Overview

Splunk Connect for Kafka

<table>
<thead>
<tr>
<th>Version</th>
<th>2.0.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vendor Products</td>
<td>Kafka Connect</td>
</tr>
<tr>
<td>Visible in Splunk Web</td>
<td>No. This product does not contain any views.</td>
</tr>
</tbody>
</table>

Splunk Connect for Kafka is a sink connector that allows a Splunk software administrator to subscribe to a Kafka topic and stream the data to the Splunk HTTP Event Collector. After the Splunk platform indexes the events, you can then directly analyze the data or use it as a contextual data feed to correlate with other Kafka-related data in the Splunk platform.

Download Splunk Connect for Kafka from Splunkbase.

For a summary of new features, fixed issues, and known issues, see the Release Notes for Splunk Connect for Kafka.

For information about installing and configuring Splunk Connect for Kafka, see the Installation section of this manual.

For developer-based documentation, visit the Splunk Connect for Kafka Github page.

Hardware and software requirements for Splunk Connect for Kafka

To install Splunk Connect for Kafka, you must meet the following requirements.

Plan your deployment

Use one of the following connector deployment options to deploy Splunk Connect for Kafka:

- Splunk Connect for Kafka in a dedicated Kafka Connect Cluster (best practice).
- Splunk Connect for Kafka in an existing Kafka Connect Cluster.

Splunk Connect for Kafka can run in containers, in virtual machines, or on physical machines. You can leverage any automation tools for deployment.

See the Plan a deployment section of the Splunk Enterprise manual for more information on planning your Splunk platform deployment.

System requirements

- A Kafka Connect environment running Kafka version 1.0.0 or later.
- Java 8 or later.
- Splunk platform environment of version 7.1 or later.
- Configured and valid HTTP Event Collector (HEC) tokens.

If you are using Splunk Cloud, use the Splunk Support Portal to request that Splunk Connect for Kafka be installed on your deployment. Splunk Support will set up and provide a URL for your HTTP Event Collector endpoint. If you are ingesting Kinesis Firehose events, you can reuse the HTTP Event Collector (HEC) endpoint setting you configured for the
Architecture requirements

Splunk Connect for Kafka supports two types of architectures:

- Directly inject data to a Splunk platform indexer cluster. For example:

  A Kafka Connect Cluster (in containers or virtual machines or physical machines) -> Splunk Indexer Cluster (HEC)

- Set up a heavy forwarder layer in front of a Splunk platform indexer cluster to offload the data injection load to your Splunk platform indexer cluster. Setting up a heavy forwarder layer can help distribute computational resources across your Splunk platform deployment. For example:

  A Kafka Connect Cluster (in containers, virtual machines, or physical machines) -> Heavy Forwarders (HEC) -> Splunk Indexer Cluster

Optionally, the Splunk Connect for Kafka can use its internal load balancing to communicate to HEC ports on the indexers directly. See the parameter `splunk.hec.uri` in the Parameters topic of this manual to learn more.

Sizing guidelines

Determine how many Kafka Connect instances to deploy by calculating how much volume per day Splunk Connect for Kafka needs to index in your Splunk platform deployment. For example, an 8 CPU, 16 GB memory machine can potentially achieve 50 - 60 MBs per second throughput from Kafka Connect into your Splunk platform deployment if your Splunk platform deployment is sized correctly.

Do not create more tasks than the number of partitions in your deployment. Creating 2 * CPU tasks per Kafka Connector is a safe estimate.

For example, if you have the following deployment:

- 5 Kafka Connects running the Splunk Connect for Kafka.
- Each host has 8 CPUs with 16 GB memory.
- There are 200 partitions to collect data from. `max.tasks` will be: `max.tasks = 2 * CPUs/host * Kafka Connect instances = 2 * 8 * 5 = 80 tasks`.
- Alternatively, if there are only 60 partitions to consume from, set `max.tasks` to 60.

Benchmark results

A single instance of Splunk Connect for Kafka can reach maximum indexed throughput of 32 MB/second with the following testbed and raw HEC endpoint in use:

Hardware specifications:

- **AWS**: EC2 c4.2xlarge, 8 vCPU and 31 GB Memory.
- **Splunk Cluster**: 3 indexer cluster without load balancer.
- **Kafka Connect**: JVM heap size configuration is "-Xmx6G -Xms2G".
• **Kafka Connect resource usage**: ~6GB memory, ~3 vCPUs.
• **Kafka records size**: 512 Bytes.
• **Batch size**: Maximum 100 Kafka records per batch which is around 50KB per batch.

**HTTP Event Collector (HEC) requirements**

• HEC token settings must be the same on all Splunk platform data injection nodes in your environment, including indexers and heavy forwarders.
• *(Optional)* When creating a HEC token, enable indexer acknowledgment in order to prevent potential data loss.
• Enable HEC token acknowledgements in order to avoid data loss. This is a best practice.

If indexer acknowledgment is enabled, set `ackIdleCleanup` to `true` in `inputs.conf`.

See Set up and use HTTP Event Collector in Splunk Web in the *Splunk Enterprise* manual and About HTTP Event Collector Indexer Acknowledgment for more information.

**Data ingestion parameters for Splunk Connect for Kafka**

Use the following parameters to specify the types of data that you want to ingest into your Splunk platform deployment.

**Required parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Connector name. A consumer group with this name will be created with tasks to be distributed evenly across the connector cluster nodes.</td>
</tr>
<tr>
<td>connector.class</td>
<td>The Java class used to perform connector jobs. Keep the default value <code>com.splunk.kafka.connect.SplunkSinkConnector</code> unless you modify the connector.</td>
</tr>
<tr>
<td>tasks.max</td>
<td>The number of tasks generated to handle data collection jobs in parallel. The tasks will be spread evenly across all Splunk Connect for Kafka connector nodes.</td>
</tr>
<tr>
<td>splunk.hec.uri</td>
<td>Splunk HTTP Event Collector (HEC) URIs. Either a list of Fully Qualified Domain Names (FQDNs) or IPs of all Splunk indexers, separated with a &quot;,&quot; or a load balancer. The connector will load balance to indexers using round robin. Splunk Connector will round robin to this list of indexers. For example, <code>&lt;code&gt;https://hec1.splunk.com:8088, https://hec2.splunk.com:8088, https://hec3.splunk.com:8088&lt;/code&gt;</code>.</td>
</tr>
<tr>
<td>splunk.hec.token</td>
<td>Splunk HEC token.</td>
</tr>
<tr>
<td>topics</td>
<td>Comma-separated list of Kafka topics for Splunk to consume. For example, <code>prod-topic1,prod-topic2,prod-topic3</code>.</td>
</tr>
</tbody>
</table>

**Header parameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>splunk.header.support</td>
<td>Header name. A consumer group with this name will be created with tasks to be distributed evenly across the connector cluster nodes. Parses Kafka headers for using metadata in generated Splunk software events. By default, this setting is set to <code>false</code>. Requires Kafka Connect version 1.1 or later.</td>
</tr>
<tr>
<td>splunk.header.custom</td>
<td>Header name. Applicable when <code>splunk.header.support</code> is set to <code>true</code>. Custom headers are configured separated by comma for multiple headers. For example, <code>custom_header_1,custom_header_2,custom_header_3</code>. This setting will look for Kafka record headers.</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>splunk.header.index</td>
<td>Header name. Applicable when splunk.header.support is set to true. This setting specifies the header to be used for the Splunk platform index. By default, it is set to splunk.header.index.</td>
</tr>
<tr>
<td>splunk.header.source</td>
<td>Header name. Applicable when splunk.header.support is set to true. This setting specifies the source to be used for the Splunk platform source. By default, it is set to splunk.header.source.</td>
</tr>
<tr>
<td>splunk.header.sourcetype</td>
<td>Header name. Applicable when splunk.header.support is set to true. This setting specifies the sourcetype to be used for the Splunk software sourcetype. By default, it is set to splunk.header.sourcetype.</td>
</tr>
<tr>
<td>splunk.header.host</td>
<td>Header name. Applicable when splunk.header.support is set to true. This setting specifies the host to be used for the Splunk software host. By default, it is set to splunk.header.host.</td>
</tr>
</tbody>
</table>

