver streaming data to Apache Iceberg Tables in Amazon S3. Your Apache Iceberg Tables can be in self-managed in Amazon S3 or hosted in Amazon S3 Tables. In self-managed Iceberg tables, you manage all the table optimizations such as compaction, and snapshot expiration. Amazon S3 Tables provide storage that is optimized for large-scale analytics workloads, with features that continuously improve query performance and reduce storage costs for tabular data. For more information on Amazon S3 Tables, see Amazon S3 Tables.

This feature allows you to route records from a single stream into different Apache Iceberg Tables. You can automatically apply insert, update, and delete operations to records in those tables. It also supports fine-grained data access control on Apache Iceberg tables in Amazon S3 with AWS Lake Formation. You can specify access controls centrally in AWS Lake Formation and provide more granular table-level and column-level permissions for Firehose.

Considerations and limitations

> **Note**
>
> Firehose supports Apache Iceberg Tables as a destination in all AWS Regions except China Regions, AWS GovCloud (US) Regions, Asia Pacific (Taipei), Asia Pacific (Malaysia), Asia Pacific (New Zealand), and Mexico (Central).

Firehose support for Apache Iceberg tables has the following considerations and limitations.

- **Throughput** – If you use **Direct PUT** as the source to deliver data to Apache Iceberg tables, then the maximum throughput per stream is 5 MiB/second in US East (N. Virginia), US West (Oregon), and Europe (Ireland) Regions and 1 MiB/second in all other AWS Regions. If you want to insert

data to Iceberg tables with no updates and deletes and you want higher throughput for your stream, then you can use the Firehose Limits form to request a throughput limit increase.

You can also set the `AppendOnly` flag to `True` if you want to only insert data and not perform updates and deletes. By setting the `AppendOnly` flag to `True`, Firehose automatically scales to match your throughput. Currently, you can set this flag only with the CreateDeliveryStream API operation.

If a **Direct PUT** stream experiences throttling due to higher data ingest volumes that exceed the throughput capacity of a Firehose stream, then Firehose automatically increases the throughput limit of the stream until the throttling is contained. Depending on increased throughput and throttling, it might take longer for Firehose to increase the throughput of a stream to the desired levels. Because of this, continue to retry the failed data ingest records. If you expect the data volume to increase in sudden large bursts, or if your new stream needs a higher throughput than the default throughput limit, request to increase the throughput limit.

- **Throughput and Partition Scaling** – The service is optimized to support either a large number of Iceberg partitions or very high ingest throughput. As ingest throughput increases, the number of partitions that can be actively written to decreases.

  Here are the limits for ingest throughput and max active partitions supported.

Ingest Throughput	Max Active Partitions supported
≤ 20 MB/s	Up to ~3,000
20–40 MB/s	1000
40–400 MB/s	100
400–750 MB/s	50
750 MB/s–1.5 GB/s	1

> **Note**
>
> Ingest throughput here refers to the volume of data Firehose processes and writes to your Iceberg tables, measured *after* any Lambda transformation – not the volume of data sent to the stream at the source. If you use a Lambda function that expands your records,

your processed throughput can be significantly higher than your source ingest volume, and the applicable max active partitions limit is based on the larger, post-transformation throughput.

- **S3 Transaction Per Second (TPS)** – To optimize S3 performance, if you are using Kinesis Data Streams or Amazon MSK as a source, we recommend that you partition the source record using a proper partition key. In that way, data records that are routed to the same Iceberg table are mapped to one or a few source partitions know as shards. If possible, spread data records belonging to different target Iceberg tables into different partitions/shards, so that you can use all the aggregate throughput available across all the partitions/shards of the source topic/stream.
- **Columns** – For column names and values, Firehose takes only the first level of nodes in a multi-level nested JSON. For example, Firehose selects the nodes that are available in the first level including the position field. The column names and the data types of the source data must exactly match those of the target tables for Firehose to deliver successfully. In this case, Firehose expects that you have either struct or map data type column in your Iceberg tables to match the position field. Firehose supports 16 levels of nesting. Following is an example of a nested JSON.

```
{
   "version":"2016-04-01",
   "deviceId":"<solution_unique_device_id>",
   "sensorId":"<device_sensor_id>",
   "timestamp":"2024-01-11T20:42:45.000Z",
   "value":"<actual_value>",
   "position":{
      "x":143.595901,
      "y":476.399628,
      "z":0.24234876
   }
}
```

  If the column names or data types do not match, then Firehose throws an error and delivers data to the S3 error bucket. If all the column names and data types match in the Apache Iceberg tables, but you have an additional field present in the source record, Firehose skips the new field.
- **One JSON object per record** – You can send only one JSON object in one Firehose record. If you aggregate and send multiple JSON objects inside a record, Firehose throws an error and delivers data to the S3 error bucket. If you aggregate records with KPL and ingest data into Firehose with

Amazon Kinesis Data Streams as source, then Firehose automatically de-aggregates and uses one JSON object per record.

- **Compaction and storage optimization** – Every time you write to Iceberg Tables using Firehose, it commits and generates snapshots, data files and delete files. Having many data files increases metadata overhead and affects read performance. To get efficient query performance, you might want to consider a solution that periodically takes small data files and rewrites them into fewer larger data files. This process is called compaction. AWS Glue Data Catalog supports automatic compaction of your Apache Iceberg Tables. For more information, see Compaction management in the *AWS Glue User Guide*. For additional information, see Automatic compaction of Apache Iceberg Tables. Alternatively, you can run the Athena Optimize command to perform compaction manually. For more information about the Optimize command, see Athena Optimize.

  Besides compaction of data files, you can also optimize storage consumption with the VACUUM statement that performs table maintenance on Apache Iceberg tables, such as snapshot expiration and orphan file removal. Alternatively, you can use AWS Glue Data Catalog that also supports managed table optimization of Apache Iceberg tables by automatically removing the data files, orphaned files, and expire snapshots that are no longer needed. For more information, see this blog post on Storage optimization of Apache Iceberg Tables.
- We do not support Amazon MSK Serverless source for Apache Iceberg Tables as a destination.
- For an update operation, Firehose puts a delete file followed by an insert operation. Putting delete files incurs Amazon S3 put charges.
- Firehose does not recommend using multiple Firehose streams to write data to the same Apache Iceberg table. This is because Apache Iceberg relies on Optimistic Concurrency Control (OCC). If multiple Firehose streams attempt to write to a single Iceberg table concurrently, then only one stream is succeed in committing the data at a given time. The other streams that fail to commit back-off, and retry the commit operation until the configured retry duration expires. Once the retry duration is exhausted, the data and delete file keys (Amazon S3 paths) are sent to the configured Amazon S3 error prefix.
- The current Iceberg Library version that Firehose supports is version 1.5.2.
- For delivering encrypted data to Amazon S3 Tables, you should configure AWS Key Management Service parameters in Amazon S3 Tables, and not in the Firehose configuration. If you configure AWS Key Management Service parameters in Firehose for delivering encrypted data to Amazon S3 Tables, then Firehose cannot use those parameters to encrypt. For more information, see Using server-side encryption with AWS KMS keys.

- Firehose streams only support delivery to databases and tables that are created through Iceberg's GlueCatalog API. Delivery to databases and tables that are created through the Glue SDK are not supported. Note that a hyphen (-) is not a supported character for the database and the table name in the Iceberg library. For more details, see the Glue Database Regex and the Glue Table Regex that are supported by the Iceberg library.
- All files written by Firehose are computed using the partition that is present in the record. This also applies to deleted files. Global deletes, such as writing unpartitioned delete files for a partitioned table, is not supported.
- Firehose does not currently support bloom filter properties when delivering data to Apache Iceberg tables. When bloom filter properties are configured on Iceberg tables, Firehose will ignore these properties during data delivery operations.

Prerequisites to use Apache Iceberg Tables as a destination

Choose from the following options to complete the required prerequisites.

Topics

- Prerequisites to deliver to Iceberg Tables in Amazon S3
- Prerequisites to deliver to Amazon S3 Tables

Prerequisites to deliver to Iceberg Tables in Amazon S3

Before you begin, complete the following prerequisites.

- **Create an Amazon S3 bucket** – You must create an Amazon S3 bucket to add metadata file path during tables creation. For more information, see Create an S3 bucket.
- **Create an IAM role with required permissions** – Firehose needs an IAM role with specific permissions to access AWS Glue tables and write data to Amazon S3. The same role is used to grant AWS Glue access to Amazon S3 buckets. You need this IAM role when you create an Iceberg Table and a Firehose stream. For more information, see Grant Firehose access to Amazon S3 Tables.
- **Create Apache Iceberg Tables** – If you are configuring unique keys in the Firehose stream for updates and deletes, Firehose validates if the table and unique keys exist as a part of stream creation. For this scenario, you must create tables before creating the Firehose stream. You can use AWS Glue to create Apache Iceberg Tables. For more information, see Creating Apache

Iceberg tables. If you are not configuring unique keys in the Firehose stream, then you don't require to create Iceberg tables before creating a Firehose stream.

> **Note**
>
> Firehose supports the following table version and format for Apache Iceberg tables.
>
> - **Table format version** – Firehose only supports V2 table format. Do not create tables in V1 format, else you get an error and data is delivered to the S3 error bucket instead.
> - **Data storage format** – Firehose writes data to Apache Iceberg Tables in Parquet format.
> - **Row level operation** – Firehose supports the Merge-on-Read (MOR) mode of writing data to Apache Iceberg Tables.

Prerequisites to deliver to Amazon S3 Tables

To deliver data to Amazon S3 table buckets, complete the following prerequisites.

- Create an S3 Table bucket, namespace, tables in the table bucket, and other integration steps outlined in Getting started with Amazon S3 Tables. Column names must be lowercase because of the limitations imposed by the S3 Tables catalog integration, as specified in S3 tables catalog integration limitations.
- **Create an IAM role with required permissions** – Firehose needs an IAM role with specific permissions to access AWS AWS Glue tables and write data to tables in an Amazon S3 table bucket. To write to tables in an Amazon S3 table bucket, you must also provide the IAM role with the required permissions. The permissions required for Amazon S3 Tables catalog depend on the access control mode you use:
  - **IAM access control** – The Firehose delivery role needs IAM permissions directly on Amazon S3 Tables resources.
  - **Lake Formation access control** – The Firehose delivery role needs AWS AWS Lake Formation permissions for managing access to your table resources. AWS Lake Formation uses its own permissions model that enables fine-grained access control for Data Catalog resources.

  You configure this IAM role when you create a Firehose stream. For more information, see Grant Firehose access to Amazon S3 Tables.

For step-by-step integration, refer to the blog Build a data lake for streaming data with Amazon S3 Tables and Amazon Data Firehose. For additional information, also refer to Using Amazon S3 Tables with AWS analytics services.

Set up the Firehose stream

To create a Firehose stream with Apache Iceberg Tables as your destination you must configure the following.

> **Note**
>
> The setup of a Firehose stream for delivering to tables in S3 table buckets is the same as Apache Iceberg Tables in Amazon S3.

Configure source and destination

To deliver data to Apache Iceberg Tables, choose the source for your stream.

To configure your source for your stream, see Configure source settings.

Next, choose **Apache Iceberg Tables** as the destination and provide a Firehose stream name.

Configure data transformation

To perform custom transformations on your data, such as adding or modifying records in your incoming stream, you can add a Lambda function to your Firehose stream. For more information on data transformation using Lambda in a Firehose stream, see Transform source data in Amazon Data Firehose.

For Apache Iceberg Tables, you must specify how you want to route incoming records to different destination tables and the operations that you want to perform. One of the ways to provide the required routing information to Firehose is using a Lambda function.

For more information, see Route records to different Iceberg tables.

Connect data catalog

Apache Iceberg requires a data catalog to write to Apache Iceberg Tables. Firehose integrates with AWS Glue Data Catalog for Apache Iceberg Tables.

You can use AWS Glue Data Catalog in the same account as your Firehose stream or in a cross-account and in the same Region as your Firehose stream (default), or in a different Region.

If you are delivering to an Amazon S3 Table and you are using the console to set up your Firehose stream, then select the catalog that corresponds to your Amazon S3 Table catalog. If you are using the CLI to set up your Firehose stream, then in the `CatalogConfiguration` input, use `CatalogARN` with the format: `arn:aws:glue:<region>:<account-id>:catalog/s3tablescatalog/<s3 table bucket name>`. For more information, see Setting up a Firehose stream to Amazon S3 tables.

> **Note**
>
> Firehose supports three operations for Iceberg tables: insert, update, and delete. Without a specified operation, Firehosedefaults to insert, adding each incoming record as a new row, and preserving duplicates. To modify existing records instead, specify the "update" operation, which uses primary keys to locate, and change existing rows.
> Example:
>
> - Default (insert): Multiple identical customer records create duplicate rows.
> - Specified update: New customer address updates the existing record.

Configure JQ expressions

For Apache Iceberg Tables, you must specify how you want to route incoming records to different destination tables and the operations such as insert, update, and delete that you want to perform. You can do this by configuring JQ expressions for Firehose to parse and get the required information. For more information, see ???.

Configure unique keys

Updates and Deletes with more than one table – Unique keys are one or more fields in your source record that uniquely identify a row in Apache Iceberg Tables. If you have insert only scenario with more than one table, then you do not have to configure unique keys. If you want to do updates and deletes on certain tables, then you must configure unique keys for those required tables. Note that update will automatically insert the row if the row in the tables is missing. If you have only a single table, then you can configure unique keys. For an update operation, Firehose puts a delete file followed by an insert.

You can either configure unique keys per table as a part of Firehose stream creation or you can set identifier-field-ids natively in Iceberg during create table or alter table operation. Configuring unique keys per table during stream creation is optional. If you don't configure unique keys per table during stream creation, Firehose checks for `identifier-field-ids` for required tables and will use them as unique keys. If both are not configured, then delivery of data with update and delete operations fails.

To configure this section, provide the database name, table name, and unique keys for the tables where you want to update or delete data. You can have only entry for each table in the configuration. You don't need to configure this section for append-only scenarios. Optionally, you can also choose to provide an error bucket prefix if data from the table fails to deliver as shown in the following example.

```
[
  {
    "DestinationDatabaseName": "MySampleDatabase",
    "DestinationTableName": "MySampleTable",
    "UniqueKeys": [
      "COLUMN_PLACEHOLDER"
    ],
    "S3ErrorOutputPrefix": "OPTIONAL_PREFIX_PLACEHOLDER"
  }
]
```

Firehose supports the configuration of unique keys if the supplied column name is unique across the entire table. However, it does not support fully qualified column names as unique keys. For instance, a key named `top._id` is not considered a unique key if the column name `_id` is also present at the top-level. If `_id` is unique across the entire table, then it is utilized regardless of its location within the table structure. This is whether it's a top-level column or a nested column. In the following example, `_id` is a valid unique key for the schema because the column name is unique across the schema.

```
[
 "schema": {
  "type": "struct",
  "fields": [
    {
      "name": "top",
      "type": {
        "type": "struct",
```

```
        "fields": [
          { "name": "_id", "type": "string" },
          { "name": "name", "type": "string" }
        ]
      }
    },
    { "name": "user", "type": "string" }
  ]
}
]
```

In the following example, `_id` is not a valid unique key for the schema because it is used in both the top-level column, and the nested struct.

```
[
"schema": {
  "type": "struct",
  "fields": [
    {
      "name": "top",
      "type": {
        "type": "struct",
        "fields": [
          { "name": "_id", "type": "string" },
          { "name": "name", "type": "string" }
        ]
      }
    },
    { "name": "_id", "type": "string" }
  ]
}
]
```

Specify retry duration

You can use this configuration to specify the duration in seconds for which Firehose should attempt to retry, if it encounters failures in writing to Apache Iceberg Tables in Amazon S3. You can set any value from 0 to 7200 seconds for performing retries. By default, Firehose retries for 300 seconds.

Handle failed delivery or processing

You must configure Firehose to deliver records to an S3 backup bucket in case it encounters failures in processing or delivering a stream after expiry of retry duration. For this, configure the **S3 backup bucket** and **S3 backup bucket error output prefix** from **Backup settings** in console.

Handle errors

Firehose sends all delivery errors to CloudWatch Logs, and Amazon S3 error buckets.

List of errors:

Error Message	Description
`Iceberg.NoSuchTable`	Firehose is writing to a table that doesn't exist, or the table is not in V2 format. Firehose doesn't support tables in V1 format.
`Iceberg.InvalidTableName`	A null or empty table name is passed, or the table is not in V2 format. Firehosedoesn't support tables in V1 format.
`S3.AccessDenied`	Ensure that the IAM role created in the prerequisites step has the required permissio ns, and trust policy.
`Glue.AccessDenied`	Ensure that the IAM role created in the prerequisites step has the required permissio ns, and trust policy.

Configure buffer hints

Firehose buffers incoming streaming data in memory to a certain size (**Buffering size**) and for a certain period of time (**Buffering interval**) before delivering it to Apache Iceberg Tables. You can choose a buffer size of 1–128 MiBs and a buffer interval of 0–900 seconds. Higher buffer hints results in a lower number of S3 writes, less cost of compaction due to larger data files, and faster query runtime, but with a higher latency. Lower buffer hint values deliver the data with lower latency.

Configure advanced settings

You can configure server-side encryption, error logging, permissions, and tags for your Apache Iceberg Tables. For more information, see Configure advanced settings. You must add the IAM role that you created as part of the ???. Firehose will assume the role to access AWS Glue tables and write to Amazon S3 buckets.

Firehose stream creation can take several minutes to complete. After you successfully create the Firehose stream, you can start ingesting data into it and can view the data in Apache Iceberg tables.

Route incoming records to a single Iceberg table

If you want Firehose to insert data to a single Iceberg table, simply configure a single database and table in your stream configuration as shown in the following example JSON. For a single table, you do not require JQ expression and Lambda function for providing the routing information to Firehose. If you provide these fields along with JQ or Lambda, then Firehose will take input from JQ or Lambda.

```
[
  {
    "DestinationDatabaseName": "UserEvents",
    "DestinationTableName": "customer_id",
    "UniqueKeys": [
      "COLUMN_PLACEHOLDER"
    ],
    "S3ErrorOutputPrefix": "OPTIONAL_PREFIX_PLACEHOLDER"
  }
]
```

In this example, Firehose routes all input records to `customer_id` table in `UserEvents` database. If you want to perform update or delete operations on a single table, then you must provide the operation for each incoming record to Firehose using either the JSONQuery method or Lambda method.

Route incoming records to different Iceberg tables

Amazon Data Firehose can route incoming records in a stream to different Iceberg tables based on the content of the record. Records are not kept in order when delivered from Amazon Data Firehose. Consider the following sample input record.

```
{
  "deviceId": "Device1234",
  "timestamp": "2024-11-28T11:30:00Z",
  "data": {
    "temperature": 21.5,
    "location": {
      "latitude": 37.3324,
      "longitude": -122.0311
    }
  },
  "powerlevel": 84,
  "status": "online"
}
```

```
{
  "deviceId": "Device4567",
  "timestamp": "2023-11-28T10:40:00Z",
  "data": {
    "pressure": 1012.4,
    "location": {
      "zipcode": 24567
    }
  },
  "powerlevel": 82,
  "status": "online"
}
```

In this example, the **`deviceId`** field has two possible values – `Device1234` and `Device4567`. When an incoming record has **`deviceId`** field as `Device1234`, we want to write the record to an Iceberg table named `Device1234`, and when an incoming record has **`deviceId`** field as `Device4567`, we want to write the record to a table named `Device4567`.

Note that the records with `Device1234` and `Device4567` might have a different set of fields that map to different columns in the corresponding Iceberg table. The incoming records might have a nested JSON structure where the **`deviceId`** can be nested within the JSON record. In the

upcoming sections, we discuss how you can route records to different tables by providing the appropriate routing information to Firehose in such scenarios.

Provide routing information to Firehose with JSONQuery expression

The simplest and most cost effective way to provide record routing information to Firehose is by providing a JSONQuery expression. With this approach, you provide JSONQuery expressions for three parameters – `Database Name`, `Table Name`, and (optionally) `Operation`. Firehose uses the expression that you provide to extract information from your incoming stream records to route the records.

The `Database Name` parameter specifies the name of the destination database. The `Table Name` parameter specifies the name of the destination table. `Operation` is an optional parameter that indicates whether to insert the incoming stream record as a new record into the destination table, or to modify or delete an existing record in the destination table. The Operation field must have one of the following values – `insert`, `update`, or `delete`.

For each of these three parameters, you can either provide a static value or a dynamic expression where the value is retrieved from the incoming record. For example, if you want to deliver all incoming stream records to a single database named `IoTevents`, the Database Name would have a static value of `“IoTevents”`. If the destination table name must be obtained from a field in the incoming record, the Table Name is a dynamic expression that specifies the field in the incoming record from which the destination table name needs to be retrieved.

In the following example, we use a static value for Database Name, a dynamic value for Table Name, and a static value for operation. Note that specifying the Operation is optional. If no operation is specified, Firehose inserts the incoming records into the destination table as new records by default.

```
Database Name : "IoTevents"
Table Name : .deviceId
Operation : "insert"
```

If the `deviceId` field is nested within the JSON record, we specify Table Name with the nested field information as `.event.deviceId`.

> **Note**
>
> - When you specify the operation as `update` or `delete`, you must either specify unique keys for the destination table when you set up your Firehose stream, or set identifier-field-ids in Iceberg when you run create table or alter table operations in Iceberg. If you fail to specify this, then Firehose throws an error and delivers data to an S3 error bucket.
> - The `Database Name` and `Table Name` values must exactly match with your destination database and table names. If they do not match, then Firehose throws an error and delivers data to an S3 error bucket.

Provide routing information using an AWS Lambda function

There might be scenarios where you have complex rules that determine how to route incoming records to a destination table. For example, you might have a rule that defined if a field contains the value A, B, or F, that should be routed to a destination table named `TableX` or you might want to augment the incoming stream record by adding additional attributes. For example, if a record contains a field `device_id` as 1, you might want to add another field that `device_type` as “modem“, and write the additional field to the destination table column. In such cases, you can transform the source stream by using an AWS Lambda function in Firehose and provide routing information as part of the output of the Lambda transformation function. To understand how you can transform the source stream by using an AWS Lambda function in Firehose, see Transform source data in Amazon Data Firehose.

When you use Lambda for transformation of a source stream in Firehose, the output must contain `recordId`, `result`, and `data` or `KafkaRecordValue` parameters. The parameter `recordId` contains the input stream record, `result` indicates whether the transformation was successful, and `data` contains the Base64-encoded transformed output of your Lambda function. For more information, see ???.

```
{
  "recordId": "49655962066601463032522589543535113056108699331451682818000000",
  "result": "Ok",
  "data": "1IiwiI6ICJmYWxsIiwgImdgU21IiwiI6ICJmYWxsIiwg==tcHV0ZXIgU2NpZW5jZSIsICJzZW1"
}
```

To specify routing information to Firehose on how to route the stream record to a destination table as part of your Lambda function, the output of your Lambda function must contain an additional section for `metadata`. The following example shows how the metadata section is added to the Lambda output for a Firehose stream that uses Kinesis Data Streams as a data source to instruct Firehose that it must insert the record as a new record into table named `Device1234` of the database `IoTevents`.

```
 {
"recordId": "49655962066601463032522589543535113056108699331451682818000000",
    "result": "Ok",
    "data":
 "1IiwiI6ICJmYWxsIiwgImdgU21IiwiI6ICJmYWxsIiwg==tcHV0ZXIgU2NpZW5jZSIsICJzZW1",

    "metadata":{
"otfMetadata":{
            "destinationTableName":"Device1234",
            "destinationDatabaseName":"IoTevents",
            "operation":"insert"
        }
    }
  }
```

Similarly, the following example shows how you can add the metadata section to the Lambda output for a Firehose that uses Amazon Managed Streaming for Apache Kafka as a data source to instruct Firehose that it must insert the record as a new record into a table named `Device1234` in the database `IoTevents`.

```
{
"recordId": "49655962066601463032522589543535113056108699331451682818000000",
    "result": "Ok",
    "kafkaRecordValue":
 "1IiwiI6ICJmYWxsIiwgImdgU21IiwiI6ICJmYWxsIiwg==tcHV0ZXIgU2NpZW5jZSIsICJzZW1",

    "metadata":{
"otfMetadata":{
            "destinationTableName":"Device1234",
            "destinationDatabaseName":"IoTevents",
            "operation":"insert"
        }
    }
  }
```

For this example,

- `destinationDatabaseName` refers to the name of the target database and is a required field.
- `destinationTableName` refers to the name of the target table and is a required field.
- `operation` is an optional field with possible values as `insert`, `update`, and `delete`. If you do not specify any values, the default operation is `insert`.

> **Note**
>
> - When you specify the operation as `update` or `delete`, you must either specify unique keys for the destination table when you set up your Firehose stream, or set identifier-field-ids in Iceberg when you run create table or alter table operations in Iceberg. If you fail to specify this, then Firehose throws an error and delivers data to an S3 error bucket.
> - The `Database Name` and `Table Name` values must exactly match with your destination database and table names. If they do not match, then Firehose throws an error and delivers data to an S3 error bucket.
> - When your Firehose stream has both a Lambda transformation function and a JSONQuery expression, Firehose first checks for the metadata field in Lambda output to determine how to route the record to the appropriate destination table, and then look at the output of your JSONQuery expression for missing fields.
>
> If the Lambda or JSONQuery expression don't provide the required routing information, then Firehose assumes this as a single table scenario and looks for single table information in the unique keys configuration.
>
> For more information, see the Route incoming records to a single Iceberg table. If Firehose fails to determine routing information and match the record to a specified destination table, it delivers the data to your specified S3 error bucket.

Sample Lambda function

This Lambda function is a sample Python code that parses the incoming stream records and adds required fields to specify how the data should be written to specific tables. You can use this sample code to add the metadata section for routing information.

```
import json
```

```
import base64


def lambda_handler(firehose_records_input, context):
    print("Received records for processing from DeliveryStream: " +
 firehose_records_input['deliveryStreamArn'])

    firehose_records_output = {}
    firehose_records_output['records'] = []


    for firehose_record_input in firehose_records_input['records']:

        # Get payload from Lambda input, it could be different with different sources
        if 'kafkaRecordValue' in firehose_record_input:
            payload_bytes =
 base64.b64decode(firehose_record_input['kafkaRecordValue']).decode('utf-8')
        else
            payload_bytes =
 base64.b64decode(firehose_record_input['data']).decode('utf-8')

        # perform data processing on customer payload bytes here

        # Create output with proper record ID, output data (may be different with
 different sources), result, and metadata
        firehose_record_output = {}

        if 'kafkaRecordValue' in firehose_record_input:
            firehose_record_output['kafkaRecordValue'] =
 base64.b64encode(payload_bytes.encode('utf-8'))
        else
            firehose_record_output['data'] =
 base64.b64encode(payload_bytes.encode('utf-8'))

        firehose_record_output['recordId'] = firehose_record_input['recordId']
        firehose_record_output['result'] =  'Ok'
        firehose_record_output['metadata'] = {
            'otfMetadata': {
                'destinationDatabaseName': 'your_destination_database',
                'destinationTableName': 'your_destination_table',
                'operation': 'insert'
                }
            }
        firehose_records_output['records'].append(firehose_record_output)
```

```
    return firehose_records_output
```

Monitor metrics

For data delivery to Apache Iceberg Tables, Firehose emits the following CloudWatch metrics at a stream level.

Metric	Description
`DeliveryToIceberg.Bytes`	The number of bytes delivered to Apache Iceberg Tables over the specified time period. Units: Bytes
`DeliveryToIceberg.IncomingRowCount`	Number of records that Firehose attempts to deliver to Apache Iceberg Tables. Units: Count
`DeliveryToIceberg.SuccessfulRowCount`	Number of successful rows delivered to Apache Iceberg Tables. Units: Count
`DeliveryToIceberg.FailedRowCount`	Number of failed rows delivered to S3 backup bucket. Units: Count
`DeliveryToIceberg.DataFreshness`	The age (from getting into Firehose to now) of the earliest record in Firehose. Any record earlier than this age has been delivered to Apache Iceberg Tables. Units: Seconds
`DeliveryToIceberg.Success`	Sum of successful commits to Apache Iceberg Tables.
`JQProcessing.Duration`	The amount of time it took to run the JQ expression. Units: Milliseconds

Understand supported data types

Firehose supports all the primitive and complex data types that Apache Iceberg supports. For more information, see Schemas and Data Types. When sending binary data as a string, you must use Firehose supported encoding types - Basic Base64, MIME Base64, URL and filename safe Base64, and Hex. For Timestamp data types, you must always send in microseconds.

