Informatica® Mass Ingestion
May 2024

Mass Ingestion Streaming

Informatica Mass Ingestion Mass Ingestion Streaming
May 2024

Publication Date: 2024-05-23

Table of Contents

CHAPTER 1

Mass Ingestion Streaming

Mass Ingestion Streaming is a separately licensed ingestion type of the Mass Ingestion service. Mass Ingestion Streaming can ingest data at scale from any streaming data sources, such as logs, clickstream, social media, and IoT sources. Use Mass Ingestion Streaming to ingest high-volume, real-time data from streaming sources to on-premises and cloud storage. You can also track and monitor the progress of the ingestion.

To gather operational intelligence from streaming data or to perform real-time data warehousing, you need to collect and analyze the data before it becomes obsolete or corrupted. Use Mass Ingestion Streaming to combine or separate data from streaming sources in real time. You can apply simple transformations on the data to ensure the data ingested is ready for analytics.

The Mass Ingestion service has an easy-to-use interface that runs in Informatica Intelligent Cloud Services. Use the Mass Ingestion Streaming service to define, deploy, undeploy, and monitor ingestion jobs. A job is an executable instance of an ingestion task. You can collect streaming and IoT data from different sources, apply simple transformations to the data, and then ingest the data to different types of targets. You can ingest data from sources, such as Amazon Kinesis, event logs, Google PubSub, JMS, Kafka, MQTT, OPC UA, and REST V2. You can stream data to targets, such as Amazon Kinesis, Amazon S3, Azure Event Hubs, Databricks Delta, Google BigQuery v2, Google Cloud Storage, Google PubSub, Kafka, and Microsoft Azure Data Lake Storage.

Use cases

Mass Ingestion Streaming can help you fulfill multiple usage requirements.

Consider using Streaming ingestion in the following scenarios:

- **Real-time analytics**. Ingest streaming and IoT data into messaging systems, such as Apache Kafka or Amazon Kinesis, for real-time analytics. Real-time analytics can help companies identify business opportunities and revenue streams that can result in increased profits and improved customer service.
- **Data integration**. Ingest streaming and IoT data into cloud data lakes, such as Amazon S3, for integrating data in real-time to provide up-to-the-minute information.

Mass Ingestion Streaming sources

You can ingest high volume, real-time data from supported streaming sources to on-premises and cloud targets that Mass Ingestion Streaming supports. You can ingest data in the form of events or messages.

You can use the following data sources in a streaming ingestion task:

- Amazon Kinesis Streams
- AMQP
- Azure Event Hubs Kafka
- Flat File
- Google PubSub
- JMS
- Kafka
  - Apache Kafka
  - Confluent Kafka
  - Amazon Managed Streaming (Amazon MSK)
- MQTT
- OPC UA
- REST V2

To determine the connectors to use for these source types, see *Connectors and Connections > Mass Ingestion Streaming connectors*.

Amazon Kinesis Streams sources

Use a Kinesis Streams source to read data from an Amazon Kinesis Stream. To create a Kinesis Streams source connection, use the Kinesis connection type.

Kinesis Streams is a real-time data stream processing service that Amazon Kinesis offers within the AWS ecosystem. Kinesis Streams is a customizable option that you can use to build custom applications to process and analyze streaming data. As Kinesis Streams cannot automatically scale to meet data in-flow demand, you must manually provision enough capacity to meet system needs.

Before you use a Kinesis stream source, perform the following tasks:

1. In the Amazon Kinesis console, create and configure an Amazon Kinesis stream.
2. In the Amazon Web Services (AWS) Identity and Access Management (IAM) service, create a user.
3. Download the access key and secret access key that are generated during the user creation process.
4. Associate the user with a group that has permissions to write to the Kinesis stream.

Mass Ingestion Streaming does not support profile based and cross account authentication. Amazon Web Services credentials used for Amazon Kinesis must have permissions to access to Amazon DynamoDB and Amazon CloudWatch services.

AMQP sources

Use an Advanced Message Queuing Protocol (AMQP) source to read messages from an AMQP message queue. To create an AMQP source connection, use the AMQP connection type.

AMQP is a message-oriented standard with queuing, routing, reliability and security features. AMQP is a wire-level, platform-agnostic protocol that you can use to facilitate business transactions by passing real-time message streams.

Using the AMQP connector, you can read messages from AMQP brokers, monitor a message queue, and handle subscribe patterns for brokered messaging. The streaming ingestion task uses RabbitMQ as the AMQP broker. RabbitMQ is a distributed message broker system that is fast, scalable, and durable. RabbitMQ uses AMQP 0-9-1 messaging protocol for the secure transfer of messages.

In a streaming ingestion task, you can use an AMQP source to subscribe to a stream of incoming messages. The AMQP broker stores the messages in the message queue until the streaming ingestion job receives the message off the queue. When the streaming ingestion job receives a message, the job acknowledges the receipt of the message. The acknowledged message is then removed from the message queue.

You can use the AMQP source when you have long-running tasks that you want to run as reliable background jobs. You can also choose to use an AMQP source for communication between applications where one part of the system needs to notify another part, such as order handling in a webshop.

Azure Event Hubs Kafka sources

You can configure a Kafka source to connect to Azure Event Hubs. To create an Azure Event Hubs Kafka source connection, use the Kafka connection type.

When you create a standard or dedicated tier Event Hubs namespace, the Kafka endpoint for the namespace is enabled by default. You can then use the Azure Event Hubs enabled Kafka connection as a source connection while configuring a streaming ingestion task. Enter the Event Hubs name as the topic name.

The Azure Event Hubs source information that you enter while configuring a streaming ingestion task is same as that of a normal Kafka source configuration. For more information about Azure Event Hubs Kafka source properties, see " Azure Event Hubs Kafka source properties" on page 25.

Configure the following properties while creating a Kafka connection in Administrator:

- Kafka Broker List: `NAMESPACENAME.servicebus.windows.net:9093`
- Additional Connection Properties:
  `security.protocol=SASL_SSL,sasl.mechanism=PLAIN,sasl.kerberos.service.name=Kafka`
- SSL Mode: One-Way
- SSL TrustStore File Path: Path to a trusted root cert on your file system. For example,
  `<AGENT_HOME>/jdk/jre/lib/security/cacerts`
- SSL TrustStore Password: Trustore password.
- Additional Security Properties:
  ```
  sasl.jaas.config=
  org.apache.kafka.common.security.plain.PlainLoginModule required
  username="$ConnectionString" password="Endpoint=sb://
  mynamespace.servicebus.windows.net/;SharedAccessKeyName=XXXXXX;SharedAccessKey=XXXXXX";
  ```

For more information about creating an Azure Event Hubs Kafka source connection, see the *Connections* help.

Note: Event Hubs for Kafka is available only on standard and dedicated tiers. The basic tier doesn't support Kafka on Event Hubs.

Flat File sources

Use a flat file as a source to read incoming real-time data. Configure a flat file connection to read data from flat files that are stored in the same directory.

A streaming ingestion task reads each row in a flat file source and ingests the data to a configured target. When a flat file is continuously updated in real time, the streaming ingestion task reads only the newly added content instead of reading the complete file again.

Streaming ingestion can read data from the delimited flat files. The delimiter character must be a carriage return (`\r`), a line feed (`\n`), or a combination of both.

Google PubSub sources

Use a Google PubSub source to read messages from the configured Google Cloud PubSub subscription. To create a Google PubSub source connection, use the Google PubSub connection type.

Google PubSub is an asynchronous messaging service that decouples services that produce events from services that process events. You can use Google PubSub as a messaging-oriented middleware or for event ingestion and delivery for streaming analytics pipelines. Google PubSub offers durable message storage and real-time message delivery with high availability and consistent performance at scale. You can run Google PubSub servers in all the available Google Cloud regions around the world.

Before you use Google PubSub connector, you must ensure that you meet the following prerequisites:

- Your organization has the Google PubSub Connector license.
- You have a Google service account JSON key to access Google PubSub.
- You have the client_email, client_id, and private_key values for the Google service account. You need these details when you create a Google PubSub connection in Administrator.

In a streaming ingestion task, you can use a Google PubSub source to subscribe to messages from a Google PubSub topic.

JMS sources

Use a JMS source to read data from a JMS provider. To create a JMS source connection, use the JMS connection type.

JMS providers are message-oriented middleware systems that send JMS messages. The JMS source reads JMS messages either from a JMS provider message queue or from a JMS provider based on the message topic.

The JMS source can read the following JMS message types:

- Message. Contains only header and properties fields
- TextMessage. Contains a string object. TextMessages can contain XML or JSON message data.
- BytesMessage. A stream of uninterpreted bytes. Use a BytesMessage for encoding a message body to match an existing message format. BytesMessages generally do not include property fields.
- MapMessage. Contains a set of name or value pairs. The names are in string format. The values are of Java primitive datatypes.

JMS message delivery destination types

You can choose one of the following JMS message delivery destination types:

- **Queue**. The JMS message producer delivers messages to a single consumer. The consumer must be registered to consume messages from the queue. If no consumers are registered to the queue, the queue retains the messages until a consumer registers to it.
- **Topic**. The JMS message producer delivers messages to all active consumers who subscribe to the topic. Several producers can send messages to the topic destination, and each message can be delivered to several subscribers. If no consumers are registered to the topic, the topic doesn't retain the message. You can make the subscription sharable, durable or both. A sharable subscription enables one or more consumers to access a single subscription. A durable subscription retains the message for inactive subscribers until subscribers consume the message or until the message expires.

Kafka sources

Use a Kafka source to read messages from a Kafka topic. To create a Kafka source connection, use the Kafka connection type.

Kafka is a publish-subscribe messaging system. It is an open-source distributed streaming platform that persists the streaming data in a Kafka topic. Any topic can then be read by any number of systems that need data in real-time. Kafka can serve as an interim staging area for streaming data that can be consumed by different downstream consumer applications can consume.

Kafka runs as a cluster comprised of one or more servers each of which is called a broker. Kafka brokers stream data in the form of messages. These messages are published to a topic. When you create a Kafka source, you create a Kafka consumer to read messages from a Kafka topic.

In a streaming ingestion task, you can use a Kafka source to subscribe to a stream of incoming data. When you configure a Kafka source to read from a Kafka topic, you can specify the topic name or use a Java supported regular expression to subscribe to all topics that match a specified pattern.

You can use the same Kafka connection to create an Amazon Managed Streaming for Apache Kafka (Amazon MSK) or a Confluent Kafka source connection. You can then use the Amazon MSK source or the Confluent Kafka source in a streaming ingestion task to read messages from an Apache Kafka or a Confluent Kafka topic.

MQTT sources

Use an MQTT source to read data from an MQ Telemetry Transport (MQTT) broker. To create an MQTT source, use the MQTT connection type.

MQTT is a publish-subscribe messaging system. It is a simple, lightweight, and persistent messaging protocol. It is designed for constrained devices and low-bandwidth, high-latency, or unreliable networks. Both publishers and subscribers are MQTT clients. MQTT decouples the publisher from the subscriber, so a broker manages the client connections.

An MQTT broker receives all messages, filters the messages, determines which client subscribed to each message, and then sends messages to the subscribed clients. If multiple MQTT sources connect to one MQTT broker, each connection must have a unique identifier. When you run a streaming ingestion job to ingest data from an MQTT source, Streaming Ingestion first writes the data to an internal queue before writing the data to a target.

Note: An MQTT source must have a unique client identifier. If two MQTT sources have the same client identifier, the MQTT broker rejects both the clients and the streaming ingestion job gets into *running with warning* state.

Mass Ingestion Streaming supports MQTT Quality of Service (QoS) level 1. Level 1 indicates that the client sends the message to the broker at least once, but the message might be delivered more than once. After the broker acknowledges the message receipt, the client deletes the message from the outbound queue. The QoS Level is restricted to client to broker or broker to client communication.

OPC UA sources

Use an OPC UA source to read messages from an OPC UA application tag. To create an OPC UA source connection, use the OPCUA connection type.

Open Platform Communications (OPC) is one of the important communication protocols for Industry 4.0 and the IIoT (Industrial Internet Of Things). OPC Unified Architecture (OPC UA) is a machine-to-machine communication protocol used for industrial automation. OPC UA provides a flexible and adaptable mechanism to move data between enterprise systems, monitoring devices, and sensors that interact with real-world data. You can use OPC UA to establish communication for simple downtime status or for massive amounts of highly complex plant-wide information.

The OPC UA source is a client that collects data from OPC servers. Data points in OPC are tags that represent data from devices and provide real-time access to data. In a streaming ingestion task, you can create an OPC UA source to read the incoming data based on the list of tags that you provide. You must mention the tags in a JSON array format.

REST V2 sources

Use a REST V2 source to read data from a web service application. To create a REST V2 source connection, use the REST V2 connection type.

REST V2 source connector is a generic connector for cloud applications with REST API. It supports Swagger specification version 2.0. The Swagger specification file contains operation ID, path parameters, query parameters, header fields, and payload details.

When you can create a REST V2 source connection in Administrator, for a streaming ingestion task, you can configure one of the following REST authentication types:

- Basic
- OAuth1.0
- OAuth2.0 Client Credentials
- OAuth2.0 Authorization Code
- JWT bearer token authentication

Mass Ingestion Streaming targets

You can ingest streaming data from a supported source to any on-premises and cloud targets that Mass Ingestion Streaming supports.

You can use the following targets in a streaming ingestion task:

- Amazon Kinesis Data Firehose
- Amazon Kinesis Streams
- Amazon S3

- Databricks Delta
- Flat file
- Google BigQuery V2
- Google Cloud Storage
- Google PubSub
- JDBC V2
- Kafka
  - Apache Kafka
  - Confluent Kafka
  - Amazon Managed Streaming (Amazon MSK)
- Microsoft Azure Data Lake Store Gen2
- Microsoft Azure Event Hubs

To determine the connectors to use for these target types, see *Connectors and Connections* > *Mass Ingestion Streaming connectors*.

Amazon Kinesis Data Firehose target

Use a Kinesis target to receive data from a source and write the data to an Amazon Kinesis Data Firehose target. To create a Kinesis target, use the Amazon Kinesis connection type.

Kinesis Firehose is a real-time data stream processing service that Amazon Kinesis offers within the AWS ecosystem. Use Kinesis Firehose to batch, encrypt, and compress data. Kinesis Firehose can automatically scale to meet system needs.

To configure access for Kinesis Firehose as a target, perform the following tasks:

- Create an AWS account with the required IAM permissions for the IAM user to use the AWS Kinesis Data Firehose service.
- Define a Firehose delivery stream. Configure source as Direct PUT or other sources.
- Grant required permissions to the IAM user credentials based on the target the user is writing to. For a list of permissions, see the AWS documentation at https://docs.aws.amazon.com/firehose/latest/dev/controlling-access.html#access-to-firehose.

Amazon Kinesis Streams target

Use a Kinesis target to receive data from source services and write the data to an Amazon Kinesis Stream. To create a Kinesis target, use the Amazon Kinesis connection type.

Kinesis Streams is a real-time data stream processing service that Amazon Kinesis offers within the AWS ecosystem. Kinesis Streams is a customizable option that you can use to build custom applications to process and analyze streaming data. As Kinesis Streams cannot automatically scale to meet data in-flow demand, you must manually provision enough capacity to meet system needs.

Before you use a Kinesis stream target, perform the following tasks:

1. In the Amazon Kinesis console, create and configure an Amazon Kinesis stream.
2. In the Amazon Web Services (AWS) Identity and Access Management (IAM) service, create a user.
3. Download the access key and secret access key that are generated during the user creation process.
4. Associate the user with a group that has permissions to write to the Kinesis stream.

Amazon S3 target

Use an Amazon S3 V2 connector to write the streaming data to an Amazon S3 target.

Amazon Simple Storage Service (Amazon S3) is storage service in which you can copy data from a streaming source and simultaneously move data to any target. You can use Amazon S3 to transfer the data from a list of configured source connections to an Amazon S3 target. You can accomplish these tasks by using the AWS Management Console web interface.

You can use an Amazon S3 object as a target in a streaming ingestion task. You can configure the Amazon S3 target and advanced properties for a target object.

Data partitioning

The streaming ingestion task can create partitions on the Amazon S3 V2 target and write data to the partitions. To use partitioning, you must select a partitioning interval according to which the task creates the partitions. Based on the selected time interval, the streaming ingestion job saves the incoming message in an /<object name>/<Year>/<month>/<day>/<hour>/<minutes> partition in the Amazon S3 V2 bucket. The streaming ingestion job adds timestamp hierarchy folders in the Amazon S3 bucket.

You can enable partitioning in the following ways when you configure a task:

- Add the `${Timestamp}` expression to the object name in the Object Name/Expression field and select a time interval.
  For example, enter `/streaming/${Timestamp}` in the **Object Name/Expression** field, select a partitioning interval of five minutes, and run the streaming ingestion task at 15:20 on June 8, 2022. The streaming ingestion job saves the incoming message in the /streaming/2022/06/08/15/20 partition in the Amazon S3 V2 bucket. The job saves data that streams during the next time interval under the hour folder, that is, 2022/06/08/15/25.

  Note: If you add a `${Timestamp}` expression to the object name in the Object Name/Expression field and don't select a partitioning interval, the streaming ingestion job saves the objects to the 0/ folder in the Amazon S3 bucket.
- Use a regular expression in the **Object Name/Expression** field and select a partitioning interval. You don't need to add the `${Timestamp}` expression to a regular expression.
  For example, run a streaming ingestion task at 12:10 on June 18, 2022 with a regular expression $"SourceTable":"((.*?))"$, and a partitioning interval of five minutes. The incoming data is {"SourceTable":"xyz"}. The streaming ingestion job saves the xyz object in the 2022/06/18 /12/10 folder hierarchy in the Amazon S3 bucket. The job saves data that streams during the next time interval under the hour folder, that is, 2022/06/08/15/25.

For more information, see "Amazon S3 target properties" on page 33.

Databricks Delta target

Use a streaming ingestion task to write data to a Databricks Delta target. To create a Databricks Delta target, use the Databricks Delta connection type. The Databricks Delta target requires a Databricks cluster version 6.3 or later.

Databricks Delta is an open source storage layer that provides ACID transactions and works on top of existing data lakes. Databricks uses proprietary Delta software to manage stored data and allow fast access to the data.

You can access Delta Lake tables built on top of the following storage types:

- Azure Data Lake Storage (ADLS) Gen2
- Amazon Web Services (AWS) S3

The Databricks Delta target writes data to one or more Delta Lake tables on Databricks. You can use the Databricks Delta target in a streaming ingestion task for the following use cases:

- Ingest bulk data from all streaming sources into Databricks Delta tables
- Merge change data capture (CDC) from all streaming sources and write to Databricks Delta tables

The Databricks Delta connection uses a JDBC URL to connect to the Databricks cluster. When you configure the target, you specify the JDBC URL and credentials to use to connect to the cluster. You also define the connection information that the target uses to connect to the staging location in Amazon S3 or Azure Data Lake Storage Gen2.