Optional parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>splunk.indexes</td>
<td>Target Splunk indexes to send data to. This can be a list of indexes, and can also be the same sequence and order as topics.</td>
</tr>
<tr>
<td></td>
<td>It is possible to inject data from different Kafka topics to different Splunk platform indexes. For example, prod-topic1, prod-topic2, and prod-topic3 can be sent to index prod-index1, prod-index2, and prod-index3.</td>
</tr>
<tr>
<td></td>
<td>If you want to index all data from multiple topics to the main index, then &quot;main&quot; can be specified. If you leave this setting unconfigured, data will route to the default index configured against the HEC token. Verify that the indexes configured here are in the index list of HEC tokens, otherwise Splunk HEC will drop the data. By default, this setting is empty.</td>
</tr>
<tr>
<td>splunk.sources</td>
<td>Splunk event source metadata for the Kafka topic data. The same configuration rules as indexes can be applied. If left unconfigured, the default source binds to the HEC token. By default, this setting is empty.</td>
</tr>
<tr>
<td>splunk.sourcetypes</td>
<td>Splunk event source metadata for the Kafka topic data. The same configuration rules as indexes can be applied here. If left unconfigured, the default source binds to the HEC token. By default, this setting is empty.</td>
</tr>
<tr>
<td>splunk.hec.backoff.threshhold.seconds</td>
<td>The amount of time Splunk Connect for Kafka waits to attempt resending after errors from a HEC endpoint.</td>
</tr>
<tr>
<td>splunk.flush.window</td>
<td>The interval, in seconds, at which the events from Kafka connect will be flushed to your Splunk platform instance. By default, this is set to 30.</td>
</tr>
<tr>
<td>splunk.hec.ssl.validate.certs</td>
<td>Valid settings are true or false, and they enable or disable HTTPS certification validation. By default, this is set to true.</td>
</tr>
<tr>
<td>splunk.hec.http.keepalive</td>
<td>Valid settings are true or false, and they enable or disable HTTPS connection keep-alive. By default, this is set to true.</td>
</tr>
<tr>
<td>splunk.hec.max.http.connection.per.channel</td>
<td>Controls how many HTTP connections will be created and cached in the HTTP pool for one HEC channel. By default, this is set to 2.</td>
</tr>
<tr>
<td>splunk.hec.max.outstanding.events</td>
<td>Maximum amount of un-acknowledged events kept in memory by connector. Will trigger back-pressure event to slow down collection if reached.</td>
</tr>
<tr>
<td>splunk.hec.lb.poll.interval</td>
<td>Specify this parameter(in seconds) to control the polling interval (increase to do less polling, decrease to do more frequent polling). Default is 120.</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>splunk.hec.total.channels</td>
<td>Controls the total channels created to perform HEC event POSTs. By default, this is set to 2.</td>
</tr>
<tr>
<td>splunk.hec.max.batch.size</td>
<td>Maximum batch size when posting events to Splunk. The size is the actual number of Kafka events, and not byte size. By default, this is set to 500.</td>
</tr>
<tr>
<td>splunk.hec.threads</td>
<td>Controls how many threads are spawned to do data injection via HEC in a single connector task. By default, this is set to 1.</td>
</tr>
<tr>
<td>splunk.hec.socket.timeout</td>
<td>Internal TCP socket timeout when connecting to Splunk. By default, this is set to 60 seconds.</td>
</tr>
<tr>
<td>splunk.hec.json.event.formatted</td>
<td>Set to true for events that are already in HEC format. Valid settings are true or false.</td>
</tr>
<tr>
<td>splunk.hec.threads</td>
<td>Controls how many threads are spawned to do data injection via HEC in a single connector task. By default, this is set to 1.</td>
</tr>
<tr>
<td>splunk.hec.socket.timeout</td>
<td>Internal TCP socket timeout when connecting to Splunk. By default, this is set to 60 seconds.</td>
</tr>
<tr>
<td>splunk.hec.ack.enabled</td>
<td>Valid settings are true or false. When set to true the Splunk Connect for Kafka connector will poll event acknowledgments (ACKs) for POST events before check-pointing the Kafka offsets. This is used to prevent data loss, as this setting implements guaranteed delivery. By default, this setting is set to true. If this setting is set to true, verify that the corresponding HEC token is also enabled with index acknowledgments, otherwise the data injection will fail, due to duplicate data. When set to false, the Splunk Connect for Kafka connector will only POST events to your Splunk platform instance. After it receives an HTTP 200 OK response, it assumes the events are indexed by Splunk. In cases where the Splunk platform crashes, there may be data loss.</td>
</tr>
<tr>
<td>splunk.hec.ack.poll.interval</td>
<td>This setting is only applicable when splunk.hec.ack.enabled is set to true. Internally it controls the event ACKs polling interval. By default, this setting is set to 10 seconds.</td>
</tr>
<tr>
<td>splunk.hec.ack.poll.threads</td>
<td>This setting is used for performance tuning and is only applicable when splunk.hec.ack.enabled is set to true. It controls how many threads should be spawned to poll event ACKs. By default, this is set to 1. For large Splunk indexer clusters (for example, 100 indexers) increase this number. Speed up ACK polling by increasing to 4 threads.</td>
</tr>
<tr>
<td>splunk.hec.event.timeout</td>
<td>This setting is applicable when splunk.hec.ack.enabled is set to true. This setting determines how long the connector will wait before timing out and resending when events are POSTed to Splunk and before they are ACKed. By default, this setting is set to 300 seconds.</td>
</tr>
</tbody>
</table>

Acknowledgment parameters (optional)

Enable HTTP Event Collector (HEC) token acknowledgments to avoid data loss. Without HEC token acknowledgment, data loss may occur, especially in the case of a system restart or crash.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>splunk.hec.raw</td>
<td>Set to true for Splunk software to ingest data using the HEC/raw endpoint. Default is false, which will use the /event endpoint.</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>splunk.hec.raw.line.breaker</td>
<td>Only applicable to HEC /raw endpoint. The setting is used to specify a custom line breaker to help Splunk separate the events correctly. For example, you can specify &quot;#####&quot; as a special line breaker. Internally, Splunk Connect for Kafka will append this line breaker to every Kafka record to form a clear event boundary. The connector performs data injection in batch mode. On the Splunk platform, configure props.conf to set up line breaker for the source types. Then the Splunk software will break events for data flowing through the HEC /raw endpoint. By default, this setting is empty.</td>
</tr>
<tr>
<td>splunk.hec.json.event.enrichment</td>
<td>Only applicable to the HEC /event endpoint. This setting is used to enrich raw data with extra metadata fields. It contains a list of key value pairs separated by &quot;,&quot;. The configured enrichment metadata will be indexed along with raw event data by Splunk software. Data enrichment for the HEC /event endpoint is only available in Splunk Enterprise 6.5 and later. By default, this setting is empty.</td>
</tr>
<tr>
<td>splunk.hec.track.data</td>
<td>Valid settings are true or false. When set to true, data loss and data injection latency metadata will be indexed along with raw data. This setting only works in conjunction with the HEC /event endpoint (splunk.hec.raw: false). By default, this setting is set to false.</td>
</tr>
</tbody>
</table>
Install

Install Splunk Connect for Kafka

The below steps have been tested on both Apache Kafka and Confluent platform deployments.

To install Splunk Connect for Kafka, perform the following steps:

1. Navigate to the Splunk Connect for Kafka repository on github and download the latest
   splunk-kafka-connect-[VERSION].jar release.
2. Start your Kafka Cluster and confirm it is running.
   curl http://<KAFKA_CONNECT_HOSTNAME>:<KAFKA_CONNECT_PORT>
   For example, curl http://localhost:8083
3. (Optional) Create a directory to store your Kafka Connect connectors. This will be used for your plugin.path
   setting.
4. Navigate to your /$KAFKA_HOME/config/ directory.
5. Modify the connect-distributed.properties file to include the below information:

   #Required configurations for Splunk Connect for Kafka
   bootstrap.servers=<BOOTSTRAP_SERVER1,BOOTSTRAP_SERVER2,BOOTSTRAP_SERVER3 >
   plugin.path=<PLUGIN_PATH>
io.confluent.connect.avro.AvroConverter>
io.confluent.connect.avro.AvroConverter>
6. Place the Splunk Connect for Kafka jar file in the plugin.path directory for all Kafka Connect hosts.
7. Restart your deployment's Kafka Connect services.
8. Run ./bin/connect-distributed.sh config/connect-distributed.properties to start Kafka Connect.
9. If this is a new install, create a test topic (for example, perf), and inject events into the topic using the Kafka
   data-gen-app or the kafka-console-producer.
10. Run the following command to confirm that Splunk Connect of Kafka has been installed and configured correctly.
    curl http://localhost:8083/connector-plugins The command will return a list of the available connectors.
    Splunk Connect for Kafka is available if the returned list of available connectors contains

   $KAFKA_HOME is the home directory where your Kafka Connect deployment is located. This could be the same as
   your Kafka home directory if you are running on a shared system.