Data types examples

The following section shows examples of different data types.

MapType


```
{
    "destination_column_0":
 {"WP5o0JOkuIQcDPcsvpJJygF1xzaOSq0wUlgTwuIeCEzgVneGxA":"PO3ReF3auyDqbfonx9Cd8NTmcQnqnw7JuZOCWwI
    "destination_column_1": "{\"{\\\"destination_nested_column_0\\\": \
\\"18:56:14.974\\\", \\\"destination_nested_column_1\\\": 241.86246}\":
\"M07kAvYdHvBh61F7RzfxtEd39YQI33LnM2NbGS67DOFFsRUyUUujKT5VnK7Wtfz1mHNeIix6FAY9cYpwTdedgr9XnFwG0
\",\"{\\\"destination_nested_column_0\\\": \\\"18:56:14.974\
\\", \\\"destination_nested_column_1\\\": 562.56384}\":
\"9GlxhDCt95LxBo51HybBZihqOqf6EU8jrDu7NMpxtGB2dY6q6kXpvxIrFuMdqHCJKIZIcDikwggLniUm8kgE4d
\",\"{\\\"destination_nested_column_0\\\": \\\"18:56:14.974\
\\", \\\"destination_nested_column_1\\\": 496.03268}\":
\"keTJZYLNvLRB50DMKzEI6M0AM4mueyNnA1m2YVnYdDwyxUpPqkb72Q6LiX0B9s8gCjZ6trW6C1PFk9KNBIpxYsj5Tc5Xs
\",\"{\\\"destination_nested_column_0\\\": \\\"18:56:14.974\\\", \\
\"destination_nested_column_1\\\": 559.0878}\":
\"mG0ZET84BUF28E312UCIWgmypyQFSUODH9NAMAnF3LJEutbooZWcBt97PP5AHaopNvC8pQZ4mGXB9hmVmjUNmuj5Qanyx
\",\"{\\\"destination_nested_column_0\\\": \\\"18:56:14.974\
\\", \\\"destination_nested_column_1\\\": 106.845245}\":
\"aidoVYrzu8gcLRkVVUyTKCN9gqTUFYi8uJQsrXEFEYl1f9ool7JhAtg9QKG5BBu67Ngb95ENsNKQyCHNImSu5x4hMnmHU
\"}"
}
```


DecimalType


```
{
    "destination_column_0": 9455262425851.1342772,
    "destination_column_1": "9455262425851.1342772",
    "destination_column_2": 9455262425852
}
```

BinaryType (base64-default, base64-mime, base64-url-safe, hex)


```
{
    "destination_column_0": "AsYhnHD\/Ra54hITl1daNV9glOjtWPEfopH
+PjgUKHYB6K7UcYi4K19b80wD4J\/93x5tyh+0y
+k5cMljVRlmfIkIuLxl9ERBiPPLhf4+yoJ2k70VavPnYWmNLs1hLDHlfeEMIfVhrqOGzJMoA
+CBAWXfIuiG420JSQP5iAx5xFG\/
mOfkM5zYothje8OGXltdthcCL6WYBiP0SlwXcE0uMeRfwclAc9fTOBz6RzdJlHhUDjoAXg
+4cvly27F82XpuGMNwpUj98AOrgbh2MoU9yvsM9ZrjD0eGVgOZP8Ky7Za4oE\/oK8j
+qABF6XV712iA6pVtTNJFvX6Ey3ssNYvno+LYF5ZsySs2rB5AbVM73RfOPqdS\/c\/
r3MEqoEqt+nPx6eGam4WSA+0swztt7aLdrlX6yK7xJeIJ0rTlIDBo0ZUaw011ykY
\/8Bvy+4byoPlmr4Z5yhN1z3ZTOkx7eDR6xMv+vDVsDBtItVazDwHgDy41r
\/hQNeNedPKrozc8TY9k7wZre\/6V2lCa3BmT8Uu9b9ydjR9z+fCSdG
+VRv35nz5kdqdKy8YIrynYs4eOcjh8jH3UwVYrYQcnWkBAfF7Xk9CoPVnL3ciHZtyiZOaTGIj9rO0xX\/
W5dGe9\/4YChs6LbD584kxLTxvHgSl4vadaTGNKci3SvNmZNsz8ducxtNXF\/Tv2DUub465hzgpaLPur3+MB
+kfdN2YXUfqb
+xJAgxThWfUe151nrH0EPow9lgSlp21rUBGznJAvPRl1ExGIAuc7JYAoUrJUkx5Hfl6PekPDhqt7+yJwCB8qxhTTryxo
+bjtai4ndRCGcuCaxT8KkOcXsS37urd3YGSDMinZdMNVc646s25415qK6nBRlqqAY8+EYmcUIVB9XcNdke4zoUfhVQoruwi
\/kFafoulo5DEoMOyaH1N2HCSxG5tZXNQocSZPaY8efZYMCpmDXsPAzkmgSkYRDSu\/r3wUqROa2tGK5\/
pQY24v+Jq0U\/jQ99GShlU283nZ85ot2ocbtMAgD\/WsrSEh6lNt9RaI3HfA7\/HcH\/
fgr9jsTtxDgZhabTBwwDwX0zjWGx1bCuTLKBN7byxg9ZvAVgqwPS4HERLer5T5UkKf74zn9Eq3HYH1Q5JpyDUx
+im7mte1sprf1+A24kksVU\/MD9aP9N8\/QDsQ13gkhOn5KwFMz3BC2Vw5gL
+gGNHFKDRL6wGIfhuYcx9LucolZ1yNy9Gbb3ioWSSufyFpyXqtndDLPI5QS1SJpJm2KDyqcH1SmRLIhd9MNRUC73EAEm
+NO5wxPzBRSjhCHZpf8SrYITWJl7K3XzGOfPFh2NgES3jMP9cvSXO6yyICcep2HBYGbFflni89+Rw==",
    "destination_column_1": "AsYhnHD\/Ra54hITl1daNV9glOjtWPEfopH
+PjgUKHYB6K7UcYi4K19b80wD4J\/93x5tyh+0y+k5c\r
\nMljVRlmfIkIuLxl9ERBiPPLhf4+yoJ2k70VavPnYWmNLs1hLDHlfeEMIfVhrqOGzJMoA+CBAWXfI\r
\nuiG420JSQP5iAx5xFG\/mOfkM5zYothje8OGXltdthcCL6WYBiP0SlwXcE0uMeRfwclAc9fTOBz6R\r
\nzdJlHhUDjoAXg+4cvly27F82XpuGMNwpUj98AOrgbh2MoU9yvsM9ZrjD0eGVgOZP8Ky7Za4oE\/oK\r
\n8j+qABF6XV712iA6pVtTNJFvX6Ey3ssNYvno+LYF5ZsySs2rB5AbVM73RfOPqdS\/c\/r3MEqoEqt+
\r\nnPx6eGam4WSA+0swztt7aLdrlX6yK7xJeIJ0rTlIDBo0ZUaw011ykY\/8Bvy+4byoPlmr4Z5yhN1z
\r\n3ZTOkx7eDR6xMv+vDVsDBtItVazDwHgDy41r\/hQNeNedPKrozc8TY9k7wZre\/6V2lCa3BmT8Uu9b
\r\n9ydjR9z+fCSdG+VRv35nz5kdqdKy8YIrynYs4eOcjh8jH3UwVYrYQcnWkBAfF7Xk9CoPVnL3ciHZ
\r\ntyiZOaTGIj9rO0xX\/W5dGe9\/4YChs6LbD584kxLTxvHgSl4vadaTGNKci3SvNmZNsz8ducxtNXF
\/\r\nTv2DUub465hzgpaLPur3+MB+kfdN2YXUfqb+xJAgxThWfUe151nrH0EPow9lgSlp21rUBGznJAvP
\r\nRl1ExGIAuc7JYAoUrJUkx5Hfl6PekPDhqt7+yJwCB8qxhTTryxo+bjtai4ndRCGcuCaxT8KkOcXs\r
\nS37urd3YGSDMinZdMNVc646s25415qK6nBRlqqAY8+EYmcUIVB9XcNdke4zoUfhVQoruwidzDU\/k\r
\nFafoulo5DEoMOyaH1N2HCSxG5tZXNQocSZPaY8efZYMCpmDXsPAzkmgSkYRDSu\/r3wUqROa2tGK5\r
\n\/pQY24v+Jq0U\/jQ99GShlU283nZ85ot2ocbtMAgD\/WsrSEh6lNt9RaI3HfA7\/HcH\/fgr9jsTtxDg
\r\nZhabTBwwDwX0zjWGx1bCuTLKBN7byxg9ZvAVgqwPS4HERLer5T5UkKf74zn9Eq3HYH1Q5JpyDUx+\r
\nim7mte1sprf1+A24kksVU\/MD9aP9N8\/QDsQ13gkhOn5KwFMz3BC2Vw5gL+gGNHFKDRL6wGIfhuYc
\r\nx9LucolZ1yNy9Gbb3ioWSSufyFpyXqtndDLPI5QS1SJpJm2KDyqcH1SmRLIhd9MNRUC73EAEm+NO\r
\n5wxPzBRSjhCHZpf8SrYITWJl7K3XzGOfPFh2NgES3jMP9cvSXO6yyICcep2HBYGbFflni89+Rw==",
```

```
    "destination_column_2": "AsYhnHD_Ra54hITl1daNV9glOjtWPEfopH-
PjgUKHYB6K7UcYi4K19b80wD4J_93x5tyh-0y-k5cMljVRlmfIkIuLxl9ERBiPPLhf4-
yoJ2k70VavPnYWmNLs1hLDHlfeEMIfVhrqOGzJMoA-
CBAWXfIuiG420JSQP5iAx5xFG_mOfkM5zYothje8OGXltdthcCL6WYBiP0SlwXcE0uMeRfwclAc9fTOBz6RzdJlHhUDjoAX
qABF6XV712iA6pVtTNJFvX6Ey3ssNYvno-LYF5ZsySs2rB5AbVM73RfOPqdS_c_r3MEqoEqt-
nPx6eGam4WSA-0swztt7aLdrlX6yK7xJeIJ0rTlIDBo0ZUaw011ykY_8Bvy-4byoPlmr4Z5yhN1z3ZTOkx7eDR6xMv-
vDVsDBtItVazDwHgDy41r_hQNeNedPKrozc8TY9k7wZre_6V2lCa3BmT8Uu9b9ydjR9z-fCSdG-
VRv35nz5kdqdKy8YIrynYs4eOcjh8jH3UwVYrYQcnWkBAfF7Xk9CoPVnL3ciHZtyiZOaTGIj9rO0xX_W5dGe9_4YChs6LbD
MB-kfdN2YXUfqb-
xJAgxThWfUe151nrH0EPow9lgSlp21rUBGznJAvPRl1ExGIAuc7JYAoUrJUkx5Hfl6PekPDhqt7-
yJwCB8qxhTTryxo-bjtai4ndRCGcuCaxT8KkOcXsS37urd3YGSDMinZdMNVc646s25415qK6nBRlqqAY8-
EYmcUIVB9XcNdke4zoUfhVQoruwidzDU_kFafoulo5DEoMOyaH1N2HCSxG5tZXNQocSZPaY8efZYMCpmDXsPAzkmgSkYRDS
Jq0U_jQ99GShlU283nZ85ot2ocbtMAgD_WsrSEh6lNt9RaI3HfA7_HcH_fgr9jsTtxDgZhabTBwwDwX0zjWGx1bCuTLKBN7
im7mte1sprf1-A24kksVU_MD9aP9N8_QDsQ13gkhOn5KwFMz3BC2Vw5gL-
gGNHFKDRL6wGIfhuYcx9LucolZ1yNy9Gbb3ioWSSufyFpyXqtndDLPI5QS1SJpJm2KDyqcH1SmRLIhd9MNRUC73EAEm-
NO5wxPzBRSjhCHZpf8SrYITWJl7K3XzGOfPFh2NgES3jMP9cvSXO6yyICcep2HBYGbFflni89-Rw==",
    "destination_column_3":
 "02c6219c70ff45ae788484e5d5d68d57d8253a3b563c47e8a47f8f8e050a1d807a2bb51c622e0ad7d6fcd300f827f
}
```


TimeType (Epoch in Microseconds, LocalTime Java Object)

```
{
    "destination_column_0": 68175096000,
    "destination_column_1": "18:56:15.096"
}
```

TimestampType.withZone (Epoch in Microseconds, OffsetDateTime Java Object, LocalDateTime Java Object)

```
{
    "destination_column_0": 1725476175099000,
    "destination_column_1": "2024-09-04T18:56:15.099Z",
    "destination_column_2": "2024-09-04T18:56:15.099"
}
```

DoubleType

```
{
    "destination_column_0": 9.18477568715142,
    "destination_column_1": "9.18477568715142"
}
```

BooleanType

```
{
    "destination_column_0": true,
    "destination_column_1": "false",
    "destination_column_2": 1,
    "destination_column_3": 0
}
```

FloatType

```
{
    "destination_column_0": 0.6242226,
    "destination_column_1": "0.6242226"
}
```

IntegerType

```
{
    "destination_column_0": 7,
    "destination_column_1": "7"
}
```

TimestampType.withoutZone (Epoch in Microseconds, LocalDateTime Java Object, OffsetDateTime Java Object, ZonedDateTime Java Object)

```
{
    "destination_column_0": 1725476175114000,
    "destination_column_1": "2024-09-04T18:56:15.114",
    "destination_column_2": "2024-09-04T18:56:15.114Z",
    "destination_column_3": "2024-09-04T18:56:15.114-07:00"
}
```

DateType

```
{
    "destination_column_0": 19970,
    "destination_column_1": "2024-09-04"
}
```

LongType

```
{
    "destination_column_0": 8,
    "destination_column_1": "8"
}
```

UUIDType (UUID Java Object)

```
{
    "destination_column_0": "21c5521c-a6d4-48d4-b2c8-7f6d842f72c3"
}
```

ListType

```
{
    "destination_column_0":
 ["s1FSrgb0lGDxfn2iYT0EtlP47aHSjwmLZgrdr1JqRs0dmbeCcQoaLr4Xhi2KIVvmus9ppFdpWIcOHnJ0omhAPhXH0yns
    "destination_column_1": "[{\"destination_nested_column_0\":\"bb00f8e6-
db82-4241-a5c5-0d9c0d2f71a4\",\"destination_nested_column_1\":907.35345},
{\"destination_nested_column_0\":\"2c77b702-d405-4fe1-beee-fb541d7ab833\",
\"destination_nested_column_1\":544.0026},{\"destination_nested_column_0\":
\"68389200-d6b1-413d-bcd9-fdb931708395\",\"destination_nested_column_1\":153.683},
{\"destination_nested_column_0\":\"bc31cbaa-39cd-4e2f-b357-9ea9ce75532b\",
\"destination_nested_column_1\":977.5165},{\"destination_nested_column_0\":
\"b7d627f9-0d5b-41b7-903a-525488259fba\",\"destination_nested_column_1\":434.17215},
{\"destination_nested_column_0\":\"06b6ec1e-1952-4582-b285-46aaf40064b8\",
\"destination_nested_column_1\":580.33124},{\"destination_nested_column_0\":
\"f04b3bbf-61ad-4c5c-8740-6f666f57c431\",\"destination_nested_column_1\":550.75793}]"
}
```

Resources

Use the following resources to learn more:

- Stream real-time data into Apache Iceberg tables in Amazon S3 using Amazon Data Firehose
- Streamline AWS WAF log analysis with Apache Iceberg and Amazon Data Firehose
- Build a data lake for streaming data with Amazon S3 Tables and Amazon Data Firehose

Tag a Firehose stream

You can assign your own metadata to Firehose streams that you create in Amazon 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.

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

- TagDeliveryStream
- ListTagsForDeliveryStream
- UntagDeliveryStream

Understand tag basics

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

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

You can use tags to categorize your Firehose streams. For example, you can categorize Firehose 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 Firehose 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`

If you specify tags in the `CreateDeliveryStream` action, Amazon Data Firehose performs an additional authorization on the `firehose:TagDeliveryStream` action to verify if users have permissions to create tags. If you do not provide this permission, requests to create new Firehose streams with IAM resource tags will fail with an `AccessDeniedException` such as following.

```
AccessDeniedException
User: arn:aws:sts::x:assumed-role/x/x is not authorized to perform:
 firehose:TagDeliveryStream on resource: arn:aws:firehose:us-east-1:x:deliverystream/x
 with an explicit deny in an identity-based policy.
```

The following example demonstrates a policy that allows users to create a Firehose stream and apply tags.

Track costs with tagging

You can use tags to categorize and track your AWS costs. When you apply tags to your AWS resources, including Firehose 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 User Guide*.

Know tag restrictions

The following restrictions apply to tags in Amazon 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: `_ . / = + - @`.

Security in Amazon 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 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 Data Firehose. The following topics show you how to configure 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 Data Firehose resources.

Topics

- Data protection in Amazon Data Firehose
- Controlling access with Amazon Data Firehose
- Authenticate with AWS Secrets Manager in Amazon Data Firehose
- Manage IAM roles through Amazon Data Firehose console
- Understand compliance for Amazon Data Firehose
- Resilience in Amazon Data Firehose
- Understand infrastructure security in Amazon Data Firehose
- Implement security best practices for Amazon Data Firehose

Data protection in Amazon Data Firehose

Amazon Data Firehose encrypts all data in transit using TLS protocol. Furthermore, for data stored in interim storage during processing, Amazon Data Firehose encrypts data using AWS Key Management Service and verifies data integrity using checksum verification.

If you have sensitive data, you can enable server-side data encryption when you use Amazon 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

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 Firehose stream reads the data from your data stream, Kinesis Data Streams first decrypts the data and then sends it to Amazon Data Firehose. Amazon 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 Firehose 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 Firehose stream. To do that, specify DeliveryStreamEncryptionConfigurationInput when you invoke CreateDeliveryStream.

To successfully use `CUSTOMER_MANAGED_CMK`, both the caller's IAM policy and the KMS key policy must allow `kms:GenerateDataKey` and `kms:Decrypt` operations. Firehose validates these permissions when you call PutRecord or PutRecordBatch with `CUSTOMER_MANAGED_CMK` encryption. Additionally, `kms:CreateGrant` permission is required when calling CreateDeliveryStream or StartDeliveryStreamEncryption with `CUSTOMER_MANAGED_CMK` encryption.

When the CMK is of type `CUSTOMER_MANAGED_CMK`, if the Amazon 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 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”.

> **Important**
>
> To encrypt your Firehose stream, use symmetric CMKs. Amazon 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.

> **Note**
>
> When you use a customer managed key (CUSTOMER_MANAGED_CMK) to enable server-side encryption (SSE) for your Firehose stream, the Firehose service sets an encryption context whenever it uses your key. Since this encryption context represents an occurrence where a key owned by your AWS account was used, it is logged as part of AWS CloudTrail

event logs for your AWS account. This encryption context is system generated by the Firehose service. Your application should not make any assumptions about the format or content of the encryption context set by the Firehose service.

Controlling access with Amazon Data Firehose

The following sections cover how to control access to and from your Amazon Data Firehose resources. The information they cover includes how to grant your application access so it can send data to your Firehose stream. They also describe how you can grant Amazon 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 Amazon 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 access to your Firehose resources
- Grant Firehose access to your private Amazon MSK cluster
- Allow Firehose to assume an IAM role
- Grant Firehose access to AWS Glue for data format conversion
- Grant Firehose access to an Amazon S3 destination
- Grant Firehose access to Amazon S3 Tables
- Grant Firehose access to an Apache Iceberg Tables destination
- Grant Firehose access to an Amazon Redshift destination
- Grant Firehose access to a public OpenSearch Service destination
- Grant Firehose access to an OpenSearch Service destination in a VPC
- Grant Firehose access to a public OpenSearch Serverless destination
- Grant Firehose access to an OpenSearch Serverless destination in a VPC
- Grant Firehose access to a Splunk destination
- Accessing Splunk in VPC

- Ingest VPC flow logs into Splunk using Amazon Data Firehose
- Accessing Snowflake or HTTP end point
- Grant Firehose access to a Snowflake destination
- Accessing Snowflake in VPC
- Grant Firehose access to an HTTP endpoint destination
- Cross-account delivery from Amazon MSK
- Cross-account delivery to an Amazon S3 destination
- Cross-account delivery to an OpenSearch Service destination
- Using tags to control access

Grant access to your Firehose resources

To give your application access to your Firehose 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:*"`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "firehose:DeleteDeliveryStream",
                "firehose:PutRecord",
                "firehose:PutRecordBatch",
                "firehose:UpdateDestination"
            ],
            "Resource": [
                "arn:aws-eusc:firehose:eusc-de-
east-1:123456789012:deliverystream/delivery-stream-name"
            ]
        }
    ]
}
```

Grant Firehose access to your private Amazon MSK cluster

If the source of your Firehose stream is a private Amazon MSK cluster, then use a policy similar to this example.

You must add a policy like this to the cluster's resource-based policy to grant Firehose service principal the permission to invoke the Amazon MSK `CreateVpcConnection` API operation.

Allow Firehose to assume an IAM role

This section describes the permissions and policies that grant Amazon Data Firehose access to ingest, process, and deliver data from source to destination.

> **Note**
>
> If you use the console to create a Firehose stream and choose the option to create a new role, AWS attaches the required trust policy to the role. If you want Amazon Data Firehose to use an existing IAM role or if you create a role on your own, attach the following trust policies to that role so that Amazon Data Firehose can assume it. Edit the policies to replace *`account-id`* with your AWS account ID. For information about how to modify the trust relationship of a role, see Modifying a Role.

Amazon Data Firehose uses an IAM role for all the permissions that the Firehose stream needs to process and deliver data. Make sure that the following trust policies are attached to that role so that Amazon Data Firehose can assume it.

If you choose Amazon MSK as the source for your Firehose stream, you must specify another IAM role that grants Amazon Data Firehose permissions to ingest source data from the specified Amazon MSK cluster. Make sure that the following trust policies are attached to that role so that Amazon Data Firehose can assume it.

JSON

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

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

Make sure that this role that grants Amazon Data Firehose permissions to ingest source data from the specified Amazon MSK cluster grants the following permissions:

Grant Firehose access to AWS Glue for data format conversion

If your Firehose stream performs data-format conversion, Amazon Data Firehose references table definitions stored in AWS Glue. To give Amazon 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.

```
{
    "Sid": "",
    "Effect": "Allow",
    "Action": [
        "glue:GetTable",
        "glue:GetTableVersion",
        "glue:GetTableVersions"
    ],
    "Resource": [
        "arn:aws-eusc:glue:eusc-de-east-1:123456789012:catalog",
        "arn:aws-eusc:glue:eusc-de-east-1:123456789012:database/b",
        "arn:aws-eusc:glue:eusc-de-east-1:123456789012:table/b/easd"
    ]
},
{
      actions: ['glue:GetSchemaVersion'],
      grantee: options.role,
      resourceArns: ['*'],
}
```

The recommended policy for getting schemas from schema registry has no resource restrictions. For more information, see IAM examples for deserializers in the AWS Glue Developer Guide.

Grant Firehose access to an Amazon S3 destination

When you're using an Amazon S3 destination, Amazon 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, Amazon 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 Firehose stream. Amazon 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 Amazon 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 Amazon Data Firehose.

JSON

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

```
                "kinesis:GetShardIterator",
                "kinesis:GetRecords",
                "kinesis:ListShards"
            ],
            "Resource": "arn:aws-eusc:kinesis:eusc-de-
east-1:123456789012:stream/stream-name"
        },
        {
           "Effect": "Allow",
           "Action": [
               "kms:Decrypt",
               "kms:GenerateDataKey"
           ],
           "Resource": [
               "arn:aws-eusc:kms:eusc-de-east-1:123456789012:key/key-id"

           ],
           "Condition": {
               "StringEquals": {
                   "kms:ViaService": "s3.eusc-de-east-1.amazonaws.com"
               },
               "StringLike": {
                   "kms:EncryptionContext:aws:s3:arn": "arn:aws-eusc:s3:::amzn-
s3-demo-bucket/prefix*"
               }
           }
        },
        {
           "Effect": "Allow",
           "Action": [
               "logs:PutLogEvents"
           ],
           "Resource": [
               "arn:aws-eusc:logs:eusc-de-east-1:123456789012:log-group:log-
group-name:log-stream:log-stream-name"
           ]
        },
        {
           "Effect": "Allow",
           "Action": [
               "lambda:InvokeFunction",
               "lambda:GetFunctionConfiguration"
           ],
           "Resource": [
```

```
                "arn:aws-eusc:lambda:eusc-de-
east-1:123456789012:function:function-name:function-version"
            ]
        }
    ]
}
```

The policy above 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. If you use Amazon MSK as your source, then you can substitute that statement with the following:

```
{
   "Sid":"",
   "Effect":"Allow",
   "Action":[
      "kafka:GetBootstrapBrokers",
      "kafka:DescribeCluster",
      "kafka:DescribeClusterV2",
      "kafka-cluster:Connect"
   ],
   "Resource":"arn:aws-eusc:kafka:{{mskClusterRegion}}:{{mskClusterAccount}}:cluster/
{{mskClusterName}}/{{clusterUUID}}"
},
{
   "Sid":"",
   "Effect":"Allow",
   "Action":[
      "kafka-cluster:DescribeTopic",
      "kafka-cluster:DescribeTopicDynamicConfiguration",
      "kafka-cluster:ReadData"
   ],
   "Resource":"arn:aws-eusc:kafka:{{mskClusterRegion}}:{{mskClusterAccount}}:topic/
{{mskClusterName}}/{{clusterUUID}}/{{mskTopicName}}"
},
{
   "Sid":"",
   "Effect":"Allow",
   "Action":[
      "kafka-cluster:DescribeGroup"
   ],
   "Resource":"arn:aws-eusc:kafka:{{mskClusterRegion}}:{{mskClusterAccount}}:group/
{{mskClusterName}}/{{clusterUUID}}/*"
```

```
}
```

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 Amazon Data Firehose access to an Amazon S3 destination in another account, see the section called "Cross-account delivery to an Amazon S3 destination".

Grant Firehose access to Amazon S3 Tables

Firehose needs an IAM role with specific permissions to access AWS AWS Glue tables and write data to tables in an Amazon S3 table bucket. To write to tables in an Amazon S3 table bucket, you must also provide the IAM role with the required permissions. The permissions required for Amazon S3 Tables catalog depend on the access control mode you use:

- **IAM access control** – The Firehose delivery role needs IAM permissions directly on Amazon S3 Tables resources.
- **Lake Formation access control** – The Firehose delivery role needs AWS AWS Lake Formation permissions for managing access to your table resources. AWS Lake Formation uses its own permissions model that enables fine-grained access control for Data Catalog resources.

You configure this IAM role when you create a Firehose stream. Choose the tab that corresponds to your access control mode.

IAM access control

If you are using IAM access control mode (without AWS Lake Formation), the Firehose delivery role needs IAM permissions directly on Amazon S3 Tables resources and AWS Glue Data Catalog objects.

Sign in to the AWS Management Console and open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.

Create a policy and choose **JSON** in the policy editor. Add the following inline policy that grants the required permissions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "S3TablesAccessPermission",
```

```
            "Effect": "Allow",
            "Action": [
                "s3tables:GetNamespace",
                "s3tables:GetTable",
                "s3tables:GetTableData",
                "s3tables:GetTableMetadataLocation",
                "s3tables:PutTableData",
                "s3tables:UpdateTableMetadataLocation"
            ],
            "Resource": [
                "arn:aws:s3tables:region:account-id:bucket/*",
                "arn:aws:s3tables:region:account-id:bucket/*/table/*"
            ]
        },
        {
            "Sid": "S3TableBucketAccessPermission",
            "Effect": "Allow",
            "Action": [
                "s3tables:GetTableBucket"
            ],
            "Resource": "arn:aws:s3tables:region:account-id:bucket/*"
        },
        {
            "Sid": "GlueCatalogAccessPermission",
            "Effect": "Allow",
            "Action": [
                "glue:GetDatabase",
                "glue:GetDatabases",
                "glue:GetTable",
                "glue:GetTables",
                "glue:UpdateTable"
            ],
            "Resource": [
                "arn:aws:glue:region:account-id:catalog",
                "arn:aws:glue:region:account-id:catalog/s3tablescatalog",
                "arn:aws:glue:region:account-id:catalog/s3tablescatalog/*",
                "arn:aws:glue:region:account-id:database/*",
                "arn:aws:glue:region:account-id:table/*/*"
            ]
        },
        {
            "Sid": "S3DeliveryErrorBucketPermission",
            "Effect": "Allow",
            "Action": [
```

```
                "s3:AbortMultipartUpload",
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:ListBucketMultipartUploads",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::error-delivery-bucket",
                "arn:aws:s3:::error-delivery-bucket/*"
            ]
        },
        {
            "Sid": "RequiredWhenUsingKinesisDataStreamsAsSource",
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStream",
                "kinesis:GetShardIterator",
                "kinesis:GetRecords",
                "kinesis:ListShards"
            ],
            "Resource": "arn:aws:kinesis:region:account-id:stream/stream-name"
        },
        {
            "Sid": "KMSPermissionForS3TablesEncryption",
            "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:s3tables:region:account-id:bucket/*/table/*"
                }
            }
        },
        {
```

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

The policy has statements that allow access to Amazon Kinesis Data Streams, invoking Lambda functions, and access to AWS KMS keys. If you don't use any of these resources, you can remove the respective statements. If error logging is enabled, Amazon Data Firehose also sends data delivery errors to your CloudWatch log group and streams. You must configure log group and log stream names to use this option. For log group and log stream names, see Monitor Amazon Data Firehose Using CloudWatch Logs.