You specify the tables in Delta Lake to which you want to write the data. The target writes data from record fields to table columns based on matching names.

Flat file target

Use a streaming ingestion task to write data from various sources to a flat file target. The task writes real-time streaming data from various sources to the file system of the Secure Agent that runs the dataflow.

A streaming ingestion task writes the data to the staging directory with the file name that you provide. When the task adds new content to the file, it precedes it with a New Line character (\n) in the target.

The flat file target performs the file rollover action and you can configure the rollover properties. The file rollover process closes the current file and creates a new file on the basis of the file size, event count, or time. To configure rollover, specify the **Rollover Size**, **Rollover Events Count**, or the **Rollover Time** properties of the target service. The rollover process moves the file from the staging directory to the target and renames the file. The file name format of the renamed file is the original file name with an addition of the time stamp and counter information (yyyy_mm_dd-hh_mm_ss_counter). For example, during rollover, the file streaming.txt is renamed to streaming-2021_08_16-17_17_30_4.txt.

You can implement a combination of the rollover properties. For example, if you set the rollover events count to 1000, the rollover size to 1 GB, and the rollover time to 1 hour, the task rolls the file over when the file reaches a size of 1 GB even if the 1000 events are not accumulated and the 1-hour period has not elapsed.

Google BigQuery V2 target

Use a streaming ingestion task to write data to a Google BigQuery V2 database target. To create a Google BigQuery V2 target, use the Google BigQuery V2 connection type.

You can use a Google BigQuery V2 target to write data to a BigQuery table. The target consumes data in the JSON format. The target ignores the records if the fields don't match the table columns of the database. The Google BigQuery V2 target accepts data in a simple JSON format or in an array of a simple JSON format.

Before you use Google BigQuery V2 Connector, you must complete the following prerequisite tasks:

- Ensure that you have a Google service account to access Google BigQuery.
- Ensure that you have the client_email, project_id, private_key, and region ID values for the service account. Enter the values in the corresponding **Service Account ID**, **Project ID**, **Service Account Key**, and **Region ID** connection properties when you create a Google BigQuery V2 connection.
- You must specify the private_key_id and the client_id properties in the **Provide Optional Properties** field. The Google BigQuery V2 connection test fails on providing these parameters. However, you can ignore the failed test connection and run the streaming ingestion job. Use the following format:

  ```
  "private_key_id":"<private key ID>" and "client_id":"<client ID>"
  ```

- If you want to configure a timeout interval for a Google BigQuery connection, specify the timeout interval property in the **Provide Optional Properties** field of the connection properties. Use the following format:

  ```
  "timeout": "<timeout_interval_in_seconds>"
  ```

- A table to write the data to must exist before you deploy the streaming ingestion task.
- You must have read and write access to the Google BigQuery datasets that contain the target tables.

Google Cloud Storage V2 target

Use a streaming ingestion task to write data to a Google Cloud Storage target. To create a Google Cloud Storage target, use the Google Cloud Storage V2 connection type.

You can use Google Cloud Storage to stream multimedia, store custom data analytics pipelines, or distribute large data objects to users through direct download. You can write data to Google Cloud Storage for data backup. In the event of a database failure, you can read the data from Google Cloud Storage and restore it to the database.

Google Cloud Storage offers different storage classes based on factors such as data availability, latency, and price. Google Cloud Storage has the following components:

- **Projects**. In Google Cloud Storage, all resources are stored within a project. Project is a top-level container that stores billing details and user details. You can create multiple projects. A project has a unique project name, project ID, and project number.
- **Buckets**. Each bucket acts like a container that stores data. You can use buckets to organize and access data. You can create more than one bucket but you cannot nest buckets. You can create multiple folders within a bucket and you can also nest folders. You can define access control lists to manage objects and buckets. An access control list consists of permission and scope entries. Permission defines the access to perform a read or write operation. Scope defines a user or a group who can perform the operation.
- **Objects**. Objects comprise the data that you upload to Google Cloud Storage. You can create objects in a bucket. Objects consist of object data and object metadata components. The object data is a file that you store in Google Cloud Storage. The object metadata is a collection of name-value pairs that describe object qualities.

Before you use Google Cloud Storage V2 Connector, you must complete the following prerequisite tasks:

1. Ensure that you have a Google service account to access Google Cloud Storage.
2. Ensure that you have the client_email, project_id, and private_key values for the service account. You will need to enter these details when you create a Google Cloud Storage connection in the Administrator.
3. Ensure that you have enabled the Google Cloud Storage JSON API for your service account. Google Cloud Storage V2 Connector uses the Google API to integrate with Google Cloud Storage.
4. Verify that you have write access to the Google Cloud Storage bucket that contains the target file.
5. Ensure that you have enabled a license to use a Cloudera CDH or Hortonworks HDP package in your organization.

When you deploy a streaming ingestion task, the Secure Agent uses the Google Cloud Storage API to perform the specified operation and writes data to Google Cloud Storage files. You can write data into a Google Cloud Storage target. You cannot perform update, upsert, or delete operations on a Google Cloud Storage target.

Google PubSub target

Use a streaming ingestion task to write data to a Google PubSub topic. To create a Google PubSub target, use the Google PubSub connection type.

Google PubSub is an asynchronous messaging service that decouples services that produce events from services that process events. You can use Google PubSub as a messaging-oriented middleware or for event ingestion and delivery for streaming analytics pipelines. Google PubSub offers durable message storage and real-time message delivery with high availability and consistent performance at scale. You can run Google PubSub servers in all the available Google Cloud regions around the world.

Before you use Google PubSub connector, you must ensure that you meet the following prerequisites:

- Your organization has the Google PubSub Connector license.
- You have a Google service account JSON key to access Google PubSub.
- You have the client_email, client_id, and private_key values for the Google service account. You need these details when you create a Google PubSub connection in Administrator.

In a streaming ingestion task, you can use a Google PubSub target to publish to messages to a Google PubSub topic.

JDBC V2 target

Use a streaming ingestion task to write data to a database target. To create a JDBC V2 target, use the JDBC V2 connection type.

You can use a JDBC V2 target to write data to a database table. The target consumes data in JSON format. The target ignores fields that don't map to the table columns of the database.

Consider the following prerequisites before you configure JDBC V2 as a target:

- A table to write data must exist before deploying the streaming ingestion task.
- Copy the database driver files to the following directory:
  `<Secure Agent installation directory>/apps/Streaming_Ingestion_Agent/ext`

Note: The JDBC V2 target accepts data only in a simple JSON or an array of a simple JSON format.

Kafka target

Use a streaming ingestion task to write data to a Kafka target. To create a Kafka target, use the Kafka connection type.

Kafka is a publish-subscribe messaging system. It is an open-source distributed streaming platform. This platform allows systems that generate data to persist their data in real-time in a Kafka topic. Any topic can then be read by any number of systems who need that data in real-time. Kafka can serve as an interim staging area for streaming data that can be consumed by different downstream consumer applications.

Kafka runs as a cluster that comprises of one or more Kafka brokers. Kafka brokers stream data in the form of messages, publishes the messages to a topic, subscribes the messages from a topic, and then writes it to the Kafka target.

When you create a Kafka target, you create a Kafka producer to write Kafka messages. You can use each Kafka target in a streaming ingestion job that writes streaming Kafka messages. When you configure a Kafka target, specify the topic to publish the messages and the IP address and port on which the Kafka broker runs. If a Kafka topic does not exist in the target, instead of manually creating topics you can also configure your brokers to auto-create topics when a non-existent topic is published to.

You can use the same Kafka connection to create an Amazon Managed Streaming for Apache Kafka (Amazon MSK) or a Confluent Kafka connection. You can then use the Amazon MSK or the Confluent Kafka target in a streaming ingestion task to write messages to an Apache Kafka or a Confluent Kafka target.

Microsoft Azure Data Lake Storage Gen2 target

Use a streaming ingestion task to write data to a Microsoft Azure Data Lake Storage Gen2 target. To create a Microsoft Azure Data Lake Storage Gen2 target, use the Microsoft Azure Data Lake Storage Gen2 connection type.

Microsoft Azure Data Lake Storage Gen2 is a next-generation data lake solution for big data analytics. You can store data in the form of directories and sub-directories, making it efficient for data access and manipulation. You can store data of any size, structure, and format. You can process large volumes of data to achieve faster business outcomes. Data scientists and data analysts can use data in the data lake to find out specific patterns before you move the analyzed data to a data warehouse. You can use big data analytics available on top of Microsoft Azure Blob storage.

The streaming ingestion task writes data to Microsoft Azure Data Lake Storage Gen2 based on the specified conditions.

For more information about Microsoft Azure Data Lake Storage Gen2, see the Microsoft Azure Data Lake Storage Gen2 documentation.

Microsoft Azure Event Hubs target

Use a streaming ingestion task to write data to an Azure Event Hubs target. To create an Azure Event Hubs target, use the Azure Event Hubs connection type.

Azure Event Hubs is a highly scalable data streaming platform and event ingestion service that receives and processes events. Azure Event Hubs can ingest and process large volumes of events with low latency and high reliability. It is a managed service that can handle message streams from a wide range of connected devices and systems.

Any entity that sends data to an event hub is an event publisher. Event publishers can publish events using HTTPS or Kafka 1.0 and later. Event publishers use a Shared Access Signature (SAS) token to identify themselves to an event hub and can have a unique identity or use a common SAS token.

For more information about Event Hubs, see the Microsoft Azure Event Hubs documentation.

Transformations in Mass Ingestion Streaming

Transformations are part of a streaming ingestion task. Transformations represent the operations that you want to perform when ingesting streaming data.

Each transformation performs a specific function. For example, a Filter transformation filters data from the ingested data based on a specified condition.

When you create a streaming ingestion task, adding a transformation is optional. Each transformation type has a unique set of options that you can configure.

You can use the following transformations in streaming ingestion tasks:

- Combiner
- Filter
- Format Converter
- Java
- Jolt

- Python
- Splitter

You can add multiple transformations to a streaming ingestion task. In such a case, the order of transformations is important because the source data undergoes each transformation in the given order. The output of one transformation becomes the input to the next one in the task flow.

In a streaming ingestion task, you can add only one Combiner transformation and one Format Converter transformation. The Format Converter transformation must be the last transformation in the task flow. If the task includes both a Combiner transformation and a Format Converter transformation, the Format Converter transformation must be the last transformation in the task flow, preceded by the Combiner transformation.

Data formats

Each transformation type processes a specific format of incoming streaming data.

Streaming ingestion transformations can process streaming data in the following formats:

- Binary. Any type of structured and unstructured data.
- JSON. Readable format for structuring data.
- XML. Structured text data.

If a task doesn't include a transformation, it consumes the incoming data in its original format.

Combiner transformation

A Combiner transformation combines multiple events from a streaming source into a single event based on the specified conditions.

A Combiner transformation processes binary data and JSON data. For JSON message formats, the Combiner transformation combines the incoming data into an array of data and returns JSON array objects as output. For binary message formats, it combines the incoming data based on the specified conditions.

In a streaming ingestion task, you can add only one Combiner transformation. If the task includes both a Combiner transformation and a Format Converter transformation, the Format Converter transformation must be the last transformation in the task flow, preceded by the Combiner transformation. If the task doesn't include a Format Converter transformation, the Combiner transformation must be the last transformation in the task flow.

You can use one of the following conditions for a Combiner transformation:

- Minimum number of events
- Maximum aggregate size
- Time limit

For example, consider the following events:

- Record created
- Record published

If you use comma (,) as a delimiter, the Combiner transformation returns the following combined event:

Record created,Record Published

Note: When you process binary data with a Combiner transformation, you cannot use a regular expression as a delimiter.

Filter transformation

The Filter transformation filters data out of the incoming streaming events based on a specified filter condition.

You can filter data based on one or more conditions. For example, to work with data within a date range, you can create conditions to remove data based on the specified dates.

Format Converter transformation

The Format Converter transformation converts the data format of XML and JSON incoming messages to Parquet format, based on the specified conditions, before streaming them into the data lake.

You can add only one Format Converter transformation to a streaming ingestion task. The Format Converter transformation must be the last transformation in the task flow.

You can specify the date, time, and timestamp format of incoming data. If the format is not specified, it is considered in milliseconds since the epoch (Midnight, January 1, 1970, GMT).

Java transformation

A Java transformation runs the Java code to process incoming messages and send the processed data to another transformation or a target.

You can use the Java transformation to define simple or moderately complex transformation functionality. A Java transformation can process binary, JSON, and XML data.

Because you can import the Java code as snippets, you don't need to write an entire Java program. You can import a sample Java code and create and compile the Java transformation.

When you import a non-standard Java package, you must set a classpath for each JAR file or the class file directory associated with the Java package. You don't need to set a classpath for built-in Java packages. For example, java.io is a built-in Java package. If you import java.io, you don't need to set the classpath for it.

The Java transformation uses the *inputData* variable and the *outputData* variable to store the incoming data and outgoing data.

The following table shows the mapping between the data types:

Incoming data	Java data type
JSON	String
XML	String
Binary	Byte[]

Sample Java script for JSON

```
ClassPath: /<Secue Agent Location>/apps/Streaming_Ingestion_Agent/ext/json-
simple-1.1.1.jar

########/* Import Code */########
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
```

```
import org.json.simple.parser.ParseException;


########/* Main code */########
JSONParser parser = new JSONParser();
try {
    JSONObject object = (JSONObject) parser.parse(inputData);
    object.put("age", 23);
    outputData = object.toJSONString();
} catch (ParseException e) {
    throw new RuntimeException();
}

########/* inputData and outputData */########
inputData: {"name":"test"}
outputData: {"name":"test","age":23}
```

Sample Java script for binary

```
ClassPath:/<Secure Agent Location>/apps/Streaming_Ingestion_Agent/ext/binary-2.3.0.jar

########/* Import Code */########
import java.io.*;


########/* Main code */########
String temp = new String(inputData);
outputData = (temp+"-text").getBytes();

########/* inputData and outputData */########
inputData: Sample
outputData: Sample-text
```

Sample Java script for XML

```
/<Secure Agent Location>/apps/Streaming_Ingestion_Agent/ext/dom-0.9.4.jar


########/* Import Code */########
import java.io.*;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.w3c.dom.*;



########/* Main code */########
try {
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    DocumentBuilder builder = null;
    builder = factory.newDocumentBuilder();
    StringBuilder xmlStringBuilder = new StringBuilder();
    xmlStringBuilder.append(inputData);
    ByteArrayInputStream input = new
ByteArrayInputStream(xmlStringBuilder.toString().getBytes("UTF-8"));
    Document doc = builder.parse(input);
    Node entreprise = doc.getFirstChild();
    Node employee = doc.getElementsByTagName("employee").item(0);
    Element job = doc.createElement("job");
    job.appendChild(doc.createTextNode("Commercial"));
    employee.appendChild(job);
    DOMSource domSource = new DOMSource(doc);
    StringWriter writer = new StringWriter();
    StreamResult result = new StreamResult(writer);
    TransformerFactory tf = TransformerFactory.newInstance();
    Transformer transformer = tf.newTransformer();
    transformer.setOutputProperty(OutputKeys.OMIT_XML_DECLARATION, "yes");
    transformer.transform(domSource, result);
    outputData = writer.toString();
```

```
} catch (Exception e) {}


########/* inputData and outputData */########
inputData: < entreprise > < employee id = "1" > < name > Alex < /name><age>25</age > <
address > San Francisco < /address></employee > < /entreprise>
outputData: < entreprise > < employee id = "1" > < name > Alex < /name><age>25</age > <
address > San Francisco < /address><job>Commercial</job > < /employee></entreprise >
```

Jolt transformation

Use the Jolt transformation to convert complex JSON data to simple JSON data.

The Jolt transformation provides a set of operations that perform the JSON-to-JSON data conversion. You can add multiple Jolt specifications sequentially (a chain) to an array of simple specifications to form an overall JSON-to-JSON transformation. Based on the specification, the Jolt transformation transforms the complex input structure to a simple JSON structure.

Python transformation

A Python transformation runs a Python script to transform incoming data from a streaming source.

A Python transformation processes binary, JSON, and XML data. The Python transformation uses two variables, *inputData* and *outputData* to store the incoming data and outgoing data.

The *inputData* variable stores incoming data of the XML and JSON message formats as string. It stores incoming data of binary message format as `numpy.ndarray`. Binary data in the *inputData* variable is encoded as ASCII characters. You must decode the data accordingly. Ensure that the Python transformation script handles non-ASCII characters present in the *inputData* variable.

The *outputData* variable stores outgoing data of the XML and JSON message formats as string. It stores outgoing data of binary message format as `bytearray`.

Before using a Python transformation, create a directory, Python home, to install Python. After installing Python in the Python home directory, ensure to install the third-party libraries, NumPy and Jep (Java Embedded Python) in the same directory as Python home. For more information about the Python installation steps, see the Knowledge Base article 000175168.

In one Secure Agent, you can't use two different versions of Python to run the same Python transformation.

Sample Python scripts for JSON

```
import json
temp=json.loads(inputData)
temp["name"]="Mr "+temp["name"]
outputData=json.dumps(temp)
#################################################################
inputData: { "name":"John", "age":30, "city":"New York"}
outputData: { "name":"Mr John", "age":30, "city":"New York"}
```

Sample Python scripts for binary

```
temp = ''.join(str(chr(c)) for c in inputData)
temp += " - this is edited again text"
outputData = bytearray(temp, 'utf-8')
#################################################################
inputData: Sample text
outputData: Sample text - this is edited again text
```

Sample Python scripts for XML

```
import xml.etree.ElementTree as ET
myroot = ET.fromstring(inputData)
for x in myroot:
```

```
     if x.tag=="body":
         x.tag="Msg"
xmlstr = ET.tostring(myroot)
outputData=xmlstr.decode('utf-8')
################################################################
inputData: <note><to>You</to><from>Me</from><heading>Message</heading><body>Happy
Coding</body></note>
outputData: <note><to>You</to><from>Me</from><heading>Message</heading><Msg>Happy
Coding</Msg></note>
```

Splitter transformation

A Splitter transformation splits multiline messages or message arrays into separate messages based on the conditions that you specify before ingesting them into targets.

The Splitter transformation splits binary, JSON, and XML messages based on the condition that you specify and passes the separated messages into new files before ingesting them into targets. Use the Splitter transformation to split complex messages into logical components. For example, if a message contains an error code and error message separated by a comma, you can use the comma to separate the code and message into different files.

Binary messages

In binary message format, the Splitter transformation divides the messages based on line boundaries or byte sequence. The maximum number of lines determines the line boundaries. Each output split file contains no more than the configured number of lines or bytes. The default value for the line boundaries is 1. The default for the byte sequence is ','.

JSON messages

In JSON message format, the Splitter transformation divides a JSON file into separate files based on the array element specified by a JSONPath expression. Each generated file is comprised of an element of the specified array. The generated file is transferred to the downstream target or transformation in the task. If the specified JSONPath is not found or does not evaluate to an array element, the original file is routed to *failure* and no files are generated. The default JSONPath Expression is '$'.