Splunk Connect for Kafka commands

Use the following commands to check the status of Splunk Connect for Kafka, to manage connectors, and to manage

tasks:

<table>
<thead>
<tr>
<th>Description</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>List active connectors</td>
<td>curl <a href="http://localhost:8083/connectors">http://localhost:8083/connectors</a></td>
</tr>
<tr>
<td>Get kafka-connect-splunk connector information</td>
<td>curl <a href="http://localhost:8083/connectors/kafka-connect-splunk">http://localhost:8083/connectors/kafka-connect-splunk</a></td>
</tr>
<tr>
<td></td>
<td>curl <a href="http://localhost:8083/connectors/kafka-connect-splunk/config">http://localhost:8083/connectors/kafka-connect-splunk/config</a></td>
</tr>
<tr>
<td>Description</td>
<td>Command</td>
</tr>
<tr>
<td>-----------------------------------------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>Pause kafka-connect-splunk workers</td>
<td></td>
</tr>
</tbody>
</table>

See the Confluent documentation for additional REST examples.

**Upgrade Splunk Kafka Connect**

See the following steps to upgrade to the latest version of Splunk Connect for Kafka.

1. Navigate to the Splunk Connect for Kafka repository on github and download the latest splunk-kafka-connect-[VERSION].jar release.
2. Stop all Splunk Connect for Kafka workers.
3. Remove the old Splunk Connect for Kafka plugin from your deployment's plugin path or classpath.
4. Install the new Splunk Connect for Kafka plugin using the installation instructions in the Install Splunk Connect for Kafka topic in this manual.
5. Start up your deployment's workers.
6. Start up the Splunk Connect for Kafka.

For more information on Kafka commands, see the Splunk Connect for Kafka commands section of the Install and Administer Splunk Connect for Kafka topic in this manual.
Configure

Configure Splunk Connect for Kafka

After you bring Kafka Connect up on every host, the Kafka Connect instances automatically form a cluster. A REST call can be executed against one of the cluster instances, and the configuration will be propagated across your entire cluster.

Create a data collection task in Splunk Connect for Kafka

Use the following steps to create a data collection task using Splunk Connect for Kafka.

1. Start Kafka Connect.
   
   
   ```
   $KAFKA_HOME/bin/connect-distributed.sh config/connect-distributed.properties
   ```
   
   where `$KAFKA_HOME` is the install directory of Kafka on your Kafka Connect host.

2. Run the following command, using your deployment's details, to create connector tasks.

   ```
   curl localhost:8083/connectors -X POST -H "Content-Type: application/json" -d '{
   "name": "kafka-connect-splunk",
   "config": {
   "connector.class": "com.splunk.kafka.connect.SplunkSinkConnector",
   "tasks.max": "3",
   "topics": "<KAFKA_TOPIC>",
   "splunk.indexes": "<SPLUNK_DESTINATION_INDEX>",
   "splunk.hec.uri": "<SPLUNK_HEC_URI:SPLUNK_HEC_PORT>",
   "splunk.hec.token": "<SPLUNK_HEC_TOKEN>",
   "splunk.hec.raw": "true",
   "splunk.hec.ack.enabled": "false"
   }
   }
   '}
   ```

   Required configurations

<table>
<thead>
<tr>
<th>Required configurations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topics</td>
<td>Configures the Kafka topic for ingestion</td>
</tr>
<tr>
<td>splunk.indexes</td>
<td>Sets the destination Splunk platform index</td>
</tr>
<tr>
<td>splunk.hec.uri</td>
<td>Configures the Splunk HTTP Event Collector (HEC) URI</td>
</tr>
<tr>
<td>splunk.hec.token</td>
<td>Adjust to set the Splunk HEC token.</td>
</tr>
<tr>
<td>splunk.hec.ack.enabled</td>
<td>Verify that the deployment's indexer acknowledgment configurations used in the REST call match those defined for the target HTTP Event Collector (HEC) token.</td>
</tr>
</tbody>
</table>

3. (Optional) Enable verbose logging. Enable verbose logging for better visibility into errors and information.

   1. Navigate to `config/connect-log4j.properties`, and add the following information:

   ```
   log4j.appender.stdout.layout.ConversionPattern= [%d] %p %X{connector.context} %m (%c:%L) %n
   ```

   2. Save your changes.

4. Verify that data is flowing into your Splunk platform instance by searching for the index from your configurations.

For information on advanced parameters, see the Parameters topic.

For more information on using the Splunk HTTP Event Collector (HEC), see the HEC documentation.
Configuration schema structure reference

Use the following schema as a reference for configuring Splunk Connect for Kafka to send data to your Splunk platform deployment.

```
{
  "name": "<connector-name>",
  "config": {
    "connector.class": "com.splunk.kafka.connect.SplunkSinkConnector",
    "tasks.max": "<number-of-tasks>",
    "topics": "<list-of-topics-separated-by-comma>",
    "splunk.indexes": "<list-of-indexes-for-topics-data-separated-by-comma>",
    "splunk.sources": "<list-of-sources-for-topics-data-separated-by-comma>",
    "splunk.sourcetypes": "<list-of-sourcetypes-for-topics-data-separated-by-comma>",
    "splunk.hec.uri": "<Splunk-HEC-URI>",
    "splunk.hec.token": "<Splunk-HEC-Token>",
    "splunk.hec.raw": "<true|false>",
    "splunk.hec.raw.line.breaker": "<line breaker separator>",
    "splunk.hec.json.event.enrichment": "<key value pairs separated by comma, only applicable to /event HEC>",
    "splunk.hec.ack.enabled": "<true|false>",
    "splunk.hec.ack.poll.interval": "<event ack poll interval>",
    "splunk.hec.ack.poll.threads": "<number of threads used to poll event acks>",
    "splunk.hec.ssl.validate.certs": "<true|false>",
    "splunk.hec.http.keepalive": "<true|false>",
    "splunk.hec.max.http.connection.per.channel": "<max number of http connections per channel>",
    "splunk.hec.total.channels": "<total number of channels>",
    "splunk.hec.max.batch.size": "<max number of kafka records post in one batch>",
    "splunk.hec.threads": "<number of threads to use to do HEC post for single task>",
    "splunk.hec.event.timeout": "<timeout in seconds>",
    "splunk.hec.socket.timeout": "<timeout in seconds>",
    "splunk.hec.track.data": "<true|false, tracking data loss and latency, for debugging lagging and data loss>",
    "splunk.header.support": "<true|false>",
    "splunk.header.custom": "<list-of-custom-headers-to-be-used-from-kafka-headers-separated-by-comma>",
    "splunk.header.index": "<header-value-to-be-used-as-splunk-index>",
    "splunk.header.source": "<header-value-to-be-used-as-splunk-source>",
    "splunk.header.sourcetype": "<header-value-to-be-used-as-splunk-sourcetype>",
    "splunk.header.host": "<header-value-to-be-used-as-splunk-host>"
  }
}
```

Collect from the current Kafka topic offsets

To collect from the latest offset in your Kafka topic, use the below steps:

1. Before starting Splunk Connect for Kafka, add the following line of code to the Kafka Connect properties file:

   ```
   consumer.auto.offset.reset=latest
   ```

2. Restart Kafka Connect

Scale your environment

Before scaling the Splunk Connect for Kafka tier, ensure that the bottleneck is in the connector tier and not in another component. Review the following scaling options:

- Increase the number of parallel tasks by adjusting the `tasks.max` parameter. Only do this if the hardware is underutilized, such as low CPU, low memory usage and low data injection throughput. You can reconfigure the
connector with more tasks.

- Increase hardware resources on cluster nodes in case of resource exhaustion, such as high CPU, or high memory usage.
- Increase the number of Kafka Connect nodes.

Do not create more tasks than the number of partitions. Creating 2 * CPU tasks per Splunk Kafka Connector is a safe estimate. For example, assume there are five Kafka Connects running the Splunk Kafka Connector. Each host is 8 CPUs with 16 GB memory. And there are 200 partitions to collect data from. max.tasks will be: max.tasks = 2 * CPUs/host * Kafka Connect instances = 2 * 8 * 5 = 80 tasks. Alternatively, if there are only 60 partitions to consume from, then set max.tasks to 60. Otherwise, the remaining 20 will be pending.

**Determine number of Kafka Connect instances**

Determine the number of Kafka Connect instances needed by estimating how much volume per day Splunk Connect for Kafka needs to index in your Splunk platform. For example, an 8 CPU, 16 GB memory machine can achieve 50 - 60 MB/s throughput from Kafka into your Splunk platform if your Splunk platform deployment is sized correctly.

**Data loss and latency monitoring**

When configuring Splunk Connect for Kafka using the REST API, "splunk.hec.track.data": "true" can be configured to allow data loss tracking and data collection latency monitoring. This is accomplished by enriching the raw data with offset, timestamp, partition, and topic metadata. This setting will only work in conjunction with HEC /event endpoint ("splunk.hec.raw" : "false")

**Data loss tracking**

Splunk Connect for Kafka uses offset to track data loss since offsets in a Kafka topic partition are sequential. If there is a gap in the Splunk software, there is data loss.