In the inline policy, replace the placeholder values with your actual resource names, AWS account number, and Region.

After you create the policy, open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/ and create an IAM role with **AWS service** as the **Trusted entity type**.

For **Service or use case**, choose **Kinesis**. For **Use case** , choose **Kinesis Firehose**.

On the next page, choose the policy created in the previous step to attach to this role. On the review page, you will find trust policy already attached to this role giving permissions to the Firehose service to assume this role. When you create the role, Amazon Data Firehose can assume it to perform required operations on AWS Glue and Amazon S3 Tables. Add the Firehose service principal to the trust policy of the role that is created. For more information, see Allow Firehose to assume an IAM role.

Lake Formation access control

If you are using AWS Lake Formation access control mode, the Firehose delivery role needs AWS Lake Formation permissions for credential vending in addition to the IAM policy.

Sign in to the AWS Management Console and open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.

Create a policy and choose **JSON** in the policy editor. Add the following inline policy that grants Amazon S3 permissions such as read/write permissions, permissions to update the table in the data catalog, and others.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "S3TableAccessViaGlueFederation",
      "Effect": "Allow",
      "Action": [
        "glue:GetTable",
        "glue:GetDatabase",
        "glue:UpdateTable"
      ],
      "Resource": [
        "arn:aws-eusc:glue:eusc-de-east-1:123456789012:catalog/s3tablescatalog/
*",
        "arn:aws-eusc:glue:eusc-de-east-1:123456789012:catalog/s3tablescatalog",
        "arn:aws-eusc:glue:eusc-de-east-1:123456789012:catalog",
        "arn:aws-eusc:glue:eusc-de-east-1:123456789012:database/*",
        "arn:aws-eusc:glue:eusc-de-east-1:123456789012:table/*/*"
      ]
    },
    {
      "Sid": "S3DeliveryErrorBucketPermission",
      "Effect": "Allow",
      "Action": [
        "s3:AbortMultipartUpload",
        "s3:GetBucketLocation",
        "s3:GetObject",
        "s3:ListBucket",
```

```
        "s3:ListBucketMultipartUploads",
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws-eusc:s3:::<error delivery bucket>",
        "arn:aws-eusc:s3:::<error delivery bucket>/*"
      ]
    },
    {
      "Sid": "RequiredWhenUsingKinesisDataStreamsAsSource",
      "Effect": "Allow",
      "Action": [
        "kinesis:DescribeStream",
        "kinesis:GetShardIterator",
        "kinesis:GetRecords",
        "kinesis:ListShards"
      ],
      "Resource": "arn:aws-eusc:kinesis:eusc-de-east-1:123456789012:stream/
<stream-name>"
    },
    {
      "Sid":
 "RequiredWhenDoingMetadataReadsANDDataAndMetadataWriteViaLakeformation",
      "Effect": "Allow",
      "Action": [
        "lakeformation:GetDataAccess"
      ],
      "Resource": "*"
    },
    {
      "Sid": "RequiredWhenUsingKMSEncryptionForS3ErrorBucketDelivery",
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
      ],
      "Resource": [
        "arn:aws-eusc:kms:eusc-de-east-1:123456789012:key/<KMS-key-id>"
      ],
      "Condition": {
        "StringEquals": {
          "kms:ViaService": "s3.eusc-de-east-1.amazonaws.com"
        },
        "StringLike": {
```

```
          "kms:EncryptionContext:aws:s3:arn": "arn:aws-eusc:s3:::<error delivery
 bucket>/prefix*"
        }
      }
    },
    {
      "Sid": "LoggingInCloudWatch",
      "Effect": "Allow",
      "Action": [
        "logs:PutLogEvents"
      ],
      "Resource": [
        "arn:aws-eusc:logs:eusc-de-east-1:123456789012:log-group:log-group-
name>:log-stream:<log-stream-name>"
      ]
    },
    {
      "Sid": "RequiredWhenAttachingLambdaToFirehose",
      "Effect": "Allow",
      "Action": [
        "lambda:InvokeFunction",
        "lambda:GetFunctionConfiguration"
      ],
      "Resource": [
        "arn:aws-eusc:lambda:eusc-de-east-1:123456789012:function:<function-
name>:<function-version>"
      ]
    }
  ]
}
```

The policy has statements that allows access to Amazon Kinesis Data Streams, invoking Lambda functions, and access to AWS KMS keys. If you don't use any of these resources, you can remove the respective statements. If error logging is enabled, Amazon Data Firehose also sends data delivery errors to your CloudWatch log group and streams. You must configure log group and log stream names to use this option. For log group and log stream names, see Monitor Amazon Data Firehose Using CloudWatch Logs.

In the inline policies, replace `<error delivery bucket>` with your Amazon S3 bucket name, `aws-account-id` and Region with a valid AWS account number and Region of the resource.

In addition to the IAM policy, you must also grant the Firehose delivery role the required permissions in AWS Lake Formation. For more information, see Grant permissions on tables.

After you create the policy, open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/ and create an IAM role with **AWS service** as the **Trusted entity type**.

For **Service or use case**, choose **Kinesis**. For **Use case** , choose **Kinesis Firehose**.

On the next page, choose the policy created in the previous step to attach to this role. On the review page, you will find trust policy already attached to this role giving permissions to the Firehose service to assume this role. When you create the role, Amazon Data Firehose can assume it to perform required operations on AWS Glue and S3 buckets. Add the Firehose service principal to the trust policy of the role that is created. For more information, see Allow Firehose to assume an IAM role.

Grant Firehose access to an Apache Iceberg Tables destination

You must have an IAM role before you create a Firehose stream and Apache Iceberg Tables using AWS Glue. Use the following steps to create a policy and an IAM role. Firehose assumes this IAM role and performs the required actions.

1. Sign in to the AWS Management Console and open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
2. Create a policy and choose **JSON** in policy editor.
3. Add the following inline policy that grants Amazon S3 permissions like the read/write permissions, permissions to update the table in the data catalog etc.

   This policy has a statement that allows access to Amazon Kinesis Data Streams, invoking Lambda functions, and access to KMS keys. If you don't use any of these resources, you can remove the respective statements.

   If error logging is enabled, Firehose also sends data delivery errors to your CloudWatch log group and streams. For this you must configure log group and log stream names. For log group and log stream names, see Monitor Amazon Data Firehose Using CloudWatch Logs.
4. In the inline policies, replace *amzn-s3-demo-bucket* with your Amazon S3 bucket name, aws-account-id and Region with a valid AWS account number and Region of the resources.

> **Note**
>
> This role gives permission to all databases and tables in your data catalog. If you want, you can give permissions only to specific tables and databases.

5. After you create the policy, open the IAM console and create an IAM role with **AWS service** as the **Trusted entity type**.
6. For **Service or use case**, choose **Kinesis**. For **Use case** choose **Kinesis Firehose**.
7. On the next page, choose the policy created in the previous step to attach to this role. On the review page, you will find trust policy already attached to this role giving permissions to Firehose service to assume this role. When you create the role, Amazon Data Firehose can assume it to perform required operations on AWS Glue and S3 buckets.

Grant Firehose access to an Amazon Redshift destination

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

Topics

- IAM role and access policy
- VPC access to an Amazon Redshift provisioned cluster or Amazon Redshift Serverless workgroup

IAM role and access policy

When you're using an Amazon Redshift destination, Amazon 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. Amazon Data Firehose then loads the data from the S3 bucket to your Amazon Redshift provisioned cluster or Amazon Redshift Serverless workgroup. If error logging is enabled, Amazon Data Firehose also sends data delivery errors to your CloudWatch log group and streams. Amazon Data Firehose uses the specified Amazon Redshift user name and password to access your provisioned cluster or Amazon Redshift Serverless workgroup, 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 Firehose stream.

Use the following access policy to enable Amazon 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 Amazon 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.

JSON

```
{
"Version":"2012-10-17",
    "Statement":
    [
        {
            "Effect": "Allow",
            "Action": [
                "s3:AbortMultipartUpload",
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:ListBucketMultipartUploads",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws-eusc:s3:::amzn-s3-demo-bucket",
                "arn:aws-eusc:s3:::amzn-s3-demo-bucket/*"
            ]
        },
        {
           "Effect": "Allow",
           "Action": [
               "kms:Decrypt",
               "kms:GenerateDataKey"
           ],
           "Resource": [
               "arn:aws-eusc:kms:eusc-de-east-1:123456789012:key/key-id"

           ],
           "Condition": {
               "StringEquals": {
                   "kms:ViaService": "s3.eusc-de-east-1.amazonaws.com"
               },
               "StringLike": {
                   "kms:EncryptionContext:aws:s3:arn": "arn:aws-eusc:s3:::amzn-
s3-demo-bucket/prefix*"
```

```
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStream",
                "kinesis:GetShardIterator",
                "kinesis:GetRecords",
                "kinesis:ListShards"
            ],
            "Resource": "arn:aws-eusc:kinesis:eusc-de-
east-1:123456789012:stream/stream-name"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws-eusc:logs:eusc-de-east-1:123456789012:log-group:log-
group-name:log-stream:log-stream-name"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "lambda:InvokeFunction",
                "lambda:GetFunctionConfiguration"
            ],
            "Resource": [
                "arn:aws-eusc:lambda:eusc-de-
east-1:123456789012: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 provisioned cluster or Amazon Redshift Serverless workgroup

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

Region	CIDR blocks
US East (Ohio)	`13.58.135.96/27`
US East (N. Virginia)	`52.70.63.192/27`
US West (N. California)	`13.57.135.192/27`
US West (Oregon)	`52.89.255.224/27`
AWS GovCloud (US-East)	`18.253.138.96/27`
AWS GovCloud (US-West)	`52.61.204.160/27`
Canada (Central)	`35.183.92.128/27`
Canada West (Calgary)	`40.176.98.192/27`
Asia Pacific (Hong Kong)	`18.162.221.32/27`
Asia Pacific (Mumbai)	`13.232.67.32/27`
Asia Pacific (Hyderabad)	`18.60.192.128/27`
Asia Pacific (Seoul)	`13.209.1.64/27`
Asia Pacific (Singapore)	`13.228.64.192/27`
Asia Pacific (Sydney)	`13.210.67.224/27`
Asia Pacific (Jakarta)	`108.136.221.64/27`

Region	CIDR blocks
Asia Pacific (Tokyo)	13.113.196.224/27
Asia Pacific (Osaka)	13.208.177.192/27
Asia Pacific (Thailand)	43.208.112.96/27
Asia Pacific (Taipei)	43.212.53.160/27
China (Beijing)	52.81.151.32/27
China (Ningxia)	161.189.23.64/27
Europe (Zurich)	16.62.183.32/27
Europe (Frankfurt)	35.158.127.160/27
Europe (Ireland)	52.19.239.192/27
Europe (London)	18.130.1.96/27
Europe (Paris)	35.180.1.96/27
Europe (Stockholm)	13.53.63.224/27
Europe (Spain)	18.100.71.96/27
Middle East (Bahrain)	15.185.91.0/27
Mexico (Central)	78.12.207.32/27
South America (São Paulo)	18.228.1.128/27
Europe (Milan)	15.161.135.128/27
Africa (Cape Town)	13.244.121.224/27
Middle East (UAE)	3.28.159.32/27
Israel (Tel Aviv)	51.16.102.0/27

Region	CIDR blocks
Asia Pacific (Melbourne)	`16.50.161.128/27`
Asia Pacific (Malaysia)	`43.216.58.0/27`

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 Firehose access to a public OpenSearch Service destination

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

Use the following access policy to enable Amazon 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 Amazon 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.

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 Amazon Data Firehose access to an OpenSearch Service cluster in another account, see the section called “Cross-account delivery to an OpenSearch Service destination”.

Grant Firehose access to an OpenSearch Service destination in a VPC

If your OpenSearch Service domain is in a VPC, make sure you give Amazon Data Firehose the permissions that are described in the previous section. In addition, you need to give Amazon 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`

> **Important**
>
> Do not revoke these permissions after you create the Firehose stream. If you revoke these permissions, your Firehose stream will be degraded or stop delivering data to your OpenSearch service domain whenever the service attempts to query or update ENIs.

> **Important**
>
> When you specify subnets for delivering data to the destination in a private VPC, make sure you have enough number of free IP addresses in chosen subnets. If there is no available free IP address in a specified subnet, Firehose cannot create or add ENIs for the data delivery in the private VPC, and the delivery will be degraded or fail.

When you create or update your Firehose stream, you specify a security group for 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 Firehose stream. If you use the same security group for both your Firehose 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 Firehose access to a public OpenSearch Serverless destination

When you're using an OpenSearch Serverless destination, Amazon Data Firehose delivers data to your OpenSearch Serverless collection, and concurrently backs up failed or all documents to

your S3 bucket. If error logging is enabled, Amazon Data Firehose also sends data delivery errors to your CloudWatch log group and streams. Amazon Data Firehose uses an IAM role to access the specified OpenSearch Serverless collection, S3 bucket, AWS KMS key, and CloudWatch log group and streams. You are required to have an IAM role when creating a Firehose stream.

Use the following access policy to enable Amazon Data Firehose to access your S3 bucket, OpenSearch Serverless 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 Amazon 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.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:AbortMultipartUpload",
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:ListBucketMultipartUploads",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws-eusc:s3:::amzn-s3-demo-bucket",
                "arn:aws-eusc:s3:::amzn-s3-demo-bucket/*"
            ]
        },
        {
           "Effect": "Allow",
           "Action": [
               "kms:Decrypt",
               "kms:GenerateDataKey"
           ],
           "Resource": [
               "arn:aws-eusc:kms:eusc-de-east-1:123456789012:key/key-id"
```

```
            ],
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "s3.eusc-de-east-1.amazonaws.com"
                },
                "StringLike": {
                    "kms:EncryptionContext:aws:s3:arn": "arn:aws-eusc:s3:::amzn-
s3-demo-bucket/prefix*"
                }
            }
        },
       {
          "Effect": "Allow",
          "Action": [
              "kinesis:DescribeStream",
              "kinesis:GetShardIterator",
              "kinesis:GetRecords",
              "kinesis:ListShards"
          ],
          "Resource": "arn:aws-eusc:kinesis:eusc-de-
east-1:123456789012:stream/stream-name"
       },
       {
          "Effect": "Allow",
          "Action": [
              "logs:PutLogEvents"
          ],
          "Resource": [
              "arn:aws-eusc:logs:eusc-de-east-1:123456789012:log-group:log-group-
name:log-stream:log-stream-name"
          ]
       },
       {
          "Effect": "Allow",
          "Action": [
              "lambda:InvokeFunction",
              "lambda:GetFunctionConfiguration"
          ],
          "Resource": [
              "arn:aws-eusc:lambda:eusc-de-east-1:123456789012:function:function-
name:function-version"
          ]
       },
       {
```

```
        "Effect": "Allow",
        "Action": "aoss:APIAccessAll",
        "Resource": "arn:aws-eusc:aoss:eusc-de-
east-1:123456789012:collection/collection-id"
      }
    ]
}
```

In addition to the policy above, you must also configure Amazon Data Firehose to have the following minimum permissions assigned in a data access policy:

```
[
   {
      "Rules":[
         {
            "ResourceType":"index",
            "Resource":[
               "index/target-collection/target-index"
            ],
            "Permission":[
               "aoss:WriteDocument",
               "aoss:UpdateIndex",
               "aoss:CreateIndex"
            ]
         }
      ],
      "Principal":[
         "arn:aws-eusc:sts::123456789012:assumed-role/firehose-delivery-role-name/*"
      ]
   }
]
```

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*.

Grant Firehose access to an OpenSearch Serverless destination in a VPC

If your OpenSearch Serverless collection is in a VPC, make sure you give Amazon Data Firehose the permissions that are described in the previous section. In addition, you need to give Amazon Data

Firehose the following permissions to enable it to access your OpenSearch Serverless collection's VPC.

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

> **Important**
>
> Do not revoke these permissions after you create the Firehose stream. If you revoke these permissions, your Firehose stream will be degraded or stop delivering data to your OpenSearch service domain whenever the service attempts to query or update ENIs.

> **Important**
>
> When you specify subnets for delivering data to the destination in a private VPC, make sure you have enough number of free IP addresses in chosen subnets. If there is no available free IP address in a specified subnet, Firehose cannot create or add ENIs for the data delivery in the private VPC, and the delivery will be degraded or fail.

When you create or update your Firehose stream, you specify a security group for Firehose to use when it sends data to your OpenSearch Serverless collection. You can use the same security group that the OpenSearch Serverless collection uses or a different one. If you specify a different security group, ensure that it allows outbound HTTPS traffic to the OpenSearch Serverless collection's security group. Also ensure that the OpenSearch Serverless collection's security group allows HTTPS traffic from the security group you specified when you configured your Firehose stream. If you use the same security group for both your Firehose stream and the OpenSearch Serverless

collection, 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 Firehose access to a Splunk destination

When you're using a Splunk destination, Amazon 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, 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 or an Application Load Balancer. Also, enable duration-based sticky sessions with cookie expiration disabled for Classic Load Balancer and expiration is set to the maximum (7 days) for Application Load Balancer. For information about how to do this, see Duration-Based Session Stickiness for Classic Load Balancer or an Application Load Balancer.

You must have an IAM role when you create a Firehose stream. 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 Amazon 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 Amazon Data Firehose. This policy also grants Amazon 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. Amazon Data Firehose doesn't use IAM to access Splunk. For accessing Splunk, it uses your HEC token.

JSON

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

```
                "s3:GetObject",
                "s3:ListBucket",
                "s3:ListBucketMultipartUploads",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws-eusc:s3:::amzn-s3-demo-bucket",
                "arn:aws-eusc:s3:::amzn-s3-demo-bucket/*"
            ]
        },
        {
           "Effect": "Allow",
           "Action": [
               "kms:Decrypt",
               "kms:GenerateDataKey"
           ],
           "Resource": [
               "arn:aws-eusc:kms:eusc-de-east-1:123456789012:key/key-id"

           ],
           "Condition": {
               "StringEquals": {
                   "kms:ViaService": "s3.eusc-de-east-1.amazonaws.com"
               },
               "StringLike": {
                   "kms:EncryptionContext:aws:s3:arn": "arn:aws-eusc:s3:::amzn-
s3-demo-bucket/prefix*"
               }
           }
        },
        {
           "Effect": "Allow",
           "Action": [
               "kinesis:DescribeStream",
               "kinesis:GetShardIterator",
               "kinesis:GetRecords",
               "kinesis:ListShards"
           ],
           "Resource": "arn:aws-eusc:kinesis:eusc-de-
east-1:123456789012:stream/stream-name"
        },
        {
           "Effect": "Allow",
           "Action": [
```

```
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws-eusc:logs:eusc-de-east-1:123456789012:log-group:log-
group-name:log-stream:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "lambda:InvokeFunction",
                "lambda:GetFunctionConfiguration"
            ],
            "Resource": [
                "arn:aws-eusc:lambda:eusc-de-
east-1:123456789012: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*.

Accessing Splunk in VPC

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

Region	CIDR blocks
US East (Ohio)	`18.216.68.160/27, 18.216.170.64/27, 18.216.170.96/27 \`
US East (N. Virginia)	`34.238.188.128/26, 34.238.188.192/26, 34.238.195.0/26`
US West (N. California)	`13.57.180.0/26`

Region	CIDR blocks
US West (Oregon)	34.216.24.32/27, 34.216.24.192/27, 34.216.24.224/27
AWS GovCloud (US-East)	18.253.138.192/26
AWS GovCloud (US-West)	52.61.204.192/26
Asia Pacific (Hong Kong)	18.162.221.64/26
Asia Pacific (Taipei)	43.212.53.192/26
Asia Pacific (Mumbai)	13.232.67.64/26
Asia Pacific (Seoul)	13.209.71.0/26
Asia Pacific (Singapore)	13.229.187.128/26
Asia Pacific (Sydney)	13.211.12.0/26
Asia Pacific (Thailand)	43.208.112.128/26
Asia Pacific (Tokyo)	13.230.21.0/27, 13.230.21.32/27
Canada (Central)	35.183.92.64/26
Canada West (Calgary)	40.176.98.128/26
Europe (Frankfurt)	18.194.95.192/27, 18.194.95.224/27, 18.195.48.0/27
Europe (Ireland)	34.241.197.32/27, 34.241.197.64/27, 34.241.197.96/27
Europe (London)	18.130.91.0/26
Europe (Paris)	35.180.112.0/26
Europe (Spain)	18.100.194.0/26

Region	CIDR blocks
Europe (Stockholm)	13.53.191.0/26
Middle East (Bahrain)	15.185.91.64/26
Mexico (Central)	78.12.207.64/26
South America (São Paulo)	18.228.1.192/26
Europe (Milan)	15.161.135.192/26
Africa (Cape Town)	13.244.165.128/26
Asia Pacific (Osaka)	13.208.217.0/26
China (Beijing)	52.81.151.64/26
China (Ningxia)	161.189.23.128/26
Asia Pacific (Jakarta)	108.136.221.128/26
Middle East (UAE)	3.28.159.64/26
Israel (Tel Aviv)	51.16.102.64/26
Europe (Zurich)	16.62.183.64/26
Asia Pacific (Hyderabad)	18.60.192.192/26
Asia Pacific (Melbourne)	16.50.161.192/26
Asia Pacific (Malaysia)	43.216.44.192/26
Asia Pacific (New Zealand)	3.102.119.128/26

Ingest VPC flow logs into Splunk using Amazon Data Firehose

To learn more about how to create a VPC flow log subscription, publish to Firehose, and send the VPC flow logs to a supported destination see Ingest VPC flow logs into Splunk using Amazon Data Firehose.

Accessing Snowflake or HTTP end point

There is no subset of AWS IP address ranges specific to Amazon Data Firehose when the destination is HTTP end point or Snowflake public clusters.

To add Firehose to an allow list for public Snowflake clusters or to your public HTTP or HTTPS endpoints, add all the current AWS IP address ranges to your ingress rules.

> **Note**
>
> Notifications aren't always sourced from IP addresses in the same AWS Region as their associated topic. You must include the AWS IP address range for all Regions.

Grant Firehose access to a Snowflake destination

When you're using Snowflake as a destination, Firehose delivers data to a Snowflake account using your Snowflake account URL. It also backs up error data to the Amazon Simple Storage Service bucket that you specify, and you can optionally use an AWS Key Management Service key that you own for Amazon S3 server-side encryption. If error logging is enabled, Firehose sends data delivery errors to your CloudWatch Logs streams.

You must have an IAM role before you create a Firehose stream. Firehose assumes that IAM role and gains access to the specified bucket, key, and CloudWatch Logs group and streams. Use the following access policy to enable Firehose to access your S3 bucket. If you don't own the S3 bucket, add `s3:PutObjectAcl` to the list of Amazon Simple Storage Service actions, which grants the bucket owner full access to the objects delivered by Firehose. This policy also grants Firehose access to CloudWatch for error logging. 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. Firehose doesn't use IAM to access Snowflake. For accessing Snowflake, it uses your Snowflake account Url and PrivateLink Vpce Id in the case of a private cluster.

JSON

```
{
"Version":"2012-10-17",
    "Statement":
    [
        {
"Effect": "Allow",
            "Action": [
                "s3:AbortMultipartUpload",
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:ListBucketMultipartUploads",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws-eusc:s3:::amzn-s3-demo-bucket",
                "arn:aws-eusc:s3:::amzn-s3-demo-bucket/*"
            ]
        },
        {
"Effect": "Allow",
           "Action": [
               "kms:Decrypt",
               "kms:GenerateDataKey"
           ],
           "Resource": [
               "arn:aws-eusc:kms:eusc-de-east-1:123456789012:key/key-id"

           ],
           "Condition": {
"StringEquals": {
"kms:ViaService": "s3.eusc-de-east-1.amazonaws.com"
               },
               "StringLike": {
"kms:EncryptionContext:aws:s3:arn": "arn:aws-eusc:s3:::amzn-s3-demo-bucket/
prefix*"
               }
           }
        },
        {
"Effect": "Allow",
```

```
            "Action": [
                "kinesis:DescribeStream",
                "kinesis:GetShardIterator",
                "kinesis:GetRecords",
                "kinesis:ListShards"
            ],
            "Resource": "arn:aws-eusc:kinesis:eusc-de-east-1:123456789012:stream/
stream-name"
        },
        {
"Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws-eusc:logs:eusc-de-east-1:123456789012:log-group:log-
group-name:log-stream:*"
            ]
        }
    ]
}
```

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*.

Accessing Snowflake in VPC

If your Snowflake cluster is private link enabled, Firehose will use one of the following VPC endpoints at time of private link creation to deliver data to your private cluster without going through public internet. For this, create Snowflake network rules to allow ingress from the following `AwsVpceIds` for the AWS Region your cluster is in. For more information, see Creating network rule in *Snowflake User Guide*.

VPC Endpoint Ids to use based on Regions your cluster is in

AWS Region	VPCE IDs
US East (Ohio)	vpce-0d96cafcd96a50aeb vpce-0cec34343d48f537b
US East (N. Virginia)	vpce-0b4d7e8478e141ba8

AWS Region	VPCE IDs
	vpce-0b75cd681fb507352 vpce-01c03e63820ec00d8 vpce-0c2cfc51dc2882422 vpce-06ca862f019e4e056 vpce-020cda0cfa63f8d1c vpce-0b80504a1a783cd70 vpce-0289b9ff0b5259a96 vpce-0d7add8628bd69a12 vpce-02bfb5966cc59b2af vpce-09e707674af878bf2 vpce-049b52e96cc1a2165 vpce-0bb6c7b7a8a86cdbb vpce-03b22d599f51e80f3 vpce-01d60dc60fc106fe1 vpce-0186d20a4b24ecbef vpce-0533906401a36e416 vpce-05111fb13d396710e vpce-0694613f4fbd6f514 vpce-09b21cb25fe4cc4f4 vpce-06029c3550e4d2399 vpce-00961862a21b033da

AWS Region	VPCE IDs
	vpce-01620b9ae33273587 vpce-078cf4ec226880ac9 vpce-0d711bf076ce56381 vpce-066b7e13cbfca6f6e vpce-0674541252d9ccc26 vpce-03540b88dedb4b000 vpce-0b1828e79ad394b95 vpce-0dc0e6f001fb1a60d vpce-0d8f82e71a244098a vpce-00e374d9e3f1af5ce vpce-0c1e3d6631ddb442f
US West (Oregon)	vpce-0f60f72da4cd1e4e7 vpce-0c60d21eb8b1669fd vpce-01c4e3e29afdafbef vpce-0cc6bf2a88da139de vpce-0797e08e169e50662 vpce-033cbe480381b5c0e vpce-00debbdd8f9eb10a5 vpce-08ec2f386c809e889 vpce-0856d14310857b545

AWS Region	VPCE IDs
Europe (Frankfurt)	vpce-068dbb7d71c9460fb vpce-0a7a7f095942d4ec9
Europe (Ireland)	vpce-06857e59c005a6276 vpce-04390f4f8778b75f2 vpce-011fd2b1f0aa172fd
Asia Pacific (Tokyo)	vpce-06369e5258144e68a vpce-0f2363cdb8926fbe8
Asia Pacific (Singapore)	vpce-049cd46cce7a12d52 vpce-0e8965a1a4bdb8941
Asia Pacific (Seoul)	vpce-0aa444d9001e1faa1 vpce-04a49d4dcfd02b884
Asia Pacific (Sydney)	vpce-048a60a182c52be63 vpce-03c19949787fd1859
Asia Pacific (Mumbai)	vpce-0d68cb822f6f0db68 vpce-0517d32692ffcbde2
Europe (London)	vpce-0fd1874a0ba3b9374 vpce-08091b1a85e206029
South America (Sao Paulo)	vpce-065169b8144e4f12e vpce-0493699f0e5762d63

AWS Region	VPCE IDs
Canada (Central)	vpce-07e6ed81689d5271f vpce-0f53239730541394c
Europe (Paris)	vpce-09419680077e6488a vpce-0ea81ba2c08140c14
Asia Pacific (Osaka)	vpce-0a9f003e6a7e38c05 vpce-02886510b897b1c5a
Europe (Stockholm)	vpce-0d96410833219025a vpce-060a32f9a75ba969f
Asia Pacific (Jakarta)	vpce-00add4b9a25e5c649 vpce-004ae2de34338a856

Grant Firehose access to an HTTP endpoint destination

You can use Amazon Data Firehose to deliver data to any HTTP endpoint destination. Amazon 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, Amazon 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 Firehose stream. Amazon 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 Amazon 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 Amazon Data Firehose. This policy also grants Amazon 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**
>
> Amazon 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 Amazon Data Firehose.