XML messages

In XML message format, the Splitter transformation splits an XML message into many files based on the level of input depth. Each of these files contain a child or descendant of the original file.

Configuring a streaming ingestion task

In Data Integration, use the streaming ingestion task wizard to configure a streaming ingestion task. You define a source and a target, and you can optionally define a transformation to transform the data.

On the wizard pages, complete the following configuration tasks:

1. Define the basic information of a task, such as the task name, project location and runtime environment.
2. Configure a source.
3. Configure a target.
4. Optionally, add one or multiple transformations.
5. Optionally, set the runtime options.

As you work through the task wizard, you can click **Save** to save your work at any time after you configure the target. When you have completed the wizard, click **Save** to save the task.

Before you begin, verify that the conditions that are described in "Before you begin" on page 22.

Before you begin

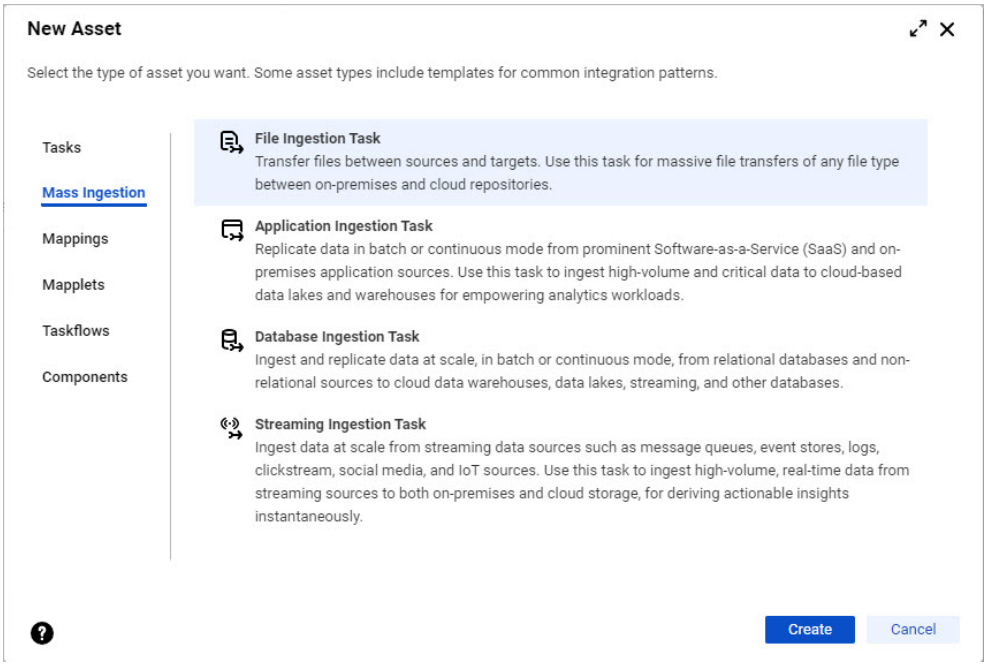
Before you create streaming ingestion tasks, verify that the following conditions are met:

- Check that your organization has licenses for Mass Ingestion Streaming and the streaming ingestion packages.
- The Mass Ingestion Streaming is running on the Secure Agent.
- Source and target connections exist.

Defining basic task information

To begin defining a streaming ingestion task, you must first enter some basic information about the task, such as task name, project or project folder location, and runtime environment.

1. Start the task wizard in one of the following ways:
   - On the Home page, click the **Ingest** panel and select **Streaming Ingestion Task**.
   - In the navigation bar on the **Explore** page or the Home page, click **New** to open the **New Asset** dialog box. Then, select **Mass Ingestion** > **Streaming Ingestion Task** and click **Create**.



   The **Definition** page of the streaming ingestion task wizard appears.

2. Configure the following properties:

Property	Description
Name	A name for the streaming ingestion task. The names of streaming ingestion tasks must be unique within the organization. Task names can contain alphanumeric characters, spaces, and underscores. Names must begin with an alphabetic character or underscore. Task names are not case-sensitive.
Location	The project folder to store the task.
Runtime Environment	Runtime environment that contains the Secure Agent. The Secure Agent runs the task.
Description	Optional. Description about the task. Maximum length is 4,000 characters.

3. Click **Next**

Configuring a source

To configure a source, select a source connection from which you want to ingest streaming data and then configure source properties. Before you configure a source, ensure that the connection to the source is created in the Administrator service.

1. On the **Source** page, select a connection.

   The streaming ingestion task supports the following sources:

   - Amazon Kinesis Streams
   - AMQP
   - Apache Kafka
   - Flat file
   - Google PubSub
   - JMS
   - MQTT
   - OPC UA
   - REST V2

   The connection type populates automatically based on the connection that you select.
2. Based on the source that you select, enter the required details.

   Options that appear on the **Source** tab of the task wizard vary based on the type of source that you select.
3. Under **Advanced Properties**, enter the required information.
4. Click **Next**.

   The **Target** tab appears.

Amazon Kinesis Streams source properties

The following table describes the Amazon Kinesis Streams source properties on the **Source** tab when you define a streaming ingestion task:

Property	Description
Connection	Name of the Amazon Kinesis Stream source connection.
Connection Type	The Amazon Kinesis connection type. The connection type populates automatically based on the connection that you select.
Stream	Name of the Kinesis Stream from which you want to read data.

The following table describes the advanced properties for Amazon Kinesis Streams sources in the **Source** tab when you define a streaming ingestion task:

Property	Description
Append GUID to DynamoDB table name	Specifies whether or not to add a GUID as a suffix to the Amazon DynamoDB table name. If disabled, you must enter the Amazon DynamoDB table name. Default is enabled.
Amazon DynamoDB	Amazon DynamoDB table name to store the checkpoint details of the Kinesis source data. The Amazon DynamoDB table name is generated automatically. However, if you enter a name of your choice, the streaming ingestion task prefixes the given name to the auto-generated name.

For more information about Kinesis Streams, see the Amazon Web Services documentation.

AMQP source properties

The following table describes the Advanced Message Queuing Protocol (AMQP) source properties on the **Source** tab when you define a streaming ingestion task:

Property	Description
Connection	Name of the AMQP source connection.
Connection Type	The AMQP connection type. The connection type populates automatically based on the connection that you select.
Queue	Name of the existing AMQP queue from which the streaming ingestion task reads the messages. This queue is pre-defined by the AMQP administrator.
Auto Acknowledge messages	You can choose True or False. If you choose True, the AMQP broker automatically acknowledges the received messages.
Batch Size	The maximum number of messages that must be pulled in a single session. Default is 10 messages.

Azure Event Hubs Kafka source properties

You can create a Kafka connection with Azure Event Hubs namespace.

When you create a standard or dedicated tier Event Hubs namespace, the Kafka endpoint for the namespace is enabled by default. You can then use the Azure Event Hubs enabled Kafka connection as a source connection while creating a streaming ingestion task. Enter the Event Hubs name as the topic name.

The following table describes the Kafka source properties on the **Source** tab when you define a streaming ingestion task:

Property	Description
Connection	Name of the Kafka source connection.
Connection Type	The Kafka connection type. The connection type populates automatically based on the connection that you select.
Topic	Name of the Event Hubs from which you want to read the events. You can either enter the topic name manually or fetch the already created metadata of the Kafka enabled Event Hubs connection. 1. Click **Select**. The **Select Source Object** dialog box appears showing all the available topics. 2. Select the required topic and click **OK**.

The following table describes the advanced properties for Kafka sources in the **Source** tab when you define a streaming ingestion task:

Property	Description
Consumer Configuration Properties	Comma-separated list of configuration properties for the consumer to connect to Kafka. Specify the values as key-value pairs. For example, `key1=value1`, `key2=value2`. The `group.id` property of Kafka consumer is autogenerated. You can override this property.

Note: Event Hubs for Kafka is available only on standard and dedicated tiers. The basic tier doesn't support Kafka on Event Hubs.

Flat File source properties

The following table describes the flat file source properties on the **Source** tab when you define a streaming ingestion task:

Property	Description
Connection	Name of the flat file source connection.
Connection Type	The Flat file connection type. The connection type appears automatically based on the connection that you select.

Property	Description
Initial Start Position	Starting position from which the data is to be read in the file to tail. You can choose one of the following positions to start reading: - Beginning of File. Read from the beginning of the file to tail. Do not ingest any data that has already been rolled over. - Current Time. Read from the most recently updated part of the file to tail. Do not ingest any data that has already been rolled over or any data in the file to tail that has already been written.
Tailing Mode	Tail a file or multiple files based on the logging pattern. You can choose one of the following modes: - Single File. Tail only one file. - Multiple Files. Tail all the files indicated in the base directory. In this mode, you can enter a regular expression to indicate the files to tail.
File	Absolute path with the name of the file you want to read. Name of the file to tail or regular expression to find the files to tail. Enter the base directory for multiple files mode.

The following table describes the advanced properties that you can configure for flat file sources on the **Source** tab when you define a streaming ingestion task:

Connection Property	Description
Rolling Filename Pattern	Name pattern for the file that rolls over. If the file to tail rolls over, the file name pattern is used to identify files that have rolled over. The underlying streaming ingestion Secure Agent recognizes this file pattern. When the Secure Agent restarts, and the file has rolled over, it picks up from where it left off. You can use asterisk (*) and question mark (?) as wildcard characters to indicate that the files are rolled over in the same directory. For example, `${filename}.log.*`. Here, asterisk (*) represents the successive version numbers that would be appended to the file name.

Google PubSub source properties

The following table describes the Google PubSub source properties on the **Source** tab when you define a streaming ingestion task:

Property	Description
Connection	Name of the Google PubSub source connection.
Connection Type	The Google PubSub connection type. The connection type populates automatically based on the connection that you select.
Subscription	Name of the subscription on the Google PubSub service from which messages should be pulled. The Google PubSub connection supports only the pull delivery type for a subscription.
Batch Size	Maximum number of messages that the Cloud service bundles together in a batch. Default is 1.

JMS source properties

The following table describes the JMS source properties on the **Source** tab when you define a streaming ingestion task:

Property	Description
Connection	Name of the JMS source connection.
Connection Type	JMS connection type. The connection type populates automatically based on the connection that you select.
Destination Type	Type of destination that the source service sends the JMS message to. You can choose one of the following destination types: - Queue. The JMS provider delivers messages to a single consumer who is registered for the queue. - Topic. The JMS provider delivers messages to all active consumers who subscribe to the topic. When you use this destination type, multiple consumers can read the message. Default is **Queue**.
Shared Subscription	Enables multiple consumers to access a single subscription. Applies to the topic destination type. Default is false.
Durable Subscription	Enables inactive subscribers to retain messages and deliver retained messages when the subscribers reconnect. Applies to the topic destination type. Default is false.
Subscription Name	Name of the subscription. Applies to the topic destination type, when the topic subscription is sharable, durable, or both. If no value is specified, the ingestion service generates a unique subscription name.
JMS Destination	Name of the queue or topic that the JMS provider delivers the message to. **Note:** If a JMS connection is created with the JMS Weblogic server, the queue or topic JMS Destination must start with a period, followed by a slash (./). For example: `./<JMS Server module name>!<Queue or topic name>` For more information about connecting Mass Ingestion Streaming to an Oracle Weblogic JMS server, see Informatica Knowledge Base article 000186952.

The following table describes the advanced properties for JMS sources in the **Source** tab when you define a streaming ingestion task:

Property	Description
Client ID	Optional. Unique identifier that identifies the JMS connection. The streaming ingestion task generates a unique client ID if a value isn't specified for an unshared durable subscription.

Kafka source properties

When you define Kafka as the source of a streaming ingestion task, you must configure the mandatory Kafka source properties on the **Source** tab. Optionally, provide a comma-separated list of consumer configuration properties.

The following table describes the mandatory Kafka source properties:

Property	Description
Connection	Name of the Kafka source connection.
Connection Type	The Kafka connection type. The connection type populates automatically based on the connection that you select.
Topic	Kafka source topic name or a Java supported regular expression for the Kafka source topic name pattern to read the events from. You can either enter the topic name manually or fetch the metadata of the Kafka connection. To select the metadata of the Kafka connection perform the following actions: 1. Click **Select**. The **Select Source Object** dialog box appears, showing all the topics or topic patterns available in the Kafka broker. 2. Select the topic and click **OK**. **Note:** When you add a new Kafka source topic to a streaming ingestion job that is in *Up and Running* state, redeploy the job immediately to avoid data loss from the new topics.

Consumer Configuration Properties

On the **Advanced Properties** section of the **Source** tab, in **Consumer Configuration Properties**, you can provide a comma-separated list of optional consumer configuration properties. Specify the values as key-value pairs.

The following table describes the consumer configuration properties that you can configure for Kafka sources:

Property	Description
group.id	Specifies the name of the consumer group the Kafka consumer belongs to. If `group.id` doesn't exist when you construct the Kafka consumer, the task creates the consumer group automatically. This property is auto-generated. You can override this property. Default is `key1=value1`, `key2=value2`.
auto.offset.reset	Specifies the behavior of the consumer when there is no committed position or when an offset is out of range. You can use the following types of auto offset reset: - Earliest. Resets the offset position to the beginning of the topic. - Latest. Resets the offset position to the latest position of the topic. - None. When you read data from a Kafka topic or use a topic pattern and the offset of the last checkpoint is deleted during message recovery, provide the following property to recover the messages from the next available offset: `auto.offset.reset=earliest` Otherwise, the streaming ingestion task reads data from the latest offset available.

Property	Description
message-demarcator	Kafka source receives messages in batches. You can contain all Kafka messages in a single batch for a given topic and partition. This property allows you to provide a string to use as a demarcation for multiple Kafka messages. If you don't provide a value, each Kafka message is triggered as a single event. You can use the following delimiters as demarcators: - New line. Separates the new content with a new line feed. Enter the following value to use a new line as a message demarcator: `message-demarcator=${literal('
'):unescapeXml()}` - Comma. Separates the new content with a comma. Enter the following value to use a comma as a message demarcator: `message-demarcator=${literal(','):unescapeXml()}` - Semicolon. Separates the new content with a semicolon. Enter the following value to use a semicolon as a message demarcator: `message-demarcator=${literal(';'):unescapeXml()}` - Tab. Separates the new content with a tab. Enter the following value to use a tab as a message demarcator: `message-demarcator=${literal(' '):unescapeXml()}`
max.poll.records	Specifies the maximum number of records returned in a single call to poll. For example, max.poll.records=100000

MQTT source properties

The following table describes the MQTT source properties on the **Source** tab when you define a streaming ingestion task:

Property	Description
Connection	Name of the MQTT source connection.
Connection Type	The MQTT connection type. The connection type populates automatically based on the connection that you select.
Topic	Name of the MQTT topic.

The following table describes the advanced properties for the MQTT source on the **Source** tab when you define a streaming ingestion task:

Connection Property	Description
Client ID	Optional. Unique identifier that identifies the connection between the MQTT source and the MQTT broker. The client ID is the file-based persistence store that the MQTT source uses to store messages when they are being processed. If you do not specify a client ID, the streaming ingestion task uses the client ID provided in the MQTT connection. However, if you have not specified the client ID even in the MQTT connection, the streaming ingestion task generates a unique client ID.
Max Queue Size	Optional. The maximum number of messages that the processor can store in memory at the same time. Default value is 1024 bytes.

OPC UA source properties

The following table describes the OPC UA source properties on the **Source** tab when you define a streaming ingestion task:

Property	Description
Connection	Name of the OPC UA source connection.
Connection Type	The OPC UA connection type. The connection type populates automatically based on the connection that you select.
Tag List Specified As	Format in which the list of tags is specified. Select one of the following formats: - **List of Tags**. List of tags to be read by the OPC client, specified as a JSON array. - **Path for Tags File**. File containing list of tags to be read by the OPC client, specified as a JSON array.
Tags or File Path	List of tags or path to the file containing the list of tags to be read, specified as a JSON array. The list of tags or file path cannot exceed 2048 characters.
Minimum Publish Interval	The minimum publish interval of subscription notification messages, in milliseconds. Set this property to a lower value to detect the rapid change of data. Default is 1,000 milliseconds.

REST V2 source properties

The following table describes the REST V2 source properties on the **Source** tab when you define a streaming ingestion task:

Property	Description
Connection	Name of the REST V2 source connection.
Connection Type	The REST V2 connection type. The connection type populates automatically based on the connection that you select.
REST Endpoints	List of REST endpoints specified in the input Swagger file. These endpoints appear based on the chosen REST connection.
Scheme	List of schemes specified in the Swagger definition. The selected scheme is used to create a the URL.
Poll Interval	Interval between two consecutive REST calls. Default is 10 seconds.
Action on Unsuccessful Response codes	Action required for unsuccessful REST calls. You can choose of the following actions: - Raise Alert - Route to Downstream. Route the response to the downstream processors. - Route to Reject Directory: Route the response to the reject directory configured in Runtime Options page.

Based on the defined operation ID in the selected **REST Endpoints** property, the dynamic properties such as, **Path**, **Query**, and **Payload** appear at the lower section of the REST V2 source page.

- **Headers**. Adds header to a REST call.
- **Path**. Consists of multiple path parameters as specified in Swagger definition. You cannot edit the **Path Key**. You can only enter corresponding values for the path keys.
- **Query**. Consists of query parameters. Query parameters are similar to path parameters.
- **Payload**.
  - **Sample Payload**. A read-only text box that shows schema of request body for a PUT, POST, or PATCH request. For example, `{ "name" : "string", "salary" : "string", "age" : "string" }`.
  - **Body**. The request body to be sent incase of PUT, POST, or PATCH request. You can copy a sample request body from a sample payload and then can replace the values as appropriate.

You can define any of these properties as mandatory in the Swagger specification file. Then, the same property is considered mandatory while configuring a streaming ingestion REST V2 source. If you do not define a REST endpoint in the Swagger specification file, the corresponding section does not appear on the streaming ingestion REST V2 page.

Note: When you configure a streaming ingestion task, the absolute path of the Swagger specification file you provide must be available in the runtime environment.

Configuring a target

To configure a target, select a target connection to which you want to transfer the streaming data and then configure the target properties. Before you configure a target, ensure that the connection to the target is created in the Administrator service.

1. On the **Target** page, select a connection.

   The streaming ingestion task supports the following targets:

   - Amazon Kinesis Data Firehose
   - Amazon Kinesis Streams
   - Amazon S3 V2
   - Apache Kafka
   - Databricks Delta
   - Flat file
   - Google BigQuery V2
   - Google PubSub
   - Google Cloud Storage V2
   - JDBC V2
   - Microsoft Azure Data Lake Storage Gen2
   - Microsoft Azure Event Hubs

2. Based on the target that you select, enter the required details.

   Options that appear on the **Target** tab of the task wizard vary based on the type of target that you select.