**Data latency tracking**

Splunk Connect for Kafka uses the timestamp of the record to track the time elapsed between the time a Kafka record was created and the time the record was indexed in Splunk.

**Data duplication and data loss**

Run the following SPL query to identify data duplication:

```
index=main sourcetype="<sourcetype>" | stats count as TotalCount, max(kafka_offset) as Offset by kafka_partition | eval loss= TotalCount - (Offset+1)
```

Run the following SPL query to identify data loss:

```
index=main sourcetype="<sourcetype>" | dedup kafka_offset kafka_partition |stats count as TotalCount, max(kafka_offset) as Offset by kafka_partition | eval loss= TotalCount - (Offset+1)
```

**Malformed data**

If the raw data of the Kafka records is a JSON object but is not able to be marshaled, or if the raw data is in bytes but it is not UTF-8 encodable, Splunk Connect for Kafka considers these records malformed. It will log the exception with Kafka specific information (topic, partition, offset) for these records within the console, as well as the malformed records information will be indexed in Splunk. Users can search
within the Splunk software to return any malformed Kafka records encountered.

Security configurations for Splunk Connect for Kafka

Splunk Connect for Kafka supports the following security processes:

- Secure Socket Layer (SSL)
- SASL/Generic Security Service Application Program Interface (GSSAPI/GSS-API) (Kerberos)
- Simple Authentication and Security Layer (SASL)/PLAIN
- SASL/Salted Challenge Response Authentication Mechanism (SCRAM)-SHA-256
- SASL/SCRAM-SHA-512

Configure SSL security for Splunk Connect for Kafka

Use the following information to configure SSL security for Splunk Connect for Kafka.

Configure a certificate for Kafka connector with Splunk

1. Create a cert directory
   mkdir ~/cert
   cd ~/cert

2. Generate a Self-Signed Certificate
   openssl req -newkey rsa:2048 -nodes -keyout kafka_connect.key -x509 -days 365 -out kafka_connect.crt

3. Generate a certificate .pem file from .crt. The Splunk HTTP Event Collector (HEC) requires .pem format.
   openssl x509 -in kafka_connect.crt -out kafka_connect.pem -outform PEM

4. Generate a new keystore. You will need to create a password when generating the new keystore.
   keytool -genkeypair -keyalg RSA -keystore keystore.jks

5. Import Signed/Root/Intermediate Certificate
   keytool -importcert -trustcacerts -file kafka_connect.crt -alias localhost -keystore keystore.jks

6. Configure HEC using your certificate.
   1. Copy the certificate and key to Splunk
      cp ~/cert/kafka_connect.key ~/splunk/etc/auth
      cp ~/cert/kafka_connect.pem ~/splunk/etc/auth
   2. In $SPLUNK_HOME, navigate to ~/etc/apps/splunk_httpinput/local/.
   3. Open inputs.conf with a text editor and add the following lines under the [http] stanza:
      [http]
      disabled = 0
      enableSSL = 1
      serverCert = ~/splunk/etc/auth/kafka_connect.pem <absolute path to your certificate>
      privKeyPath = ~/splunk/etc/auth/kafka_connect.key
      sslPassword = <your password of certificate private key>
   4. Restart your Splunk platform instance.
      cd ~/splunk/bin
      ./splunk restart
Configure a certificate authority for your Kafka broker and the Splunk Connect for Kafka

1. Generate your own certificate authority (CA) certificate, and add the same CA certificate to each client and broker's truststore. The following bash script generates the keystore and truststore for brokers (kafka.server.keystore.jks and kafka.server.truststore.jks) and clients (kafka.client.keystore.jks and kafka.client.truststore.jks):

```bash
#!/bin/bash

PASSWORD=test1234
VALIDITY=365

keytool -keystore kafka.server.keystore.jks -alias localhost -validity $VALIDITY -genkey
openssl req -new -x509 -keyout ca-key -out ca-cert -days $VALIDITY
keytool -keystore kafka.server.truststore.jks -alias CARoot -import -file ca-cert
keytool -keystore kafka.client.truststore.jks -alias CARoot -import -file ca-cert
keytool -keystore kafka.server.keystore.jks -alias localhost -certreq -file cert-file
openssl x509 -req -CA ca-cert -CAkey ca-key -in cert-file -out cert-signed -days $VALIDITY
-CAcreateserial -passin pass:$PASSWORD
keytool -keystore kafka.server.keystore.jks -alias CARoot -import -file ca-cert
keytool -keystore kafka.server.keystore.jks -alias localhost -import -file cert-signed
keytool -keystore kafka.client.keystore.jks -alias localhost -validity $VALIDITY -genkey
keytool -keystore kafka.client.keystore.jks -alias localhost -certreq -file cert-file
openssl x509 -req -CA ca-cert -CAkey ca-key -in cert-file -out cert-signed -days $VALIDITY
-CAcreateserial -passin pass:$PASSWORD
keytool -keystore kafka.client.keystore.jks -alias CARoot -import -file ca-cert
keytool -keystore kafka.client.keystore.jks -alias localhost -import -file cert-signed
```

2. Configure Kafka brokers.

1. Navigate to config/server.properties, and add the following lines:

   ```properties
   listeners=SSL://localhost:9092
   security.inter.broker.protocol=SSL
   ssl.enabled.protocols=TLSv1.2,TLSv1.1,TLSv1
   ssl.client.auth=none
   ssl.keystore.type = JKS
   ssl.keystore.location=~/cert/kafka.server.keystore.jks
   ssl.keystore.password=test1234
   ssl.key.password=test1234
   ssl.truststore.type = JKS
   ssl.truststore.location=~/cert/kafka.server.truststore.jks
   ssl.truststore.password=test1234
   ```

2. Save your changes.

3. Configure Kafka Connect

1. Navigate to config/connect-distributed.properties, and add the following lines:

   ```properties
   bootstrap.servers=https://localhost:9092
   security.protocol=SSL
   ssl.key.password=test1234
   ssl.keystore.location=/cert/kafka.client.keystore.jks
   ssl.keystore.password=test1234
   ssl.truststore.location=/cert/kafka.client.truststore.jks
   ssl.truststore.password=test1234
   ssl.enabled.protocols=TLSv1.2,TLSv1.1
   ssl.truststore.type=JKS
   # Authentication settings for Connect consumers used with sink connectors
   consumer.security.protocol=SSL
   consumer.ssl.key.password=test1234
   ```
consumer.ssl.keystore.location=~/cert/kafka.client.keystore.jks
consumer.ssl.keystore.password=test1234
consumer.ssl.truststore.location=~/cert/kafka.client.truststore.jks
consumer.ssl.truststore.password=test1234
consumer.ssl.enabled.protocols=TLSv1.2,TLSv1.1
c consumer.ssl.truststore.type=JKS

2. Save your changes.

4. Start your Kafka server and Kafka Connect
   cd $KAFKA_HOME
   bin/zookeeper-server-start.sh config/zookeeper.properties
   bin/kafka-server-start.sh config/server.properties
   ./bin/connect-distributed.sh config/connect-distributed.properties

5. Create a Kafka topic
   1. Create a new properties file named client.properties. This file is referenced when you use the command line tools to open a connection. Configure it to use your key store and trust store JKS files.
      security.protocol=SSL
      ssl.keystore.location=~/cert/kafka.client.keystore.jks
      ssl.keystore.password=test1234 <keystore password>
      ssl.key.password=test1234 <pkcs12 password>
      ssl.truststore.location=~/cert/kafka.client.truststore.jks
      ssl.truststore.password=test1234 <truststore password>
   2. Change the access permissions for your file system, in order to hide passwords.
      chmod 0600 client.properties
   3. Use your client.properties file to make connections to a broker from the Kafka command line tools.
      $ bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 -topic mytopic
         --consumer.config client.properties
      $ bin/kafka-console-producer.sh --broker-list localhost:9092 -topic mytopic --producer.config client.properties
   6. Run the following command to create connector tasks. Use the following table as reference to adjust the command to fit your deployment.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topics</td>
<td>Configure the Kafka topic to be ingested.</td>
</tr>
<tr>
<td>splunk.indexes</td>
<td>Set the destination Splunk indexes.</td>
</tr>
<tr>
<td>splunk.hec.token</td>
<td>Set your Http Event Collector (HEC) token.</td>
</tr>
<tr>
<td>splunk.hec.uri</td>
<td>URI for your destination Splunk HEC endpoint.</td>
</tr>
</tbody>
</table>

   curl localhost:8083/connectors -X POST -H "Content-Type: application/json" -d '{ "name": "ssl_validate_certs_true",
   "config": {
      "connector.class": "com.splunk.kafka.connect.SplunkSinkConnector",
      "tasks.max": "1",
      "topics": "<YOUR_TOPIC>",
      "splunk.indexes": "<SPLUNK_INDEXES>", // mytopic
      "splunk.hec.uri": "<SPLUNK_HEC_URI:SPLUNK_HEC_PORT>",
      "splunk.hec.token": "<YOUR_TOKEN>",
      "splunk.hec.ssl.trust.store.path": "<KEYSTORE_LOCATION>", // ~/cert/keystore.jks
      "splunk.hec.ssl.trust.store.password": "<PASSWORD_KEYSTORE>",
      "splunk.hec.ack.enabled": "false",
      "splunk.hec.ssl.validate.certs": "true"
   }'
Configure workers and SinkTasks to work with your SSL secured cluster

1. Navigate to `$KAFKA_HOME/config/connect-distributed.properties` to configure the Kafka Connect worker and consumer settings to use SSL.