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 Amazon Data Firehose does NOT support data delivery to HTTP endpoints in a VPC.

Cross-account delivery from Amazon MSK

When you're creating a Firehose stream from your Firehose account (for example, Account B) and your source is an MSK cluster in another AWS account (Account A), you must have the following configurations in place.

Account A:

1. In the Amazon MSK console, choose the provisioned cluster and then choose **Properties**.
2. Under **Network settings**, choose **Edit** and turn on **Multi-VPC connectivity**.
3. Under **Security settings** choose **Edit cluster policy**.
   a. If the cluster does not already have a policy configured, check **Include Firehose service principal** and **Enable Firehose cross-account S3 delivery**. The AWS Management Console will automatically generate a policy with the appropriate permissions.

b. If the cluster already has a policy configured, add the following permissions to the existing policy:

```
{
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws-eusc:iam::eusc-de-east-1:role/mskaasTestDeliveryRole"
      },
      "Action": [
        "kafka:GetBootstrapBrokers",
        "kafka:DescribeCluster",
        "kafka:DescribeClusterV2",
        "kafka-cluster:Connect"
      ],
      "Resource": "arn:aws-eusc:kafka:eusc-de-east-1:123456789012:cluster/
DO-NOT-TOUCH-mskaas-provisioned-privateLink/xxxxxxxxx-2f3a-462a-ba09-
xxxxxxxxxx-20"  // ARN of the cluster
    },
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws-eusc::iam::eusc-de-east-1:role/mskaasTestDeliveryRole"
      },
      "Action": [
        "kafka-cluster:DescribeTopic",
        "kafka-cluster:DescribeTopicDynamicConfiguration",
        "kafka-cluster:ReadData"
      ],
      "Resource": "arn:aws-eusc:kafka:eusc-de-east-1:arn:topic/DO-NOT-TOUCH-
mskaas-provisioned-privateLink/xxxxxxxxx-2f3a-462a-ba09-xxxxxxxxxx-20/*"//topic
 of the cluster
    },
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws-eusc:iam::eusc-de-east-1:role/mskaasTestDeliveryRole"
      },
      "Action": "kafka-cluster:DescribeGroup",
      "Resource": "arn:aws-eusc:kafka:eusc-de-east-1:arn:group/DO-NOT-TOUCH-
mskaas-provisioned-privateLink/xxxxxxxxx-2f3a-462a-ba09-xxxxxxxxxx-20/*" //
topic of the cluster
    },
```

```
}
```

4. Under **AWS principal**, enter the principal ID from Account B.
5. Under **Topic**, specify the Apache Kafka topic from which you want your Firehose stream to ingest data. Once the Firehose stream is created, you cannot update this topic.
6. Choose **Save changes**

Account B:

1. In the Firehose console, choose **Create Firehose stream** using Account B.
2. Under **Source**, choose **Amazon Managed Streaming for Apache Kafka**.
3. Under **Source settings**, for the **Amazon Managed Streaming for Apache Kafka cluster**, enter the ARN of the Amazon MSK cluster in Account A.
4. Under **Topic**, specify the Apache Kafka topic from which you want your Firehose stream to ingest data. Once the Firehose stream is created, you cannot update this topic.
5. In **Delivery stream name** specify the name for your Firehose stream.

In Account B when you're creating your Firehose stream, you must have an IAM role (created by default when using the AWS Management Console) that grants the Firehose stream 'read' access to the cross-account Amazon MSK cluster for the configured topic.

The following is what gets configured by the AWS Management Console:

```
{
    "Sid": "",
    "Effect": "Allow",
    "Action": [
        "kafka:GetBootstrapBrokers",
        "kafka:DescribeCluster",
        "kafka:DescribeClusterV2",
        "kafka-cluster:Connect"
        ],
    "Resource": "arn:aws-eusc:kafka:eusc-de-east-1:arn:aws-eusc::cluster/DO-NOT-TOUCH-
mskaas-provisioned-privateLink/xxxxxxxxx-2f3a-462a-ba09-xxxxxxxxxx-20/*" //topic of the
 cluster
    },
    {
    "Sid": "",
```

```
    "Effect": "Allow",
    "Action": [
        "kafka-cluster:DescribeTopic",
        "kafka-cluster:DescribeTopicDynamicConfiguration",
        "kafka-cluster:ReadData"
    ],
    "Resource": "arn:aws-eusc:kafka:eusc-de-east-1:arn:aws-eusc::topic/DO-NOT-
TOUCH-mskaas-provisioned-privateLink/xxxxxxxxx-2f3a-462a-ba09-xxxxxxxxxx-20/
mskaas_test_topic" //topic of the cluster
    },
    {
    "Sid": "",
    "Effect": "Allow",
    "Action": [
        "kafka-cluster:DescribeGroup"
    ],
    "Resource": "arn:aws-eusc:kafka:eusc-de-east-1:arn:aws-eusc::group/DO-NOT-TOUCH-
mskaas-provisioned-privateLink/xxxxxxxxx-2f3a-462a-ba09-xxxxxxxxxx-20/*" //topic of the
 cluster
    },
 }
```

Next, you can complete the optional step of configuring record transformation and record format conversion. For more information, see (Optional) Configure record transformation and format conversion.

Cross-account delivery to an Amazon S3 destination

You can use the AWS CLI or the Amazon Data Firehose APIs to create a Firehose stream in one AWS account with an Amazon S3 destination in a different account. The following procedure shows an example of configuring a Firehose 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 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 Data

Firehose. This permission is required for cross account delivery. Amazon 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.

   JSON

```
{

    "Version":"2012-10-17",
    "Id": "PolicyID",
    "Statement": [
        {
            "Sid": "StmtID",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws-eusc:iam::123456789012:role/iam-role-name"
            },
            "Action": [
                "s3:AbortMultipartUpload",
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:ListBucketMultipartUploads",
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": [
                "arn:aws-eusc:s3:::amzn-s3-demo-bucket",
                "arn:aws-eusc:s3:::amzn-s3-demo-bucket/*"
            ]
        }
    ]
}
```

3. Create a Firehose 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 Amazon Data Firehose APIs to create a Firehose 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 Firehose 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 Firehose access to a public OpenSearch Service destination".
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.

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws-eusc:iam::123456789012:role/firehose_delivery_role "
      },
      "Action": "es:ESHttpGet",
      "Resource": [
        "arn:aws-eusc:es:eusc-de-east-1:123456789012:domain/cross-account-
cluster/_all/_settings",
        "arn:aws-eusc:es:eusc-de-east-1:123456789012:domain/cross-account-
cluster/_cluster/stats",
        "arn:aws-eusc:es:eusc-de-east-1:123456789012:domain/cross-account-
cluster/roletest*/_mapping/roletest",
        "arn:aws-eusc:es:eusc-de-east-1:123456789012:domain/cross-account-
cluster/_nodes",
        "arn:aws-eusc:es:eusc-de-east-1:123456789012:domain/cross-account-
cluster/_nodes/stats",
        "arn:aws-eusc:es:eusc-de-east-1:123456789012:domain/cross-account-
cluster/_nodes/*/stats",
        "arn:aws-eusc:es:eusc-de-east-1:123456789012:domain/cross-account-
cluster/_stats",
        "arn:aws-eusc:es:eusc-de-east-1:123456789012:domain/cross-account-
cluster/roletest*/_stats",
```

```
        "arn:aws-eusc:es:eusc-de-east-1:123456789012:domain/cross-account-
cluster/"
      ]
    }
  ]
}
```

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

> **Note**
>
> To create a Firehose stream in one AWS account with an OpenSearch Service destination in a different account, you must use the AWS CLI or the Amazon 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 Amazon Data Firehose operations based on tag keys and values. The following subsections describe how to do this for the different Amazon 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

For the `CreateDeliveryStream` operation, use the `aws:RequestTag` condition key. In the following example, `MyKey` and `MyValue` represent the key and corresponding value for a tag. For more information, see Understand tag basics

JSON

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

```
            "firehose:CreateDeliveryStream",
            "firehose:TagDeliveryStream"
        ],
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "aws:RequestTag/MyKey": "MyValue"
            }
        }
    }]
}
```

TagDeliveryStream

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

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "firehose:TagDeliveryStream",
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "aws:TagKeys": "MyKey"
                 }
            }
        }
    ]
}
```

UntagDeliveryStream

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

JSON

```
{
    "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 operations

For all 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.

`ListDeliveryStreams`, use the `firehose:ResourceTag` condition key to control access based on the tags on that Firehose stream.

In the following example, `MyKey` and `MyValue` represent the key and corresponding value for a tag. The policy would only apply to Data Firehose streams having a tag named `MyKey` with a value of `MyValue`. For more information about controlling access based on resource tags, see Controlling access to AWS resources using tags in the *IAM User Guide*.

Authenticate with AWS Secrets Manager in Amazon Data Firehose

Amazon Data Firehose integrates with AWS Secrets Manager to provide secure access to your secrets and automate credential rotation. This integration allows Firehose to retrieve a secret from Secrets Manager at runtime to connect to previously mentioned streaming destinations and deliver your data streams. With this, your secrets are not visible in plain text during stream creation workflow either in AWS Management Console or API parameters. It provides a secure practice to manage your secrets and relieves you from complex credential management activities such as setting up custom Lambda functions to manage password rotations.

For more information, see the AWS Secrets Manager User Guide.

Topics

- Understand secrets
- Create a secret
- Use the secret
- Rotate the secret

Understand secrets

A secret can be a password, a set of credentials such as a user name and password, an OAuth token, or other secret information that you store in an encrypted form in Secrets Manager.

For each destination, you must specify the secret key-value pair in the correct JSON format as shown in the following section. Amazon Data Firehose will fail to connect to your destination if your secret doesn't have the correct JSON format as per the destination.

Format of secret for databases such as MySQL and PostgreSQL

```
{
    "username":  "<username>",
    "password":  "<password>"
}
```

Format of secret for Amazon Redshift Provisioned cluster and Amazon Redshift Serverless workgroup

```
{
    "username":  "<username>",
    "password":  "<password>"
}
```

Format of secret for Splunk

```
{
    "hec_token":  "<hec token>"
}
```

Format of secret for Snowflake

```
{
    "user":  "<snowflake-username>",
    "private_key":  "<snowflake-private-key>", // without the beginning and ending
 private key, remove all spaces and newlines
    "key_passphrase":  "<snowflake-private-key-passphrase>" // optional
}
```

Format of secret for HTTP endpoint, Coralogix, Datadog, Dynatrace, Elastic, Honeycomb, LogicMonitor, Logz.io, MongoDB Cloud, and New Relic

```
{
    "api_key":  "<apikey>"
}
```

Create a secret

To create a secret, follow the steps in Create an AWS Secrets Manager secret in the *AWS Secrets Manager User Guide*.

Use the secret

We recommend that you use AWS Secrets Manager to store your credentials or keys to connect to streaming destinations such as Amazon Redshift, HTTP endpoint, Snowflake, Splunk, Coralogix, Datadog, Dynatrace, Elastic, Honeycomb, LogicMonitor, Logz.io, MongoDB Cloud, and New Relic.

You can configure authentication with Secrets Manager for these destinations through the AWS Management Console at the time of Firehose stream creation. For more information, see

Configure destination settings. Alternatively, you can also use the CreateDeliveryStream and UpdateDestination API operations to configure authentication with Secrets Manager.

Firehose caches the secrets with an encryption and uses them for every connection to destinations. It refreshes the cache every 10 minutes to ensure that the latest credentials are used.

You can choose to turn off the capability of retrieving secrets from Secrets Manager at any time during the lifecycle of the stream. If you don't want to use Secrets Manager to retrieve secrets, you can use the username/password or API key instead.

> **Note**
>
> Although, there is no additional cost for this feature in Firehose, you are billed for access and maintenance of Secrets Manager. For more information, see AWS Secrets Manager pricing page.

Grant access to Firehose to retrieve the secret

For Firehose to retrieve a secret from AWS Secrets Manager, you must provide Firehose the required permissions to access the secret and the key that encrypts your secret.

When using AWS Secrets Manager to store and retrieve secrets, there are a few different configuration options depending on where the secret is stored and how it is encrypted.

- If the secret is stored in the same AWS account as your IAM role and it's encrypted with the default AWS managed key (`aws/secretsmanager`), the IAM role that Firehose assumes only needs `secretsmanager:GetSecretValue` permission on the secret.

```
// secret role policy
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "secretsmanager:GetSecretValue",
            "Resource": "Secret ARN"
        }
    ]
}
```

For more information on IAM policies, see Permissions policy examples for AWS Secrets Manager.

- If the secret is stored in the same account as the role but encrypted with a customer managed key (CMK), the role needs both `secretsmanager:GetSecretValue` and `kms:Decrypt` permissions. The CMK policy also needs to allow the IAM role to perform `kms:Decrypt`.
- If the secret is stored in a different AWS account than your role, and it is encrypted with the default AWS managed key, this configuration is not possible as Secrets Manager does not allow cross-account access when secret is encrypted with AWS managed key.
- If the secret is stored in a different account and encrypted with a CMK, IAM role needs `secretsmanager:GetSecretValue` permission on the secret and `kms:Decrypt` permission on the CMK. The secret's resource policy and the CMK policy in the other account also need to allow the IAM role the necessary permissions. For more information, see Cross-account access.

Rotate the secret

Rotation is when you periodically update a secret. You can configure AWS Secrets Manager to automatically rotate the secret for you on a schedule that you specify. This way, you can replace long-term secrets with short-term ones. This helps to reduce the risk of compromise. For more information, see Rotate AWS Secrets Manager secrets in the *AWS Secrets Manager User Guide*.

Manage IAM roles through Amazon Data Firehose console

Amazon Data Firehose is a fully managed service that delivers real-time streaming data to destinations. You can also configure Firehose to transform and convert the format of your data before delivery. To use these features, you must first provide IAM roles to grant permissions to Firehose when you create or edit a Firehose stream. Firehose uses this IAM role for all the permissions that the Firehose stream needs.

For example, consider a scenario where you create a Firehose stream that delivers data to Amazon S3, and this Firehose stream has Transform source records with AWS Lambda feature enabled. In this case, you must provide IAM roles to grant Firehose permissions to access the S3 bucket and invoke the Lambda function, as shown in the following.

JSON

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

```
    "Statement": [{
        "Sid": "lambdaProcessing",
        "Effect": "Allow",
        "Action": ["lambda:InvokeFunction", "lambda:GetFunctionConfiguration"],
        "Resource": "arn:aws-eusc:lambda:eusc-de-
east-1:123456789012:function:<lambda function name>:<lambda function version>"
    }, {
        "Sid": "s3Permissions",
        "Effect": "Allow",
        "Action": ["s3:AbortMultipartUpload", "s3:GetBucketLocation",
 "s3:GetObject", "s3:ListBucket", "s3:ListBucketMultipartUploads",
 "s3:PutObject"],
        "Resource": ["arn:aws-eusc:s3:::<bucket name>", "arn:aws-
eusc:s3:::<bucket name>/*"]
    }]
}
```

Firehose console allows you to choose how you want to provide these roles. You can choose from one of the following options.

- Choose an existing IAM role
- Create a new IAM role from console

Choose an existing IAM role

You can choose from an existing IAM role. With this option, make sure that the IAM role you choose has a proper trust policy and permissions required for your source and destination. For more information, see Controlling access with Amazon Data Firehose.

Create a new IAM role from console

Alternatively, you could also use the Firehose console to create a new role on your behalf.

When Firehose creates an IAM role on your behalf, the role automatically includes all permission and trust policies that grant the required permissions based on the Firehose stream configuration.

For example, if you didn't enable **Transform source records with AWS Lambda** feature then console generates the following statement in the permission policy.

```
{
```

```
  "Sid": "lambdaProcessing",
  "Effect": "Allow",
   "Action": [
     "lambda:InvokeFunction",
     "lambda:GetFunctionConfiguration"
   ],
   "Resource": "arn:aws-eusc:lambda:eusc-de-east-1123456789012:function:
%FIREHOSE_POLICY_TEMPLATE_PLACEHOLDER%"
}
```

Note

It's safe to ignore the policy statements that contain `%FIREHOSE_POLICY_TEMPLATE_PLACEHOLDER%` as they don't grant permissions on any resources.

The console create and edit Firehose stream workflows also create a trust policy and attach it to the IAM role. The trust policy allows Firehose to assume the IAM role. Following is a example of a trust policy.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [{
        "Sid": "firehoseAssume",
        "Effect": "Allow",
        "Principal": {
            "Service": "firehose.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
    }]
}
```

Important

- You should avoid using the same console-managed IAM role for multiple Firehose streams. Otherwise, the IAM role could become overly permissive or result in errors.

- To use different policy statements within a permission policy from a console-managed IAM role, you can create your own IAM role, and copy the policy statements to a permission policy attached to the new role. To attach the role to the Firehose stream, select the **Choose existing IAM role** option in the **Service access**.
- Console manages any IAM role that contains the string *service-role* in its ARN. When you choose the existing IAM role option, make sure to select an IAM role without the *service-role* string in its ARN so that console doesn't make any changes to it.

Steps to create an IAM role from console

1. Open the Firehose console at https://eusc-de-east-1.console.amazonaws-eusc.eu/firehose/.
2. Choose **Create Firehose stream**.
3. Choose a source and destination. For more information, see Tutorial: Create a Firehose stream from console.
4. Choose the destination settings. For more information, see Configure destination settings.
5. Under **Advanced settings**, for **Service access**, choose **Create or update IAM role**.

   > **Note**
   >
   > This is a default option. To use an existing role, select the **Choose existing IAM role** option. Firehose console won't make any changes to your own role.

6. Choose **Create Firehose stream**.

Edit IAM role from console

When you edit a Firehose stream, Firehose updates the corresponding permission policy accordingly to reflect the configuration and permission changes.

For example, when you edit the Firehose stream and enable **Transform source records with AWS Lambda** feature using the latest version of Lambda function as `exampleLambdaFunction`, you get the following policy statement in the permission policy.

```
{
  "Sid": "lambdaProcessing",
```

```
  "Effect": "Allow",
  "Action": [
    "lambda:InvokeFunction",
    "lambda:GetFunctionConfiguration"
  ],
  "Resource": "arn:aws-eusc:lambda:eusc-de-
east-1:123456789012:function:exampleLambdaFunction:$LATEST"
}
```

> **Important**
>
> A console-managed IAM role is designed to be autonomous. We don't recommend that you modify the permission policy or trust policy outside of the console.

Steps to edit IAM role from console

1. Open the Firehose console at https://eusc-de-east-1.console.amazonaws-eusc.eu/firehose/.
2. Choose **Firehose streams** and choose the name of a Firehose stream you want to update.
3. On the **Configuration** tab, in the **Server access** section, choose **Edit**.
4. Update the IAM role option.

   > **Note**
   >
   > By default, the console always updates an IAM role with the pattern *service-role* in its ARN. When you choose the existing IAM role option, make sure to select an IAM role without the *service-role* string in its ARN so that console doesn't make any changes to it.

5. Choose **Save changes**.

Understand compliance for Amazon Data Firehose

Third-party auditors assess the security and compliance of Amazon 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 Data Firehose is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. If your use of 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 CSPM – 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 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, Data Firehose offers several features to help support your data resiliency and backup needs.

Disaster recovery

Amazon 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, Amazon Data Firehose ensures that the Firehose stream is migrated without any loss of data.

Understand infrastructure security in Amazon Data Firehose

As a managed service, Amazon Data Firehose is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access Firehose through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

> **Note**
>
> For outgoing HTTPS requests, Amazon Data Firehose uses an HTTP library that automatically selects the highest TLS protocol version supported at the destination side.

Using Amazon Data Firehose with AWS PrivateLink

You can use an interface VPC endpoint (AWS PrivateLink) to access Amazon Data Firehose from your VPC without requiring an Internet Gateway or NAT Gateway. Interface VPC endpoints don't require an internet gateway, NAT device, VPN connection, or 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 Firehose

To get started, create an interface VPC endpoint in order for your Amazon 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 Amazon Data Firehose. For more about using policies to control access from a VPC endpoint to Amazon 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 Amazon Data Firehose service. In this example, you use a policy that allows the Lambda function to list the Firehose streams in the current Region but not to describe any Firehose stream.

Create a VPC endpoint

1. Sign in to the AWS Management Console and open the Amazon VPC console at https://eusc-de-east-1.console.amazonaws-eusc.eu/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://eusc-de-east-1.console.amazonaws-eusc.eu/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 `AmazonDataFirehoseReadOnlyAccess`.

> **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://eusc-de-east-1.console.amazonaws-eusc.eu/lambda/.
2. Choose **Create function**.

3. Choose **Author from scratch**.
4. Enter a name for the function, then set **Runtime** to `Python 3.9` or higher.
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 Firehose streams in the Region, as well as the following output:

    ```
    Calling describe_delivery_stream.

    AccessDeniedException

    Access denied test succeeded.
    ```

Supported AWS Regions

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 (Thailand)
- Asia Pacific (Tokyo)
- Asia Pacific (Hong Kong)
- Canada (Central)

- Canada West (Calgary)
- China (Beijing)
- China (Ningxia)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (London)
- Europe (Paris)
- Mexico (Central)
- South America (São Paulo)
- AWS GovCloud (US-East)
- AWS GovCloud (US-West)
- Europe (Spain)
- Middle East (UAE)
- Asia Pacific (Jakarta)
- Asia Pacific (Osaka)
- Israel (Tel Aviv)
- Asia Pacific (Malaysia)

Implement security best practices for Amazon Data Firehose

Amazon 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 Amazon 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 Firehose streams, and your Firehose 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 Firehose 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 Amazon Data Firehose. For more information, see Data protection in Amazon Data Firehose.

Use CloudTrail to monitor API calls

Amazon 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 Amazon Data Firehose.

Using the information collected by CloudTrail, you can determine the request that was made to Amazon 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 “Log Firehose API calls”.

Monitor Amazon Data Firehose

You can monitor Amazon Data Firehose using the following features:

Topics

- Implement best practices with CloudWatch Alarms
- Monitor Amazon Data Firehose with CloudWatch metrics
- Access CloudWatch Metrics for Amazon Data Firehose
- Monitor Amazon Data Firehose Using CloudWatch Logs
- Access CloudWatch logs for Amazon Data Firehose
- Monitor Kinesis Agent health
- Log Amazon Data Firehose API calls with AWS CloudTrail

Implement best practices with CloudWatch Alarms

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

- `DeliveryToS3.DataFreshness`
- `DeliveryToIceberg.DataFreshness`
- `DeliveryToSplunk.DataFreshness`
- `DeliveryToAmazonOpenSearchService.DataFreshness`
- `DeliveryToAmazonOpenSearchServerless.DataFreshness`
- `DeliveryToHttpEndpoint.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 *Troubleshoot errors*.

Monitor Amazon Data Firehose with CloudWatch metrics

> **Important**
>
> Be sure to enable alarms on all CloudWatch metrics that belong to your destination in order to identify errors in timely manner.

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

Amazon Data Firehose collects and publishes CloudWatch metrics every minute. However, if bursts of incoming data occur only for a few seconds, they may not be fully captured or visible in the one-minute metrics. This is because CloudWatch metrics are aggregated from Amazon Data Firehose over one-minute intervals.

The metrics collected for Firehose streams are free of charge. For information about Kinesis agent metrics, see Monitor Kinesis Agent health.

Topics

- CloudWatch metrics for dynamic partitioning
- CloudWatch metrics for data delivery
- Data ingestion metrics
- API-level CloudWatch metrics
- Data Transformation CloudWatch Metrics
- CloudWatch Logs Decompression Metrics
- Format Conversion CloudWatch Metrics
- Server-Side Encryption (SSE) CloudWatch Metrics
- Dimensions for Amazon Data Firehose
- Amazon Data Firehose Usage Metrics

CloudWatch metrics for dynamic partitioning

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

Metric	Description
`ActivePartitionsLimit`	The maximum number of active partitions that a Firehose stream processes before sending data to the error bucket. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`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). Statistics: Minimum, Maximum, Average, Sum, Samples 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 throughput that is being processed per partition. This metric enables you to monitor the per partition throughput. Units: StandardUnit.BytesSecond
`DeliveryToS3.ObjectCount`	Indicates the number of objects that are being delivered to your S3 bucket.

Metric	Description
	Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count

CloudWatch metrics for data delivery

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`, `DeliveryToAmazonOpenSearchService.Success`, or `DeliveryToRedshift.Success`, that doesn't indicate that there's data loss. Amazon 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.

Topics

- Delivery to OpenSearch Service
- Delivery to OpenSearch Serverless
- Delivery to Amazon Redshift
- Delivery to Amazon S3
- Delivery to Snowflake
- Delivery to Splunk
- Delivery to HTTP Endpoints

Delivery to OpenSearch Service

Metric	Description
`DeliveryToAmazonOpenSearchService.Bytes`	The number of bytes indexed to OpenSearch Service over the specified time period. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Bytes

Metric	Description
`DeliveryToAmazonOpenSearchService.DataFreshness`	The age (from getting into Amazon Data Firehose to now) of the oldest record in Amazon Data Firehose. Any record older than this age has been delivered to OpenSearch Service. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Seconds
`DeliveryToAmazonOpenSearchService.Records`	The number of records indexed to OpenSearch Service over the specified time period. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`DeliveryToAmazonOpenSearchService.Success`	The sum of the successfully indexed records.
`DeliveryToS3.Bytes`	The number of bytes delivered to Amazon S3 over the specified time period. Amazon Data Firehose emits this metric only when you enable backup for all documents. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`DeliveryToS3.DataFreshness`	The age (from getting into Amazon Data Firehose to now) of the oldest record in Amazon Data Firehose. Any record older than this age has been delivered to the S3 bucket. Amazon Data Firehose emits this metric only when you enable backup for all documents. Units: Seconds

Metric	Description
`DeliveryToS3.Records`	The number of records delivered to Amazon S3 over the specified time period. Amazon Data Firehose emits this metric only when you enable backup for all documents. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`DeliveryToS3.Success`	The sum of successful Amazon S3 put commands. Amazon Data Firehose always emits this metric regardless of whether backup is enabled for failed documents only or for all documents.
`DeliveryToAmazonOpenSearchService.AuthFailure`	Authentication/authorization error. Verify the OS/ES cluster policy and role permissions. 0 indicates that there is no issue. 1 indicates authentication failure.
`DeliveryToAmazonOpenSearchService.DeliveryRejected`	Delivery rejected error. Verify the OS/ES cluster policy and role permissions. 0 indicates that there is no issue. 1 indicates that there's a delivery failure.