3. Under **Advanced Properties**, enter the required information.
4. Perform one of the following tasks:

   - To add a transformation, click **Next**.
     The **Transformation** tab appears.
   - To save the task, click **Save**.
     You can then deploy the streaming ingestion task. For more information about deploying the streaming ingestion task, see "Deploying a streaming ingestion task" on page 47.

Amazon Kinesis Data Firehose target properties

The following table describes the Amazon Kinesis Data Firehose target properties on the **Target** tab when you define a streaming ingestion task:

Property	Description
Connection	Name of the Amazon Kinesis Data Firehose target connection.
Connection Type	The Amazon Kinesis connection type. The connection type populates automatically based on the connection that you select.
Stream Name/ Expression	Kinesis stream name or a regular expression for the Kinesis stream name pattern. Use the `$expression$` format for the regular expression. `$expression$` evaluates the data and sends the matching data to capturing group 1.

For more information about Kinesis Data Firehose, see the Amazon Web Services documentation.

Amazon Kinesis Streams target properties

The following table describes the Amazon Kinesis Streams target properties on the **Target** tab when you define a streaming ingestion task:

Property	Description
Connection	Name of the Amazon Kinesis Stream target connection.
Connection Type	The Amazon Kinesis connection type. The connection type populates automatically based on the connection that you select.
Stream Name/ Expression	Kinesis stream name or a regular expression for the Kinesis stream name pattern. Use the `$expression$` format for the regular expression. `$expression$` evaluates the data and sends the matching data to capturing group 1.

For more information about Kinesis Streams, see the Amazon Web Services documentation.

Amazon S3 target properties

The following table describes the Amazon S3 target properties on the **Target** tab when you define a streaming ingestion task:

Property	Description
Connection	Name of the Amazon S3 target connection.
Connection Type	The Amazon S3 V2 connection type. The connection type populates automatically based on the connection that you select.
Object Name/ Expression	Amazon S3 file name or a regular expression for the Amazon S3 file name pattern. Use the `$expression$` format for a regular expression. `$expression$` evaluates the data and sends the matching data to capturing group 1.

The following table describes the Amazon S3 advanced target properties that you can configure on the **Target** tab when you define a streaming ingestion task:

Property	Description
Partitioning Interval	Optional. The time interval according to which the streaming ingestion task creates partitions in the Amazon S3 bucket. To use this option, you must add a `${Timestamp}` expression to the object name in the **Object Name/Expression** field. Default is none. For more information, see "Amazon S3 target" on page 12.
Minimum Upload Part Size	Optional. Minimum upload part size when uploading a large file as a set of multiple independent parts, in megabytes. Use this property to tune the file load to Amazon S3. Default value is 5120 MB.
Multipart Upload Threshold	Optional. Multipart download minimum threshold to determine when to upload objects in multiple parts in parallel. Default value is 5120 MB.

Azure Event Hubs target properties

The following table describes the Azure Event Hubs target properties on the **Target** tab when you define a streaming ingestion task:

Property	Description
Connection	Name of the Azure Event Hubs target connection.
Connection Type	The Azure Event Hubs connection type. The connection type populates automatically based on the connection that you select.
Event Hub	The name of the Azure Event Hubs.

The following table describes the Azure Event Hubs advanced target properties that you can configure on the **Target** tab when you define a streaming ingestion task:

Property	Description
Shared Access Policy Name	Optional. The name of the Event Hub Namespace Shared Access Policy. The policy must apply to all data objects that are associated with this connection. To read from Event Hubs, you must have Listen permission. To write to an Event Hub, the policy must have Send permission.
Shared Access Policy Primary Key	Optional. The primary key of the Event Hub Namespace Shared Access Policy.

Databricks Delta target properties

The following table describes the Databricks Delta target properties on the **Target** tab when you define a streaming ingestion task:

Property	Description
Connection	Name of the Databricks Delta target connection.
Connection Type	The Databricks Delta connection type. The connection type populates automatically based on the connection that you select.
Staging Location	Relative directory path to store the staging files. - If the Databricks cluster is deployed on AWS, use the relative path of the Amazon S3 staging bucket. - If the Databricks cluster is deployed on Azure, use the relative path of the Azure Data Lake Store Gen2 staging file system name.
Target Table Name	Name of the Databricks Delta table to append.

The following table describes the Databricks Delta target advanced properties that you can configure on the **Target** tab when you define a streaming ingestion task:

Property	Description
Target Database Name	Overrides the database name provided in the Databricks Delta connection in Administrator.

For a Databricks Delta target, the source messages must be only in JSON format.

Note: In a streaming ingestion job with Databricks Delta target, when you change the source schema to include additional data columns, Informatica recommends that you redeploy the job to include the change data capture.

When you use a Filter transformation in a streaming ingestion task with a Databricks Delta target, ensure that the ingested data conforms to a valid JSON data format. The Filter transformation with JSONPath filter type validates the incoming data. If the incoming data does not conform to a valid JSON data format, the streaming ingestion task rejects the data. The rejected data then moves into the configured reject directory. If you do not have a reject directory already configured, the rejected data is lost.

Informatica recommends that you use a Combiner transformation in the streaming ingestion task that contains a Databricks Delta target. Add the Combiner transformation before writing to the target. The streaming ingestion task then combines all the staged data before writing into the Databricks Delta target. To optimize performance, batch 100 records or more.

Flat file target properties

The following table describes the flat file target properties on the **Target** tab when you define a streaming ingestion task:

Property	Description
Connection	Name of the flat file target connection.
Connection Type	The flat file connection type. The connection type appears based on the connection that you select.
Staging Directory Location	Path to the staging directory on the Secure Agent. Specify the staging directory where to stage the files when you write data to a flat file target. Ensure that the directory has sufficient space and you have write permissions to the directory.
Rollover Size *	The file size, in KB, at which the task moves the file from the staging directory to the target. For example, set the rollover size to 1 MB and name the file target.log. If the source service sends 5 MB to the target, the streaming ingestion task first creates the target.log.<timestamp> file. When the size of target.log.<timestamp> reaches 1 MB, the task rolls the file over.
Rollover Events Count *	Number of events or messages to accumulate for file rollover. For example, if you set the rollover events count to 1000, the task rolls the file over when the file accumulates 1000 events.
Rollover Time *	Length of time, in milliseconds, for a target file to roll over. After the time period has elapsed, the target file rolls over. For example, if you set rollover time as 1 hour, the task rolls the file over when the file reaches a period of 1 hour.

Property	Description
File Name	The name of the file that the task creates on the target.
* Specify a value for at least one rollover option to perform target file rollover.	

Google BigQuery V2 target properties

The following table describes the Google BigQuery V2 target properties on the **Target** tab when you define a streaming ingestion task:

Property	Description
Connection	Name of the Google BigQuery V2 target connection.
Connection Type	The Google BigQuery V2 connection type. The connection type populates automatically based on the connection that you select.
Dataset Name	Name of the Google BigQuery dataset. The dataset must exist in the Google Cloud Platform.
Table name	Name of the Google BigQuery table to insert data to in JSON format.

Google Cloud Storage target properties

The following table describes the Google Cloud Storage target properties on the **Target** tab when you define a streaming ingestion task:

Property	Description
Connection	Name of the Google Cloud Storage target connection.
Connection Type	The Google Cloud Storage connection type. The connection type populates automatically based on the connection that you select.
Number of Retries	The number of times the streaming ingestion task retries to write to the Google Cloud Storage target. Default is 6.
Bucket	The container to store, organize, and access objects that you upload to Google Cloud Storage.
Key	Name of the Google Cloud Storage target object.

The following table describes the Google Cloud Storage advanced target properties on the **Target** tab when you define a streaming ingestion task:

Property	Description
Proxy Host	Host name of the outgoing proxy server that the Secure Agent uses.
Proxy Port	Port number of the outgoing proxy server.

Property	Description
Content Type	The file content type. You can specify any MIME types, such as application.json, multipart, text, or html. These values are not case sensitive. Default is text.
Object ACL	Access control associated with the uploaded object. Choose one of the following types of authentication: - **Authenticated Read**. Gives the bucket or object owner FULL_CONTROL permission and gives all authenticated Google account holders READ permission. - **Bucket Owner Full Control**. Grants full control permission to the bucket or object owner and grants read permission to all the authenticated Google account holders. - **Bucket Owner Read Only**. Grants full control permission to the object owner and grants read permission to the bucket owner. Use this type only with objects. - **Private**. Gives the bucket or object owner FULL_CONTROL permission for a bucket or object. - **Project Private**. Gives permission to the project team based on their roles. Anyone who is part of the team has READ permission and project owners and project editors have FULL_CONTROL permission. This is the default ACL for newly created buckets. - **Public Read Only**. Gives the bucket owner FULL_CONTROL permission and gives all anonymous users READ and WRITE permission. This ACL applies only to buckets. When you apply this to a bucket, anyone on the Internet can list, create, overwrite, and delete objects without authenticating.
Server Side Encryption Key	Server-side encryption key for the Google Cloud Storage bucket. Required if the Google Cloud Storage bucket is encrypted with SSE-KMS.
Content Disposition Type	Type of RFC-6266 Content Disposition to be attached to the object. Choose either **Inline** or **Attachment**.

Google PubSub target properties

The following table describes the Google PubSub target properties on the **Target** tab when you define a streaming ingestion task:

Property	Description
Connection	Name of the Google PubSub target connection.
Connection Type	The Google PubSub connection type. The connection type populates automatically based on the connection that you select.
Topic	Name of the target Google PubSub topic.
Batch Size	Maximum number of messages that the Cloud service bundles together in a batch. Default is 1.

JDBC V2 target properties

The following table describes the JDBC V2 target properties on the **Target** tab when you define a streaming ingestion task:

Property	Description
Connection	Name of the JDBC V2 target connection.
Connection Type	The JDBC V2 connection type. The connection type populates automatically based on the connection that you select.
Table name	Name of the table to insert data to in JSON format.

Kafka target properties

The following table describes the Kafka target properties that on the **Target** tab when you define a streaming ingestion task:

Property	Description
Connection	Name of the Kafka target connection.
Connection Type	The Kafka connection type. The connection type populates automatically based on the connection that you select.
Topic Name/ Expression	Kafka topic name or a Java supported regular expression for the Kafka topic name pattern. Use the `$expression$` format for the regular expression. `$expression$` evaluates the data and sends the matching data to capturing group 1. You can either enter the topic name manually or fetch the already created metadata of the Kafka connection. 1. Click **Select**. The **Select Target Object** dialog box appears showing all the topics available in the Kafka broker. However, Kafka topic name patterns do not appear in the list. 2. Select the required topic and click **OK**.

The following table describes the Kafka advanced target properties on the **Target** tab when you define a streaming ingestion task:

Property	Description
Producer Configuration Properties	The configuration properties for the producer.
Metadata Fetch Timeout in milliseconds	The time after which the metadata is not fetched.
Batch Flush Size in bytes	The batch size of the events after which a streaming ingestion task writes data to the target.

Microsoft Azure Data Lake Storage Gen2 target properties

The following table describes the Microsoft Azure Data Lake Storage Gen2 (ADLS Gen2) target properties on the **Target** tab when you define a streaming ingestion task:

Property	Description
Connection	Name of the Microsoft Azure Data Lake Storage Gen2 target connection.
Connection Type	The ADLS Gen2 connection type. The connection type populates automatically based on the connection that you select.
Write Strategy	The operation type to write data to ADLS Gen2 file. If the file exists in ADLS Gen2 storage, you can select to overwrite, append, fail, or rollover the existing file. Default is **Append**. - **Append**. Add data to an existing file inside a directory. - **Overwrite**. Delete existing data in an existing file and insert newly read data. - **Fail**. Write data to an existing file fails. - **Rollover**. Close the current file to which data is being written to and create a new file based on the configured rollover value. **Note:** When you edit or redeploy a streaming ingestion job that contains a target with the rollover strategy, all the files in the staging directory are moved to the target directory even if the defined rollover conditions are not met.
Interim Directory	Path to the staging directory in ADLS Gen2. Specify the staging directory where you want to stage the files when you write data to ADLS Gen2. Ensure that the directory has sufficient space and you have write permissions to the directory. Applicable when you select the **Write Strategy** as Rollover. While configuring an ADLS Gen 2 target in a streaming ingestion job, if you do not specify any value for the rollover properties, the files remain in the interim directory. When you stop or undeploy the streaming ingestion job, these files in the interim directory are moved to the target location, by default.
Rollover Size	Target file size, in kilobytes (KB), at which to trigger rollover. Applicable when you select the **Write Strategy** as Rollover.
Rollover Events Count	Number of events or messages that you want to accumulate for the rollover. Applicable when you select the **Write Strategy** as Rollover.
Rollover Time	Length of time, in milliseconds, for a target file to roll over. After the time period has elapsed, the target file rolls over. Applicable when you select the **Write Strategy** as Rollover.
File Name/ Expression	File name or a regular expression for the file name pattern. Use the `$expression$` format for the regular expression. `$expression$` evaluates the data and sends the matching data to capturing group 1.

The following table describes the Microsoft Azure Data Lake Storage Gen2 (ADLS Gen2) advanced target properties on the **Target** tab when you define a streaming ingestion task:

Property	Description
Filesystem Name Override	Overrides the default file system name provided in connection. This file system name is used write to a file at run time.
Directory Override	Overrides the default directory path. The ADLS Gen2 directory that you use to write data. Default is root directory. The directory path specified while creating the target overrides the path specified while creating a connection.
Compression Format	Optional. Compression format to use before the streaming ingestion task writes data to the target file. Use one of the following formats: - None - Gzip - Bzip2 - Zlib - Deflate Default is **None**. To read a compressed file from the data lake storage, the compressed file must have specific extensions. If the extensions used to read the compressed file are not valid, the Secure Agent does not process the file.

Configuring a transformation

You can specify the data format of the streaming data. Based on the data format, you can configure a transformation.

1. On the **Transformation** page, select the format of the streaming data.

   The streaming ingestion transformations support the following data formats:

   - Binary
   - JSON
   - XML
2. Based on the selected data format, select one of the supported transformations, and configure it.
3. To add more than one transformation, click **Add Transformations**.
   a. On the **Transformation** tab, click **Add Transformation**.

      The **New Transformation** dialog box appears.

   b. Based on the transformation type you select, enter the required properties.

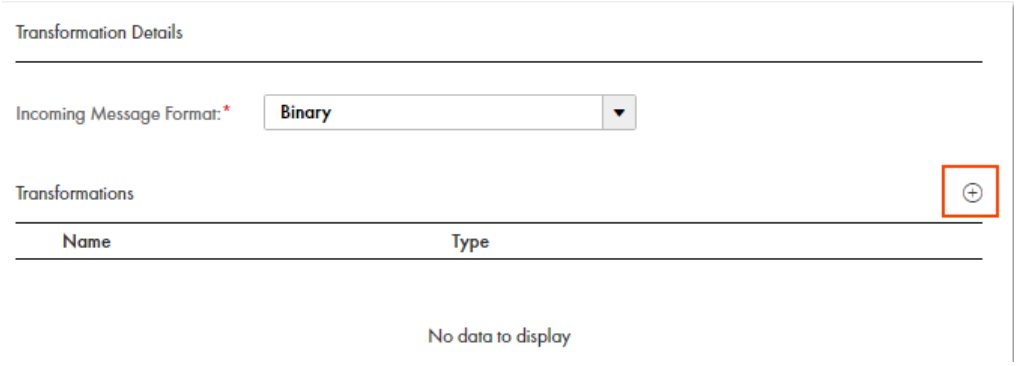
   c. Click **Save**.

      The saved transformation appears under **Transformations** in the **Transformation Details** wizard.
4. Perform one of the following tasks:
   - To configure the runtime options for the task, click **Next**.
     The **Runtime Options** tab appears.
   - To save the task, click **Save**.

You can then deploy the streaming ingestion task. For more information about deploying the streaming ingestion task, see "Deploying a streaming ingestion task" on page 47.

Adding a transformation

1. On the **Transformation** tab, click **+** to add a transformation.



   The **New Transformation** dialog box appears.
2. Based on the transformation type you select, enter the required properties.
3. Click **Save**.

   The saved transformation appears under **Transformations** in the **Transformation Details** wizard.

Combiner transformation properties

The following table describes the properties you can configure for a Combiner transformation:

Property	Description
Transformation Type	Select Combiner.
Transformation Name	Name of the Combiner transformation.
Minimum Number of Events	Minimum number of events to collect before the transformation combines the events into a single event. Default is 1.
Maximum Aggregate Size	Maximum size of the combined events in megabytes. If not specified, this transformation waits to meet any of the other two conditions before combining the events.
Time Limit	Maximum time to wait before combining the events. If not specified, this transformation waits for the other conditions before combining the events or waits forever.
Delimiter	Symbols used to specify divisions between data strings in the transformed data. Applicable only for the binary data format.
Append the delimiter character to the last record in each batch	When there are many batches with events or records, you can choose whether to use the delimiter character at the end of the last record in each batch. This enables the delimiter character to act as the division between each batch.

Filter transformation properties

The following table describes the properties you can configure for a Filter transformation:

Property	Description
Transformation Type	Select Filter.
Transformation Name	Name of the Filter transformation.
Filter Type	Type of filter to evaluate the incoming data. Use one of the following filter types: - JSON Path. An expression that consists of a sequence of JSON properties. - Regular Expression. A range or pattern of values. - XPath. An expression that selects nodes or node-sets in an XML document.
Expression	Expression for the filter type that you select.

Format Converter transformation properties

When you define a streaming ingestion task and add a Format Converter transformation, provide values for transformation properties on the **New Transformation** page of the task wizard.

The following table describes the properties you can configure for a Format Converter transformation:

Property	Description
Transformation Type	Select Format Converter.
Transformation Name	Name of the Format Converter transformation.
Convert to Format	The streaming ingestion task converts incoming data to the selected format. Currently, the Format Converter transformation converts the incoming data only to Parquet format.
Date Format *	Enter the format of dates in input fields. For example, MM/dd/yyyy.
Time Format *	Enter the format of time in input fields. For example, HH/mm/ss.
Timestamp Format *	Enter the format of timestamps in input fields. For example, the epoch timestamp for 10/11/2021 12:04:41 GMT (MM/dd/yyyy HH:mm:ss) is 1633953881 and the timestamp in milliseconds is 1633953881000.
Expect Records as Array	Determines whether to expect a single record or an array of records. Select this property to expect arrays in each record. Applies only to XML incoming messages. By default, this property is deselected.
* If the format is not specified, it is considered in milliseconds since the epoch (Midnight, January 1, 1970, GMT).	