   ```
   # Worker security are located at the top level
   security.protocol=SSL
   ssl.truststore.location=/var/private/ssl/kafka.client.truststore.jks
   ssl.truststore.password=test1234
   
   # Sink security settings are prefixed with "consumer."
   consumer.security.protocol=SSL
   consumer.ssl.truststore.location=/var/private/ssl/kafka.client.truststore.jks
   consumer.ssl.truststore.password=test1234
   ```

2. Adjust the settings `consumer.ssl.truststore.location` and `ssl.truststore.password` to reflect your setup.

There is currently no way to change the configuration for connectors individually, but if your server supports client authentication over SSL, use a separate principal for the worker and the connectors. See Confluent’s documentation on configuring workers and connectors with security for more information.

3. Start Kafka Connect.

   ```
   ./bin/connect-distributed.sh config/connect-distributed-quickstart.properties
   ```

SASL/GSSAPI (Kerberos)

Configure Kafka Connect when your Kafka cluster is secured using Kerberos.

1. Configure the Kafka Connect worker and consumer settings to use Kerberos in `$KAFKA_HOME/config/connect-distributed.properties`.

   ```
   # Worker security are located at the top level
   security.protocol=SASL_PLAINTEXT
   sasl.mechanism=GSSAPI
   
   # Sink security settings are prefixed with "consumer."
   consumer.sasl.mechanism=GSSAPI
   consumer.security.protocol=SASL_PLAINTEXT
   sasl.kerberos.service.name=kafka
   ```

2. Modify `bin/connect-distributed.sh` by editing the `EXTRA_ARGS` environment variable.

3. Pass in the location of the JAAS conf file. Optionally, you can specify the path to your Kerberos configuration file and set Kerberos debugging to `true` for troubleshooting connection issues.

   ```
   EXTRA_ARGS=
   ```

For example, a Kafka Client JAAS file using the principal connect:

```java
KafkaClient {
    com.sun.security.auth.module.Krb5LoginModule required
    useKeyTab=true
    storeKey=true
    keyTab="/etc/security/keytabs/connect.keytab"
    principal="connect/_HOST@REALM";
};
```

Modify the keytab and principal settings to reflect your environment.
4. Start Kafka Connect.

```
./bin/connect-distributed.sh config/connect-distributed.properties
```

See Confluent's documentation for more information on configuring Kafka Connect using JAAS.

**SASL/PLAIN**

Do not run SASL/PLAIN in production without SSL.

Configure Kafka Connect worker and consumer settings to use SASL/PLAIN:

1. **Configure the Kafka Connect worker and consumer settings to use SASL/PLAIN in**
   
   
   $KAFKA_HOME/config/connect-distributed.properties.
   
   ```
   # Worker security are located at the top level
   security.protocol=SASL_SSL
   sasl.mechanism=PLAIN
   
   # Sink security settings are prefixed with "consumer."
   consumer.security.protocol=SASL_SSL
   consumer.sasl.mechanism=PLAIN
   ```

2. **Navigate to** $KAFKA_HOME/config/connect-distributed.properties **and edit the EXTRA_ARGS environment variable.**

```
EXTRA_ARGS=${EXTRA_ARGS-'-name connectDistributed
-Djava.security.auth.login.config=/root/kafka_connect_jaas.conf'}
```

For example, a Kafka Client JAAS file for SASL/PLAIN.

```java

KafkaClient {
    org.apache.kafka.common.security.plain.PlainLoginModule required
    username="alice"
    password="alice-secret";
};
```

3. **Pass in the location of the JAAS conf file.**

```
EXTRA_ARGS=${EXTRA_ARGS-'-name connectDistributed
-Djava.security.auth.login.config=/root/kafka_connect_jaas.conf'}
```

For example, a Kafka Client JAAS file for SASL/PLAIN.

4. **Start Kafka Connect.**

```
./bin/connect-distributed.sh config/connect-distributed.properties
```

See Confluent's documentation for more information on configuring Kafka Connect using SASL/PLAIN.

**SASL/SCRAM-SHA-256 and SASL/SCRAM-SHA-512**

Configure the Kafka Connect worker and consumer settings to use SASL/SCRAM:

1. **Navigate to** $KAFKA_HOME/config/connect-distributed.properties **and make the following adjustments:**

```
# Worker security are located at the top level
security.protocol=SASL_SSL
sasl.mechanism=SCRAM-SHA-256 (or SCRAM-SHA-512)

# Sink security settings are prefixed with "consumer."
consumer.security.protocol=SASL_SSL
consumer.sasl.mechanism=SCRAM-SHA-256 (or SCRAM-SHA-512)
```
2. Modify `bin/connect-distributed.sh` by editing the `EXTRA_ARGS` environment variable. Pass in the location of the JAAS configuration file.

```
EXTRA_ARGS=${EXTRA_ARGS-"-name connectDistributed -Djava.security.auth.login.config=/root/kafka_connect_jaas.conf"}
```

For example, a Kafka Client JAAS file for SASL/SCRAM:

```
KafkaClient {
    org.apache.kafka.common.security.scram.ScramLoginModule required
    username="alice"
    password="alice-secret";
}
```

3. Start Kafka Connect

```
./bin/connect-distributed.sh config/connect-distributed.properties
```

Workers and SinkTasks now work with your SASL/SCRAM secured cluster. See Confluent's documentation for more information on configuring Kafka Connect using JAAS.

### Load balancing configurations for Splunk Connect for Kafka

Splunk Connect for Kafka supports the following types of load balancing configurations:

- A comma-separated list of HTTP Event Collectors (HEC) endpoints for internal load balancing.
- A preconfigured hardware or software load balancer forwarding requests to HEC endpoints.

Include a load balancer in front of the HECs that are enabled on your Splunk platform indexer cluster or collection of Splunk platform heavy forwarders. Load balancing is supported when HEC token acknowledgment is enabled.

If you are using Splunk Cloud, submit a case on the Splunk Support Portal to create or modify a load balancer for use with Splunk Connect for Kafka. If you are ingesting Kinesis Firehose events, you can reuse the Elastic Load Balancing (ELB) URL that you configured for the Splunk Add-on for Amazon Kinesis Firehose.

If your deployment is configured with load balancing, and HEC acknowledgment is enabled (splunk.hec.ack.enabled:true), follow these steps to ensure data ingests correctly.

1. Enable **sticky sessions** on your load balancer with the maximum configurable cookie timeout period.
2. Set HEC channels (splunk.hec.total.channels) to multiple HEC endpoints (for example, multiple indexers or 2 * indexers behind your load balancer). This will ensure the data flow is evenly balanced across the Splunk platform indexers.
3. Review indexer polling configurations.
   - The parameter `splunk.hec.lb.poll.interval` controls the load balancer polling interval. By default, it is set to 120 seconds (2 minutes). Users can specify this parameter, in seconds, to control the polling interval. Increase this parameter to do less polling, or decrease it to perform more frequent polling.
4. Save your changes.

Data duplication might occur even with sticky sessions when requests are offloaded to a different endpoint under load.

See the [Configuration examples](#) page to see examples of load balancing with a list of HEC enabled endpoints, and load balancing with a preconfigured load balancer.
Index routing configurations for Splunk Connect for Kafka

Index routing is an optional Splunk Connect for Kafka configuration that can be done in either your Splunk software or your Kafka deployment. If you want to route indexes based on your Kafka topics, configure index routing in your Kafka deployment. If you want to route indexes based on the data stream, configure index routing in your Splunk platform deployment.