Delivery to OpenSearch Serverless

Metric	Description
`DeliveryToAmazonOpenSearchServerless.Bytes`	The number of bytes indexed to OpenSearch Serverless over the specified time period. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Bytes

Metric	Description
`DeliveryToAmazonOpenSearchServerless.DataFreshness`	The age (from getting into Amazon Data Firehose to now) of the oldest record in Amazon Data Firehose. Any record older than this age has been delivered to OpenSearch Serverless. Units: Seconds
`DeliveryToAmazonOpenSearchServerless.Records`	The number of records indexed to OpenSearch Serverless over the specified time period. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`DeliveryToAmazonOpenSearchServerless.Success`	The sum of the successfully indexed records.
`DeliveryToS3.Bytes`	The number of bytes delivered to Amazon S3 over the specified time period. Amazon Data Firehose emits this metric only when you enable backup for all documents. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`DeliveryToS3.DataFreshness`	The age (from getting into Amazon Data Firehose to now) of the oldest record in Amazon Data Firehose. Any record older than this age has been delivered to the S3 bucket. Amazon Data Firehose emits this metric only when you enable backup for all documents. Units: Seconds

Metric	Description
`DeliveryToS3.Records`	The number of records delivered to Amazon S3 over the specified time period. Amazon 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. Amazon Data Firehose always emits this metric regardless of whether backup is enabled for failed documents only or for all documents.
`DeliveryToAmazonOpenSearchServerless.AuthFailure`	Authentication/authorization error. Verify the OS/ES cluster policy and role permissions. 0 indicates that there is no issue. 1 indicates that there is an authentication failure.
`DeliveryToAmazonOpenSearchServerless.DeliveryRejected`	Delivery rejected error. Verify the OS/ES cluster policy and role permissions. 0 indicates that there is no issue. 1 indicates that there is a delivery failure.

Delivery to Amazon Redshift

Metric	Description
`DeliveryToRedshift.Bytes`	The number of bytes copied to Amazon Redshift over the specified time period. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`DeliveryToRedshift.Records`	The number of records copied to Amazon Redshift over the specified time period.

Metric	Description
	Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`DeliveryToRedshift.Success`	The sum of successful Amazon Redshift COPY commands.
`DeliveryToS3.Bytes`	The number of bytes delivered to Amazon S3 over the specified time period. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Bytes
`DeliveryToS3.DataFreshness`	The age (from getting into Amazon Data Firehose to now) of the oldest record in Amazon Data Firehose. Any record older than this age is delivered to the S3 bucket. Units: Seconds
`DeliveryToS3.Records`	The number of records delivered to Amazon S3 over the specified time period. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`DeliveryToS3.Success`	The sum of successful Amazon S3 put commands.
`DeliveryToRedshift.DataFreshness`	The age (from getting into Amazon Data Firehose to now) of the oldest record in Amazon Data Firehose. Any record that is older than this age is delivered to the Amazon Redshift cluster.

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

Metric	Description
`DeliveryToS3.Bytes`	The number of bytes delivered to Amazon S3 over the specified time period. When data transformation is enabled, this metric reflects the pre-processed byte size before transformation. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Bytes
`DeliveryToS3.DataFreshness`	The age (from getting into Amazon Data Firehose to now) of the oldest record in Amazon Data Firehose. Any record older than this age has been delivered to the S3 bucket. Statistics: Minimum, Maximum, Average, Samples Units: Seconds
`DeliveryToS3.Records`	The number of records delivered to Amazon S3 over the specified time period. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`DeliveryToS3.Success`	The sum of successful Amazon S3 put commands.
`BackupToS3.Bytes`	The number of bytes delivered to Amazon S3 for backup over the specified time period. Amazon Data Firehose emits this metric when backup is enabled (which is only possible when data transformation is also enabled). Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`BackupToS3.DataFreshness`	Age (from getting into Amazon Data Firehose to now) of the oldest record in Amazon Data Firehose. Any record

Metric	Description
	older than this age has been delivered to the Amazon S3 bucket for backup. Amazon Data Firehose emits this metric when backup is enabled (which is only possible when data transformation is also enabled). Statistics: Minimum, Maximum, Average, Samples Units: Seconds
`BackupToS3.Records`	The number of records delivered to Amazon S3 for backup over the specified time period. Amazon Data Firehose emits this metric when backup is enabled (which is only possible when data transformation is also enabled). Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`BackupToS3.Success`	The sum of successful Amazon S3 put commands for backup. Amazon Data Firehose emits this metric when backup is enabled (which is only possible when data transformation is also enabled).

Delivery to Snowflake

Metric	Description
`DeliveryToSnowflake.Bytes`	The number of bytes delivered to Snowflake over the specified time period. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Bytes
`DeliveryToSnowflake.DataFreshness`	Age (from getting into Firehose to now) of the oldest record in Firehose. Any record older than this age has

Metric	Description
	been delivered to Snowflake. Note that it can take a few seconds to commit data to Snowflake after Firehose insert call is successful. For the time it takes to commit data to Snowflake, refer to the `DeliveryToSnowflake.DataCommitLatency` metric. Statistics: Minimum, Maximum, Average, Samples Units: Seconds
`DeliveryToSnowflake.DataCommitLatency`	The time it takes for the data to be committed to Snowflake after Firehose inserted records successfully. Statistics: Minimum, Maximum, Average, Samples Units: Seconds
`DeliveryToSnowflake.Records`	The number of records delivered to Snowflake over the specified time period. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`DeliveryToSnowflake.Success`	The sum of successful insert calls made to Snowflake.
`DeliveryToS3.Bytes`	The number of bytes delivered to Amazon S3 over the specified time period. This metric is only available when delivery to Snowflake fails and Firehose attempts to backup failed data to S3. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Bytes

Metric	Description
`DeliveryToS3.Records`	The number of records delivered to Amazon S3 over the specified time period. This metric is only available when delivery to Snowflake fails and Firehose attempts to backup failed data to S3. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`DeliveryToS3.Success`	The sum of successful Amazon S3 put commands. This metric is only available when delivery to Snowflake fails and Firehose attempts to backup failed data to S3.
`BackupToS3.DataFreshness`	Age (from into Firehose to now) of the oldest record in Firehose. Any record older than this age is backed up to the Amazon S3 bucket. This metric is available when the Firehose stream is configured to back up all data. Statistics: Minimum, Maximum, Average, Samples Units: Seconds
`BackupToS3.Records`	The number of records delivered to Amazon S3 for backup over the specified time period. This metric is available when the Firehose stream is configured to back up all data. Statistics: Minimum, Maximum, Average, Sum, Samples Units:Count

Metric	Description
`BackupToS3.Bytes`	The number of bytes delivered to Amazon S3 for backup over the specified time period. This metric is available when the Firehose stream is configured to back up all data. Statistics: Minimum, Maximum, Average, Sum, Samples Units:Count
`BackupToS3.Success`	The sum of successful Amazon S3 put commands for backup. Firehose emits this metric when the Firehose stream is configured to back up all data.

Delivery to Splunk

Metric	Description
`DeliveryToSplunk.Bytes`	The number of bytes delivered to Splunk over the specified time period. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Bytes
`DeliveryToSplunk.DataAckLatency`	The approximate duration it takes to receive an acknowledgement from Splunk after Amazon 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. Statistics: Minimum, Maximum, Average, Samples Units: Seconds

Metric	Description
`DeliveryToSplunk.DataFreshness`	Age (from getting into Amazon Data Firehose to now) of the oldest record in Amazon Data Firehose. Any record older than this age has been delivered to Splunk. Statistics: Minimum, Maximum, Average, Samples Units: Seconds
`DeliveryToSplunk.Records`	The number of records delivered to Splunk over the specified time period. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`DeliveryToSplunk.Success`	The sum of the successfully indexed records.
`DeliveryToS3.Success`	The sum of successful 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. Amazon Data Firehose emits this metric when the Firehose stream is configured to back up all documents. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count

Metric	Description
`BackupToS3.DataFreshness`	Age (from getting into Amazon Data Firehose to now) of the oldest record in Amazon Data Firehose. Any record older than this age has been delivered to the Amazon S3 bucket for backup. Amazon Data Firehose emits this metric when the Firehose stream is configured to back up all documents. Statistics: Minimum, Maximum, Average, Samples Units: Seconds
`BackupToS3.Records`	The number of records delivered to Amazon S3 for backup over the specified time period. Amazon Data Firehose emits this metric when the Firehose stream is configured to back up all documents. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`BackupToS3.Success`	Sum of successful Amazon S3 put commands for backup. Amazon Data Firehose emits this metric when the Firehose 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. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Bytes

Metric	Description
`DeliveryToHttpEndpoint.Records`	The number of records delivered successfully to the HTTP endpoint. This metric is only emitted for successful delivery attempts and is not emitted when a delivery attempt fails. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Counts
`DeliveryToHttpEndpoint.DataFreshness`	Age of the oldest record in Amazon Data Firehose. Statistics: Minimum, Maximum, Average, Samples Units: Seconds
`DeliveryToHttpEndpoint.Success`	The number of records successfully delivered to the HTTP endpoint per delivery attempt. Unlike `DeliveryToHttpEndpoint.Records`, this metric is emitted for every delivery attempt. On success, the value equals the number of records in the delivery attempt. On failure of all records in the delivery attempt, the value is 0. Use the Minimum statistic to monitor for delivery failures. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`DeliveryToHttpEndpoint.ProcessedBytes`	The number of attempted processed bytes, including retries.
`DeliveryToHttpEndpoint.ProcessedRecords`	The number of attempted records including retries.

Data ingestion metrics

Topics

- Data ingestion through Kinesis Data Streams
- Data ingestion through Direct PUT
- Data ingestion from MSK

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. Statistics: Minimum, Maximum, Average, Sum, Samples 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. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`ThrottledDescribeStream`	The total number of times the `DescribeStream` operation is throttled when the data source is a Kinesis data stream. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`ThrottledGetRecords`	The total number of times the `GetRecords` operation is throttled when the data source is a Kinesis data stream. Statistics: Minimum, Maximum, Average, Sum, Samples

Metric	Description
	Units: Count
`ThrottledGetShardIterator`	The total number of times the `GetShardIterator` operation is throttled when the data source is a Kinesis data stream. Statistics: Minimum, Maximum, Average, Sum, Samples 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. Statistics: Minimum, Maximum, Average, Samples Units: Millisecond

Data ingestion through Direct PUT

Metric	Description
`BackupToS3.Bytes`	The number of bytes delivered to Amazon S3 for backup over the specified time period. Amazon Data Firehose emits this metric when data transformation is enabled for Amazon S3 or Amazon Redshift destinations. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Bytes
`BackupToS3.DataFreshness`	Age (from getting into Amazon Data Firehose to now) of the oldest record in Amazon Data Firehose. Any record older than this age has been delivered to the Amazon S3 bucket for backup. Amazon Data Firehose emits this

Metric	Description
	metric when data transformation is enabled for Amazon S3 or Amazon Redshift destinations. Statistics: Minimum, Maximum, Average, Samples Units: Seconds
`BackupToS3.Records`	The number of records delivered to Amazon S3 for backup over the specified time period. Amazon Data Firehose emits this metric when data transformation is enabled for Amazon S3 or Amazon Redshift destinati ons. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`BackupToS3.Success`	Sum of successful Amazon S3 put commands for backup. Amazon 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 Firehose 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**.
`DeliveryToAmazonOp enSearchService.Bytes`	The number of bytes indexed to OpenSearch Service over the specified time period. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Bytes

Metric	Description
`DeliveryToAmazonOpenSearchService.DataFreshness`	The age (from getting into Amazon Data Firehose to now) of the oldest record in Amazon Data Firehose. Any record older than this age has been delivered to OpenSearch Service. Statistics: Minimum, Maximum, Average, Samples Units: Seconds
`DeliveryToAmazonOpenSearchService.Records`	The number of records indexed to OpenSearch Service over the specified time period. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`DeliveryToAmazonOpenSearchService.Success`	The sum of the successfully indexed records.
`DeliveryToRedshift.Bytes`	The number of bytes copied to Amazon Redshift over the specified time period. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Bytes
`DeliveryToRedshift.Records`	The number of records copied to Amazon Redshift over the specified time period. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`DeliveryToRedshift.Success`	The sum of successful Amazon Redshift COPY commands.

Metric	Description
`DeliveryToS3.Bytes`	The number of bytes delivered to Amazon S3 over the specified time period. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Bytes
`DeliveryToS3.DataFreshness`	The age (from getting into Amazon Data Firehose to now) of the oldest record in Amazon Data Firehose. Any record older than this age has been delivered to the S3 bucket. Statistics: Minimum, Maximum, Average, Samples Units: Seconds
`DeliveryToS3.Records`	The number of records delivered to Amazon S3 over the specified time period. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`DeliveryToS3.Success`	The sum of successful Amazon S3 put commands.
`DeliveryToSplunk.Bytes`	The number of bytes delivered to Splunk over the specified time period. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Bytes

Metric	Description
`DeliveryToSplunk.DataAckLatency`	The approximate duration it takes to receive an acknowledgement from Splunk after Amazon 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. Statistics: Minimum, Maximum, Average, Samples Units: Seconds
`DeliveryToSplunk.DataFreshness`	Age (from getting into Amazon Data Firehose to now) of the oldest record in Amazon Data Firehose. Any record older than this age has been delivered to Splunk. Statistics: Minimum, Maximum, Average, Samples Units: Seconds
`DeliveryToSplunk.Records`	The number of records delivered to Splunk over the specified time period. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`DeliveryToSplunk.Success`	The sum of the successfully indexed records.
`IncomingBytes`	The number of bytes ingested successfully into the Firehose stream over the specified time period. Data ingestion could be throttled when it exceeds one of the Firehose stream limits. Throttled data will not be counted for `IncomingBytes` . Statistics: Minimum, Maximum, Average, Sum, Samples Units: Bytes

Metric	Description
`IncomingPutRequests`	The number of successful PutRecord and PutRecordBatch requests over a specified period of time. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`IncomingRecords`	The number of records ingested successfully into the Firehose stream over the specified time period. Data ingestion could be throttled when it exceeds one of the Firehose stream limits. Throttled data will not be counted for `IncomingRecords` . Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`RecordsPerSecondLimit`	The current maximum number of records per second that a Firehose stream can ingest before throttling. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`ThrottledRecords`	The number of records that were throttled because data ingestion exceeded one of the Firehose stream limits. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count

Data ingestion from MSK

Metric	Description
`DataReadFromSource.Records`	The number of records read from the source Kafka Topic.

Metric	Description
	Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`DataReadFromSource.Bytes`	The number of bytes read from the source Kafka Topic. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Bytes
`SourceThrottled.Delay`	The amount of time that the source Kafka cluster is delayed in returning the records from the source Kafka Topic. Statistics: Minimum, Maximum, Average, Samples Units: Milliseconds
`BytesPerSecondLimit`	Current limit of throughput at which Firehose is going to read from each partition of the source Kafka Topic. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Bytes/sec
`KafkaOffsetLag`	The difference between the largest offset of the record that Firehose has read from the source Kafka Topic and the largest offset of the record available from the source Kafka Topic. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`FailedValidation.Records`	The number of records that failed record validation. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count

Metric	Description
`FailedValidation.Bytes`	The number of bytes that failed record validation. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Bytes
`DataReadFromSource.Backpressured`	Indicates that a Firehose stream is delayed in reading records from the source partition either because BytesPerSecondLimit per partition has exceeded or that the normal flow of delivery is slow or has stopped Units: Boolean

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. Statistics: Minimum, Maximum, Average, Samples Units: Milliseconds
`DescribeDeliveryStream.Requests`	The total number of `DescribeDeliveryStream` requests. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`ListDeliveryStreams.Latency`	The time taken per `ListDeliveryStream` operation, measured over the specified time period. Statistics: Minimum, Maximum, Average, Samples

Metric	Description
	Units: Milliseconds
`ListDeliveryStreams.Requests`	The total number of `ListFirehose` requests. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`PutRecord.Bytes`	The number of bytes put to the Firehose stream using `PutRecord` over the specified time period. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Bytes
`PutRecord.Latency`	The time taken per `PutRecord` operation, measured over the specified time period. Statistics: Minimum, Maximum, Average, Samples Units: Milliseconds
`PutRecord.Requests`	The total number of `PutRecord` requests, which is equal to total number of records from `PutRecord` operations. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`PutRecordBatch.Bytes`	The number of bytes put to the Firehose stream using `PutRecordBatch` over the specified time period. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Bytes

Metric	Description
`PutRecordBatch.Latency`	The time taken per `PutRecordBatch` operation, measured over the specified time period. Statistics: Minimum, Maximum, Average, Samples Units: Milliseconds
`PutRecordBatch.Records`	The total number of records from `PutRecordBatch` operations. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`PutRecordBatch.Requests`	The total number of `PutRecordBatch` requests. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`PutRequestsPerSecondLimit`	The maximum number of put requests per second that a Firehose stream can handle before throttling. This number includes PutRecord and PutRecordBatch requests. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`ThrottledDescribeStream`	The total number of times the `DescribeStream` operation is throttled when the data source is a Kinesis data stream. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count

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

CloudWatch Logs Decompression Metrics

If decompression is enabled for CloudWatch Logs delivery, the `AWS/Firehose` namespace includes the following metrics.

Metric	Description
`OutputDecompressedBytes.Success`	Successful decompressed data in bytes Statistics: Minimum, Maximum, Average, Sum, Samples Units: Bytes
`OutputDecompressedBytes.Failed`	Failed decompressed data in bytes Statistics: Minimum, Maximum, Average, Sum, Samples Units: Bytes

Metric	Description
`OutputDecompressedRecords.Success`	Number of successful decompressed records Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`OutputDecompressedRecords.Failed`	Number of failed decompressed records Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count

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 Firehose stream. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`KMSKeyDisabled`	The number of times the service encounters a `KMSDisabledException` for the Firehose stream. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`KMSKeyInvalidState`	The number of times the service encounters a `KMSInvalidStateException` for the Firehose stream. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count
`KMSKeyNotFound`	The number of times the service encounters a `KMSNotFoundException` for the Firehose stream. Statistics: Minimum, Maximum, Average, Sum, Samples Units: Count

Dimensions for Amazon Data Firehose

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

Amazon 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 three minutes.

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 3-minute period.

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

Dimension	Description
`Service`	The name of the AWS service containing the resource. For Amazon Data Firehose usage metrics, the value for this dimension is `Firehose`.
`Class`	The class of resource being tracked. Amazon 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` .