Java transformation properties

The following table describes the properties you can configure for a Java transformation:

Property	Description
Transformation Type	Select Java.
Transformation Name	Name of the Java transformation.
Classpath	The JAR file used to run the Java code. You can use a separator to include multiple JAR files. On UNIX, use a colon to separate multiple classpath entries. On Windows, use a semicolon to separate multiple classpath entries. For example, `/home/user/commons-text-1.9.jar; /home/user/json-simple-1.1.1.jar`
Import code	Import third-party, built-in, and custom Java packages. You can import multiple packages. Use a semicolon to separate multiple packages. You can use the following syntax to import packages: import <package name> For example, import java.io.*;

Main code	A Java code that provides the transformation logic.
	For example,
	`JSONParser parser = new JSONParser();`
	`try {`
	`JSONObject object = (JSONObject) parser.parse(inputData);`
	`object.put("age", 23);`
	`outputData=object.toJSONString();`
	`} catch (ParseException e) {`
	`throw new RuntimeException();`
	`}`

Jolt transformation properties

The following table describes the properties you have to configure for a Jolt transformation:

Property	Description
Transformation Type	Select Jolt.
Transformation Name	Name of the Jolt transformation.

Jolt Specification	Enter a JSON structure to add a chain of multiple operations using the following operations:
	- shift. Reads values of the input JSON and adds them to specified locations in the output JSON.
	For example,
	`[ { "operation": "shift", "spec": { "breadbox": "counterTop" } } ]`
	- default. Adds values or arrays of values to the output JSON.
	For example,
	`[ { "operation": "default", "spec": { "counterTop": { "loaf1": { "slices": [ "slice1", "slice2", "slice3", "slice4 ] } } } } ] }`
	- cardinality. Transforms elements in the input JSON to single values or to arrays (lists) in the output JSON.
	For example,
	`[ { "operation": "cardinality", "spec": { "counterTop": { "loaf1": { "slices": "ONE" } } } } ]`
	- remove. Deletes elements if found in the input JSON.
	For example,
	`[ { "operation": "remove", "spec": { "counterTop": { "loaf2": "", "jar1": "" } } } ]`
	- modify. Writes calculated values to elements in the output JSON.
	For example,
	`[ { "operation": "modify-overwrite-beta", "spec": { "counterTop": { "jar2": { "contents": "=toUpper" } } } } ]`
	- sort. Sorts all arrays and maps from the input JSON into the output JSON.
	For example,
	`[ { "operation": "sort" } ]`
	The following is an example of a Jolt specification involving multiple operations:
	`Input: {"name":"test"}`
	`[{ "operation": "shift", "spec": { "name": "testname" } }, { "operation": "default", "spec": { "city": ["Anantapur", "Bangalore", "Hyderabad"] } }, { "operation": "cardinality", "spec": { "city": "ONE" } }, {`

Property	Description

	<pre>"operation": "remove",
	"spec": {
	"age": ""
	}
	
	}, {
	"operation": "modify-overwrite-beta",
	"spec": {
	"city": "=toUpper"
	}
	}, {
	"operation": "sort"}]</pre>
	Note: If the input records doesn't match the Jolt specification, the transformation writes null records to the target.

Python transformation properties

The following table describes the properties you can configure for a Python transformation:

Property	Description
Transformation Type	Select Python.
Transformation Name	Name of the Python transformation.
Script Input Type	Python script input type. You can either enter the Python script in **Script Body** or provide the path to the Python script available in the **Script Path**.
Python Path	Directory to the Python path libraries.

Splitter transformation properties

The following table describes the properties you can configure for a Splitter transformation for binary message format:

Property	Description
Transformation Type	Select Splitter.
Transformation Name	Name of the Splitter transformation.
Split Type	Split condition to evaluate the incoming data. Use one of the following split types: - Line Split. - Content Split.
Line Split Count	The maximum number of lines that each output split file contains, excluding header lines.
Byte Sequence	Specified sequence of bytes on which to split the content.

The following table describes the properties you can configure for a Splitter transformation for JSON message format:

Property	Description
Split Expression	Split condition to evaluate the incoming data. Use one of the following split types: - Array Split. - JSONPath Expression.
JSONPath Expression	A JSONPath expression that specifies the array element to split into JSON or scalar fragments. The default JSONpath Expression is $.

The following table describes the properties you can configure for a Splitter transformation for XML message format:

Property	Description
Split Depth	The XML nesting depth to start splitting the XML fragments. The default split depth is 1.

Configuring runtime options

You can configure additional runtime options for the streaming ingestion task. Runtime options include settings to manage reject events and to notify users about errors.

1. On the **Runtime Options** page, under **Notification Management**, select to either disable the notifications or set the time limit after which you want to receive a notification if an error occurs.

   You can set the time in minutes or hours. Default is 10 minutes.
2. Enter a list of email addresses to which you want to send notifications if a task runs with error.

   Use commas to separate a list of email addresses. Note that email notification options configured for the organization are not used here.
3. Under **Agent Parameters**, perform the following tasks:
   a. Specify a directory to store the rejected events.

      Rejected events are not stored by default.

      Note that filtered events in a regular expression are not moved to the reject directory.
   b. To purge the log files, specify the maximum file size for the log file after which the log file is purged.

      You can set the file size in megabytes or gigabytes. Default is 10 MB.
   c. Choose the severity of an event that you want to log.

      The default log level is `Info`.

      The supported log levels are:

      - Debug. Logs all messages along with additional debug level messages.
      - Error. Logs only the error messages.
      - Info. Logs all errors, warnings, and important informational messages.

- Warn. Logs all errors and warning messages.

4. Under **Advanced Parameters**, perform the following tasks to improve the performance of the streaming ingestion task:

   a. Click the + icon next to **Advanced Parameters**.

      The **Key** and **Value** fields appear.

   b. Specify a valid parameter property and its value.

5. To save the task, click **Save**.

   You can then deploy the streaming ingestion task. For more information about deploying the streaming ingestion task, see "Deploying a streaming ingestion task" on page 47.

Deploying a streaming ingestion task

After you create a streaming ingestion task, you must deploy it on the Secure Agent for the task to run as a job. Before deploying the task, ensure that the Secure Agent configured for the task is running.

▶ To deploy a task, perform one of the following actions:

- After you save a task, click **Deploy**.
- In Data Integration, on the **Explore** page, open the project that contains the task, and then select **Deploy** from the **Actions** menu for the task.

A message about successful deployment of the task appears. Your job is now queued to run.

When you edit or undeploy a job, deploying the job again does not affect any other jobs running on the same Secure Agent or Secure Agent group.

When you edit or redeploy a streaming ingestion job that contains a target with the rollover strategy, all the files in the staging directory are moved to the target directory even if the defined rollover conditions are not met.

Undeploying a streaming ingestion job

You can undeploy a streaming ingestion job from the Secure Agent.

In Data Integration, on the **Explore** page, open the project that contains the job, and then select **Undeploy** from the **Actions** menu for the job.

Navigate to the row for the job that you want to undeploy in any of the following monitoring interfaces:

- **My Jobs** page that's accessed from the navigation bar of the Home page
- **All Jobs** page in Monitor
- **All Jobs** tab on the **Mass Ingestion** page in Operational Insights

When you undeploy a streaming ingestion job, the Secure Agent does not save the previous state histories of the job. So, when you deploy the streaming ingestion task again, the task runs as completely new job.

Stopping and resuming streaming ingestion jobs

You can stop and resume streaming ingestion jobs. Stop or resume a job on the **My Jobs** page in Data Integration, the **All Jobs** and **Running Jobs** pages in Monitor, or the Mass Ingestion page in Operational Insights.

Stop a job.

You can stop a streaming ingestion job that is Up and Running, Running with Error, or Running with Warnings.

To stop a job, open the **My Jobs** page in Data Integration, or the **All Jobs** and **Running Jobs** pages in Monitor, and then select **Stop** from the **Actions** menu for the job.

Alternatively, you can stop a streaming ingestion job from the **Actions** menu in a job row on the **Mass Ingestion** page in Operational Insights.

You can also use the Stop icon beside the **Actions** menu to stop the job.

Resume a job.

You can resume a stopped streaming ingestion job.

To resume a job, open the **My Jobs** page in Data Integration, or the **All Jobs** and **Running Jobs** pages in Monitor, and then select **Resume** from the **Actions** menu for the job.

Alternatively, you can resume a streaming ingestion job from the **Actions** menu in a job row on the **Mass Ingestion** page in Operational Insights.

You can also use the Resume icon beside the **Actions** menu to resume the job.

When you stop a streaming ingestion job, the Secure Agent saves the previous state histories of the job. When you resume the stopped streaming ingestion job, the job starts running from the last saved state.

Frequently asked questions for Mass Ingestion Streaming behavior

Review the following frequently asked questions to understand the product behaviour.

Are the checkpoints stored on the Secure Agent, cloud repository, source, or target?

Depending on the source and target, the checkpoints are stored on the Secure Agent, cloud repository, source, or target. The storage of checkpoints varies for different sources and targets.

For example, for an Amazon Kinesis Streams source, streaming ingestion creates a DynamoDB checkpoint table in a data warehouse such as Amazon S3.

Can I set the checkpoint after deploying or undeploying a task?

No, you can't set the checkpoint after deploying or undeploying a task.

In a streaming ingestion task with a Kafka source and a Kafka target, can I rely on the offset stored in Kafka as a restart checkpoint?

Yes, you can treat the offset stored in Kafka as a restart checkpoint.

In a streaming ingestion task with a Kafka source and a Kafka target, does the group.id property help to restart the checkpoint after the tasks are undeployed and deployed?

The **group.id** is a shared property in Kafka that helps in checkpoint recovery. When a Kafka source within a group processes a message successfully, it updates its offset in the topic partition. This offset is stored along with the **group.id**. When a Kafka dataflow transitions from the undeployed to the deployed status, and is up and running, it uses the same **group.id** and the stored offset to resume processing from where the dataflow stopped.

Can you guarantee At least once delivery allowing potential duplicates in the target after a task failure?

Yes, **At least once delivery** is guaranteed.

Is there any difference in delivery depending on the source and target connectors?

Each connector has a unique implementation, but all are designed to ensure **At least once delivery** guaranteed delivery.

How does recovery occur after a failure?

After the Secure Agent recovers from a failure, the runtime engine restarts, and the reconciliation process takes place.

Does Mass Ingestion Streaming support a Secure Agent group with more than one Secure Agent?

Mass Ingestion Streaming supports multiple Secure Agents under a single agent group. However, a dataflow can't run across multiple Secure Agents at the same time.

Does Mass Ingestion Streaming support shared Secure Agent groups?

Mass Ingestion Streaming doesn't support shared Secure Agent groups.

CHAPTER 2

Mass Ingestion Streaming REST API

Use streaming ingestion resources to deploy, undeploy, start, stop, copy, and update streaming ingestion tasks and to monitor streaming ingestion jobs.

When you use the streaming ingestion resource, use the following request header format:

```
<METHOD><base URL>
Content-Type: application/json
Accept: application/json
IDS-SESSION-ID: <SessionId>
```

Dataflows resource

Use the Dataflows resource to deploy, undeploy, start, and stop streaming ingestion tasks.

Use the following base URL:

```
<server URI>/sisvc/api/v1/Dataflows('<dataflow ID>')/OData.SI.<API name>
```

Note: If you use a tool such as Postman that automatically includes the HTTP version, do not enter the HTTP version in the URL. If the HTTP version appears twice in the URL, the request fails.

Deploying a streaming ingestion task

Use a POST request to deploy a streaming ingestion task.

POST request

To deploy a streaming ingestion task, use the following URL:

```
<server URI>/sisvc/api/v1/Dataflows('<dataflow ID>')/OData.SI.Deploy
```

A request body is not required because the URL passes the dataflow ID.

POST request example

To deploy a streaming ingestion task, you might send a request similar to the following example:

```
POST <serverUrl>/sisvc/api/v1/Dataflows('50077311-d4a4-437c-9218-c3596d1f182f')/
OData.SI.Deploy
Content-Type: application/json
Accept:application/json
IDS-SESSION-ID:2l0oeVx22Rujiej7yTokmT
```

POST response example

If the request is successful, you might receive a response similar to the following example:

```
{
    "@odata.context": "$metadata#OData.SI.DeploymentResult",
    "successful": true,
    "code": null,
    "errorMessage": null
}
```

Undeploying a streaming ingestion task

Use a POST request to undeploy a streaming ingestion task.

POST request

To undeploy a streaming ingestion task, use the following URL:

```
<server URI>/sisvc/api/v1/Dataflows('<dataflow ID>')/OData.SI.Undeploy
```

A request body is not required because the URL passes the dataflow ID.

POST request example

To undeploy a streaming ingestion task, you might send a request similar to the following example:

```
POST <serverUrl>/sisvc/api/v1/Dataflows('50077311-d4a4-437c-9218-c3596d1f182f')/
OData.SI.Undeploy
Content-Type: application/json
Accept:application/json
IDS-SESSION-ID:2l0oeVx22Rujiej7yTokmT
```

POST response example

If the request is successful, you might receive a response similar to the following example:

```
{
    "@odata.context": "$metadata#OData.SI.DeploymentResult",
    "successful": true,
    "code": null,
    "errorMessage": null
}
```

Starting a streaming ingestion task

Use a POST request to start a streaming ingestion task.

POST request

To start a streaming ingestion task, use the following URL:

```
<server URI>/sisvc/api/v1/Dataflows('<dataflow ID>')/OData.SI.Start
```

A request body is not required because the URL passes the dataflow ID.

POST request example

To start a streaming ingestion task, you might send a request similar to the following example:

```
POST <serverUrl>/sisvc/api/v1/Dataflows('50077311-d4a4-437c-9218-c3596d1f182f')/
OData.SI.Start
Content-Type: application/json
Accept:application/json
IDS-SESSION-ID:2l0oeVx22Rujiej7yTokmT
```

POST response example

If the request is successful, you might receive a response similar to the following example:

```
{
    "@odata.context": "$metadata#OData.SI.DeploymentResult",
    "successful": true,
    "code": null,
    "errorMessage": null
}
```

Stopping a streaming ingestion task

Use a POST request to stop a streaming ingestion task.

POST request

To stop a streaming ingestion task, use the following URL:

```
<server URI>/sisvc/api/v1/Dataflows('<dataflowID>')/OData.SI.Stop
```

A request body is not required because the URL passes the dataflow ID.

POST request example

To stop a streaming ingestion task, you might send a request similar to the following example:

```
POST <serverUrl>/sisvc/api/v1/Dataflows('d7572789-dc4c-4c56-bbeb-3772736d61aa')/
OData.SI.Stop
Content-Type: application/json
Accept:application/json
IDS-SESSION-ID:2l0oeVx22Rujiej7yTokmT
```

POST response example

If the request is successful, you might receive a response similar to the following example:

```
{
    "@odata.context": "$metadata#OData.SI.DeploymentResult",
    "successful": true,
    "code": null,
    "errorMessage": null
}
```

CopyEntities resource

Use the CopyEntities resource to copy streaming ingestion tasks.

POST request

To copy streaming ingestion tasks, use the following URL:

```
<server URI>/sisvc/restapi/v1/CopyEntities
```

You can include the following fields in the request:

Field	Type	Required	Description
targetLocationID	String	Yes	ID of the target location to copy the objects to.
sourceEntities	Array	Yes	Configuration of the source and target connections.
sourceId	String	Yes	ID of the source object.
targetName	String	Yes	Name of the target object.
targetDescription	String	-	Description of the target object.

POST request example

To copy two streaming ingestion tasks, you might send a request similar to the following example:

```
POST <serverUrl>/sisvc/restapi/v1/CopyEntities
Content-Type: application/json
Accept:application/json
IDS-SESSION-ID:2l0oeVx22Rujiej7yTokmT
{
  "targetLocationID": "2RGmVdwN15PbfnQP5PSoSB",
  "sourceEntities": [
    {
      "sourceId": "5Ff6jeaSh2UfAqiV01ldKD",
      "targetName": "Test_Copy_A",
      "targetDescription": "Description_1"
    },
    {
      "sourceId": "fZnCSqcWTOQkJOr8VCWZQE",
      "targetName": "Test_Copy_B",
      "targetDescription": "Description_2"
    }
  ]
}
```

If the request is unsuccessful, the response includes a reason for the failure.

POST response

When you use a POST request to copy streaming ingestion tasks, it returns a success response if successful or an error object if an error occurs.

POST response example

If the request is successful, you might receive a response similar to the following example:

```
{
  "Status Message": "Operation succeeded on 2 artifacts.",
  "Success": {
    "Test_Copy_A": "ideNJw6l54gizxofF53HQH",
    "Test_Copy_B": "cOQ3gcWKSYikzVqqg6IOok"
  }
}
```

UpdateEntity resource

Use the UpdateEntity resource to update a streaming ingestion task. You can update streaming ingestion tasks that use the following connectors: Amazon Kinesis, Amazon S3 V2, Microsoft Azure Event Hub, Microsoft Azure Data Lake Storage Gen2, Flat file, JDBC V2, JMS, Kafka, or MQTT.

POST request

Use a POST request to update a streaming ingestion task.

To update a streaming ingestion task, use the following URL:

```
<server URI>/sisvc/restapi/v1/UpdateEntity/Documents('<document ID>')
```

You can include the following fields in the request:

Field	Type	Required	Description
name	String	Yes	Name of the task.
description	String	-	Description of the task.
runtimeId	String	Yes	ID of the runtime environment.
currentVersion	String	Yes	The latest dataflow object version.
nodes	Array	Yes	Details of the task source and target connections.

Fields of the nodes array

The fields in the array provide the name, type, and connection ID of the connection. It includes the configuration of the source and target connections in key-value pairs which you can edit. You can include the following fields in the nodes array:

Field	Type	Required	Description
name	String	Yes	Name of the connection.
type	String	Yes	The connection type, source or target.
connectionId	String	Yes	ID of the connection.
transformationType	String	-	Not applicable.
config	Array	Yes	Configuration of the source and target connections.

Connection configuration for tasks with MQTT as a source

When the source connection of the task source is MQTT, you can include the following fields and key-value pairs in the config array of the source connection:

Key	Type	Required	Description
ClientID	String	-	Unique identifier of the connection between the MQTT source and the MQTT broker. The client ID is the file-based persistence store that the MQTT source uses to store messages while they are processed. You can enter a string of up to 255 characters.
MaxQueueSize	Integer	-	The maximum number of messages that the processor can store in memory. You can enter a value between 1 and 2147483647.
Topic	String	Yes	Name of the MQTT topic.