Configure index routing in your Kafka deployment

Use the following formatting to configure index routing when creating Splunk Connect for Kafka tasks:

topics
splunk.indexes

topics are the Kafka topics where data is stored. You can specify multiple topics in the same SinkTask. For each topic, you can also specify a Splunk platform index, source, and source type.

In this case, one data stream of a particular topic will be delivered to the specified index. If only one index is specified, data stream from all the topics will be delivered to the specified index.

In this example, there are three topics (test-1, test-2, test-3) delivering all data to one Kafka index.

"topics": "test-1,test-2,test-3",
"splunk.indexes": "kafka"

To deliver the data to kafka-1, kafka-2, kafka-3 indexes, respectively, add each index to the task configuration.

"topics": "test-1,test-2,test-3",
"splunk.indexes": "kafka-1,kafka-2,kafka-3"

Depending on your deployment, there could be another layer of index configuration on the Splunk platform through the use of HEC configurations. If your deployment’s HTTP Event Collector (HEC) configurations are set to not overwrite any indexers, the rules configured by Splunk Connect for Kafka will be followed. Otherwise, your HEC will overwrite the indexes specified by the task.

Configure index routing in your Splunk platform deployment

On the Splunk platform of your Kafka connector deployment, logs can be routed using any indexed fields. Configure the Splunk configuration files props.conf and transforms.conf on your Splunk platform indexers or Splunk platform heavy forwarders.

The following examples uses files and fields relevant to an AWS CloudWatch event.

{
  "owner": "123456789012",
  "logGroup": "CloudTrail",
  "logStream": "123456789012_CloudTrail_us-east-1",
  "subscriptionFilters": ["Destination"],
  "messageType": "DATA_MESSAGE",
  "logEvents": {
}
Example 1: Route owner with ID 123456789012 to a Splunk production index

1. Navigate to $SPLUNK_HOME/etc/system/local/, and create a props.conf file and a transforms.conf file.
2. Update the props.conf with this configuration:
   ```
   [kafka:events]
   TRANSFORMS-index_routing = route_data_to_index_by_field_owner_id
   ```
3. Update the $SPLUNK_HOME/etc/system/local/transforms.conf file with this configuration:
   ```
   [route_data_to_index_by_field_owner_id]
   REGEX = "\w+":"123456789012"
   DEST_KEY = _MetaData:Index
   FORMAT = prod
   ```
4. Save your changes.

Example 2: Route AWS CloudWatch logs from a certain region to an index dedicated to that region

1. Navigate to $SPLUNK_HOME/etc/system/local/, and create a props.conf file.
2. Update the props.conf with this configuration:
   ```
   [kafka:events]
   TRANSFORMS-index_routing = route_data_to_index_by_aws_region
   ```
3. Update the $SPLUNK_HOME/etc/system/local/transforms.conf file with this configuration:
   ```
   [route_data_to_index_by_aws_region]
   REGEX = "logStream":"(.us-east-1)"
   DEST_KEY = _MetaData:Index
   FORMAT = aws-cloudwatch-us-east-1
   ```
4. Save your changes.

If your Splunk platform deployment has index clustering set up, make sure your props.conf and transforms.conf files are in sync on each indexer.

See the Managing Indexers and Clusters of Indexers manual to learn more about managing and updating indexer cluster configurations.

Configuration examples for Splunk Connect for Kafka

Depending on your deployment, use the following configuration examples to configure your Splunk Connect for Kafka deployment.

Enable HEC token acknowledgements to avoid data loss. Without HEC token acknowledgement, data loss may occur, especially in case of a system restart or crash.

If raw events need to go through Splunk's index time extraction, use the HEC /raw event endpoint. When using the /raw endpoint,
HEC endpoint and when your raw data does not contain a timestamp or contains multiple timestamps or carriage returns, you must configure the `splunk.hec.raw.line.breaker` and setup a corresponding `props.conf` inside your Splunk platform to honor this line breaker setting. This will assist Splunk to do event breaking. For example, in **Connection configuration**, set "splunk.hec.raw.line.breaker":"####" for sourcetype "s1".

In `props.conf`, you can set up the line breaker as follows:

```
[s1] # sourcetype name
LINE_BREAKER = (####)
SHOULD_LINEMERGE = false
```

The auto-assigned timestamp will work for all deployments that use the /event HEC endpoint.

### Splunk indexing with acknowledgment

#### Using HEC /raw endpoint

```
curl <hostname>:8083/connectors -X POST -H "Content-Type: application/json" -d'{
    "name": "splunk-prod-financial",
    "config": {
        "connector.class": "com.splunk.kafka.connect.SplunkSinkConnector",
        "tasks.max": "10",
        "topics": "t1,t2,t3,t4,t5,t6,t7,t8,t9,t10",
        "splunk.hec.token": "1B901D2B-576D-40CD-AF1E-98141B499534",
        "splunk.hec.ack.enabled": "true",
        "splunk.hec.ack.poll.interval": "20",
        "splunk.hec.ack.poll.threads": "2",
        "splunk.hec.event.timeout": "300",
        "splunk.hec.raw": "true",
        "splunk.hec.raw.line.breaker": "####"
    }
}
```

#### Using HEC /event endpoint

```
curl <hostname>:8083/connectors -X POST -H "Content-Type: application/json" -d'{
    "name": "splunk-prod-financial",
    "config": {
        "connector.class": "com.splunk.kafka.connect.SplunkSinkConnector",
        "tasks.max": "10",
        "topics": "t1,t2,t3,t4,t5,t6,t7,t8,t9,t10",
        "splunk.hec.token": "1B901D2B-576D-40CD-AF1E-98141B499534",
        "splunk.hec.ack.enabled": "true",
        "splunk.hec.ack.poll.interval": "20",
        "splunk.hec.ack.poll.threads": "2",
        "splunk.hec.event.timeout": "300",
        "splunk.hec.raw": "false",
        "splunk.hec.json.event.enrichment": "org=fin,bu=south-east-us",
        "splunk.hec.track.data": "true"
    }
}
```

20
Splunk indexing without acknowledgment

Using HEC /raw endpoint

curl <hostname>:8083/connectors -X POST -H "Content-Type: application/json" -d'{
  "name": "splunk-prod-financial",
  "config": {
    "connector.class": "com.splunk.kafka.connect.SplunkSinkConnector",
    "tasks.max": "10",
    "topics": "t1,t2,t3,t4,t5,t6,t7,t8,t9,t10",
    "splunk.hec.token": "1B901D2B-576D-40CD-AF1E-98141B499534",
    "splunk.hec.ack.enabled": "false",
    "splunk.hec.raw": "true",
    "splunk.hec.raw.line.breaker": "####"
  }
}

Using HEC /event endpoint

curl <hostname>:8083/connectors -X POST -H "Content-Type: application/json" -d'{
  "name": "splunk-prod-financial",
  "config": {
    "connector.class": "com.splunk.kafka.connect.SplunkSinkConnector",
    "tasks.max": "10",
    "topics": "t1,t2,t3,t4,t5,t6,t7,t8,t9,t10",
    "splunk.hec.token": "1B901D2B-576D-40CD-AF1E-98141B499534",
    "splunk.hec.ack.enabled": "false",
    "splunk.hec.raw": "false",
    "splunk.hec.json.event.enrichment": "org=fin,bu=south-east-us",
    "splunk.hec.track.data": "true"
  }
}

Example of a connector with header support enabled

curl <hostname>:8083/connectors -X POST -H "Content-Type: application/json" -d'{
  "name": "splunk-prod-financial",
  "config": {
    "connector.class": "com.splunk.kafka.connect.SplunkSinkConnector",
    "tasks.max": "10",
    "topics": "t1,t2,t3,t4,t5,t6,t7,t8,t9,t10",
    "splunk.sourcetypes": "collectd_http",
    "splunk.hec.token": "1B901D2B-576D-40CD-AF1E-98141B499534",
    "splunk.hec.ack.enabled": "true",
    "splunk.hec.ack.poll.interval": "20",
    "splunk.hec.ack.poll.threads": "2",
    "splunk.hec.event.timeout": "120",
    "splunk.hec.raw": "false",
    "splunk.header.support": "true",
    "splunk.header.index": "destination_storage",
    "splunk.header.source": "Financial_Application",
    "splunk.header.sourcetype": "ledger_format",
    "splunk.header.host": "finance.company.host"
  }
}'}
Example of a connector for custom Java keystore location

curl <hostname>:8083/connectors -X POST -H "Content-Type: application/json" -d' {
  "name": "splunk-prod-financial",
  "config": {
    "connector.class": "com.splunk.kafka.connect.SplunkSinkConnector",
    "tasks.max": "20",
    "topics": "t1",
    "splunk.hec.token": "1B901D2B-576D-40CD-AF1E-98141B499534",
    "splunk.sink.ssl.trust.store.path": "/keystore.jks",
    "splunk.sink.ssl.trust.store.password": "password"
  }
}'