Access CloudWatch Metrics for Amazon Data Firehose

You can monitor metrics for Amazon 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://eusc-de-east-1.console.amazonaws-eusc.eu/cloudwatch/.
2. On the navigation bar, choose a region.
3. In the navigation pane, choose **Metrics**.
4. Choose the **Firehose** namespace.
5. Choose **Firehose 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
```

Monitor Amazon Data Firehose Using CloudWatch Logs

Amazon 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 Amazon Data Firehose error logging when you create your Firehose stream.

If you enable Amazon Data Firehose error logging in the Amazon Data Firehose console, a log group and corresponding log streams are created for the Firehose 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 Firehose stream. `DestinationDelivery` is the log stream that is created and used to log any errors related to the

delivery to the primary destination. Another log stream called `BackupDelivery` is created only if S3 backup is enabled for the destination. However, which log stream receives S3 backup delivery errors depends on the `S3BackupMode` setting of the Firehose stream. When the `S3BackupMode` setting is `AllData` (or `Enabled` for Amazon Redshift), S3 backup delivery errors are logged to the `BackupDelivery` log stream. When the `S3BackupMode` setting is `FailedDataOnly`, S3 backup delivery errors (for example, `S3.AccessDenied`) are logged to the `DestinationDelivery` log stream along with the primary destination errors. They are not logged to the `BackupDelivery` log stream. The `FailedDataOnly` setting is available for the HTTP endpoint, Splunk, OpenSearch, Snowflake, and Iceberg destinations.

For example, if you create a Firehose stream "MyStream" with Amazon Redshift as the destination and enable Amazon Data Firehose error logging, the following are created on your behalf: a log group named `aws/kinesisfirehose/MyStream` and two log streams named `DestinationDelivery` and **BackupDelivery**. In this example, `DestinationDelivery` will be used to log any errors related to the delivery to the Amazon Redshift destination and also to the intermediate S3 destination. `BackupDelivery`, in case S3 backup is enabled, will be used to log any errors related to the delivery to the S3 backup bucket.

As another example, suppose you create a Firehose stream with an HTTP endpoint as the destination and set `S3BackupMode` to `FailedDataOnly`. Errors for the records that failed delivery and were backed up to S3 (for example, `S3.AccessDenied`) are logged to the `DestinationDelivery` log stream. They are not logged to the `BackupDelivery` log stream.

You can enable Amazon Data Firehose error logging through the AWS CLI, the API, or 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 Amazon Data Firehose error logging exclusively. Also ensure that the associated IAM policy has `"logs:putLogEvents"` permission. For more information, see Controlling access with Amazon Data Firehose.

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

Data delivery errors

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

Errors

- Amazon S3 Data delivery errors
- Apache Iceberg Tables Data Delivery Errors
- Amazon Redshift Data delivery errors
- Snowflake Data delivery errors
- Splunk Data delivery errors
- ElasticSearch Data delivery errors
- HTTPS Endpoint Data delivery errors
- Amazon OpenSearch Service Data delivery errors
- Lambda invocation errors
- Kinesis invocation errors
- Kinesis DirectPut invocation errors
- AWS Glue invocation errors
- DataFormatConversion invocation errors

Amazon S3 Data delivery errors

Amazon 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*.

Error Code	Error Message and Information
`S3.AccessDenied`	"Access was denied. Ensure that the trust policy for the provided IAM role allows Amazon 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."
`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."
`S3.KMS.KeyDisabled`	"The provided KMS key is disabled. Enable the key or use a different key."
`S3.KMS.InvalidStateException`	"The provided KMS key is in an invalid state. Please use a different key."
`KMS.InvalidStateException`	"The provided KMS key is in an invalid state. Please use a different key."

Error Code	Error Message and Information
`KMS.DisabledException`	"The provided KMS key is disabled. Please fix the key or use a different key."
`S3.SlowDown`	"The rate of put request to the specified bucket was too high. Increase Firehose stream buffer size or reduce put requests from other applications."
`S3.SubscriptionRequired`	"Access was denied when calling S3. Ensure that the IAM role and the KMS Key (if provided) passed in has Amazon S3 subscription."
`S3.InvalidToken`	"The provided token is malformed or otherwise invalid. Please check the credentials provided."
`S3.KMS.KeyNotConfigured`	"KMS key not configured. Configure your KMSMasterKeyID, or disable encryption for your S3 bucket."
`S3.KMS.AsymmetricCMKNotSupported`	"Amazon S3 supports only symmetric CMKs. You cannot use an asymmetric CMK to encrypt your data in Amazon S3. To get the type of your CMK, use the KMS DescribeKey operation."
`S3.IllegalLocationConstraintException`	"Firehose currently uses s3 global endpoint for data delivery to the configured s3 bucket. The region of the configured s3 bucket doesn't support s3 global endpoint. Please create a Firehose stream in the same region as the s3 bucket or use s3 bucket in the region that supports global endpoint."
`S3.InvalidPrefixConfigurationException`	"The custom s3 prefix used for the timestamp evaluation is invalid. Check your s3 prefix contains valid expressions for the current date and time of the year."
`DataFormatConversion.MalformedData`	"Illegal character found between tokens."

Apache Iceberg Tables Data Delivery Errors

For Apache Iceberg Tables data delivery errors, see *Deliver data to Apache Iceberg Tables*.

Amazon Redshift Data delivery errors

Amazon 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 Amazon 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 Amazon 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."

Error Code	Error Message and Information
`Redshift.Incorrect CopyOptions`	"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."
`Redshift.ConnectionFailed`	"The connection to the specified Amazon Redshift cluster failed. Ensure that security settings allow Amazon 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."

Error Code	Error Message and Information
`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."
`Redshift.ArgumentNotSupported`	"The COPY command contains unsupported options."
`Redshift.AnalyzeTableAccessDenied`	"Access denied. Copy from S3 to Redshift is failing because analyze table can only be done by table or database owner."
`Redshift.SchemaNotFound`	"The schema specified in the DataTableName of Amazon Redshift destination configuration was not found. Specify a valid schema name."
`Redshift.ColumnSpecifiedMoreThanOnce`	"There is a column specified more than once in the column list. Ensure that duplicate columns are removed."

Error Code	Error Message and Information
`Redshift.ColumnNotNullWithoutDefault`	"There is a non-null column without DEFAULT that is not included in the column list. Ensure that such columns are included in the column list."
`Redshift.IncorrectBucketRegion`	"Redshift attempted to use a bucket in a different region from the cluster. Please specify a bucket within the same region as the cluster."
`Redshift.S3SlowDown`	"High request rate to S3. Reduce the rate to avoid getting throttled."
`Redshift.InvalidCopyOptionForJson`	"Please use either auto or a valid S3 path for json copyOption."
`Redshift.InvalidCopyOptionJSONPathFormat`	"COPY failed with error \"Invalid JSONPath format. Array index is out of range\". Please rectify the JSONPath expression."
`Redshift.InvalidCopyOptionRBACAclNotAllowed`	"COPY failed with error \"Cannot use RBAC acl framework while permission propagation is not enabled.\"
`Redshift.DiskSpaceQuotaExceeded`	"Transaction aborted due to disk space quota exceed. Free up disk space or request increased quota for the schema(s)."
`Redshift.ConnectionsLimitExceeded`	"Connection limit exceeded for user."

Error Code	Error Message and Information
`Redshift.SslNotSupported`	"The connection to the specified Amazon Redshift cluster failed because the server does not support SSL. Please check your cluster settings."
`Redshift.HoseNotFound`	"The hose has been deleted. Please check the status of your hose."
`Redshift.Delimiter`	"The copyOptions delimiter in the copyCommand is invalid. Ensure that it is a single character."
`Redshift.QueryCancelled`	"The user has canceled the COPY operation."
`Redshift.CompressionMismatch`	"Hose is configured with UNCOMPRESSED, but copyOption includes a compression format."
`Redshift.EncryptionCredentials`	"The ENCRYPTED option requires credentials in the format: 'aws_iam_role=...;master_symmetric_key=...' or 'aws_access_key_id=...;aws_secret_access_key=...[;token=...];master_symmetric_key=...'"
`Redshift.InvalidCopyOptions`	"Invalid COPY configuration options."
`Redshift.InvalidMessageFormat`	"Copy command contains an invalid character."
`Redshift.TransactionIdLimitReached`	"Transaction ID limit reached."

Error Code	Error Message and Information
`Redshift.DestinationRemoved`	"Please verify that the redshift destination exists and is configured correctly in the Firehose configuration."
`Redshift.OutOfMemory`	"The Redshift cluster is running out of memory. Please ensure the cluster has sufficient capacity."
`Redshift.CannotForkProcess`	"The Redshift cluster is running out of memory. Please ensure the cluster has sufficient capacity."
`Redshift.SslFailure`	"The SSL connection closed during the handshake."
`Redshift.Resize`	"The Redshift cluster is resizing. Firehose will not be able to deliver data while the cluster is resizing."
`Redshift.ImproperQualifiedName`	"The qualified name is improper (too many dotted names)."
`Redshift.InvalidJsonPathFormat`	"Invalid JSONPath Format."
`Redshift.TooManyConnectionsException`	"Too many connections to Redshift."
`Redshift.PSQLException`	"PSQlException observed from Redshift."
`Redshift.DuplicateSecondsSpecification`	"Duplicate seconds specification in date/time format."

Error Code	Error Message and Information
`Redshift.RelationCouldNotBeOpened`	"Encountered Redshift error, relation could not be opened. Check Redshift logs for the specified DB."
`Redshift.TooManyClients`	"Encountered too many clients exception from Redshift. Revisit max connections to the database if there are multiple producers writing to it simultaneously."

Snowflake Data delivery errors

Firehose can send the following Snowflake-related errors to CloudWatch Logs.

Error Code	Error Message and Information
`Snowflake.InvalidUrl`	"Firehose is unable to connect to Snowflake. Please make sure that Account url is specified correctly in Snowflake destination configuration."
`Snowflake.InvalidUser`	"Firehose is unable to connect to Snowflake. Please make sure that User is specified correctly in Snowflake destination configuration."
`Snowflake.InvalidRole`	"The specified snowflake role does not exist or is not authorized. Please make sure that the role is granted to the user specified"
`Snowflake.InvalidTable`	"The supplied table does not exist or is not authorized"
`Snowflake.InvalidSchema`	"The supplied schema does not exist or is not authorized"
`Snowflake.InvalidDatabase`	"The supplied database does not exist or is not authorized"

Error Code	Error Message and Information
`Snowflake.InvalidPrivateKeyOrPassphrase`	"The specified private key or passphrase is not valid. Note that the private key provided should be a valid PEM RSA private key"
`Snowflake.MissingColumns`	"The insert request is rejected due to missing columns in input payload. Make sure that values are specified for all non-nullable columns"
`Snowflake.ExtraColumns`	"The insert request is rejected due to extra columns. Columns not present in table shouldn't be specified"
`Snowflake.InvalidInput`	"Delivery failed due to invalid input format. Make sure that the input payload provided is in the JSON format acceptable"
`Snowflake.IncorrectValue`	"Delivery failed due to incorrect data type in the input payload. Make sure that the JSON values specified in input payload adhere to the datatype declared in Snowflake table definition"

Splunk Data delivery errors

Amazon 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 Amazon 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 Amazon Data Firehose with a valid HEC token."

Error Code	Error Message and Information
`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. Amazon Data Firehose will retry sending the data if the retry duration in your Amazon Data Firehose is greater than 0. If all the retries fail, Amazon 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. Amazon 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. Amazon 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 Amazon 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 Amazon Data Firehose configuration might guard against such transient failures."

Error Code	Error Message and Information
`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."
`Splunk.URLNotFound`	"The requested URL was not found on the Splunk server. Please check the Splunk cluster and make sure it is configured correctly."
`Splunk.ServerError.ContentTooLarge`	"Data delivery to Splunk failed due to a server error with a statusCode: 413, message: the request your client sent was too large. See splunk docs to configure max_content_length."
`Splunk.IndexerBusy`	"Data delivery to Splunk failed due to a server error from the HEC node. Make sure HEC endpoint or the Elastic Load Balancer is reachable and is healthy."
`Splunk.ConnectionRecycled`	"The connection from Firehose to Splunk has been recycled. Delivery will be retried."
`Splunk.AcknowledgementsDisabled`	"Could not get acknowledgements on POST. Make sure that acknowledgements are enabled on HEC endpoint."
`Splunk.InvalidHecResponseCharacter`	"Invalid characters found in HEC response, make sure to check to the service and HEC configuration."

ElasticSearch Data delivery errors

Amazon Data Firehose can send the following ElasticSearch errors to CloudWatch Logs.

Error Code	Error Message and Information
`ES.AccessDenied`	"Access was denied. Ensure that the provided IAM role associated with firehose is not deleted."
`ES.ResourceNotFound`	"The specified AWS Elasticsearch domain does not exist."

HTTPS Endpoint Data delivery errors

Amazon 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."

Error Code	Error Message and Information
`HttpEndpoint.ConnectionFailed`	"Unable to connect to the destination endpoint. Contact the owner of the endpoint to resolve this issue."
`HttpEndpoint.ConnectionReset`	"Unable to maintain connection with the endpoint. Contact the owner of the endpoint to resolve this issue."
`HttpEndpoint.ConnectionReset`	"Trouble maintaining connection with the endpoint. Please reach out to the owner of the endpoint."
`HttpEndpoint.ResponseReasonPhraseExceededLimit`	"The response reason phrase received from the endpoint exceed the configured limit of 64 characters."
`HttpEndpoint.InvalidResponseFromDestination`	"The response received from the endpoint is invalid. See Troubleshooting HTTP Endpoints in the Firehose documentation for more information. Reason: "
`HttpEndpoint.DestinationException`	"Delivery to the endpoint was unsuccessful. See Troubleshooting HTTP Endpoints in the Firehose documentation for more information. Response received with status code "
`HttpEndpoint.InvalidStatusCode`	"Received an invalid response status code."
`HttpEndpoint.SSLHandshakeFailure`	"Unable to complete an SSL Handshake with the endpoint. Contact the owner of the endpoint to resolve this issue."

Error Code	Error Message and Information
`HttpEndpoint.SSLHandshakeFailure`	"Unable to complete an SSL Handshake with the endpoint. Contact the owner of the endpoint to resolve this issue."
`HttpEndpoint.SSLFailure`	"Unable to complete TLS handshake with the endpoint. Contact the owner of the endpoint to resolve this issue."
`HttpEndpoint.SSLHandshakeCertificatePathFailure`	"Unable to complete an SSL Handshake with the endpoint due to invalid certification path. Contact the owner of the endpoint to resolve this issue."
`HttpEndpoint.SSLHandshakeCertificatePathValidationFailure`	"Unable to complete an SSL Handshake with the endpoint due to certification path validation failure. Contact the owner of the endpoint to resolve this issue."
`HttpEndpoint.MakeRequestFailure.IllegalUriException`	"HttpEndpoint request failed due to invalid input in URI. Please make sure all the characters in the input URI are valid."
`HttpEndpoint.MakeRequestFailure.IllegalCharacterInHeaderValue`	"HttpEndpoint request failed due to illegal response error. Illegal character '\n' in header value."

Error Code	Error Message and Information
`HttpEndpoint.IllegalResponseFailure`	"HttpEndpoint request failed due to illegal response error. HTTP message must not contain more than one Content-Type header."
`HttpEndpoint.IllegalMessageStart`	"HttpEndpoint request failed due to illegal response error. Illegal HTTP message start. See Troubleshooting HTTP Endpoints in the Firehose documentation for more information."

Amazon OpenSearch Service Data delivery errors

For the OpenSearch Service destination, Amazon Data Firehose sends errors to CloudWatch Logs as they are returned by OpenSearch Service.

In addition to errors that may return from OpenSearch clusters, you may encounter the following two errors:

- Authentication/authorization error occurs during attempt to deliver data to destination OpenSearch Service cluster. This can happen due to any permission issues and/or intermittently when your Amazon Data Firehose target OpenSearch Service domain configuration is modified. Please check the cluster policy and role permissions.
- Data couldn’t be delivered to destination OpenSearch Service cluster due to authentication/authorization failures. This can happen due to any permission issues and/or intermittently when your Amazon Data Firehose target OpenSearch Service domain configuration is modified. Please check the cluster policy and role permissions.

Error Code	Error Message and Information
`OS.AccessDenied`	"Access was denied. Ensure that the trust policy for the provided IAM role allows Firehose to assume the role, and the access policy allows access to the Amazon OpenSearch Service API."

Error Code	Error Message and Information
`OS.AccessDenied`	"Access was denied. Ensure that the trust policy for the provided IAM role allows Firehose to assume the role, and the access policy allows access to the Amazon OpenSearch Service API."
`OS.AccessDenied`	"Access was denied. Ensure that the provided IAM role associated with firehose is not deleted."
`OS.AccessDenied`	"Access was denied. Ensure that the provided IAM role associated with firehose is not deleted."
`OS.ResourceNotFound`	"The specified Amazon OpenSearch Service domain does not exist."
`OS.ResourceNotFound`	"The specified Amazon OpenSearch Service domain does not exist."
`OS.AccessDenied`	"Access was denied. Ensure that the trust policy for the provided IAM role allows Firehose to assume the role, and the access policy allows access to the Amazon OpenSearch Service API."
`OS.RequestTimeout`	"Request to the Amazon OpenSearch Service cluster or OpenSearch Serverless collection timed out. Ensure that the cluster or collection has sufficient capacity for the current workload."
`OS.ClusterError`	"The Amazon OpenSearch Service cluster returned an unspecified error."
`OS.RequestTimeout`	"Request to the Amazon OpenSearch Service cluster timed out. Ensure that the cluster has sufficient capacity for the current workload."
`OS.ConnectionFailed`	"Trouble connecting to the Amazon OpenSearch Service cluster or OpenSearch Serverless collection. Ensure that the cluster or collection is healthy and reachable."
`OS.ConnectionReset`	"Unable to maintain connection with the Amazon OpenSearch Service cluster or OpenSearch Serverless collection. Contact the owner of the cluster or collection to resolve this issue."

Error Code	Error Message and Information
`OS.ConnectionReset`	"Trouble maintaining connection with the Amazon OpenSearch Service cluster or OpenSearch Serverless collection. Ensure that the cluster or collection is healthy and has sufficient capacity for the current workload."
`OS.ConnectionReset`	"Trouble maintaining connection with the Amazon OpenSearch Service cluster or OpenSearch Serverless collection. Ensure that the cluster or collection is healthy and has sufficient capacity for the current workload."
`OS.AccessDenied`	"Access was denied. Ensure that the access policy on the Amazon OpenSearch Service cluster grants access to the configured IAM role."
`OS.ValidationException`	"The OpenSearch cluster returned a ESServiceException. One of the reasons is that the cluster has been upgraded to OS 2.x or higher, but the hose still has the TypeName parameter configured. Update the hose configuration by setting the TypeName to an empty string, or change the endpoint to the cluster, that supports the Type parameter."
`OS.ValidationException`	"Member must satisfy regular expression pattern: [a-z][a-z0-9\\-]+
`OS.JsonParseException`	"The Amazon OpenSearch Service cluster returned a JsonParse Exception. Ensure that the data being put is valid."
`OS.AmazonOpenSearchServiceParseException`	"The Amazon OpenSearch Service cluster returned an AmazonOpenSearchServiceParseException. Ensure that the data being put is valid."
`OS.ExplicitIndexInBulkNotAllowed`	"Ensure rest.action.multi.allow_explicit_index is set to true on the Amazon OpenSearch Service cluster."
`OS.ClusterError`	"The Amazon OpenSearch Service cluster or OpenSearch Serverless collection returned an unspecified error."

Error Code	Error Message and Information
`OS.ClusterBlockException`	"The cluster returned a ClusterBlockException. It may be overloaded."
`OS.InvalidARN`	"The Amazon OpenSearch Service ARN provided is invalid. Please check your DeliveryStream configuration."
`OS.MalformedData`	"One or more records are malformed. Please ensure that each record is single valid JSON object and that it does not contain newlines."
`OS.InternalError`	"An internal error occurred when attempting to deliver data. Delivery will be retried; if the error persists, it will be reported to AWS for resolution."
`OS.AliasWithMultipleIndicesNotAllowed`	"Alias has more than one indices associated with it. Ensure that the alias has only one index associated with it."
`OS.UnsupportedVersion`	"Amazon OpenSearch Service 6.0 is not currently supported by Amazon Data Firehose. Contact AWS Support for more information."
`OS.CharConversionException`	"One or more records contained an invalid character."
`OS.InvalidDomainNameLength`	"The domain name length is not within valid OS limits."
`OS.VPCDomainNotSupported`	"Amazon OpenSearch Service domains within VPCs are currently not supported."
`OS.ConnectionError`	"The http server closed the connection unexpectedly, please verify the health of the Amazon OpenSearch Service cluster or OpenSearch Serverless collection."

Error Code	Error Message and Information
`OS.LargeFieldData`	"The Amazon OpenSearch Service cluster aborted the request as it contained a field data larger than allowed."
`OS.BadGateway`	"The Amazon OpenSearch Service cluster or OpenSearch Serverless collection aborted the request with a response: 502 Bad Gateway."
`OS.ServiceException`	"Error received from the Amazon OpenSearch Service cluster or OpenSearch Serverless collection. If the cluster or collection is behind a VPC, ensure network configuration allows connectivity."
`OS.GatewayTimeout`	"Firehose encountered timeout errors when connecting to the Amazon OpenSearch Service cluster or OpenSearch Serverless collection."
`OS.MalformedData`	"Amazon Data Firehose does not support Amazon OpenSearch Service Bulk API commands inside the Firehose record."
`OS.ResponseEntryCountMismatch`	"The response from the Bulk API contained more entries than the number of records sent. Ensure that each record contains only one JSON object and that there are no newlines."

Lambda invocation errors

Amazon 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 Amazon Data Firehose to assume the role."
`Lambda.InvokeAccessDenied`	"Access was denied. Ensure that the access policy allows access to the Lambda function."

Error Code	Error Message and Information
`Lambda.JsonProcessingException`	"There was an error parsing returned records from the Lambda function. Ensure that the returned records follow the status model required by Amazon Data Firehose." For more information, see Required parameters for data transformation.
`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 Required parameters for data transformation.
`Lambda.MissingRecordId`	"One or more record IDs were not returned. Ensure that the Lambda function returns all received record IDs." For more information, see Required parameters for data transformation.
`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*.
`Lambda.FunctionTimedOut`	The Lambda function invocation timed out. Increase the Timeout setting in the Lambda function. For more information, see Configuring function timeout.
`Lambda.FunctionError`	This can be due to any of the following errors: • Invalid output structure. Check your function and make sure the output is in the required format. Also, make sure the processed records contain a valid result status of `Dropped`, `Ok`, or `ProcessingFailed` . • The Lambda function was successfully invoked but it returned an error result. • Lambda was unable to decrypt the environment variables because KMS access was denied. Check the function's KMS key settings as well as the key policy. For more information, see Troubleshooting Key Access.

Error Code	Error Message and Information
`Lambda.FunctionRequestTimedOut`	Amazon Data Firehose encountered Request did not complete before the request timeout configuration error when invoking Lambda. Revisit the Lambda code to check if the Lambda code is meant to run beyond the configured timeout. If so, consider tuning Lambda configuration settings, including memory, timeout. For more information, see Configuring Lambda function options.
`Lambda.TargetServerFailedToRespond`	Amazon Data Firehose encountered an error. Target server failed to respond error when calling the AWS Lambda service.
`Lambda.InvalidZipFileException`	Amazon Data Firehose encountered InvalidZipFileException when invoking the Lambda function. Check your Lambda function configuration settings and the Lambda code zip file.
`Lambda.InternalServerError`	"Amazon Data Firehose encountered InternalServerError when calling the AWS Lambda service. Amazon Data Firehose will retry sending data a fixed number of times. You can specify or override the retry options using the `CreateDeliveryStream` or `UpdateDestination` APIs. If the error persists, contact AWS Lambda support team.
`Lambda.ServiceUnavailable`	Amazon Data Firehose encountered ServiceUnavailableException when calling the AWS Lambda service. Amazon Data Firehose will retry sending data a fixed number of times. You can specify or override the retry options using the `CreateDeliveryStream` or `UpdateDestination` APIs. If the error persists, contact AWS Lambda support.
`Lambda.InvalidSecurityToken`	Cannot invoke Lambda function due to invalid security token. Cross partition Lambda invocation is not supported.

Error Code	Error Message and Information
`Lambda.InvocationFailure`	This can be due to any of the following errors: • Amazon Data Firehose encountered errors when calling AWS Lambda. The operation will be retried; if the error persists, it will be reported to AWS for resolution." • Amazon Data Firehose encountered a KMSInvalidStateException from Lambda. Lambda was unable to decrypt the environment variables because the KMS key used is in an invalid state for Decrypt. Check the lambda function's KMS key. • Amazon Data Firehose encountered an AWSLambdaException from Lambda. Lambda was unable to initialize the provided container image. Verify the image. • Amazon Data Firehose encountered timeout errors when calling AWS Lambda. The maximum supported function timeout is 5 minutes. For more information, see Data Transformation Execution Duration.
`Lambda.JsonMappingException`	There was an error parsing returned records from the Lambda function. Ensure that data field is base-64 encoded.

Kinesis invocation errors

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

Error Code	Error Message and Information
`Kinesis.AccessDenied`	"Access was denied when calling Kinesis. Ensure the access policy on the IAM role used allows access to the appropriate Kinesis APIs."
`Kinesis.ResourceNotFound`	"Firehose failed to read from the stream. If the Firehose is attached with Kinesis Stream, the stream may not exist, or the shard may have been merged or split. If the Firehose is of DirectPut type, the Firehose may not exist any more."

Error Code	Error Message and Information
`Kinesis.SubscriptionRequired`	"Access was denied when calling Kinesis. Ensure that the IAM role passed for Kinesis stream access has AWS Kinesis subscription."
`Kinesis.Throttling`	"Throttling error encountered when calling Kinesis. This can be due to other applications calling the same APIs as the Firehose stream, or because you have created too many Firehose streams with the same Kinesis stream as the source."
`Kinesis.Throttling`	"Throttling error encountered when calling Kinesis. This can be due to other applications calling the same APIs as the Firehose stream, or because you have created too many Firehose streams with the same Kinesis stream as the source."
`Kinesis.AccessDenied`	"Access was denied when calling Kinesis. Ensure the access policy on the IAM role used allows access to the appropriate Kinesis APIs."
`Kinesis.AccessDenied`	"Access was denied while trying to call API operations on the underlying Kinesis Stream. Ensure that the IAM role is propagated and valid."
`Kinesis.KMS.AccessDeniedException`	"Firehose does not have access to the KMS Key used to encrypt/decrypt the Kinesis Stream. Please grant the Firehose delivery role access to the key."
`Kinesis.KMS.KeyDisabled`	"Firehose is unable to read from the source Kinesis Stream because the KMS key used to encrypt/decrypt it is disabled. Enable the key so that reads can proceed."
`Kinesis.KMS.InvalidStateException`	"Firehose is unable to read from the source Kinesis Stream because the KMS key used to encrypt it is in an invalid state."

Error Code	Error Message and Information
`Kinesis.KMS.NotFoundException`	"Firehose is unable to read from the source Kinesis Stream because the KMS key used to encrypt it was not found."

Kinesis DirectPut invocation errors

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

Error Code	Error Message and Information
`Firehose.KMS.AccessDeniedException`	"Firehose does not have access to the KMS Key. Please check the key policy."
`Firehose.KMS.InvalidStateException`	"Firehose is unable to decrypt the data because the KMS key used to encrypt it is in an invalid state."
`Firehose.KMS.NotFoundException`	"Firehose is unable to decrypt the data because the KMS key used to encrypt it was not found."
`Firehose.KMS.KeyDisabled`	"Firehose is unable to decrypt the data because the KMS key used to encrypt the data is disabled. Enable the key so that data delivery can proceed."

AWS Glue invocation errors

Amazon Data Firehose can send the following AWS Glue invocation errors to CloudWatch Logs.

Error Code	Error Message and Information
`DataFormatConversion.InvalidSchema`	"The schema is invalid."
`DataFormatConversion.EntityNotFound`	"The specified table/database could not be found. Please ensure that the table/database exists and that the values provided in the schema configuration are correct, especially with regards to casing."
`DataFormatConversion.InvalidInput`	"Could not find a matching schema from glue. Please make sure the specified database with the supplied catalog ID exists."
`DataFormatConversion.InvalidInput`	"Could not find a matching schema from glue. Please make sure the passed ARN is in the correct format."
`DataFormatConversion.InvalidInput`	"Could not find a matching schema from glue. Please make sure the catalogId provided is valid."
`DataFormatConversion.InvalidVersionId`	"Could not find a matching schema from glue. Please make sure the specified version of the table exists."
`DataFormatConversion.NonExistentColumns`	"Could not find a matching schema from glue. Please make sure the table is configured with a non-null storage descriptor containing the target columns."

Error Code	Error Message and Information
`DataFormatConversion.AccessDenied`	"Access was denied when assuming role. Please ensure that the role specified in the data format conversion configuration has granted the Firehose service permission to assume it."
`DataFormatConversion.ThrottledByGlue`	"Throttling error encountered when calling Glue. Either increase the request rate limit or reduce the current rate of calling glue through other applications."
`DataFormatConversion.AccessDenied`	"Access was denied when calling Glue. Please ensure that the role specified in the data format conversion configuration has the necessary permissions."
`DataFormatConversion.InvalidGlueRole`	"Invalid role. Please ensure that the role specified in the data format conversion configuration exists."
`DataFormatConversion.InvalidGlueRole`	"The security token included in the request is invalid. Ensure that the provided IAM role associated with firehose is not deleted."
`DataFormatConversion.GlueNotAvailableInRegion`	"AWS Glue is not yet available in the region you have specified; please specify a different region."
`DataFormatConversion.GlueEncryptionException`	"There was an error retrieving the master key. Ensure that the key exists and has the correct access permissions."

Error Code	Error Message and Information
`DataFormatConversion.SchemaValidationTimeout`	"Timed out while retrieving table from Glue. If you have a large number of Glue table versions, please add 'glue:GetTableVersion' permission (recommended) or delete unused table versions. If you do not have a large number of tables in Glue, please contact AWS Support."
`DataFirehose.InternalError`	"Timed out while retrieving table from Glue. If you have a large number of Glue table versions, please add 'glue:GetTableVersion' permission (recommended) or delete unused table versions. If you do not have a large number of tables in Glue, please contact AWS Support."
`DataFormatConversion.GlueEncryptionException`	"There was an error retrieving the master key. Ensure that the key exists and state is correct."

DataFormatConversion invocation errors

Amazon Data Firehose can send the following DataFormatConversion invocation errors to CloudWatch Logs.

Error Code	Error Message and Information
`DataFormatConversion.InvalidSchema`	"The schema is invalid."
`DataFormatConversion.ValidationException`	"Column names and types must be non-empty strings."

Error Code	Error Message and Information
`DataFormatConversion.ParseError`	"Encountered malformed JSON."
`DataFormatConversion.MalformedData`	"Data does not match the schema."
`DataFormatConversion.MalformedData`	"Length of json key must not be greater than 262144"
`DataFormatConversion.MalformedData`	"The data cannot be decoded as UTF-8."
`DataFormatConversion.MalformedData`	"Illegal character found between tokens."
`DataFormatConversion.InvalidTypeFormat`	"The type format is invalid. Check the type syntax."
`DataFormatConversion.InvalidSchema`	"Invalid Schema. Please ensure that there are no special characters or white spaces in column names."

Error Code	Error Message and Information
`DataFormatConversion.InvalidRecord`	"Record is not as per schema. One or more map keys were invalid for map<string,string>."
`DataFormatConversion.MalformedData`	"The input JSON contained a primitive at the top level. The top level must be an object or array."
`DataFormatConversion.MalformedData`	"The input JSON contained a primitive at the top level. The top level must be an object or array."
`DataFormatConversion.MalformedData`	"The record was empty or contained only whitespace."
`DataFormatConversion.MalformedData`	"Encountered invalid characters."
`DataFormatConversion.MalformedData`	"Encountered invalid or unsupported timestamp format. Please see the Firehose developer guide for supported timestamp formats."
`DataFormatConversion.MalformedData`	"A scalar type was found in the data but a complex type was specified on the schema."

Error Code	Error Message and Information
`DataFormatConversion.MalformedData`	"Data does not match the schema."
`DataFormatConversion.MalformedData`	"A scalar type was found in the data but a complex type was specified on the schema."
`DataFormatConversion.ConversionFailureException`	"ConversionFailureException"
`DataFormatConversion.DataFormatConversionCustomerErrorException`	"DataFormatConversionCustomerErrorException"
`DataFormatConversion.DataFormatConversionCustomerErrorException`	"DataFormatConversionCustomerErrorException"

Error Code	Error Message and Information
`DataFormatConversion.MalformedData`	"Data does not match the schema."
`DataFormatConversion.InvalidSchema`	"The schema is invalid."
`DataFormatConversion.MalformedData`	"Data does not match the schema. Invalid format for one or more dates."
`DataFormatConversion.MalformedData`	"Data contains a highly nested JSON structure that is not supported."
`DataFormatConversion.EntityNotFound`	"The specified table/database could not be found. Please ensure that the table/database exists and that the values provided in the schema configuration are correct, especially with regards to casing."
`DataFormatConversion.InvalidInput`	"Could not find a matching schema from glue. Please make sure the specified database with the supplied catalog ID exists."
`DataFormatConversion.InvalidInput`	"Could not find a matching schema from glue. Please make sure the passed ARN is in the correct format."

Error Code	Error Message and Information
`DataFormatConversion.InvalidInput`	"Could not find a matching schema from glue. Please make sure the catalogId provided is valid."
`DataFormatConversion.InvalidVersionId`	"Could not find a matching schema from glue. Please make sure the specified version of the table exists."
`DataFormatConversion.NonExistentColumns`	"Could not find a matching schema from glue. Please make sure the table is configured with a non-null storage descriptor containing the target columns."
`DataFormatConversion.AccessDenied`	"Access was denied when assuming role. Please ensure that the role specified in the data format conversion configuration has granted the Firehose service permission to assume it."
`DataFormatConversion.ThrottledByGlue`	"Throttling error encountered when calling Glue. Either increase the request rate limit or reduce the current rate of calling glue through other applications."
`DataFormatConversion.AccessDenied`	"Access was denied when calling Glue. Please ensure that the role specified in the data format conversion configuration has the necessary permissions."
`DataFormatConversion.InvalidGlueRole`	"Invalid role. Please ensure that the role specified in the data format conversion configuration exists."

Error Code	Error Message and Information
`DataFormatConversion.GlueNotAvailableInRegion`	"AWS Glue is not yet available in the region you have specified; please specify a different region."
`DataFormatConversion.GlueEncryptionException`	"There was an error retrieving the master key. Ensure that the key exists and has the correct access permissions."
`DataFormatConversion.SchemaValidationTimeout`	"Timed out while retrieving table from Glue. If you have a large number of Glue table versions, please add 'glue:GetTableVersion' permission (recommended) or delete unused table versions. If you do not have a large number of tables in Glue, please contact AWS Support."
`DataFirehose.InternalError`	"Timed out while retrieving table from Glue. If you have a large number of Glue table versions, please add 'glue:GetTableVersion' permission (recommended) or delete unused table versions. If you do not have a large number of tables in Glue, please contact AWS Support."
`DataFormatConversion.MalformedData`	"One or more fields have incorrect format."

Access CloudWatch logs for Amazon Data Firehose

You can view the error logs related to Amazon Data Firehose data delivery failure using the Amazon 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 Amazon Data Firehose console

1. Sign in to the AWS Management Console and open the Firehose console at https://console.aws.amazon.com/firehose
2. On the navigation bar, choose an AWS Region.
3. Choose a Firehose stream name to go to the Firehose stream details page.
4. 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://eusc-de-east-1.console.amazonaws-eusc.eu/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.

Monitor 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 Amazon 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 Firehose 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 Specify agent configuration settings.

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.

Monitor with CloudWatch

Kinesis Agent sends the following metrics to CloudWatch.

Metric	Description
`BytesSent`	The number of bytes sent to the Firehose 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

Log Amazon Data Firehose API calls with AWS CloudTrail

Amazon 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 Amazon Data Firehose. CloudTrail captures all API calls for Amazon Data Firehose as events. The calls captured include calls from the Amazon Data Firehose console and code calls to the Amazon 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 Amazon 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 Amazon 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.

Firehose information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When supported event activity occurs in Amazon 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 Amazon 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

Amazon 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: 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:::amzn-s3-demo-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:::amzn-s3-demo-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:::amzn-s3-demo-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"
    }
  ]
}
```

Code examples for Firehose using AWS SDKs

The following code examples show how to use Firehose with an AWS software development kit (SDK).

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Scenarios are code examples that show you how to accomplish specific tasks by calling multiple functions within a service or combined with other AWS services.

For a complete list of AWS SDK developer guides and code examples, see Using Firehose with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Code examples

- Basic examples for Firehose using AWS SDKs
  - Actions for Firehose using AWS SDKs
    - Use PutRecord with an AWS SDK or CLI
    - Use PutRecordBatch with an AWS SDK or CLI
- Scenarios for Firehose using AWS SDKs
  - Use Amazon Data Firehose to process individual and batch records

Basic examples for Firehose using AWS SDKs

The following code examples show how to use the basics of Amazon Data Firehose with AWS SDKs.

Examples

- Actions for Firehose using AWS SDKs
  - Use PutRecord with an AWS SDK or CLI
  - Use PutRecordBatch with an AWS SDK or CLI

Actions for Firehose using AWS SDKs

The following code examples demonstrate how to perform individual Firehose actions with AWS SDKs. Each example includes a link to GitHub, where you can find instructions for setting up and running the code.

These excerpts call the Firehose API and are code excerpts from larger programs that must be run in context. You can see actions in context in Scenarios for Firehose using AWS SDKs.

The following examples include only the most commonly used actions. For a complete list, see the Amazon Data Firehose API Reference.

Examples

- Use PutRecord with an AWS SDK or CLI
- Use PutRecordBatch with an AWS SDK or CLI

Use `PutRecord` with an AWS SDK or CLI

The following code examples show how to use `PutRecord`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Put records to Firehose

CLI

AWS CLI

To write a record to a stream

The following `put-record` example writes data to a stream. The data is encoded in Base64 format.

```
aws firehose put-record \
    --delivery-stream-name my-stream \
    --record '{"Data":"SGVsbG8gd29ybGQ="}'
```

Output:

```
{
    "RecordId": "RjB5K/nnoGFHqwTsZlNd/
TTqvjE8V5dsyXZTQn2JXrdpMTOwssyEb6nfC8fwf1whhwnItt4mvrn+gsqeK5jB7QjuLg283+Ps4Sz/
j1Xujv31iDhnPdaLw4BOyM9Amv7PcCuB2079RuM0NhoakbyUymlwY8yt20G8X2420wu1jlFafhci4erAt7QhDEvpw
    "Encrypted": false
}
```

For more information, see Sending Data to an Amazon Kinesis Data Firehose Delivery Stream in the *Amazon Kinesis Data Firehose Developer Guide*.