POST request example

To update a streaming ingestion task with an MQTT source and a flat file target, you might send a request similar to the following example:

```
{
  "name": "mqtt to flatfile",
  "description": "mqtt to flatfile",
  "runtimeId": "0100002500000000003",
  "locationId": "5sJ0JDyJyWLlrosS5qJjsQ",
  "currentVersion": "2",
  "messageFormat": "binary",
  "nodes": [
    {
      "name": "mqtt to flatfile_source",
      "type": "source",
      "connectionId": "012MGS0B00000000001O",
      "transformationType": "",
      "config": [
        {
          "key": "ClientID",
          "value": "test"
        },
        {
          "key": "MaxQueueSize",
          "value": 1024
        },
        {
          "key": "Topic",
          "value": "test"
        }
      ]
    },
    {
      "name": "mqtt to flatfile_target",
      "type": "target",
      "connectionId": "012MGS0B00000000002N",
      "transformationType": "",
      "config": [
        {
          "key": "interimDirectory",
          "value": "/home/agent/test"
        },
        {
          "key": "rolloverSize",
          "value": 1024
        },
        {
```

```
            "key": "rolloverEvents",
            "value": 100
          },
          {
            "key": "rolloverTime",
            "value": 300000
          },
          {
            "key": "File Name",
            "value": "test"
          }
        ]
      }
    ],
    "edges": [
      {
        "from": "mqtt to flatfile_source",
        "to": "mqtt to flatfile_target"
      }
    ]
  }
```

Connection configuration for tasks with JMS as a source

When the source connection of the task source is JMS, you can include the following fields and key-value pairs in the config array of the source connection:

Key	Type	Required	Description
destinationType	String	Yes	Type of destination that the source service sends JMS messages to. Enter one of the following values: - `QUEUE`. The JMS provider delivers messages to a single consumer who is registered for the queue. - `TOPIC`. The JMS provider delivers messages to all active consumers that subscribe to the topic.
clientId	String	Yes	Unique ID of the JMS connection. You can enter a string of up to 255 characters.
sharedSubscription	String	Yes	Enables multiple consumers to access a single subscription. Applies to the TOPIC destination type. Enter one of the following values: - `True` - `False`
durableSubscription	String	Yes	When set to True, the JMS source service enables inactive subscribers to retain messages and then deliver them when the subscriber reconnects. Applies to the TOPIC destination type. Enter one of the following values: - `True` - `False`
subscriptionName	String	Yes	Name of the subscription. Applies to the TOPIC destination type, when the topic subscription type is shared, durable, or both.
JMS Destination	String	Yes	Name of the queue or topic that the JMS provider delivers messages to.

POST request example

To update a streaming ingestion task with an JMS source and a flat file target, you might send a request similar to the following example:

```
{
  "name": "crud",
```

```
  "description": "JMS to FileToFile",
  "runtimeId": "01000025000000000003",
  "locationId": "5sJ0JDyJyWLlrosS5qJjsQ",
  "currentVersion": "2",
  "messageFormat": "binary",
  "nodes": [
    {
      "name": "crud_source",
      "type": "source",
      "connectionId": "012MGS0B000000000003",
      "transformationType": "",
      "config": [
        {
          "key": "destinationType",
          "value": "QUEUE"
        },
        {
          "key": "clientId",
          "value": ""
        },
        {
          "key": "JMS Destination",
          "value": "test"
        }
      ]
    },
    {
      "name": "crud_target",
      "type": "target",
      "connectionId": "012MGS0B00000000000H",
      "transformationType": "",
      "config": [
        {
          "key": "interimDirectory",
          "value": "/home/agent/test"
        },
        {
          "key": "rolloverSize",
          "value": 1024
        },
        {
          "key": "rolloverEvents",
          "value": 100
        },
        {
          "key": "rolloverTime",
          "value": 300000
        },
        {
          "key": "File Name",
          "value": "test"
        }
      ]
    }
  ],
  "edges": [
    {
      "from": "crud_source",
      "to": "crud_target"
    }
  ]
}
```

Connection configuration for tasks with Microsoft Azure Data Lake Storage Gen2 (ADLS Gen2) as a target

When the target connection of the task target is ADLS Gen2, you can include the following fields and key-value pairs in the config array of the target connection:

Key	Type	Required	Description
writeStrategy	String	Yes	The action to take when a file by the same name exists in the ADLS Gen2 storage. Enter one of the following values: - Append. Add data to the existing file. - Overwrite. Replaces the existing file with the new file. - Fail. Fail the request. - Rollover. Close the current file and create a new file based on the configured rollover value.
rolloverSize *	Integer	-	Target file size, in KB, at which to trigger rollover. Applies to a Rollover write strategy. You can enter a value between 1 and 2147483647.
rolloverEvents *	Integer	-	Number of events or messages to accumulate before a rollover. Applies to a Rollover write strategy. You can enter a value between 1 and 2147483647.
rolloverTime *	Integer	-	Length of time, in milliseconds, after which to trigger a rollover. Applies to a Rollover write strategy. You can enter a value between 1 and 2147483647.
filesystemNameOverride	String	-	Overrides the default file system name provided in the connection. This file system name is used write to a file at run time. You can enter a string of up to 1,280 characters.
directoryOverride	String	-	Overrides the default directory path. The ADLS Gen2 directory path to write data to. If left blank, the default directory path is used. You can enter a string of up to 1,280 characters.
compressionFormat	String	-	Compression format to use before the streaming ingestion task writes data to the target file. Enter one of the following values: - None - GZIP - BZIP2 - DEFAULT1 Enter this value to use the Zlib format. - DEFAULT2 Enter this value to use the Deflate format.
File Name/Expression	String	Yes	ADLS Gen2 file name or a regular expression. You can enter a string of up to 249 characters.
* Enter a value for at least one of the fields.			

POST request example

To update a streaming ingestion task with a flat file source and an ADLS Gen2 target, you might send a request similar to the following example:

```
{
  "name": "flatfile to adls",
  "description": "flatfile to adls",
  "runtimeId": "01000025000000000003",
  "locationId": "5sJ0JDyJyWLlrosS5qJjsQ",
  "currentVersion": "2",
  "messageFormat": "binary",
  "nodes": [
    {
      "name": "flatfile to adls_source",
      "type": "source",
      "connectionId": "012MGS0B00000000002N",
      "transformationType": "",
      "config": [
        {
          "key": "File",
          "value": "logfile"
        },
        {
          "key": "initialPosition",
          "value": "Current Time"
        },
        {
          "key": "rolloverPattern",
          "value": "test"
        },
        {
          "key": "tailingMode",
          "value": "Single file"
        }
      ]
    },
    {
      "name": "flatfile to adls_target",
      "type": "target",
      "connectionId": "012MGS0B00000000003D",
      "transformationType": "",
      "config": [
        {
          "key": "writeStrategy",
          "value": "Rollover"
        },
        {
          "key": "filesystemNameOverride",
          "value": "test"
        },
        {
          "key": "File Name/Expression",
          "value": "test"
        },
        {
          "key": "compressionFormat",
          "value": "NONE"
        },
        {
          "key": "directoryOverride",
          "value": "/test"
        },
        {
          "key": "interimDirectory",
          "value": "/home/agent/test"
        },
        {
          "key": "rolloverSize",
          "value": 1024
        },
```

```
          {
            "key": "rolloverEvents",
            "value": 100
          },
          {
            "key": "rolloverTime",
            "value": 300000
          }
        ]
      }
    ]
  }
```

Connection configuration for tasks with Amazon S3 as a target

When the target connection of the task target is Amazon S3, you can include the following fields and key-value pairs in the config array of the target connection:

Key	Type	Required	Description
partitionTime	String	-	The time interval according to which the streaming ingestion task creates partitions in the Amazon S3 bucket. Enter one of the following values: - None - 5min - 10min - 15min - 20min - 30min - 1hr - 1day
minUploadPartSize	Integer	-	Minimum part size when uploading a large file as a set of multiple independent parts, in megabytes. Use this property to tune the file load to Amazon S3. You can enter a value between 50 and 5120.
multipartUploadThreshold	Integer	-	Multipart threshold when uploading objects in multiple parts in parallel. You can enter a value between 50 and 5120.
Object Name/Expression	String	Yes	Amazon S3 target file name or a regular expression for the Amazon S3 file name pattern.

POST request example

To update a streaming ingestion task with a flat file source and an Amazon S3 as target, you might send a request similar to the following example:

```
{
  "name": "flatfile to amazon S3",
  "description": "flatfile to amazon S3",
  "runtimeId": "0100002500000000003",
  "locationId": "5sJ0JDyJyWLlrosS5qJjsQ",
  "currentVersion": "2",
  "messageFormat": "binary",
  "nodes": [
    {
      "name": "flatfile to amazon S3_source",
      "type": "source",
      "connectionId": "012MGS0B00000000002N",
```

```
      "transformationType": "",
      "config": [
        {
          "key": "File",
          "value": "logfile"
        },
        {
          "key": "initialPosition",
          "value": "Current Time"
        },
        {
          "key": "rolloverPattern",
          "value": "test"
        },
        {
          "key": "tailingMode",
          "value": "Single file"
        }
      ]
    },
    {
      "name": "flatfile to amazon S3_target",
      "type": "target",
      "connectionId": "012MGS0B0000000000I7",
      "transformationType": "",
      "config": [
        {
          "key": "partitionTime",
          "value": "None"
        },
        {
          "key": "minUploadPartSize",
          "value": 5120
        },
        {
          "key": "multipartUploadThreshold",
          "value": 5120
        },
        {
          "key": "Object Name/Expression",
          "value": "test"
        }
      ]
    }
  ],
  "edges": [
    {
      "from": "flatfile to amazon S3_source",
      "to": "flatfile to amazon S3_target"
    }
  ]
}
```

Connection configuration for tasks with Azure Event Hubs as a target

When the target connection of the task target is Azure Event Hubs, you can include the following fields and key-value pairs in the config array of the target connection:

Key	Type	Required	Description
sasPolicyName	String	-	The name of the Event Hub Namespace Shared Access Policy. You can enter a string of up to 255 characters.
sasPolicyPrimaryKey	String	-	The primary key of the Event Hub Namespace Shared Access Policy. You can enter a string of up to 255 characters.
Event Hub	String	Yes	The name of the Azure Event Hubs. You can enter a string up to 255 characters. The name can contain lower case characters, upper case characters, numbers, and the special characters - and _.

POST request example

To update a streaming ingestion task with a flat file source and an Azure Event Hubs target, you might send a request similar to the following example:

```
{
  "name": "flatfile to azure event hub",
  "description": "flatfile to azure event hub",
  "runtimeId": "01000025000000000003",
  "locationId": "5sJ0JDyJyWLlrosS5qJjsQ",
  "currentVersion": "2",
  "messageFormat": "binary",
  "nodes": [
    {
      "name": "flatfile to azure event hub_source",
      "type": "source",
      "connectionId": "012MGS0B00000000002N",
      "transformationType": "",
      "config": [
        {
          "key": "File",
          "value": "logfile"
        },
        {
          "key": "initialPosition",
          "value": "Current Time"
        },
        {
          "key": "rolloverPattern",
          "value": "test"
        },
        {
          "key": "tailingMode",
          "value": "Single file"
        }
      ]
    },
    {
      "name": "flatfile to azure event hub_target",
      "type": "target",
      "connectionId": "012MGS0B00000000001S",
      "transformationType": "",
      "config": [
        {
          "key": "sasPolicyName",
          "value": "test"
        },
```

```
        {
          "key": "sasPolicyPrimaryKey",
          "value": "test"
        },
        {
          "key": "Event Hub",
          "value": "test"
        }
      ]
    }
  ],
  "edges": [
    {
      "from": "flatfile to azure event hub_source",
      "to": "flatfile to azure event hub_target"
    }
  ]
}
```

Connection configuration for tasks with JDBC V2 as a target

When the target connection of the task target is JDBC V2, you can include the following fields and key-value pairs in the config array of the target connection:

Key	Type	Required	Description
Table Name	String	Yes	Name of the table to insert data to in JSON format. Enter a string of up to 988 characters.

POST request example

To update a streaming ingestion task with a flat file source and a JDBC V2 target, you might send a request similar to the following example:

```
{
  "name": "FileFile to jdbc",
  "description": "FileToFile to jdbc_target",
  "runtimeId": "0100002500000000003",
  "locationId": "5sJ0JDyJyWLlrosS5qJjsQ",
  "currentVersion": "2",
  "messageFormat": "binary",
  "nodes": [
    {
      "name": "flatfile to jdbc_source",
      "type": "source",
      "connectionId": "012MGS0B00000000002N",
      "transformationType": "",
      "config": [
        {
          "key": "initialPosition",
          "value": "Current Time"
        },
        {
          "key": "tailingMode",
          "value": "Single file"
        },
        {
          "key": "rolloverPattern",
          "value": "test"
        },
        {
          "key": "File",
          "value": "logfile"
        }
      ]
```

```
    },
    {
      "name": "flatfile to jdbc_target",
      "type": "target",
      "connectionId": "012MGS0B0000000000KF",
      "transformationType": "",
      "config": [
        {
          "key": "Table Name",
          "value": "table"
        }
      ]
    }
  ],
  "edges": [
    {
      "from": "flatfile to jdbc_source",
      "to": "flatfile to jdbc_target"
    }
  ]
}
```

Connection configuration for tasks with Amazon Kinesis Streams as a source and as a target

When the source and target connection of the task is Amazon Kinesis Streams, you can include the following fields and key-value pairs in the config array of the source and target connection:

Key	Type	Required	Description
appendGUID	Boolean		Specifies whether or not to add a GUID as a suffix to the Amazon DynamoDB table name. Enter one of the following values: - true - false
dynamoDB	String		Amazon DynamoDB table name where to store the checkpoint details of the Kinesis source data. You can enter a string of up to 128 characters.
Stream	String	Yes	Name of the Kinesis Stream to read data from. Enter a string of up to 128 characters. Appears in the source node.
Stream Name/ Expression	String	Yes	Kinesis Stream name or a regular expression to write data to. Enter a string of up to 128 characters. Appears in the target node.

POST request example

To update a streaming ingestion task with an Amazon Kinesis Streams source and target, you might send a request similar to the following example:

```
{
  "name": "kinesis to kinesis",
  "description": "kinesis to kinesis",
  "runtimeId": "0100002500000000003",
  "locationId": "5sJ0JDyJyWLlrosS5qJjsQ",
  "currentVersion": "2",
  "messageFormat": "binary",
  "nodes": [
```

```
    {
      "name": "kinesis to kinesis_source",
      "type": "source",
      "connectionId": "012MGS0B00000000000F",
      "transformationType": "",
      "config": [
        {
          "key": "appendGUID",
          "value": true
        },
        {
          "key": "dynamoDB",
          "value": "table"
        },
        {
          "key": "Stream",
          "value": "test"
        }
      ]
    },
    {
      "name": "kinesis to kinesis_target",
      "type": "target",
      "connectionId": "012MGS0B00000000000F",
      "transformationType": "",
      "config": [
        {
          "key": "Stream Name/Expression",
          "value": "trgt"
        }
      ]
    }
  ],
  "edges": [
    {
      "from": "kinesis to kinesis_source",
      "to": "kinesis to kinesis_target"
    }
  ]
}
```

Connection configuration for tasks with flat file as a source and as a target

When the source and target connection of the task is flat file, you can include the following fields and key-value pairs in the config array of the source and target connection:

Key	Type	Required	Description
File	String	Yes	Absolute path and name of the source file. Enter the base directory for multiple files mode.
initialPosition	String	Yes	Starting position to read data from the file to tail. Enter one of the following values: - Beginning of File. Read from the beginning of the file. Don't ingest any data that has already been rolled over. - Current Time. Read from the most recently updated part of the file. Don't ingest data that was rolled over or data in the file that was written.

Key	Type	Required	Description
rolloverPattern	String	-	File name pattern for the file that rolls over. If the file to tail rolls over, the Secure Agent uses the file name pattern to identify files that have rolled over. If the Secure Agent stops during a file rollover, when it restarts, it picks up the file where it was left off. You can use asterisk (*) and question mark (?) as wildcard characters to indicate that the files are rolled over in the same directory. For example,`${filename}.log.*`. Here, asterisk (*) represents the successive version numbers that would be appended to the file name.
tailingMode	String	Yes	Tail a file or multiple files based on the logging pattern. Enter one of the following values: - Single file. Tail one file. - Multiple files. Tail all the files indicated in the base directory. You can enter a regular expression to indicate the files to tail.
File Name	String	Yes	The name of the target file.
interimDirectory	String	Yes	Path to the staging directory on the Secure Agent.
rolloverSize	Integer	Yes	The file size, in KB, at which the task moves the file from the staging directory to the target. You can enter a value between 1 and 2147483647.
rolloverEvents	Integer	Yes	Number of events or messages to accumulate before a file rollover. You can enter a value between 1 and 2147483647.
rolloverTime	Integer	-	Length of time, in milliseconds, after which the target file rolls over. You can enter a value between 1 and 2147483647.
edges	Array	-	Sequence of dataflow execution.

POST request example

To update a streaming ingestion task with a flat file source and target, you might send a request similar to the following example:

```
{
  "name": "FileToFile",
  "description": "FileToFile_V2",
  "runtimeId": "0100002500000000003",
  "locationId": "5sJ0JDyJyWLlrosS5qJjsQ",
  "currentVersion": "2",
  "messageFormat": "binary",
  "nodes": [
    {
      "name": "FileToFile_source",
      "type": "source",
      "connectionId": "0100000B000000000002",
      "transformationType": "",
      "config": [
        {
          "key": "File",
          "value": "siagent.log"
        },
        {
          "key": "initialPosition",
          "value": "Current Time"
        },
        {
```

```
            "key": "rolloverPattern",
            "value": ""
          },
          {
            "key": "tailingMode",
            "value": "Single file"
          }
        ]
      },
      {
        "name": "FileToFile_target",
        "type": "target",
        "connectionId": "0100000B000000000002",
        "transformationType": "",
        "config": [
          {
            "key": "File Name",
            "value": "testing.log"
          },
          {
            "key": "interimDirectory",
            "value": "/home/agent/infa/test_file_target"
          },
          {
            "key": "rolloverSize",
            "value": 100
          },
          {
            "key": "rolloverEvents",
            "value": 100
          },
          {
            "key": "rolloverTime",
            "value": 100
          }
        ]
      }
    ],
    "edges": [
      {
        "from": "FileToFile_source",
        "to": "FileToFile_target"
      }
    ],
    "runtimeOptions": {
      "maxLogSize": {
        "value": 10,
        "unit": "MB"
      },
      "logLevel": "INFO"
    }
  }
```

Connection configuration for tasks with Kafka as a source and as a target

When the source and target connection of the task is Kafka, you can include the following fields and key-value pairs in the config array of the source and target connection:

Key	Type	Required	Description
Topic	String	Yes	Kafka source topic name or a Java supported regular expression for the Kafka source topic name pattern to read the events from. Enter a string of up to 249 characters.
consumerProperties	String	-	Provide a comma-separated list of optional consumer configuration properties. Specify the values as key-value pairs. For example, `key1=value1, key2=value2`. You can enter a string of up to 4000 characters.
producerProperties	String	-	The configuration properties for the producer. Provide a comma-separated list and specify the values as key-value pairs. You can enter a string of up to 4000 characters.
mdFetchTimeout	Integer	-	The time after which the metadata is not fetched. Enter a value between 1 and 2147483647.
batchSize	Integer	-	The batch size of the events after which a streaming ingestion task writes data to the target. Enter a value between 1 and 2147483647.
Topic Name/ Expression	String	Yes	Kafka topic name or a Java supported regular expression for the Kafka topic name pattern. You can enter a string of up to 249 characters.