Example of a connector for events already in HEC format

curl <hostname>:8083/connectors -X POST -H "Content-Type: application/json" -d' {
  "name": "splunk-prod-financial",
  "config": {
    "connector.class": "com.splunk.kafka.connect.SplunkSinkConnector",
    "tasks.max": "20",
    "topics": "t1",
    "splunk.hec.token": "1B901D2B-576D-40CD-AF1E-98141B499534",
    "splunk.hec.json.event.formatted": "true"
  }
}'

Example of a connector to send collectd metrics to a Splunk metrics index

The Splunk metrics index is optimized for ingesting and retrieving metrics. For more information, see the Metrics manual.

curl <hostname>:8083/connectors -X POST -H "Content-Type: application/json" -d'{
  "name": "splunk-prod-financial",
  "config": {
    "connector.class": "com.splunk.kafka.connect.SplunkSinkConnector",
    "tasks.max": "10",
    "topics": "t1,t2,t3,t4,t5,t6,t7,t8,t9,t10",
    "splunk.sink.sourcetypes": "collectd_http",
    "splunk.hec.token": "1B901D2B-576D-40CD-AF1E-98141B499534",
    "splunk.hec.ack.enabled": "true",
    "splunk.hec.poll.interval": "20",
    "splunk.hec.poll.threads": "2",
    "splunk.hec.event.timeout": "120",
    "splunk.hec.raw": "true",
    "splunk.hec.raw.line.breaker": "####"
  }
}'

Example of a connector with 10 topics and 10 parallelized tasks

Use the following command to create a connector called splunk-prod-financial for 10 topics and 10 parallelized tasks. The connector will use the HEC /event endpoint with acknowledgments enabled. The data is injected into a three-server Splunk platform indexer cluster.
curl <hostname>:8083/connectors -X POST -H "Content-Type: application/json" -d '{
  "name": "splunk-prod-financial",
  "config": {
    "connector.class": "com.splunk.kafka.connect.SplunkSinkConnector",
    "tasks.max": "10",
    "topics": "t1,t2,t3,t4,t5,t6,t7,t8,t9,t10",
    "splunk.hec.token": "1B901D2B-576D-40CD-AF1E-98141B499534"
  }
}

**Example of a connector with 20 parallelized tasks**

Use the following command to update the connector to use 20 parallelized tasks.

curl <hostname>:8083/connectors -X POST -H "Content-Type: application/json" -d '{
  "name": "splunk-prod-financial",
  "config": {
    "connector.class": "com.splunk.kafka.connect.SplunkSinkConnector",
    "tasks.max": "20",
    "topics": "t1,t2,t3,t4,t5,t6,t7,t8,t9,t10",
    "splunk.hec.token": "1B901D2B-576D-40CD-AF1E-98141B499534"
  }
}

**Example of load balancing with list of HEC enabled endpoints**

curl <KAFKA_CONNECT_HOST>:8083/connectors -X POST -H "Content-Type: application/json" -d '{
  "name": "splunk-prod-financial",
  "config": {
    "connector.class": "com.splunk.kafka.connect.SplunkSinkConnector",
    "tasks.max": "1",
    "topics": "t1",
    "splunk.hec.token": "1B901D2B-576D-40CD-AF1E-98141B499534",
    "splunk.hec.ack.enabled": "true",
    "splunk.hec.raw": "true",
    "splunk.hec.raw.line.breaker": "####"
  }
}

**Example of load balancing with a preconfigured load balancer**

curl <KAFKA_CONNECT_HOST>:8083/connectors -X POST -H "Content-Type: application/json" -d '{
  "name": "splunk-prod-financial",
  "config": {
    "connector.class": "com.splunk.kafka.connect.SplunkSinkConnector",
    "tasks.max": "1",
    "topics": "t1",
    "splunk.hec.uri": "https://elb-kafka:8088",
    "splunk.hec.token": "1B901D2B-576D-40CD-AF1E-98141B499534",
    "splunk.hec.ack.enabled": "true",
    "splunk.hec.raw": "true",
    "splunk.hec.raw.line.breaker": "####"
  }
}'
Troubleshooting

Troubleshoot issues with Splunk Connect for Kafka

Use the below examples to diagnose troubleshooting issues with Splunk Connect for Kafka.

No events are arriving in Splunk

If no events arrive in your Splunk platform deployment, perform the following steps:

1. Navigate to your HTTP Event Collector (HEC) token configurations.
2. Verify the indexer acknowledgment configurations used in the REST call (splunk.hec.ack.enabled) match the configurations defined for the target HEC token.
3. Make sure that a valid value for the index is provided in the REST call (splunk.indexes) if there is no default index connected to the target HEC token.
4. Save any changes.

Enable verbose logging

If you need to enable more verbose logging for Splunk Connect for Kafka, perform the following steps:

1. On your Kafka deployment, navigate to the config/connect-log4j.properties file.
2. Append the log4j.logger.com.splunk line to log4j.logger.com.splunk=DEBUG.
3. Save your changes.

Can’t see any connector information on third party UI

If Splunk Connect for Kafka is not showing on Confluent Control Center, perform the following steps:

1. Enable cross-origin access for Kafka Connect.
2. Depending on your deployment, navigate to connect-distributed.properties or connect-distributed-quickstart.properties.
3. Append the following two lines to connect the configuration:
   access.control.allow.origin=*  
   access.control.allow.methods=GET,OPTIONS,HEAD,POST,PUT,DELETE
4. Restart Kafka Connect.

Malformed data

If the raw data of the Kafka records is a JSON object but it is not marshaled, or if the raw data is in bytes, but is not UTF-8 encodable, Splunk Connect for Kafka considers these records malformed. It logs the exception with Kafka-specific information for these records within the console, and the malformed records are indexed in Splunk. You can search "type=malformed" within your Splunk platform deployment to return any malformed Kafka records.

Performance decline over time

If events are processed at a normal rate, but after approximately 10 minutes or more, the rate suddenly drops, leading to a decrease in performance. Check the logs to see if the tasks are re-balanced and the following error populates.
ERROR WorkerSinkTask{id=testtest-1} Commit of offsets threw an unexpected exception for sequence number 50:
{sharon-test-2-0=OffsetAndMetadata{offset=436090, metadata=''},
 username-test-2-8=OffsetAndMetadata{offset=436280, metadata=''},
 username-test-2-7=OffsetAndMetadata{offset=435938, metadata=''},
 username-test-2-6=OffsetAndMetadata{offset=436119, metadata=''},
 username-test-2-5=OffsetAndMetadata{offset=435440, metadata=''},
 username-test-2-4=OffsetAndMetadata{offset=436940, metadata=''},
 username-test-2-3=OffsetAndMetadata{offset=435703, metadata=''},
 username-test-2-2=OffsetAndMetadata{offset=436149, metadata=''},
 username-test-2-1=OffsetAndMetadata{offset=435978, metadata=''}
}
(org.apache.kafka.connect.runtime.WorkerSinkTask:215)
org.apache.kafka.clients.consumer.CommitFailedException: Commit cannot be completed since the group has already rebalanced and assigned the partitions to another member.

This happens when event batches time out and Kafka identifies that task as down and triggers a re-balance on the remaining tasks. Meanwhile, Kafka connect is committing the acknowledgments that it receives from your Splunk platform deployment. When the acknowledgments are committed and got back from Kafka, Splunk Connect for Kafka is already rebalanced.

You can address this issue by:

1. Increasing the session timeout for HEC events.
2. Reducing the maximum size of batches returned in poll() with max.poll.records.
3. Checking your Splunk platform license. When your daily license volume overfills, your deployment's indexing speed slows.

**Duplicate data at the start of data collection**

If you see duplicate data at the start of data collection in a new connector deployment when no offsets are saved, or if you have added data to a topic and started a connector with a number of tasks that are greater than the number of partitions, ensure the number of tasks configured does not exceed the number of partitions.