- For API details, see PutRecord in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /**
    * Puts a record to the specified Amazon Kinesis Data Firehose delivery
stream.
    *
    * @param record The record to be put to the delivery stream. The record must
be a {@link Map} of String keys and Object values.
    * @param deliveryStreamName The name of the Amazon Kinesis Data Firehose
delivery stream to which the record should be put.
    * @throws IllegalArgumentException if the input record or delivery stream
name is null or empty.
    * @throws RuntimeException if there is an error putting the record to the
delivery stream.
    */
   public static void putRecord(Map<String, Object> record, String
deliveryStreamName) {
       if (record == null || deliveryStreamName == null ||
deliveryStreamName.isEmpty()) {
           throw new IllegalArgumentException("Invalid input: record or delivery
stream name cannot be null/empty");
```

```
        }
        try {
            String jsonRecord = new ObjectMapper().writeValueAsString(record);
            Record firehoseRecord = Record.builder()

 .data(SdkBytes.fromByteArray(jsonRecord.getBytes(StandardCharsets.UTF_8)))
                .build();

            PutRecordRequest putRecordRequest = PutRecordRequest.builder()
                .deliveryStreamName(deliveryStreamName)
                .record(firehoseRecord)
                .build();

            getFirehoseClient().putRecord(putRecordRequest);
            System.out.println("Record sent: " + jsonRecord);
        } catch (Exception e) {
            throw new RuntimeException("Failed to put record: " + e.getMessage(),
 e);
        }
    }
```

- For API details, see PutRecord in *AWS SDK for Java 2.x API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class FirehoseClient:
    """
    AWS Firehose client to send records and monitor metrics.

    Attributes:
        config (object): Configuration object with delivery stream name and
 region.
        delivery_stream_name (str): Name of the Firehose delivery stream.
```

```
        region (str): AWS region for Firehose and CloudWatch clients.
        firehose (boto3.client): Boto3 Firehose client.
        cloudwatch (boto3.client): Boto3 CloudWatch client.
    """

    def __init__(self, config):
        """
        Initialize the FirehoseClient.

        Args:
            config (object): Configuration object with delivery stream name and
region.
        """
        self.config = config
        self.delivery_stream_name = config.delivery_stream_name
        self.region = config.region
        self.firehose = boto3.client("firehose", region_name=self.region)
        self.cloudwatch = boto3.client("cloudwatch", region_name=self.region)


    @backoff.on_exception(
        backoff.expo, Exception, max_tries=5, jitter=backoff.full_jitter
    )
    def put_record(self, record: dict):
        """
        Put individual records to Firehose with backoff and retry.

        Args:
            record (dict): The data record to be sent to Firehose.

        This method attempts to send an individual record to the Firehose
delivery stream.
        It retries with exponential backoff in case of exceptions.
        """
        try:
            entry = self._create_record_entry(record)
            response = self.firehose.put_record(
                DeliveryStreamName=self.delivery_stream_name, Record=entry
            )
            self._log_response(response, entry)
        except Exception:
            logger.info(f"Fail record: {record}.")
            raise
```

- For API details, see PutRecord in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
TRY.
    DATA(lo_record) = NEW /aws1/cl_frhrecord( iv_data = iv_data ).

    DATA(lo_result) = lo_frh->putrecord(
      iv_deliverystreamname = iv_deliv_stream_name
      io_record             = lo_record ).

    MESSAGE 'Record sent to Firehose delivery stream.' TYPE 'I'.
  CATCH /aws1/cx_frhresourcenotfoundex.
    MESSAGE 'Delivery stream not found.' TYPE 'E'.
  CATCH /aws1/cx_frhinvalidargumentex.
    MESSAGE 'Invalid argument provided.' TYPE 'E'.
  CATCH /aws1/cx_frhserviceunavailex.
    MESSAGE 'Service temporarily unavailable.' TYPE 'E'.
ENDTRY.
```

- For API details, see PutRecord in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Firehose with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `PutRecordBatch` with an AWS SDK or CLI

The following code examples show how to use `PutRecordBatch`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Put records to Firehose

CLI

AWS CLI

To write multiple records to a stream

The following `put-record-batch` example writes three records to a stream. The data is encoded in Base64 format.

```
aws firehose put-record-batch \
    --delivery-stream-name my-stream \
    --records file://records.json
```

Contents of `myfile.json`:

```
[
    {"Data": "Rmlyc3QgdGhpbmc="},
    {"Data": "U2Vjb25kIHRoaW5n"},
    {"Data": "VGhpcmQgdGhpbmc="}
]
```

Output:

```
{
    "FailedPutCount": 0,
    "Encrypted": false,
    "RequestResponses": [
        {
            "RecordId": "9D2OJ6t2EqCTZTXwGzeSv/EVHxRoRCw89xd+o3+sXg8DhYOaWKPSmZy/
CGlRVEys1u1xbeKh6VofEYKkoeiDrcjrxhQp9iF7sUW7pujiMEQ5LzlrzCkGosxQn
+3boDnURDEaD42V7GiixpOyLJkYZcae1i7HzlCEoy9LJhMr8EjDSi4Om/9Vc2uhwwuAtGE0XKpxJ2WD7ZRWtAnYlK
        },
        {
            "RecordId": "jFirejqxCLlK5xjH/UNmlMVcjktEN76I7916X9PaZ
+PVaOSXDfU1WGOqEZhxq2js7xcZ552eoeDxsuTU1MSq9nZTbVfb6cQTIXnm/
GsuF37Uhg67GKmR5z90l6XKJ+/
```

```
+pDloFv7Hh9a3oUS6wYm3DcNRLTHHAimANp1PhkQvWpvLRfzbuCUkBphR2QVzhP9OiHLbzGwy8/
DfH8sqWEUYASNJKS8GXP5s"
        },
        {
            "RecordId":
 "oy0amQ40o5Y2YV4vxzufdcMOOw6n3EPr3tpPJGoYVNKH4APPVqNcbUgefo1stEFRg4hTLrf2k6eliHu/9+YJ5R3
DTBt3qBlmTj7Xq8SKVbO1S7YvMTpWkMKA86f8JfmT8BMKoMb4XZS/sOkQLe+qh0sYKXWl"
        }
    ]
}
```

For more information, see Sending Data to an Amazon Kinesis Data Firehose Delivery Stream in the *Amazon Kinesis Data Firehose Developer Guide*.

- For API details, see PutRecordBatch in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /**
     * Puts a batch of records to an Amazon Kinesis Data Firehose delivery
 stream.
     *
     * @param records           a list of maps representing the records to be
 sent
     * @param batchSize         the maximum number of records to include in each
 batch
     * @param deliveryStreamName the name of the Kinesis Data Firehose delivery
 stream
     * @throws IllegalArgumentException if the input parameters are invalid (null
 or empty)
     * @throws RuntimeException         if there is an error putting the record
 batch
     */
```

```
   public static void putRecordBatch(List<Map<String, Object>> records, int
batchSize, String deliveryStreamName) {
       if (records == null || records.isEmpty() || deliveryStreamName == null ||
deliveryStreamName.isEmpty()) {
           throw new IllegalArgumentException("Invalid input: records or
delivery stream name cannot be null/empty");
       }
       ObjectMapper objectMapper = new ObjectMapper();

       try {
           for (int i = 0; i < records.size(); i += batchSize) {
               List<Map<String, Object>> batch = records.subList(i, Math.min(i +
batchSize, records.size()));

               List<Record> batchRecords = batch.stream().map(record -> {
                   try {
                       String jsonRecord =
objectMapper.writeValueAsString(record);
                       return Record.builder()

.data(SdkBytes.fromByteArray(jsonRecord.getBytes(StandardCharsets.UTF_8)))
                           .build();
                   } catch (Exception e) {
                       throw new RuntimeException("Error creating Firehose
record", e);
                   }
               }).collect(Collectors.toList());

               PutRecordBatchRequest request = PutRecordBatchRequest.builder()
                   .deliveryStreamName(deliveryStreamName)
                   .records(batchRecords)
                   .build();

               PutRecordBatchResponse response =
getFirehoseClient().putRecordBatch(request);

               if (response.failedPutCount() > 0) {
                   response.requestResponses().stream()
                       .filter(r -> r.errorCode() != null)
                       .forEach(r -> System.err.println("Failed record: " +
r.errorMessage()));
               }
               System.out.println("Batch sent with size: " +
batchRecords.size());
```

```
            }
        } catch (Exception e) {
            throw new RuntimeException("Failed to put record batch: " +
 e.getMessage(), e);
        }
    }
```

- For API details, see PutRecordBatch in *AWS SDK for Java 2.x API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class FirehoseClient:
    """
    AWS Firehose client to send records and monitor metrics.

    Attributes:
        config (object): Configuration object with delivery stream name and
 region.
        delivery_stream_name (str): Name of the Firehose delivery stream.
        region (str): AWS region for Firehose and CloudWatch clients.
        firehose (boto3.client): Boto3 Firehose client.
        cloudwatch (boto3.client): Boto3 CloudWatch client.
    """

    def __init__(self, config):
        """
        Initialize the FirehoseClient.

        Args:
            config (object): Configuration object with delivery stream name and
 region.
        """
        self.config = config
```

```
        self.delivery_stream_name = config.delivery_stream_name
        self.region = config.region
        self.firehose = boto3.client("firehose", region_name=self.region)
        self.cloudwatch = boto3.client("cloudwatch", region_name=self.region)


    @backoff.on_exception(
        backoff.expo, Exception, max_tries=5, jitter=backoff.full_jitter
    )
    def put_record_batch(self, data: list, batch_size: int = 500):
        """
        Put records in batches to Firehose with backoff and retry.

        Args:
            data (list): List of data records to be sent to Firehose.
            batch_size (int): Number of records to send in each batch. Default is
500.

        This method attempts to send records in batches to the Firehose delivery
stream.
        It retries with exponential backoff in case of exceptions.
        """
        for i in range(0, len(data), batch_size):
            batch = data[i : i + batch_size]
            record_dicts = [{"Data": json.dumps(record)} for record in batch]
            try:
                response = self.firehose.put_record_batch(
                    DeliveryStreamName=self.delivery_stream_name,
Records=record_dicts
                )
                self._log_batch_response(response, len(batch))
            except Exception as e:
                logger.info(f"Failed to send batch of {len(batch)} records.
Error: {e}")
```

- For API details, see PutRecordBatch in *AWS SDK for Python (Boto3) API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
async fn put_record_batch(
    client: &Client,
    stream: &str,
    data: Vec<Record>,
) -> Result<PutRecordBatchOutput, SdkError<PutRecordBatchError>> {
    client
        .put_record_batch()
        .delivery_stream_name(stream)
        .set_records(Some(data))
        .send()
        .await
}
```

- For API details, see PutRecordBatch in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        DATA(lo_result) = lo_frh->putrecordbatch(
          iv_deliverystreamname = iv_deliv_stream_name
          it_records            = it_records ).
```

```
        DATA(lv_failed_count) = lo_result->get_failedputcount( ).

        IF lv_failed_count > 0.
          MESSAGE |{ lv_failed_count } records failed to send.| TYPE 'I'.
        ELSE.
          MESSAGE 'All records sent successfully to Firehose delivery stream.'
TYPE 'I'.
        ENDIF.
      CATCH /aws1/cx_frhresourcenotfoundex.
        MESSAGE 'Delivery stream not found.' TYPE 'E'.
      CATCH /aws1/cx_frhinvalidargumentex.
        MESSAGE 'Invalid argument provided.' TYPE 'E'.
      CATCH /aws1/cx_frhserviceunavailex.
        MESSAGE 'Service temporarily unavailable.' TYPE 'E'.
    ENDTRY.
```

- For API details, see PutRecordBatch in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Firehose with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Scenarios for Firehose using AWS SDKs

The following code examples show you how to implement common scenarios in Firehose with AWS SDKs. These scenarios show you how to accomplish specific tasks by calling multiple functions within Firehose or combined with other AWS services. Each scenario includes a link to the complete source code, where you can find instructions on how to set up and run the code.

Scenarios target an intermediate level of experience to help you understand service actions in context.

Examples

- Use Amazon Data Firehose to process individual and batch records

Use Amazon Data Firehose to process individual and batch records

The following code examples show how to use Firehose to process individual and batch records.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

This example puts individual and batch records to Firehose.

```
/**
 * Amazon Firehose Scenario example using Java V2 SDK.
 *
 * Demonstrates individual and batch record processing,
 * and monitoring Firehose delivery stream metrics.
 */
public class FirehoseScenario {

    private static FirehoseClient firehoseClient;
    private static CloudWatchClient cloudWatchClient;

    public static void main(String[] args) {
        final String usage = """
                Usage:
                    <deliveryStreamName>
                Where:
                    deliveryStreamName - The Firehose delivery stream name.
                """;

        if (args.length != 1) {
            System.out.println(usage);
            return;
        }

        String deliveryStreamName = args[0];

        try {
            // Read and parse sample data.
            String jsonContent = readJsonFile("sample_records.json");
            ObjectMapper objectMapper = new ObjectMapper();
```

```
            List<Map<String, Object>> sampleData =
objectMapper.readValue(jsonContent, new TypeReference<>() {});

            // Process individual records.
            System.out.println("Processing individual records...");
            sampleData.subList(0, 100).forEach(record -> {
                try {
                    putRecord(record, deliveryStreamName);
                } catch (Exception e) {
                    System.err.println("Error processing record: " +
e.getMessage());
                }
            });

            // Monitor metrics.
            monitorMetrics(deliveryStreamName);

            // Process batch records.
            System.out.println("Processing batch records...");
            putRecordBatch(sampleData.subList(100, sampleData.size()), 500,
deliveryStreamName);
            monitorMetrics(deliveryStreamName);

        } catch (Exception e) {
            System.err.println("Scenario failed: " + e.getMessage());
        } finally {
            closeClients();
        }
    }

    private static FirehoseClient getFirehoseClient() {
        if (firehoseClient == null) {
            firehoseClient = FirehoseClient.builder()
                    .region(Region.US_EAST_1)
                    .build();
        }
        return firehoseClient;
    }

    private static CloudWatchClient getCloudWatchClient() {
        if (cloudWatchClient == null) {
            cloudWatchClient = CloudWatchClient.builder()
                    .region(Region.US_EAST_1)
                    .build();
```

```
        }
        return cloudWatchClient;
    }

    /**
     * Puts a record to the specified Amazon Kinesis Data Firehose delivery
stream.
     *
     * @param record The record to be put to the delivery stream. The record must
be a {@link Map} of String keys and Object values.
     * @param deliveryStreamName The name of the Amazon Kinesis Data Firehose
delivery stream to which the record should be put.
     * @throws IllegalArgumentException if the input record or delivery stream
name is null or empty.
     * @throws RuntimeException if there is an error putting the record to the
delivery stream.
     */
    public static void putRecord(Map<String, Object> record, String
deliveryStreamName) {
        if (record == null || deliveryStreamName == null ||
deliveryStreamName.isEmpty()) {
            throw new IllegalArgumentException("Invalid input: record or delivery
stream name cannot be null/empty");
        }
        try {
            String jsonRecord = new ObjectMapper().writeValueAsString(record);
            Record firehoseRecord = Record.builder()

.data(SdkBytes.fromByteArray(jsonRecord.getBytes(StandardCharsets.UTF_8)))
                .build();

            PutRecordRequest putRecordRequest = PutRecordRequest.builder()
                .deliveryStreamName(deliveryStreamName)
                .record(firehoseRecord)
                .build();

            getFirehoseClient().putRecord(putRecordRequest);
            System.out.println("Record sent: " + jsonRecord);
        } catch (Exception e) {
            throw new RuntimeException("Failed to put record: " + e.getMessage(),
e);
        }
    }
```

```
   /**
    * Puts a batch of records to an Amazon Kinesis Data Firehose delivery
stream.
    *
    * @param records           a list of maps representing the records to be
sent
    * @param batchSize         the maximum number of records to include in each
batch
    * @param deliveryStreamName the name of the Kinesis Data Firehose delivery
stream
    * @throws IllegalArgumentException if the input parameters are invalid (null
or empty)
    * @throws RuntimeException         if there is an error putting the record
batch
    */
   public static void putRecordBatch(List<Map<String, Object>> records, int
batchSize, String deliveryStreamName) {
       if (records == null || records.isEmpty() || deliveryStreamName == null ||
deliveryStreamName.isEmpty()) {
           throw new IllegalArgumentException("Invalid input: records or
delivery stream name cannot be null/empty");
       }
       ObjectMapper objectMapper = new ObjectMapper();

       try {
           for (int i = 0; i < records.size(); i += batchSize) {
               List<Map<String, Object>> batch = records.subList(i, Math.min(i +
batchSize, records.size()));

               List<Record> batchRecords = batch.stream().map(record -> {
                   try {
                       String jsonRecord =
objectMapper.writeValueAsString(record);
                       return Record.builder()

.data(SdkBytes.fromByteArray(jsonRecord.getBytes(StandardCharsets.UTF_8)))
                           .build();
                   } catch (Exception e) {
                       throw new RuntimeException("Error creating Firehose
record", e);
                   }
               }).collect(Collectors.toList());
```

```
                PutRecordBatchRequest request = PutRecordBatchRequest.builder()
                    .deliveryStreamName(deliveryStreamName)
                    .records(batchRecords)
                    .build();

                PutRecordBatchResponse response =
getFirehoseClient().putRecordBatch(request);

                if (response.failedPutCount() > 0) {
                    response.requestResponses().stream()
                        .filter(r -> r.errorCode() != null)
                        .forEach(r -> System.err.println("Failed record: " +
r.errorMessage()));
                }
                System.out.println("Batch sent with size: " +
batchRecords.size());
            }
        } catch (Exception e) {
            throw new RuntimeException("Failed to put record batch: " +
e.getMessage(), e);
        }
    }

    public static void monitorMetrics(String deliveryStreamName) {
        Instant endTime = Instant.now();
        Instant startTime = endTime.minusSeconds(600);

        List<String> metrics = List.of("IncomingBytes", "IncomingRecords",
"FailedPutCount");
        metrics.forEach(metric -> monitorMetric(metric, startTime, endTime,
deliveryStreamName));
    }

    private static void monitorMetric(String metricName, Instant startTime,
Instant endTime, String deliveryStreamName) {
        try {
            GetMetricStatisticsRequest request =
GetMetricStatisticsRequest.builder()
                .namespace("AWS/Firehose")
                .metricName(metricName)

.dimensions(Dimension.builder().name("DeliveryStreamName").value(deliveryStreamName).bui
                .startTime(startTime)
                .endTime(endTime)
```

```
                .period(60)
                .statistics(Statistic.SUM)
                .build();

            GetMetricStatisticsResponse response =
 getCloudWatchClient().getMetricStatistics(request);
            double totalSum =
 response.datapoints().stream().mapToDouble(Datapoint::sum).sum();
            System.out.println(metricName + ": " + totalSum);
        } catch (Exception e) {
            System.err.println("Failed to monitor metric " + metricName + ": " +
 e.getMessage());
        }
    }

    public static String readJsonFile(String fileName) throws IOException {
        try (InputStream inputStream =
 FirehoseScenario.class.getResourceAsStream("/" + fileName);
             Scanner scanner = new Scanner(inputStream, StandardCharsets.UTF_8))
 {
            return scanner.useDelimiter("\\\\A").next();
        } catch (Exception e) {
            throw new RuntimeException("Error reading file: " + fileName, e);
        }
    }

    private static void closeClients() {
        try {
            if (firehoseClient != null) firehoseClient.close();
            if (cloudWatchClient != null) cloudWatchClient.close();
        } catch (Exception e) {
            System.err.println("Error closing clients: " + e.getMessage());
        }
    }
}
```

- For API details, see the following topics in *AWS SDK for Java 2.x API Reference*.
  - PutRecord
  - PutRecordBatch

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

This script puts individual and batch records to Firehose.

```
import json
import logging
import random
from datetime import datetime, timedelta

import backoff
import boto3

from config import get_config


def load_sample_data(path: str) -> dict:
    """
    Load sample data from a JSON file.

    Args:
        path (str): The file path to the JSON file containing sample data.

    Returns:
        dict: The loaded sample data as a dictionary.
    """
    with open(path, "r") as f:
        return json.load(f)


# Configure logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)


class FirehoseClient:
```

```
    """
    AWS Firehose client to send records and monitor metrics.

    Attributes:
        config (object): Configuration object with delivery stream name and
region.
        delivery_stream_name (str): Name of the Firehose delivery stream.
        region (str): AWS region for Firehose and CloudWatch clients.
        firehose (boto3.client): Boto3 Firehose client.
        cloudwatch (boto3.client): Boto3 CloudWatch client.
    """

    def __init__(self, config):
        """
        Initialize the FirehoseClient.

        Args:
            config (object): Configuration object with delivery stream name and
region.
        """
        self.config = config
        self.delivery_stream_name = config.delivery_stream_name
        self.region = config.region
        self.firehose = boto3.client("firehose", region_name=self.region)
        self.cloudwatch = boto3.client("cloudwatch", region_name=self.region)


    @backoff.on_exception(
        backoff.expo, Exception, max_tries=5, jitter=backoff.full_jitter
    )
    def put_record(self, record: dict):
        """
        Put individual records to Firehose with backoff and retry.

        Args:
            record (dict): The data record to be sent to Firehose.

        This method attempts to send an individual record to the Firehose
delivery stream.
        It retries with exponential backoff in case of exceptions.
        """
        try:
            entry = self._create_record_entry(record)
            response = self.firehose.put_record(
```

```
                DeliveryStreamName=self.delivery_stream_name, Record=entry
            )
            self._log_response(response, entry)
        except Exception:
            logger.info(f"Fail record: {record}.")
            raise


    @backoff.on_exception(
        backoff.expo, Exception, max_tries=5, jitter=backoff.full_jitter
    )
    def put_record_batch(self, data: list, batch_size: int = 500):
        """
        Put records in batches to Firehose with backoff and retry.

        Args:
            data (list): List of data records to be sent to Firehose.
            batch_size (int): Number of records to send in each batch. Default is
500.

        This method attempts to send records in batches to the Firehose delivery
stream.
        It retries with exponential backoff in case of exceptions.
        """
        for i in range(0, len(data), batch_size):
            batch = data[i : i + batch_size]
            record_dicts = [{"Data": json.dumps(record)} for record in batch]
            try:
                response = self.firehose.put_record_batch(
                    DeliveryStreamName=self.delivery_stream_name,
Records=record_dicts
                )
                self._log_batch_response(response, len(batch))
            except Exception as e:
                logger.info(f"Failed to send batch of {len(batch)} records.
Error: {e}")


    def get_metric_statistics(
        self,
        metric_name: str,
        start_time: datetime,
        end_time: datetime,
        period: int,
```

```
        statistics: list = ["Sum"],
    ) -> list:
        """
        Retrieve metric statistics from CloudWatch.

        Args:
            metric_name (str): The name of the metric.
            start_time (datetime): The start time for the metric statistics.
            end_time (datetime): The end time for the metric statistics.
            period (int): The granularity, in seconds, of the returned data
points.
            statistics (list): A list of statistics to retrieve. Default is
['Sum'].

        Returns:
            list: List of datapoints containing the metric statistics.
        """
        response = self.cloudwatch.get_metric_statistics(
            Namespace="AWS/Firehose",
            MetricName=metric_name,
            Dimensions=[
                {"Name": "DeliveryStreamName", "Value":
self.delivery_stream_name},
            ],
            StartTime=start_time,
            EndTime=end_time,
            Period=period,
            Statistics=statistics,
        )
        return response["Datapoints"]

    def monitor_metrics(self):
        """
        Monitor Firehose metrics for the last 5 minutes.

        This method retrieves and logs the 'IncomingBytes', 'IncomingRecords',
and 'FailedPutCount' metrics
        from CloudWatch for the last 5 minutes.
        """
        end_time = datetime.utcnow()
        start_time = end_time - timedelta(minutes=10)
        period = int((end_time - start_time).total_seconds())

        metrics = {
```

```
            "IncomingBytes": self.get_metric_statistics(
                "IncomingBytes", start_time, end_time, period
            ),
            "IncomingRecords": self.get_metric_statistics(
                "IncomingRecords", start_time, end_time, period
            ),
            "FailedPutCount": self.get_metric_statistics(
                "FailedPutCount", start_time, end_time, period
            ),
        }

        for metric, datapoints in metrics.items():
            if datapoints:
                total_sum = sum(datapoint["Sum"] for datapoint in datapoints)
                if metric == "IncomingBytes":
                    logger.info(
                        f"{metric}: {round(total_sum)} ({total_sum / (1024 *
1024):.2f} MB)"
                    )
                else:
                    logger.info(f"{metric}: {round(total_sum)}")
            else:
                logger.info(f"No data found for {metric} over the last 5
minutes")


    def _create_record_entry(self, record: dict) -> dict:
        """
        Create a record entry for Firehose.

        Args:
            record (dict): The data record to be sent.

        Returns:
            dict: The record entry formatted for Firehose.

        Raises:
            Exception: If a simulated network error occurs.
        """
        if random.random() < 0.2:
            raise Exception("Simulated network error")
        elif random.random() < 0.1:
            return {"Data": '{"malformed": "data"'}
        else:
```

```
            return {"Data": json.dumps(record)}

    def _log_response(self, response: dict, entry: dict):
        """
        Log the response from Firehose.

        Args:
            response (dict): The response from the Firehose put_record API call.
            entry (dict): The record entry that was sent.
        """
        if response["ResponseMetadata"]["HTTPStatusCode"] == 200:
            logger.info(f"Sent record: {entry}")
        else:
            logger.info(f"Fail record: {entry}")

    def _log_batch_response(self, response: dict, batch_size: int):
        """
        Log the batch response from Firehose.

        Args:
            response (dict): The response from the Firehose put_record_batch API
 call.
            batch_size (int): The number of records in the batch.
        """
        if response.get("FailedPutCount", 0) > 0:
            logger.info(
                f'Failed to send {response["FailedPutCount"]} records in batch of
 {batch_size}'
            )
        else:
            logger.info(f"Successfully sent batch of {batch_size} records")


if __name__ == "__main__":
    config = get_config()
    data = load_sample_data(config.sample_data_file)
    client = FirehoseClient(config)

    # Process the first 100 sample network records
    for record in data[:100]:
        try:
            client.put_record(record)
        except Exception as e:
            logger.info(f"Put record failed after retries and backoff: {e}")
```

```
    client.monitor_metrics()

    # Process remaining records using the batch method
    try:
        client.put_record_batch(data[100:])
    except Exception as e:
        logger.info(f"Put record batch failed after retries and backoff: {e}")
    client.monitor_metrics()
```

This file contains config for the above script.

```
class Config:
    def __init__(self):
        self.delivery_stream_name = "ENTER YOUR DELIVERY STREAM NAME HERE"
        self.region = "us-east-1"
        self.sample_data_file = (
            "../../../../../scenarios/features/firehose/resources/
sample_records.json"
        )


def get_config():
    return Config()
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
  - PutRecord
  - PutRecordBatch

For a complete list of AWS SDK developer guides and code examples, see Using Firehose with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Troubleshoot errors in Amazon Data Firehose

If 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, 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, Firehose keeps retrying until the retention period ends.

For information about tracking delivery errors using CloudWatch, see the section called "Monitor with CloudWatch Logs".

Direct PUT

For `DirectPut` Firehose streams, Firehose retains the records for 24 hours. For a Firehose stream whose data source is a Kinesis data stream, you can change the retention period as described in Changing the Data Retention Period. In this case, Firehose retries the following operations indefinitely: `DescribeStream`, `GetRecords`, and `GetShardIterator`.

If the Firehose 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.

Kinesis Data Stream

If the Firehose 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 Firehose stream.

Common issues

The following are troubleshooting tips to help you solve common issues while you work with a Firehose stream.

Firehose stream unavailable

Firehose stream is not available as a target for CloudWatch Logs, CloudWatch Events, or AWS IoT action as some AWS services can only send messages and events to a Firehose stream that is in the

same AWS Region. Verify that your Firehose stream is located in the same Region as your other services.

No data at destination

If there are no data ingestion problems and the metrics emitted for the Firehose 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 data.

Data freshness metric increasing or not emitted

Data freshness is a measure of how current your data is within your Firehose stream. It is the age of the oldest data record in the Firehose stream, measured from the time that Firehose ingested the data to the present time. 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".

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 Firehose 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 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 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 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 Firehose 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 "Monitor with CloudWatch Logs".
- If the data source of the Firehose 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 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 "Monitor with CloudWatch Logs".
- 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 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 Firehose 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 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://eusc-de-east-1.console.amazonaws-eusc.eu/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**.

Missing fields for transformed object for Lambda

When you use Lambda data transformation to change JSON data to Parquet object, some fields might be missing after the transformation. It happens if your JSON object has capital letters and the case sensitivity is set to `false`, which can lead to a mismatch in JSON keys after data transformation causing missing data in the resulting Parquet object in the s3 bucket.