POST request example

To update a streaming ingestion task with a Kafka source and target, you might send a request similar to the following example:

```
{
  "name": "kafka to kafka",
  "description": "kafka to kafka",
  "runtimeId": "01000025000000000003",
  "locationId": "5sJ0JDyJyWLlrosS5qJjsQ",
  "currentVersion": "2",
  "messageFormat": "binary",
  "nodes": [
    {
      "name": "kafka to kafka_source",
      "type": "source",
      "connectionId": "012MGS0B000000000002",
      "transformationType": "",
      "config": [
        {
          "key": "consumerProperties",
          "value": "key=value"
        },
        {
          "key": "Topic",
          "value": "test"
        }
      ]
    },
```

```
    {
      "name": "kafka to kafka_target",
      "type": "target",
      "connectionId": "012MGS0B000000000002",
      "transformationType": "",
      "config": [
        {
          "key": "producerProperties",
          "value": "key=value"
        },
        {
          "key": "mdFetchTimeout",
          "value": 5000
        },
        {
          "key": "batchSize",
          "value": 1048576
        },
        {
          "key": "Topic Name/Expression",
          "value": "test"
        }
      ]
    }
  ],
  "edges": [
    {
      "from": "kafka to kafka_source",
      "to": "kafka to kafka_target"
    }
  ]
}
```

POST response

When the REST API successfully performs an action, it returns a 200 or 201 success response. When the REST API encounters an error, it returns an appropriate error code.

If the request is successful, the response returns the following fields:

Field	Type	Description
name	String	Name of the task.
description	String	Description of the task, if available.
runtimeId	String	ID of the runtime environment.
currentVersion	String	The latest dataflow object version.
nodes	Array	Details of the task source and target connections.

Fields of the nodes array

The response includes the following fields in the nodes array:

Field	Type	Description
name	String	Name of the connection.
type	String	The connection type.

Field	Type	Description
connectionId	String	ID of the connection.
transformationType	String	The type of transformation.
config	String	Configuration of the source and target connections in key-value pairs. The keys in the array depend on the type of source and target connections.

If the request is unsuccessful, the response includes a reason for the failure.

Configuration information in the config array MQTT as a source

If the request is successful, the response returns the following fields:

Key	Type	Description
ClientID	String	Unique identifier that identifies the connection between the MQTT source and the MQTT broker. The client ID is the file-based persistence store that the MQTT source uses to store messages when they are being processed.
MaxQueueSize	Integer	The maximum number of messages that the processor can store in memory.
Topic	String	Name of the MQTT topic.

If the request is unsuccessful, the response includes a reason for the failure.

POST response example

If the request is successful, you might receive a response similar to the following example in a `Success` node:

```
{
  "Success": {
    "name": "mqtt to flatfile",
    "description": "mqtt to flatfile",
    "runtimeId": "0100002500000000003",
    "locationId": "5sJ0JDyJyWLlrosS5qJjsQ",
    "currentVersion": "2",
    "messageFormat": "binary",
    "nodes": [
      {
        "name": "mqtt to flatfile_source",
        "type": "source",
        "connectionId": "012MGS0B000000000010",
        "transformationType": "",
        "config": [
          {
            "key": "ClientID",
            "value": "test"
          },
          {
            "key": "MaxQueueSize",
            "value": 1024
          },
          {
            "key": "Topic",
            "value": "test"
          }
        ]
      },
      {
```

```
        "name": "mqtt to flatfile_target",
        "type": "target",
        "connectionId": "012MGS0B00000000002N",
        "transformationType": "",
        "config": [
          {
            "key": "interimDirectory",
            "value": "/home/agent/test"
          },
          {
            "key": "rolloverSize",
            "value": 1024
          },
          {
            "key": "rolloverEvents",
            "value": 100
          },
          {
            "key": "rolloverTime",
            "value": 300000
          },
          {
            "key": "File Name",
            "value": "test"
          }
        ]
      }
    ],
    "edges": [
      {
        "from": "mqtt to flatfile_source",
        "to": "mqtt to flatfile_target"
      }
    ]
  }
}
```

Configuration information in the config array for JMS as a source

The response returns only the fields that you entered in the request.

If the request is successful, the response returns the following fields:

Key	Type	Description
destinationType	String	Type of destination that the source service sends JMS messages to.
clientId	String	Unique ID of the JMS connection.
sharedSubscription	String	Enables multiple consumers to access a single subscription. Applies to the TOPIC destination type.
durableSubscription	String	The JMS source service enables inactive subscribers to retain messages and then deliver them when the subscriber reconnects. Applies to the TOPIC destination type.
subscriptionName	String	Name of the subscription. Applies to the TOPIC destination type, when the topic subscription type is shared, durable, or both.
JMS Destination	String	Name of the queue or topic that the JMS provider delivers messages to.

If the request is unsuccessful, the response includes a reason for the failure.

POST response example

If the request is successful, you might receive a response similar to the following example in a `Success` node:


```
{
  "Success": {
    "name": "crud",
    "description": "JMS to FileToFile",
    "runtimeId": "01000025000000000003",
    "locationId": "5sJ0JDyJyWLlrosS5qJjsQ",
    "currentVersion": "2",
    "messageFormat": "binary",
    "nodes": [
      {
        "name": "crud_source",
        "type": "source",
        "connectionId": "012MGS0B000000000003",
        "transformationType": "",
        "config": [
          {
            "key": "destinationType",
            "value": "QUEUE"
          },
          {
            "key": "clientId",
            "value": ""
          },
          {
            "key": "JMS Destination",
            "value": "test"
          }
        ]
      },
      {
        "name": "crud_target",
        "type": "target",
        "connectionId": "012MGS0B00000000000H",
        "transformationType": "",
        "config": [
          {
            "key": "interimDirectory",
            "value": "/home/agent/test"
          },
          {
            "key": "rolloverSize",
            "value": 1024
          },
          {
            "key": "rolloverEvents",
            "value": 100
          },
          {
            "key": "rolloverTime",
            "value": 300000
          },
          {
            "key": "File Name",
            "value": "test"
          }
        ]
      }
    ],
    "edges": [
      {
        "from": "crud_source",
        "to": "crud_target"
      }
    ]
  }
}
```

Configuration information in the config array for ADLS Gen2 as a target

The response returns only the fields that you entered in the request.

If the request is successful, the response returns the following fields:

Key	Type	Description
writeStrategy	String	The action to take when a file exists in the ADLS Gen2 storage.
rolloverSize *	Integer	Target file size, in KB, at which to trigger rollover. Applies to a Rollover write strategy.
rolloverEvents *	Integer	Number of events or messages to accumulate before a rollover. Applies to a Rollover write strategy.
rolloverTime *	Integer	Length of time, in milliseconds, after which to trigger a rollover. Applies to a Rollover write strategy.
filesystemNameOverride	String	Overrides the default file system name provided in the connection. This file system name is used write to a file at run time.
directoryOverride	String	Overrides the default directory path. The ADLS Gen2 directory path to write data to. If left blank, the default directory path is used.
compressionFormat	String	Compression format to use before the streaming ingestion task writes data to the target file.
File Name/Expression	String	ADLS Gen2 file name or a regular expression.
* Enter a value for at least one of the fields.		

If the request is unsuccessful, the response includes a reason for the failure.

POST response example

If the request is successful, you might receive a response similar to the following example in a `Success` node:

```
{
  "Success": {
    "name": "flatfile to adls",
    "description": "flatfile to adls",
    "runtimeId": "0100002500000000003",
    "locationId": "5sJ0JDyJyWLlrosS5qJjsQ",
    "currentVersion": "2",
    "messageFormat": "binary",
    "nodes": [
      {
        "name": "flatfile to adls_source",
        "type": "source",
        "connectionId": "012MGS0B00000000002N",
        "transformationType": "",
        "config": [
          {
            "key": "File",
            "value": "logfile"
          },
          {
            "key": "initialPosition",
            "value": "Current Time"
          },
          {
            "key": "rolloverPattern",
            "value": "test"
```

```
            },
            {
              "key": "tailingMode",
              "value": "Single file"
            }
          ]
        },
        {
          "name": "flatfile to adls_target",
          "type": "target",
          "connectionId": "012MGS0B00000000003D",
          "transformationType": "",
          "config": [
            {
              "key": "writeStrategy",
              "value": "Rollover"
            },
            {
              "key": "filesystemNameOverride",
              "value": "test"
            },
            {
              "key": "File Name/Expression",
              "value": "test"
            },
            {
              "key": "compressionFormat",
              "value": "NONE"
            },
            {
              "key": "directoryOverride",
              "value": "/test"
            },
            {
              "key": "interimDirectory",
              "value": "/home/agent/test"
            },
            {
              "key": "rolloverSize",
              "value": 1024
            },
            {
              "key": "rolloverEvents",
              "value": 100
            },
            {
              "key": "rolloverTime",
              "value": 300000
            }
          ]
        }
      ]
    }
  }
```

Configuration information in the config array for Amazon S3 as a target

The response returns only the fields that you entered in the request.

If the request is successful, the response returns the following fields:

Key	Type	Description
partitionTime	String	The time interval according to which the streaming ingestion task creates partitions in the Amazon S3 bucket.
minUploadPartSize	Integer	Minimum part size when uploading a large file as a set of multiple independent parts, in megabytes. Use this property to tune the file load to Amazon S3.
multipartUploadThreshold	Integer	Multipart threshold when uploading objects in multiple parts in parallel.
Object Name/Expression	String	Amazon S3 target file name or a regular expression for the Amazon S3 file name pattern.

If the request is unsuccessful, the response includes a reason for the failure.

POST response example

If the request is successful, you might receive a response similar to the following example in the `Success` node:


```
{
    "Success": {
    "name": "flatfile to amazon S3",
    "description": "flatfile to amazon S3",
    "runtimeId": "0100002500000000003",
    "locationId": "5sJ0JDyJyWLlrosS5qJjsQ",
    "currentVersion": "2",
    "messageFormat": "binary",
    "nodes": [
      {
        "name": "flatfile to amazon S3_source",
        "type": "source",
        "connectionId": "012MGS0B00000000002N",
        "transformationType": "",
        "config": [
          {
            "key": "File",
            "value": "logfile"
          },
          {
            "key": "initialPosition",
            "value": "Current Time"
          },
          {
            "key": "rolloverPattern",
            "value": "test"
          },
          {
            "key": "tailingMode",
            "value": "Single file"
          }
        ]
      },
      {
        "name": "flatfile to amazon S3_target",
        "type": "target",
        "connectionId": "012MGS0B0000000000I7",
        "transformationType": "",
        "config": [
          {
            "key": "partitionTime",
            "value": "None"
          },
          {
            "key": "minUploadPartSize",
```

```
            "value": 5120
          },
          {
            "key": "multipartUploadThreshold",
            "value": 5120
          },
          {
            "key": "Object Name/Expression",
            "value": "test"
          }
        ]
      }
    ],
    "edges": [
      {
        "from": "flatfile to amazon S3_source",
        "to": "flatfile to amazon S3_target"
      }
    ]
  }
}
```

Configuration information in the config array for Azure Event Hubs as a target

The response returns only the fields that you entered in the request.

If the request is successful, the response returns the following fields:

Key	Type	Description
sasPolicyName	String	The name of the Event Hub Namespace Shared Access Policy.
sasPolicyPrimaryKey	String	The primary key of the Event Hub Namespace Shared Access Policy.
Event Hub	String	The name of the Azure Event Hubs.

If the request is unsuccessful, the response includes a reason for the failure.

POST response example

If the request is successful, you might receive a response similar to the following example in a `Success` node:

```
{
  "Success": {
    "name": "flatfile to azure event hub",
    "description": "flatfile to azure event hub",
    "runtimeId": "0100002500000000003",
    "locationId": "5sJ0JDyJyWLlrosS5qJjsQ",
    "currentVersion": "2",
    "messageFormat": "binary",
    "nodes": [
      {
        "name": "flatfile to azure event hub_source",
        "type": "source",
        "connectionId": "012MGS0B00000000002N",
        "transformationType": "",
        "config": [
          {
            "key": "File",
            "value": "logfile"
          },
          {
            "key": "initialPosition",
            "value": "Current Time"
          },
          {
```

```
            "key": "rolloverPattern",
            "value": "test"
          },
          {
            "key": "tailingMode",
            "value": "Single file"
          }
        ]
      },
      {
        "name": "flatfile to azure event hub_target",
        "type": "target",
        "connectionId": "012MGS0B000000000001S",
        "transformationType": "",
        "config": [
          {
            "key": "sasPolicyName",
            "value": "test"
          },
          {
            "key": "sasPolicyPrimaryKey",
            "value": "test"
          },
          {
            "key": "Event Hub",
            "value": "test"
          }
        ]
      }
    ],
    "edges": [
      {
        "from": "flatfile to azure event hub_source",
        "to": "flatfile to azure event hub_target"
      }
    ]
  }
}
```

Configuration information in the config array for JDBC as a target

The response returns only the fields that you entered in the request.

If the request is successful, the response returns the following field:

Key	Type	Description
Table Name	String	Name of the table to insert data to in JSON format.

If the request is unsuccessful, the response includes a reason for the failure.

POST response example

If the request is successful, you might receive a response similar to the following example in a `Success` node:

```
{
  "Success": {
    "name": "FileFile to jdbc",
    "description": "FileToFile to jdbc_target",
    "runtimeId": "0100002500000000003",
    "locationId": "5sJ0JDyJyWLlrosS5qJjsQ",
    "currentVersion": "2",
    "messageFormat": "binary",
    "nodes": [
      {
        "name": "flatfile to jdbc_source",
        "type": "source",
```

```
        "connectionId": "012MGS0B00000000002N",
        "transformationType": "",
        "config": [
          {
            "key": "initialPosition",
            "value": "Current Time"
          },
          {
            "key": "tailingMode",
            "value": "Single file"
          },
          {
            "key": "rolloverPattern",
            "value": "test"
          },
          {
            "key": "File",
            "value": "logfile"
          }
        ]
      },
      {
        "name": "flatfile to jdbc_target",
        "type": "target",
        "connectionId": "012MGS0B0000000000KF",
        "transformationType": "",
        "config": [
          {
            "key": "Table Name",
            "value": "table"
          }
        ]
      }
    ],
    "edges": [
      {
        "from": "flatfile to jdbc_source",
        "to": "flatfile to jdbc_target"
      }
    ]
  }
}
```

Configuration information in the config array for Amazon Kinesis Streams as a source and as a target

The response returns only the fields that you entered in the request.

If the request is successful, the response returns the following fields:

Key	Type	Description
appendGUID	Boolean	Specifies whether or not to add a GUID as a suffix to the Amazon DynamoDB table name.
dynamoDB	String	Amazon DynamoDB table name where to store the checkpoint details of the Kinesis source data.

Key	Type	Description
Stream	String	Name of the Kinesis Stream from where to read data. Applies when you use Amazon Kinesis Streams as a source.
Stream Name/ Expression	String	Kinesis stream name or a regular expression for the Kinesis stream name pattern. Applies when you use Amazon Kinesis Streams as a target.

If the request is unsuccessful, the response includes a reason for the failure.

POST response example

If the request is successful, you might receive a response similar to the following example in a `Success` node:

```
{
    "Success": {
    "name": "kinesis to kinesis",
    "description": "kinesis to kinesis",
    "runtimeId": "0100002500000000003",
    "locationId": "5sJ0JDyJyWLlrosS5qJjsQ",
    "currentVersion": "2",
    "messageFormat": "binary",
    "nodes": [
      {
        "name": "kinesis to kinesis_source",
        "type": "source",
        "connectionId": "012MGS0B00000000000F",
        "transformationType": "",
        "config": [
          {
            "key": "appendGUID",
            "value": true
          },
          {
            "key": "dynamoDB",
            "value": "table"
          },
          {
            "key": "Stream",
            "value": "test"
          }
        ]
      },
      {
        "name": "kinesis to kinesis_target",
        "type": "target",
        "connectionId": "012MGS0B00000000000F",
        "transformationType": "",
        "config": [
          {
            "key": "Stream Name/Expression",
            "value": "trgt"
          }
        ]
      }
    ],
    "edges": [
      {
        "from": "kinesis to kinesis_source",
        "to": "kinesis to kinesis_target"
      }
    ]
  }
}
```

Configuration information in the config array for flat file as a source and as a target

The response returns only the fields that you entered in the request.

If the request is successful, the response returns the following fields:

Key	Type	Required	Description
File	String	Yes	Absolute path and name of the source file you want to read.
initialPosition	String	Yes	Starting position to read data from the file to tail.
rolloverPattern	String	-	File name pattern for the file that rolls over.
tailingMode	String	Yes	Tail a file or multiple files based on the logging pattern.
File Name	String	Yes	The name of the target file.
interimDirectory	String	Yes	Path to the staging directory on the Secure Agent.
rolloverSize	Integer	Yes	The file size, in KB, at which the task moves the file from the staging directory to the target.
rolloverEvents	Integer	Yes	Number of events or messages to accumulate before a file rollover.
rolloverTime	Integer	-	Length of time, in milliseconds, after which the target file rolls over.

If the request is unsuccessful, the response includes a reason for the failure.

POST response example

If the request is successful, you might receive a response similar to the following example:

```
{
  "Success": {
    "name": "FileToFile",
    "description": "FileToFile_V2",
    "runtimeId": "0100002500000000003",
    "locationId": "5sJ0JDyJyWLlrosS5qJjsQ",
    "currentVersion": "2",
    "messageFormat": "binary",
    "nodes": [
      {
        "name": "FileToFile_source",
        "type": "source",
        "connectionId": "0100000B000000000002",
        "transformationType": "",
        "config": [
          {
            "key": "File",
            "value": "siagent.log"
          },
          {
            "key": "initialPosition",
            "value": "Current Time"
          },
          {
            "key": "rolloverPattern",
            "value": ""
          },
          {
            "key": "tailingMode",
            "value": "Single file"
```

```
            }
          ]
        },
        {
          "name": "FileToFile_target",
          "type": "target",
          "connectionId": "0100000B000000000002",
          "transformationType": "",
          "config": [
            {
              "key": "File Name",
              "value": "testing.log"
            },
            {
              "key": "interimDirectory",
              "value": "/home/agent/infa/test_file_target"
            },
            {
              "key": "rolloverSize",
              "value": 100
            },
            {
              "key": "rolloverEvents",
              "value": 100
            },
            {
              "key": "rolloverTime",
              "value": 100
            }
          ]
        }
      ],
      "edges": [
        {
          "from": "FileToFile_source",
          "to": "FileToFile_target"
        }
      ],
      "runtimeOptions": {
        "maxLogSize": {
          "value": 10,
          "unit": "MB"
        },
        "logLevel": "INFO"
      }
    }
  }
```

Configuration information in the config array for Kafka as a source and as a target

The response returns only the fields that you entered in the request.

If the request is successful, the response returns the following fields:

Key	Type	Description
Topic	String	Kafka source topic name or a Java supported regular expression for the Kafka source topic name pattern to read the events from.
consumerProperties	String	A comma-separated list of optional consumer configuration properties.
producerProperties	String	The configuration properties for the producer.

Key	Type	Description
mdFetchTimeout	Integer	The time after which the metadata is not fetched.
batchSize	Integer	The batch size of the events after which a streaming ingestion task writes data to the target.
Topic Name/Expression	String	Kafka topic name or a Java supported regular expression for the Kafka topic name pattern.

If the request is unsuccessful, the response includes a reason for the failure.

POST response example

If the request is successful, you might receive a response similar to the following example in a `Success` node:


```
{
  "Success": {
    "name": "kafka to kafka",
    "description": "kafka to kafka",
    "runtimeId": "0100002500000000003",
    "locationId": "5sJ0JDyJyWLlrosS5qJjsQ",
    "currentVersion": "2",
    "messageFormat": "binary",
    "nodes": [
      {
        "name": "kafka to kafka_source",
        "type": "source",
        "connectionId": "012MGS0B000000000002",
        "transformationType": "",
        "config": [
          {
            "key": "consumerProperties",
            "value": "key=value"
          },
          {
            "key": "Topic",
            "value": "test"
          }
        ]
      },
      {
        "name": "kafka to kafka_target",
        "type": "target",
        "connectionId": "012MGS0B000000000002",
        "transformationType": "",
        "config": [
          {
            "key": "producerProperties",
            "value": "key=value"
          },
          {
            "key": "mdFetchTimeout",
            "value": 5000
          },
          {
            "key": "batchSize",
            "value": 1048576
          },
          {
            "key": "Topic Name/Expression",
            "value": "test"
          }
        ]
      }
    ],
    "edges": [
```

```
        {
          "from": "kafka to kafka_source",
          "to": "kafka to kafka_target"
        }
      ]
    }
  }
```

jobs resource

Use the jobs resource to get the details of a streaming ingestion job.

GET request

To request the details of a streaming ingestion job, use the following URL:

```
<server URI>/sisvc/monitor/v1/jobs/<dataflow ID>/<run ID of the job>
```

GET request example

To request the details of a streaming ingestion job, you might send a request similar to the following example:

```
POST https://usw1-ing.dm2-us.informaticacloud.com/sisvc/monitor/v1/jobs/
1948938e-3923-4602-aba8-f122e3d66faf/42559
Content-Type: application/json
Accept:application/json
IDS-SESSION-ID:2l0oeVx22Rujiej7yTokmT
```

GET response

Returns the jobs object if successful or an error object if an error occurs.

If successful, the response includes the following information about a streaming ingestion job:

Parameter	Type	Description
assetId	String	ID of the streaming ingestion job.
assetName	String	Name of the streaming ingestion job.
duration	Integer	The time it took to deploy the job.
endTime	Integer	End time of deploying the job, in UTC time.
startTime	Integer	Start time of deploying the job, in UTC time.
extraData	String	Additional information including the task ID, deployed version, and the Secure Agent group ID.
runId	Integer	Run ID of the streaming ingestion job. The ID changes for every deployment.
orgId	String	ID of the organization the logged in user belongs to.
runtimeEnv	String	ID of the Secure Agent that deployed the streaming ingestion job.

Parameter	Type	Description
startedBy	String	Name of the user who created the streaming ingestion task.
status	String	The status of the streaming ingestion job. A job can be in one of the following status: - Deploying. The job is being deployed. - Up and Running. The job is running. - Running with Warning. The job is running with warnings. - Running with Error. The job is running with error. - Undeployed. The job is undeployed. - Stopped. The job was intentionally stopped.

GET response example

If the request to get the details of a streaming ingestion job is successful, you might receive a response similar to the following example:

```
{
    "assetId": "1948938e-3923-4602-aba8-f122e3d66faf",
    "assetName": "testmonitor",
    "assetType": "SI_DATAFLOW",
    "correlationId": null,
    "duration": 1543,
    "endTime": "2022-02-14T04:04:13.000+0000",
    "extraData": "{\"id\":\"0RwiUUb9bVwjL67dWOKjoI\",\"version\":1,\"agentGroupId
\":null}",
    "location": "Default",
    "runId": 42559,
    "orgId": "2lFy0UUNnlnbjhaoT3TSqw",
    "runtimeEnv": "011ZFB2500000000000N",
    "startedBy": "siqa_new",
    "status": "Undeployed",
    "startTime": "2022-02-14T03:38:30.000+0000",
    "deployedVersion": 1
}
```

MIJobs resource

Use the MIJobs resource to get a list of the available streaming ingestion jobs.

GET request

To request a list of the available streaming ingestion jobs, use the following URL:

```
<server URI>/mijobmonitor/api/v1/MIJobs
```

You can include the following query parameters in the URI:

Parameter	Type	Required	Description
$count	Boolean	No	Displays the number of ingestion jobs in the database.
$filter	String	No	Filters the job based on the input. You can filter using one of the following fields: - assetName - assetType - startedBy - status You can filter jobs using single or multiple fields.
$orderby	String	No	Sorts the order of the jobs. You can sort the jobs using the following fields: - assetName - assetType - status - runtimeEnv - startTime You can sort jobs using single or multiple fields.
$skip	Integer	No	Skips the number of streaming ingestion jobs that you specify. For example, you might want to skip the first five streaming ingestion jobs. Consider the $filter and $orderby parameter values, if specified.
$top	Integer	No	Displays the number of top streaming ingestion jobs that you specify. For example, you might want to view the top ten streaming ingestion jobs. Consider the $filter and $orderby parameter values, if specified.

GET request example

To get a list of the available streaming ingestion jobs, you might send a request similar to the following example:

```
POST https://usw1-ing.dm2-us.informaticacloud.com/mijobmonitor/api/v1/MIJobs?$count=true&
$filter=(startedBy eq 'siqa_new')&$orderby=deployTime desc&$skip=0&$top=25
Content-Type: application/json
Accept:application/json
IDS-SESSION-ID:2l0oeVx22Rujiej7yTokmT
```

GET response

Returns the MIjobs object if successful or an error object if an error occurs.

If successful, the response includes the following information about the streaming ingestion job:

Parameter	Type	Description
assetName	String	Name of the streaming ingestion job.
runId	Integer	Run ID of the streaming ingestion job. The ID changes for every deployment.
orgId	String	ID of the organization the logged in user belongs to.
runtimeEnv	String	ID of the Secure Agent that deployed the streaming ingestion job.
startTime	Integer	Date and start time of deploying the job, in UTC time.

Parameter	Type	Description
endTime	Integer	Date and end time of deploying the job, in UTC time.
deployTime	Integer	Date and time of deploying the job, in UTC time.
undeployTime	Integer	Date and time of undeploying the job, in UTC time.
startedBy	Integer	Name of the user who created the streaming ingestion task.
status	String	The status of the streaming ingestion job. A job can be in one of the following status: - Deploying. The job is being deployed. - Up and Running. The job is running. - Running with Warning. The job is running with warnings. - Running with Error. The job is running with error. - Undeployed. The job is undeployed. - Stopped. The job was intentionally stopped.
extraData	String	Additional information including the task ID, the location of the streaming ingestion job, and the Secure Agent ID.

GET response example

If the request to get a list of available streaming ingestion jobs is successful, you might receive a response similar to the following example:


```
{
    "@odata.context": "$metadata#Collection(OData.MI.JobMonitor.MIJob)",
    "@odata.count": 421,
    "value": [
        {
            "assetId": "7ce6bbc7-f0e2-4278-bd6d-d1187f4a1420",
            "assetName": "SIdeployJms",
            "assetType": "SI_DATAFLOW",
            "runId": 33015,
            "duration": 300000,
            "orgId": "1Pm6cSfPcAqfgeV57Fn3u4",
            "runtimeEnv": "011U5M08000000000003",
            "startTime": "2021-04-29T13:09:48.000+0000",
            "endTime": "2021-04-29T13:14:48.000+0000",
            "deployTime": "2021-04-29T13:09:48.000+0000",
            "undeployTime": "2021-04-29T13:14:48.000+0000",
            "startedBy": "siqa_new",
            "status": "Undeployed",
            "outOfSync": true,
            "extraData": "{\"taskId\":\"7Z4ZZjXc9QViT4t2okiHuz\",\"runtimeEnv
\":\"011U5M25000000000002\",\"location\":\"RestAutomation\"}",
            "deployedVersion": 1,
            "replace": null,
            "lastUpdateTime": 0
        },
        {
            "assetId": "a03b9aa1-4a4a-47ee-808d-ddc0ee7b3a4a",
            "assetName": "kafka to kafka test",
            "assetType": "SI_DATAFLOW",
            "runId": 33527,
            "duration": 204988000,
            "orgId": "1Pm6cSfPcAqfgeV57Fn3u4",
            "runtimeEnv": "011U5M08000000000002",
            "startTime": "2021-05-04T05:41:39.000+0000",
            "endTime": "2021-05-06T14:38:07.000+0000",
            "deployTime": "2021-05-04T05:41:39.000+0000",
            "undeployTime": "2021-05-06T14:38:07.000+0000",
            "startedBy": "siqa_new",
            "status": "Undeployed",
```

```
            "outOfSync": true,
            "extraData": "{\"taskId\":\"8V21nib7Sqgiw3QoDRi5uK\",\"runtimeEnv
\":\"011U5M25000000000002\",\"location\":\"Default\"}",
            "deployedVersion": 1,
            "replace": null,
            "lastUpdateTime": 0
        }
    ]
}
```

status resource

Use the status resource to get the status of a streaming ingestion job.

GET request

To request the status of a streaming ingestion job, use the following URL:

```
<server URI>/sisvc/monitor/v1/status/dataflows/<dataflow ID>
```

GET request example

To get the status of a streaming ingestion job, you might send a request similar to the following example:

```
POST https://usw1-ing.dm2-us.informaticacloud.com/sisvc/monitor/v1/status/dataflows/
1948938e-3923-4602-aba8-f122e3d66faf
Content-Type: application/json
Accept:application/json
IDS-SESSION-ID:2l0oeVx22Rujiej7yTokmT
```

GET response

Returns the job status object if successful or an error object if an error occurs.

If successful, the response includes the following information about the status of a streaming ingestion job:

Parameter	Type	Description
dataflowName	String	Name of the streaming ingestion job.
dataflowId	Integer	ID of the streaming ingestion job.
status	String	The status of the streaming ingestion job. A job can be in one of the following status: - Deploying. The job is being deployed. - Up and Running. The job is running. - Running with Warning. The job is running with warnings. - Running with Error. The job is running with error. - Undeployed. The job is undeployed. - Stopped. The job was intentionally stopped.
timestamp	Integer	Time, in milliseconds, when the Secure Agent records the status of the streaming ingestion job.
reports	Array	Status details of each node.

Parameter	Type	Description
graph	String	The throughput information for the source and target of the job.
runId	Integer	Run ID of the streaming ingestion job. The ID changes for every deployment.

GET response example

If the request to get the status of a streaming ingestion job is successful, you might receive a response similar to the following example:


```
{
    "dataflowName": "testmonitor",
    "dataflowId": "1948938e-3923-4602-aba8-f122e3d66faf",
    "status": "Running",
    "timestamp": 1644839755000,
    "reports": [
        {
            "name": "testmonitor_testmonitor_source",
            "id": "a5684428-f41f-4d24-b73f-33c232314a91",
            "status": "Running",
            "timestamp": 1644839756000,
            "message": null
        },
        {
            "name": "testmonitor_testmonitor_target",
            "id": "4f59b5fb-b5b2-4b83-994b-0d3e56f67e22",
            "status": "Running",
            "timestamp": 1644839756000,
            "message": null
        }
    ],
    "graph": "{\"agentId\":\"011ZFB0800000000000N\",\"nodes\":[{\"id\":\"a5684428-
f41f-4d24-b73f-33c232314a91\",\"name\":\"testmonitor_source\",\"serviceType\":\"source
\",\"config\":[{\"key\":\"_nativeName\",\"value\":\"src\"},{\"key\":\"consumerProperties
\",\"value\":null}],\"connectionId\":\"011ZFB0B0000000000KJ\",\"type
\":\"\",\"metaMetadata\":\"\"},{\"id\":\"4f59b5fb-b5b2-4b83-994b-0d3e56f67e22\",\"name
\":\"testmonitor_target\",\"serviceType\":\"target\",\"config\":[{\"key\":\"_nativeName
\",\"value\":\"trgt\"},{\"key\":\"batchSize\",\"value\":\"1048576\"},{\"key
\":\"mdFetchTimeout\",\"value\":\"5000\"},{\"key\":\"producerProperties\",\"value
\":null}],\"connectionId\":\"011ZFB0B0000000000KJ\",\"type\":\"\",\"metaMetadata
\":\"\"}],\"edges\":[{\"id\":\"6ae185ea-7e6e-4bf6-bd9e-0be5ef3a8e78\",\"name
\":\"testmonitor_source_testmonitor_target\",\"from\":\"testmonitor_source\",\"to
\":\"testmonitor_target\",\"type\":\"success\",\"config\":[],\"metaMetadata
\":\"\"}],\"runtimeOptions\":null}",
    "version": 1,
    "runId": 42563
}
```


statistics resource

Use the statistics resource to get the statistics of a streaming ingestion job.

The streaming ingestion job should be in one of the following status before you can view its statistics:

- Deploying
- Up and Running
- Running with Warning

- Running with Error
- Stopped

GET request

To request the statistics of a streaming ingestion job, use the following URL:

```
<server URI>/sisvc/monitor/v1/statistics/dataflows/<dataflow ID>
```

You can include the following query parameters in the URI:

Parameter	Type	Required	Description
intervals	Integer	Yes	Time, in seconds, to display statistics for a streaming ingestion job. For example, if you specify 30 seconds, the response displays job statistics for the last 30 seconds.
overall	Boolean	No	Displays the statistics from the time the job is deployed.

GET request example

To request the statistics of a streaming ingestion job, you might send a request similar to the following example:

```
POST https://usw1-ing.dm2-us.informaticacloud.com/sisvc/monitor/v1/statistics/
dataflows/7f1daca9-3983-4677-930f-a9529802c56b?intervals=30&overall=true
Content-Type: application/json
Accept:application/json
IDS-SESSION-ID:2l0oeVx22Rujiej7yTokmT
```

GET response

Returns the statistics object if successful or an error object if an error occurs.

If successful, the response includes the following information about the statistics of a streaming ingestion job:

Parameter	Type	Description
dataflowId	String	ID of the streaming ingestion job.
dataflowRunId	Integer	Run ID of the streaming ingestion job.
startTime	Integer	Start time of the streaming ingestion job, in milliseconds.
stopTime	Integer	Stop time of the streaming ingestion job, in milliseconds.
inMessages	Integer	The number of messages that arrive at a node. A node is a source, transformation, or target, that is used in the streaming ingestion task. The value is zero for a source node.
outMessages	Integer	The number of messages that transfer from a node. The value is zero for a target node.
inBytes	Integer	The total size of incoming messages in bytes. The value is zero for a source node.

Parameter	Type	Description
outBytes	Integer	The total size of outgoing messages in bytes. The value is zero for a target node.
nodes	Array	Information about streaming data in the source and the target used in the task.
intervals	Integer	The statistics of the job for the time interval you specify in the request. Applies when you set an interval.

GET response example

If the request to get the statistics of a streaming ingestion job is successful, you might receive a response similar to the following example:


```
{
    "dataflowId": "7f1daca9-3983-4677-930f-a9529802c56b",
    "dataflowName": "newnew",
    "dataflowVersion": 1,
    "dataflowRunId": 54231,
    "snapshotCount": 171,
    "overall": {
        "dataflowId": "7f1daca9-3983-4677-930f-a9529802c56b",
        "dataflowName": "newnew",
        "dataflowVersion": 1,
        "dataflowRunId": 54231,
        "traits": {},
        "interval": 6007,
        "startTime": 1646649995000,
        "stopTime": 1646656000000,
        "nodes": [
            {
                "name": "newnew_newnew_source",
                "id": "17a51cdf-1f27-481e-81b8-d2e8ff60ec28",
                "inMessages": 0,
                "outMessages": 0,
                "inBytes": 0,
                "outBytes": 0,
                "nodeType": "Unknown"
            },
            {
                "name": "newnew_newnew_target",
                "id": "c30d6db4-6a3b-40d3-adfb-88779a972098",
                "inMessages": 0,
                "outMessages": 0,
                "inBytes": 0,
                "outBytes": 0,
                "nodeType": "Unknown"
            }
        ]
    },
    "intervals": {
        "30": {
            "dataflowId": "7f1daca9-3983-4677-930f-a9529802c56b",
            "dataflowName": "newnew",
            "dataflowVersion": null,
            "dataflowRunId": 54231,
            "traits": {},
            "interval": 30,
            "startTime": 1646655972683,
            "stopTime": 1646656002683,
            "nodes": []
        }
    }
}
```

history resource

Use the history resource to get the history of a streaming ingestion job.

GET request

To request the history of a streaming ingestion job, use the following URL:

```
<server URI>/sisvc/monitor/v1/history/dataflows/<dataflow ID>
```

GET request example

To get the history of a streaming ingestion job, you might send a request similar to the following example:

```
POST https://usw1-ing.dm2-us.informaticacloud.com/siscv/monitor/v1/history/dataflows/
1948938e-3923-4602-aba8-f122e3d66faf
Content-Type: application/json
Accept:application/json
IDS-SESSION-ID:2l0oeVx22Rujiej7yTokmT
```

GET response

Returns the job history object if successful or an error object if an error occurs.

If successful, the response includes the following information about the history of a streaming ingestion job:

Parameter	Type	Description
dataflowName	String	Streaming ingestion job name.
dataflowId	Integer	Streaming ingestion job ID.
deployedAt	Integer	The start time of deploying the job, in UTC time.
undeployedAt	Integer	The time when the job finished undeploying, in UTC.
runID	Integer	Run ID of the streaming ingestion job. The ID changes for every deployment.

GET response example

If the request to get the history of a streaming ingestion job is successful, you might receive a response similar to the following example:

```
[
    {
        "dataflowId": "1948938e-3923-4602-aba8-f122e3d66faf",
        "dataflowName": "testmonitor",
        "deployedAt": 1644809910000,
        "undeployedAt": 1644811453000,
        "dataflowVersion": 1,
        "runId": 42559,
        "overall": null,
        "intervals": {},
        "graph": null
    },
    {
        "dataflowId": "1948938e-3923-4602-aba8-f122e3d66faf",
        "dataflowName": "testmonitor",
        "deployedAt": 1644811513000,
        "undeployedAt": 1644838813000,
        "dataflowVersion": 1,
        "runId": 42561,
        "overall": null,
        "intervals": {},
        "graph": null
```

```
        }
    ]
```

INDEX

T