To fix this, make sure the hose configuration has the `deserializationOption: case.insensitive` set to `true` so that the JSON keys matches after the transformation.

Troubleshooting Amazon S3

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

- Check the Firehose `IncomingBytes` and `IncomingRecords` metrics to make sure that data is sent to your Firehose stream successfully. For more information, see Monitor Amazon Data Firehose with CloudWatch metrics.

- If data transformation with Lambda is enabled, check the Firehose `ExecuteProcessingSuccess` metric to make sure that Firehose has tried to invoke your Lambda function. For more information, see Monitor Amazon Data Firehose with CloudWatch metrics.
- Check the Firehose `DeliveryToS3.Success` metric to make sure that Firehose has tried putting data to your Amazon S3 bucket. For more information, see Monitor Amazon Data Firehose with CloudWatch metrics.
- Enable error logging if it is not already enabled, and check error logs for delivery failure. For more information, see Monitor Amazon Data Firehose Using CloudWatch Logs.
- If you see an error message in the log saying *"Firehose encountered InternalServerError when calling Amazon S3 service. The operation will be retried; if the error persists, please contact S3 for resolution."*, it could be due to the significant increase in request rates on a single partition in S3. You can optimize S3 prefix design patterns to mitigate the issue. For more information, see Best practices design patterns: optimizing Amazon S3 performance. If this does not resolve the issue, contact AWS Support for further assistance.
- Make sure that the Amazon S3 bucket that is specified in your Firehose stream still exists.
- If data transformation with Lambda is enabled, make sure that the Lambda function that is specified in your Firehose stream still exists.
- Make sure that the IAM role that is specified in your Firehose stream has access to your S3 bucket and your Lambda function (if data transformation is enabled). Also, make sure that the IAM role has access to CloudWatch log group and log streams to check error logs. For more information, see Grant Firehose access to an Amazon S3 destination.
- 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 Data FirehoseData Transformation.

Troubleshooting Amazon Redshift

Check the following if data is not delivered to your Amazon Redshift provisioned cluster or Amazon Redshift Serverless workgroup.

Data is delivered to your S3 bucket before loading into Amazon Redshift. If the data was not delivered to your S3 bucket, see Troubleshooting Amazon S3.

- Check the Firehose `DeliveryToRedshift.Success` metric to make sure that Firehose has tried to copy data from your S3 bucket to the Amazon Redshift provisioned cluster or Amazon Redshift Serverless workgroup. For more information, see Monitor Amazon Data Firehose with CloudWatch metrics.
- Enable error logging if it is not already enabled, and check error logs for delivery failure. For more information, see Monitor Amazon Data Firehose Using CloudWatch Logs.
- Check the Amazon Redshift `STL_CONNECTION_LOG` table to see if 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 Firehose stream is accurate and valid.
- Make sure that the IAM role that is specified in your Firehose 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). Also, make sure that the IAM role has access to CloudWatch log group and log streams to check error logs. For more information, see Grant Firehose access to an Amazon Redshift destination.
- If your Amazon Redshift provisioned cluster or Amazon Redshift Serverless workgroup is in a virtual private cloud (VPC), make sure that the cluster allows access from Firehose IP addresses. For more information, see Grant Firehose access to an Amazon Redshift destination.
- Make sure that the Amazon Redshift provisioned cluster or Amazon Redshift Serverless workgroup 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 Data FirehoseData Transformation.

Troubleshooting Amazon OpenSearch Service

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

Data can be backed up to your Amazon S3 bucket concurrently. If data was not delivered to your S3 bucket, see Troubleshooting Amazon S3.

- Check the Firehose `IncomingBytes` and `IncomingRecords` metrics to make sure that data is sent to your Firehose stream successfully. For more information, see Monitor Amazon Data Firehose with CloudWatch metrics.
- If data transformation with Lambda is enabled, check the Firehose `ExecuteProcessingSuccess` metric to make sure that Firehose has tried to invoke your Lambda function. For more information, see Monitor Amazon Data Firehose with CloudWatch metrics.
- Check the Firehose `DeliveryToAmazonOpenSearchService.Success` metric to make sure that Firehose has tried to index data to the OpenSearch Service cluster. For more information, see Monitor Amazon Data Firehose with CloudWatch metrics.
- Enable error logging if it is not already enabled, and check error logs for delivery failure. For more information, see Monitor Amazon Data Firehose Using CloudWatch Logs.
- Make sure that the OpenSearch Service configuration in your Firehose stream is accurate and valid.
- If data transformation with Lambda is enabled, make sure that the Lambda function that is specified in your Firehose stream still exists. Also, make sure that the IAM role has access to CloudWatch log group and log streams to check error logs. For more information, see Grant FirehoseAccess to a Public OpenSearch Service Destination.
- Make sure that the IAM role that is specified in your Firehose stream can access your OpenSearch Service cluster, S3 backup bucket, and Lambda function (if data transformation is enabled). Also, make sure that the IAM role has access to CloudWatch log group and log streams to check error logs. For more information, see Grant Firehose access to a public OpenSearch Service destination.
- 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 Data FirehoseData Transformation.
- Amazon Data Firehosecurrently does not support the delivery of CloudWatch Logs to Amazon OpenSearch Service destination because Amazon CloudWatch combines multiple log events into one Firehose record and Amazon OpenSearch Service cannot accept multiple log events in one record. As an alternative, you can consider Using subscription filter for Amazon OpenSearch Service in CloudWatch Logs.

Troubleshooting Splunk

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

- If your Splunk platform is in a VPC, make sure that Firehose can access it. For more information, see Accessing Splunk in VPC.
- If you use an AWS load balancer, make sure that it is a Classic Load Balancer or an Application Load Balancer. Also, enable duration-based sticky sessions with cookie expiration disabled for Classic Load Balancer and expiration is set to the maximum (7 days) for Application Load Balancer. For information about how to do this, see Duration-Based Session Stickiness for Classic Load Balancer or an Application Load Balancer.
- Review the Splunk platform requirements. The Splunk add-on for 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 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.
- 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.
- Increase the value of `HECAcknowledgmentTimeoutInSeconds` in the Splunk destination configuration of your Firehose stream.
- Increase the value of `DurationInSeconds` under `RetryOptions` in the Splunk destination configuration of your Firehose stream.
- Check your HEC health. Enabling health check is a pre-requsite for data transfer to Splunk.
- 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 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\`.
- Make sure that the IAM role that is specified in your Firehose stream can access the S3 backup bucket and the Lambda function for data transformation (if data transformation is enabled). Also, make sure that the IAM role has access to CloudWatch Logs group and log streams to check error logs. For more information, see Grant FirehoseAccess to a Splunk Destination.
- To redrive the data that was delivered to S3 error bucket (S3 backup) back to Splunk, follow the steps mentioned in the Splunk documentation.
- See Troubleshoot the Splunk Add-on for Amazon Kinesis Firehose.

Troubleshooting Snowflake

This section describes common troubleshooting steps while using Snowflake as a destination

Firehose stream creation fails

If Firehose stream creation fails for a stream delivering data to a PrivateLink-enabled Snowflake Cluster, it indicates that the VPCE-ID is not reachable by Firehose. This can be due to one of the following reasons:

- Incorrect VPCE-ID. Confirm that there are no typographic errors.
- Firehose does not support region-less Snowflake URLs in preview. Provide the URL using Snowflake Account Locator. See Snowflake documentation for more details.
- Confirm that the Firehose stream is created in the same AWS Region as the Snowflake Region.

- If the issue persists, reach out to AWS support.

Delivery failures

Check the following if data is not getting delivered to your Snowflake table. Snowflake delivery failed data will be delivered to the S3 error bucket along with an error code and an error message that corresponds to the payload. Following are few a common error scenarios. For the entire list of error codes, see Snowflake Data delivery errors.

- **Error code: Snowflake.DefaultRoleMissing**: Indicates that snowflake role is not configured while creating Firehose stream. If Snowflake role is not configured, make sure you set a default role to the Snowflake user specified.
- **Error code: Snowflake.ExtraColumns**: Indicates that insert to Snowflake is rejected due to extra columns in the input payload. Columns not present in table shouldn't be specified. Note that Snowflake column names are case-sensitive. If the delivery is failing with this error despite column being present in table, make sure that the case of the column name in input payload matches the column name declared in table definition.
- **Error code: Snowflake.MissingColumns**: Indicates that insert to Snowflake is rejected due to missing columns in input payload. Make sure that values are specified for all non-nullable columns.
- **Error code: Snowflake.InvalidInput**: This could happen when Firehose failed to parse the input payload provided into valid JSON format. Make sure that the json payload is well formed, doesn't have extra double quotes, quotes, escape characters etc. Currently Firehose supports only single JSON item as record payload, JSON arrays are not supported.
- **Error code: Snowflake.InvalidValue**: Indicates that delivery failed due to incorrect data type in the input payload. Make sure that the JSON values specified in input payload adhere to the datatype declared in Snowflake table definition.
- **Error code: Snowflake.InvalidTableType**: Indicates that table type configured in the Firehose stream is not supported. Refer to the limitations at Limitations) of snowpipe streaming for the supported tables, columns and data types.

> **Note**
>
> For any reason, if the table definition or role permissions are changed on your Snowflake destination after creating the Firehose stream, it can take several minutes for Firehose

to detect those changes. If you are seeing delivery errors due to this, try deleting and recreating the Firehose stream.

Troubleshooting Firehose endpoint reachability

If the Firehose API encounters a timeout, perform the following steps to test endpoint reachability:

- Check if API requests are made from a host in a VPC. All traffic from a VPC requires setting up a Firehose VPC endpoint. For more information, see Using Firehose with AWS PrivateLink.
- If traffic is coming from a public network or VPC with the Firehose VPC endpoint set up in a particular subnet, run the following commands from the host to check network connectivity. The Firehose endpoint can be found at Firehose endpoints and quotas.
  - Use tools like **traceroute** or **tcping** to check if the network setup is correct. If that fails, check your network setting:

    For example:

    ```
    traceroute firehose.us-east-2.amazonaws.com
    ```

    or

    ```
    tcping firehose.us-east-2.amazonaws.com 443
    ```

  - If it appears the network setting is correct and the following command fails, check whether the Amazon CA (Certficate Authority) is in the trust chain.

    For example:

    ```
    curl firehose.us-east-2.amazonaws.com
    ```

  If the above commands succeed, try the API again to see if there is a response returned from the API.

Troubleshooting HTTP Endpoints

This section describes common troubleshooting steps when dealing with Amazon 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. Make sure that the IAM role that is specified in your Firehose stream can access the S3 backup bucket and the Lambda function for data transformation (if data transformation is enabled). Also, make sure that the IAM role has access to CloudWatch log group and log streams to check error logs. For more information, see Grant Firehose Access to an HTTP Endpoint Destination.

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

CloudWatch Logs

It is highly recommended that you enable CloudWatch Logging for . 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. Amazon Data Firehose logs the response code and a truncated response payload received from the configured endpoint to CloudWatch Logs. Because Amazon 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 Amazon 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 Amazon 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 Firehose stream is incorrect or missing.
- **403**: Indicates that the access key you configured for your Firehose stream does not have permissions to deliver data to the configured endpoint.
- **413**: Indicates that the request payload that Amazon 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 Amazon 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 Amazon 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 Amazon Data Firehose received an invalid response from the endpoint destination. The response must conform to the response specifications or Amazon Data Firehose will consider the delivery attempt a failure and will redeliver the same data until the configured retry duration is exceeded. Amazon 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 Amazon 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: Amazon 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 Amazon Data Firehose compatible endpoint which will likely be different than the public ingestion endpoint. For example Datadog's Amazon 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 Amazon 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.

Troubleshooting MSK As Source

This section describes common troubleshooting steps while using MSK As Source

> **Note**
>
> For troubleshooting processing, transformation or S3 delivery issues, please refer the earlier sections

Hose creation fails

Check the following if your hose with MSK As Source is failing creation:

- Check that the source MSK cluster is in Active state.
- If you are using Private connectivity, ensure that Private Link on the cluster is turned on.

  If you are using Public connectivity, ensure that Public access on the cluster is turned on.
- If you are using Private connectivity, make sure that you add a resource based policy that allows Firehose to create Private Link. Also refer: MSK cross account permissions.
- Ensure that the role in source configuration has permission to ingest data from cluster's Topic.
- Ensure that your VPC security groups allow incoming traffic on ports used by the cluster's bootstrap servers.

Hose Suspended

Check the following if your hose is in SUSPENDED state

- Check that the source MSK cluster is in Active state.
- Check that the source topic exists. In case the topic was deleted and re-created, you will have to delete and re-create the Firehose stream as well.

Hose Backpresurred

The value of DataReadFromSource.Backpressured will be 1 when BytesPerSecondLimit per partition is exceeded or that the normal flow of delivery is slow or stopped.

- If you are hitting BytesPerSecondLimit please check DataReadFromSource.Bytes metric and request a limit increase.
- Check the CloudWatch logs, destination metrics, Data Transformation metrics and Format Conversion metrics to identify the bottlenecks.

Incorrect Data Freshness

Data freshness seems incorrect

- Firehose calculates the data freshness based on the timestamp of the consumed record. To ensure that this timestamp is correctly recorded when the producer record is persisted in the Kafka's broker logs, set the Kafka topic timestamp type configuration to be `message.timestamp.type=LogAppendTime`.

MSK cluster connection issues

The following procedure explain how you can validate connectivity to MSK clusters. For details about setting up Amazon MSK client, see Getting started using Amazon MSK in the *Amazon Managed Streaming for Apache Kafka Developer Guide*.

To validate connectivity to MSK clusters

1. Create a Unix-based (preferably AL2) Amazon EC2 instance. If you have only VPC connectivity enabled on your cluster then make sure your EC2 instance runs in the same VPC. SSH into the instance once its available. For more information, see this tutorial in the *Amazon EC2 User Guide*.
2. Install Java using the Yum package manager by running the following command. For more information, see the installation instructions in the Amazon Corretto 8 User Guide.

```
sudo yum install java-1.8.0
```

3. Install the AWS client by running the following command.

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
```

4. Download the Apache Kafka client 2.6* version by running the following command.

```
wget https://archive.apache.org/dist/kafka/2.6.2/kafka_2.12-2.6.2.tgz
tar -xzf kafka_2.12-2.6.2.tgz
```

5. Go to the `kafka_2.12-2.6.2/libs` directory, then run the following command to download the Amazon MSK IAM JAR file.

```
wget https://github.com/aws/aws-msk-iam-auth/releases/download/v1.1.3/aws-msk-iam-
auth-1.1.3-all.jar
```

6. Create `client.properties` file in Kafka bin folder.
7. Replace `awsRoleArn` with the role ARN that you have used in your Firehose `SourceConfiguration` and verify the cert location. Allow your AWS client user to assume role `awsRoleArn`. AWS client user will attempt to assume the role that you specified here.

```
[ec2-user@ip-xx-xx-xx-xx bin]$ cat client.properties
security.protocol=SASL_SSL
sasl.mechanism=AWS_MSK_IAM
sasl.jaas.config=software.amazon.msk.auth.iam.IAMLoginModule required
 awsRoleArn="<role arn>" awsStsRegion="<region name>";
sasl.client.callback.handler.class=software.amazon.msk.auth.iam.IAMClientCallbackHandler
awsDebugCreds=true
ssl.truststore.location=/usr/lib/jvm/java-1.8.0-
openjdk-1.8.0.342.b07-1.amzn2.0.1.x86_64/jre/lib/security/cacerts
ssl.truststore.password=changeit
```

8. Run the following Kafka command to list topics. If your connection is public, use the public endpoint Bootstrap servers. If your connection is private, use the private endpoint Bootstrap servers.

```
bin/kafka-topics.sh --list --bootstrap-server <bootstrap servers> --command-config
 bin/client.properties
```

If the request is successful, you should see an output similar to the following example.

```
[ec2-user@ip-xx-xx-xx-xx kafka_2.12-2.6.2]$ bin/kafka-topics.sh --list --bootstrap-
server <bootstrap servers> --command-config bin/client.properties

[xxxx-xx-xx 05:49:50,877] WARN The configuration 'awsDebugCreds' was supplied but
 isn't a known config. (org.apache.kafka.clients.admin.AdminClientConfig)
[xxxx-xx-xx 05:49:50,878] WARN The configuration 'ssl.truststore.location' was
 supplied but isn't a known config.
 (org.apache.kafka.clients.admin.AdminClientConfig)
[xxxx-xx-xx 05:49:50,878] WARN The configuration 'sasl.jaas.config' was supplied
 but isn't a known config. (org.apache.kafka.clients.admin.AdminClientConfig)
[xxxx-xx-xx 05:49:50,878] WARN The configuration
 'sasl.client.callback.handler.class' was supplied but isn't a known config.
 (org.apache.kafka.clients.admin.AdminClientConfig)
```

```
[xxxx-xx-xx 05:49:50,878] WARN The configuration 'ssl.truststore.password' was
 supplied but isn't a known config.
 (org.apache.kafka.clients.admin.AdminClientConfig)
[xxxx-xx-xx 05:50:21,629] WARN [AdminClient clientId=adminclient-1] Connection to
 node...
__amazon_msk_canary
__consumer_offsets
```

9. If you have any issues running the previous script, verify that the bootstrap servers you provided are reachable on the specified port. To do this, you could download and use **telnet** or a similar utility as shown in the following command.

```
sudo yum install telnet
telnet <bootstrap servers><port>
```

If the request is successful, you will get the following output. This means that you're able to connect to your MSK cluster within your local VPC and bootstrap servers are healthy on the specified port.

```
Connected to ..
```

10. If the request is unsuccessful, check inbound rules on your VPC security group. As an example, you could use the following properties on the inbound rule.

```
Type: All traffic
Port: Port used by the bootstrap server (e.g. 14001)
Source: 0.0.0.0/0
```

Retry the **telnet** connection as shown in the previous step. If you're still unable to connect or your Firehose connection is still failing, contact the AWS support.

Amazon Data Firehose Quota

This section describes current quotas, formerly referred to as limits, within Amazon Data Firehose. Each quota applies on a per-Region basis unless otherwise specified.

The Service Quotas console is a central location where you can view and manage your quotas for AWS services, and request a quota increase for many of the resources that you use. Use the quota information that we provide to manage your AWS infrastructure. Plan to request any quota increases in advance of the time that you'll need them.

For more information, see Amazon Data Firehose endpoints and quotas in the Amazon Web Services General Reference.

The following section shows Amazon Data Firehose has the following quota.

- With Amazon MSK as the source for the Firehose stream, each Firehose stream has a default quota of 10 MB/sec of read throughput per partition and 10MB max record size.
- With Amazon MSK as the source for the Firehose stream, there is a 6 MB maximum record size if AWS Lambda is enabled, and 10 MB maximum record size if Lambda is disabled. AWS Lambda caps its incoming record to 6 MB, and Amazon Data Firehose forwards records above 6Mb to an error S3 bucket. If Lambda is disabled, Firehose cap its incoming record to 10 MB. If Amazon Data Firehose receives a record size from Amazon MSK that is larger than 10 MB, then Amazon Data Firehose delivers this record to S3 error bucket and emits Cloudwatch metrics to your account. For more information on AWS Lambda limits, see Lambda quotas.
- When dynamic partitioning on a Firehose stream is enabled, there is a default quota of 500 active partitions that can be created for that Firehose stream. The active partition count is the total number of active partitions within the delivery buffer. For example, if the dynamic partitioning query constructs 3 partitions per second and you have a buffer hint configuration that triggers delivery every 60 seconds, then, on average, you would have 180 active partitions. Once data is delivered in a partition, then this partition is no longer active. If you need more partitions, you can create more Firehose streams and distribute the active partitions across them.
- When dynamic partitioning on a Firehose stream is enabled, a max throughput of 1 GB per second is supported for each active partition.
- Each account will have following quota for the number of Firehose streams per Region:
  - US East (N. Virginia), US East (Ohio), US West (Oregon), Europe (Ireland), Asia Pacific (Tokyo): 5,000 Firehose streams

- Europe (Frankfurt), Europe (London), Asia Pacific (Singapore), Asia Pacific (Sydney), Asia Pacific (Seoul), Asia Pacific (Mumbai), AWS GovCloud (US-West), Canada (West), Canada (Central): 2,000 Firehose streams
- Europe (Paris), Europe (Milan), Europe (Stockholm), Asia Pacific (Hong Kong), Asia Pacific (Osaka), South America (Sao Paulo), China (Ningxia), China (Beijing), Middle East (Bahrain), AWS GovCloud (US-East), Africa (Cape Town): 500 Firehose streams
- Europe (Zurich), Europe (Spain), Asia Pacific (Hyderabad), Asia Pacific (Jakarta), Asia Pacific (Melbourne), Middle East (UAE), Israel (Tel Aviv), Canada West (Calgary), Canada (Central), Asia Pacific (Malaysia), Asia Pacific (Thailand), Mexico (Central): 100 Firehose streams
- 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.

- When **Direct PUT** is configured as the data source, each Firehose 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 other AWS Regions: 100,000 records/second, 1,000 requests/second, and 1 MiB/second.

  If a Direct PUT stream experiences throttling due to higher data ingest volumes that exceed the throughput capacity of a Firehose stream, Amazon Data Firehose automatically increases the throughput limit of the stream until the throttling is contained. Depending on increased throughput and throttling, it might take longer for Firehose to increase the throughput of a stream to the desired levels. Because of this, continue to retry the failed data ingest records. If you expect the data volume to increase in sudden large bursts, or if your new stream needs a higher throughput than the default throughput limit, request to increase the throughput limit.

  There is three quota scale proportionally for quotas. 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.

  > **Note**
  >
  > - Do not use resource-level limits and quotas as a way to control your usage of the service.

- When Kinesis Data Streams is configured as the data source, this quota doesn't apply, and Amazon Data Firehose scales up and down with no limit.
- 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.
- Smaller data records can lead to higher costs. 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 Amazon Data Firehose in the AWS Calculator.

- Each Firehose 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 Amazon 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.
- Each of the following operations can provide up to five invocations per second, which is a hard limit.
  - `CreateDeliveryStream`
  - `DeleteDeliveryStream`
  - `DescribeDeliveryStream`
  - `ListDeliveryStreams`
  - `UpdateDestination`
  - `TagDeliveryStream`
  - `UntagDeliveryStream`
  - `ListTagsForDeliveryStream`
  - `StartDeliveryStreamEncryption`
  - `StopDeliveryStreamEncryption`

- The buffer interval hints range from 60 seconds to 900 seconds.
- For delivery from Amazon 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 OpenSearch Service delivery.
- When the destination is Amazon S3, Amazon Redshift, or OpenSearch Service, Amazon 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 Firehose streams.

Document history

The following table describes the important changes to the Amazon Data Firehose documentation.

Change	Description	Date Changed
Removal of Database as a source (public preview)	Database as a source (public preview) is now removed.	September 24, 2025
Added support for Glue multi-catalog hierarchy	This simplifies the Firehose integration with Amazon S3 Tables without the need of resource links between the default data catalog and `S3TablesCatalog` . See Setting up a Firehose stream to Amazon S3 tables.	May 14, 2025
Added Database as a source (public preview)	You can now replicate database changes to Apache Iceberg Tables in Amazon S3.	November 15, 2024
General Availability (GA) release for Added Apache Iceberg Tables as a destination	You can create a Firehose stream with Apache Iceberg Tables as the destination. See *Deliver data to Apache Iceberg Tables*.	September 30, 2024
Added data types examples	Added examples of supported data types for Apache Iceberg Tables. See Understand supported data types.	August 22, 2024
New Region launch	Amazon Data Firehose is now available in Asia Pacific (Malaysia). See Amazon Data Firehose Quota.	August 22, 2024
Added Apache Iceberg Tables as a	You can create a Firehose stream with Apache Iceberg Tables as the destination. See *Deliver data to Apache Iceberg Tables*.	July 25, 2024

Change	Description	Date Changed
destination (public preview)		
Buffering hints for Snowflake	Snowflake now supports buffering hints. See the section called "Configure destination settings for Snowflake".	July 25, 2024
Snowflake as a destination in new regions	Snowflake is now available as a destination in Asia Pacific (Singapore), Asia Pacific (Seoul), and Asia Pacific (Sydney). See the section called "Configure destination settings for Snowflake".	July 25, 2024
Restructured user guide sections	Simplified navigation for sections in user guide. See *Send data to a Firehose stream* and *Troubleshoot errors*.	July 5, 2024
Amazon Data Firehose integrates with AWS Secrets Manager	You can now access to your secrets and automate credential rotation securely with Secrets Manager. See the section called "Authenticate with AWS Secrets Manager".	June 06, 2024
Added support for ingesting logs for Dynatrace	You can now send logs and events to Dynatrace for further analysis. See the section called "Configure destination settings for Dynatrace (Classic cloud platform)".	April 18, 2024
General Availability (GA) release for Snowflake as a destination	Snowflake is now generally available as a destination. See the section called "Configure destination settings for Snowflake".	April 17, 2024
Amazon Kinesis Data Firehose is now known as Amazon Data Firehose	Amazon Kinesis Data Firehose has rebranded to Amazon Data Firehose. See *What is Amazon Data Firehose*	February 9, 2024

Change	Description	Date Changed
Added Snowflake as a destination (public preview)	You can create a Firehose stream with Snowflake as the destination. See the section called "Configure destination settings for Snowflake".	January 19, 2024
Added automatic decompression of CloudWatch Logs	You can enable decompression on new or existing streams to send decompressed CloudWatch Logs data to Firehose destinations. See the section called "Send CloudWatch Logs to Firehose".	December 15, 2023
Added Splunk Observability Cloud as a destination	You can create a Firehose stream with Splunk Observability Cloud as the destination. See the section called "Configure destination settings for Splunk Observability Cloud".	October 3, 2023
Added Amazon Managed Streaming for Apache Kafka as a data source	You can now configure Amazon MSK to send information to a Firehose stream. See the section called "Configure source settings for Amazon MSK".	September 26th, 2023
Added support for DocumentID type for the OpenSearch Service destination	If OpenSearch Service is your Firehose stream's destination, DocumentID type indicates the method for setting up document ID. The supported methods are Firehose generated document ID and OpenSearch Service generated document ID. See the section called "Configure destination settings".	May 10th, 2023
Added support dynamic partitioning	Added support for continuous dynamic partitioning of the streaming data in Amazon Data Firehose. See *Partition streaming data*.	August 31, 2021
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 the section called "Understand custom prefixes for Amazon S3 objects".	December 20, 2018

Change	Description	Date Changed
Added New Amazon Data Firehose Tutorial	Added a tutorial that demonstrates how to send Amazon VPC flow logs to Splunk through Amazon Data Firehose. See Ingest VPC flow logs into Splunk using Amazon Data Firehose.	October 30, 2018
Added Four New Amazon Data Firehose Regions	Added Paris, Mumbai, Sao Paulo, and London. For more information, see Amazon Data Firehose Quota.	June 27, 2018
Added Two New Amazon Data Firehose Regions	Added Seoul and Montreal. For more information, see Amazon Data Firehose Quota.	June 13, 2018
New Kinesis Streams as Source feature	Added Kinesis Streams as a potential source for records for a Firehose stream. For more information, see Choose source and destination for your Firehose stream.	August 18, 2017
Update to console documentation	The Firehose stream creation wizard was updated. For more information, see Tutorial: Create a Firehose stream from console.	July 19, 2017
New data transformation	You can configure Amazon Data Firehose to transform your data before data delivery. For more information, see Transform source data in Amazon Data Firehose.	December 19, 2016
New Amazon Redshift COPY retry	You can configure Amazon Data Firehose to retry a COPY command to your Amazon Redshift cluster if it fails. For more information, see Tutorial: Create a Firehose stream from console, Understand data delivery in Amazon Data Firehose, and Amazon Data Firehose Quota.	May 18, 2016

Change	Description	Date Changed
New Amazon Data Firehose destination, Amazon OpenSearch Service	You can create a Firehose stream with Amazon OpenSearch Service as the destination. For more information, see Tutorial: Create a Firehose stream from console, Understand data delivery in Amazon Data Firehose, and Grant Firehose access to a public OpenSearch Service destination.	April 19, 2016
New enhanced CloudWatch metrics and troubleshooting features	Updated Monitor Amazon Data Firehose and Troubleshoot errors in Amazon Data Firehose.	April 19, 2016
New enhanced Kinesis agent	Updated Configure Kinesis agent to send data.	April 11, 2016
New Kinesis agents	Added Configure Kinesis agent to send data.	October 2, 2015
Initial release	Initial release of the Amazon Data Firehose *Developer Guide*.	October 4, 2015