**Acknowledgments are unsuccessful**

If Splunk Connect for Kafka polls the acknowledgments for the last few batches in your logs, but never successfully polls the acknowledgements, you might see the following error:

```
[2018-01-26 20:24:25,799] DEBUG ackPollResponse="acks":{"0":false}
(com.splunk.hecclient.HecAckPoller:249)
[2018-01-26 20:24:25,799] DEBUG ackPollResponse="acks":{"0":false}
(com.splunk.hecclient.HecAckPoller:249)
(com.splunk.hecclient.HecAckPoller:263)
(com.splunk.hecclient.HecAckPoller:263)
[2018-01-26 20:24:25,808] INFO start polling 2 outstanding acks for 2 channels
(com.splunk.hecclient.HecAckPoller:188)
[2018-01-26 20:24:25,808] WARN timed out event batch after 60 seconds not acked
(com.splunk.hecclient.EventBatch:66)
[2018-01-26 20:24:25,808] WARN timed out event batch after 60 seconds not acked
(com.splunk.hecclient.EventBatch:66)
[2018-01-26 20:24:25,808] WARN detected 2 event batches timedout
(com.splunk.hecclient.HecAckPoller:208)
```

To fix this acknowledgment issue, increase the splunk.hec.event.timeout.
This might happen for the last few event batches in Kafka. The Splunk platform buffers events into batches to index. The last few batches of events may not fill the buffer on the Splunk platform, so the events stay in the buffer until they time out. If Splunk Connect for Kafka is configured to have a HEC event timeout smaller than two minutes, Splunk Connect for Kafka will timeout the events before the events are indexed.

**Splunk Connect for Kafka tasks fail due to serialization error**

If you encounter a serialization error, update your worker properties (connect-distributed.properties) file to make sure the following settings are correctly configured:

```properties
```

For StringConverter and JsonConverter only:

- key.converter.schemas.enable=false
- value.converter.schemas.enable=false

For AvroConverter only:

- key.converter.schema.registry.url=<Location of Avro schema registry>
- value.converter.schema.registry.url=<Location of Avro schema registry>

The error may look like this:

```java
org.apache.kafka.connect.errors.DataException: Converting byte[] to Kafka Connect data failed due to serialization error:
??at org.apache.kafka.connect.json.JsonConverter.toConnectData(JsonConverter.java:304)
??at org.apache.kafka.connect.runtime.WorkerSinkTask.convertMessages(WorkerSinkTask.java:425)
??at org.apache.kafka.connect.runtime.WorkerSinkTask.poll(WorkerSinkTask.java:264)
??at org.apache.kafka.connect.runtime.WorkerSinkTask.iteration(WorkerSinkTask.java:182)
??at org.apache.kafka.connect.runtime.WorkerSinkTask.execute(WorkerSinkTask.java:150)
??at org.apache.kafka.connect.runtime.WorkerTask.doRun(WorkerTask.java:146)
??at org.apache.kafka.connect.runtime.WorkerTask.run(WorkerTask.java:190)
??at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511)
??at java.util.concurrent.FutureTask.run(FutureTask.java:266)
??at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
??at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
??at java.lang.Thread.run(Thread.java:748)
```

See the Installation section of this manual to learn more.

**Error: I/O exception**

If you encounter an I/O exception error, test it against one of the following solutions.

- If this happens intermittently, the HEC is too busy to process the requests.
- If you see this error repeatedly, lower the rate of post data. Increase the event batch size to lower the number of requests.
- If no events can be delivered to Splunk, check the connection between Splunk Connect for Kafka and your Splunk HEC endpoint.

The error may look like this:

```java
ERROR encountered io exception (com.splunk.hecclient.Indexer:141)
```

26
java.net.SocketException: Socket closed
  at java.net.SocketInputStream.read(SocketInputStream.java:204)
  at java.net.SocketInputStream.read(SocketInputStream.java:141)
  at sun.security.ssl.InputRecord.readFully(InputRecord.java:465)
  at sun.security.ssl.InputRecord.read(InputRecord.java:503)
  at sun.security.ssl.SSLSocketImpl.readRecord(SSLSocketImpl.java:983)
  at sun.security.ssl.SSLSocketImpl.readDataRecord(SSLSocketImpl.java:940)
  at sun.security.ssl.AppInputStream.read(AppInputStream.java:105)
  at org.apache.http.impl.io.SessionInputBufferImpl.streamRead(SessionInputBufferImpl.java:137)
  at org.apache.http.impl.io.SessionInputBufferImpl.fillBuffer(SessionInputBufferImpl.java:153)
  at org.apache.http.impl.conn.DefaultHttpResponseParser.parseHead(DefaultHttpResponseParser.java:56)
  at org.apache.http.impl.conn.CPoolProxy.receiveResponseHeader(CPoolProxy.java:165)
  at org.apache.http.protocol.HttpRequestExecutor.execute(HttpRequestExecutor.java:125)
  at org.apache.http.impl.execchain.MainClientExec.execute(MainClientExec.java:272)
  at org.apache.http.impl.execchain.RetryExec.execute(RetryExec.java:89)
  at com.splunk.hecclient.Indexer.executeHttpRequest(Indexer.java:138)
  at com.splunk.hecclient.HecPoller$RunAckQuery.run(HecPoller.java:208)
  at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511)
  at java.util.concurrent.FutureTask.run(FutureTask.java:266)
  at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
  at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
  at java.lang.Thread.run(Thread.java:748)

Error: Conflicting operation

If you encounter a conflicting operation error when creating a Splunk Connect for Kafka task using the REST API, there might be another Splunk Connect for Kafka instance running on your deployment and the two are not in sync. You may need to stop one of the running instances.

The error may look like this:

"error_code":409,"message":"Cannot complete request because of a conflicting operation (e.g. worker rebalance)"

Error: Out of memory

If you encounter an "out of memory" error, review the current JVM memory allocated to Kafka Connect by checking the value of environment variable KAFKA_HEAP_OPTS.

Depending on your physical resource, you can increase the memory by updating the environment variable (for example, -Xmx16G -Xms2G) and restarting Kafka Connect.
**Error: Workers require a list of topics**

If you encounter an error that says SinkTasks require a list of topics, the Kafka topic name was not provided. Provide the names of the topics as part of the worker configuration.

The error may look like this:

```
ERROR Task kafka-connect-splunk-20m-ack2-1-1 threw an uncaught and unrecoverable exception
(org.apache.kafka.connect.runtime.WorkerTask:148)
org.apache.kafka.connect.errors.ConnectException: Sink tasks require a list of topics.
```

**Error: Invalid token**

If you encounter an error that says that your deployment has an invalid Splunk HEC token, provide the correct HEC token.

The error may look like this:

```
ERROR failed to post events resp={"text":"Invalid token","code":4}, status=403
(com.splunk.hecclient.Indexer:172)
```

**Error: Connection timed out**

If you encounter a time out connection error, the Splunk platform is not reachable. Verify the provided HEC URI is up, running, and reachable.

The error may look like this:

```
ERROR encountered io exception (com.splunk.hecclient.Indexer:141)
org.apache.http.conn.HttpHostConnectException: Connect to x.x.x.x:8088 [/x.x.x.x] failed: Connection timed out (Connection timed out)
```

**Error: Invalid enrichment**

If you encounter an error indicating an invalid enrichment, your deployment has an invalid, non key-value pair data enrichment parameter. Provide values in key-value format only.

The error may look like this:

```
ERROR Task kafka-connect-splunk-jan26-ack-2-1 threw an uncaught and unrecoverable exception
(org.apache.kafka.connect.runtime.WorkerTask:148)
```

**Error: Unrecognized SSL message**

If you encounter an error indicating an unrecognized SSL message, your deployment's HEC URI contains HTTP instead of HTTPS. Either enable SSL on your Splunk HEC, or use HTTP in the HEC URI. If possible, do not disable SSL.

The error may look like this:
Error: Unable to find valid certification path

If you encounter an error indicating an inability to find a valid certification path, the SSL certificate has not been provided. When SSL certificate validation setting is set to true, please provide a valid SSL certificate path.

The error may look like this:

Error: ACK is disabled

If you encounter an error that indicates that acknowledgment has been disabled, indexer acknowledgement has been disabled on your Splunk software HEC. Enable indexer acknowledgment on the Splunk platform side of your deployment, or use `splunk.hec.ack.enabled=false` in your Kafka Connect configurations.

The error may look like this:
Release Notes

Release Notes for Splunk Connect for Kafka

About this release

Version 2.0.0 of Splunk Connect for Kafka is compatible with the following software and platforms:

<table>
<thead>
<tr>
<th>Splunk platform versions</th>
<th>7.1 and later</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platforms</td>
<td>Platform independent</td>
</tr>
<tr>
<td>Vendor products</td>
<td>Kafka Connect</td>
</tr>
</tbody>
</table>

New features

Version 2.0.0 of Splunk Connect for Kafka contains the following new features:

- Configurable flush window. Users can now set a flush rate depending on their data-rate and event frequency.
- Clear Record Tracker (internal memory buffer) on acknowledgments to reduce memory footprint.
- Polling for Indexer fault tolerance.
- Improved ability to handle Null/Empty events.
- Improved efficiency on memory utilization and flushing of events from memory for situations when ACK is not enabled in the Splunk software.
- Ability to track HTTP Event Collector (HEC) telemetry.

Fixed issues

Version 2.0.0 of Splunk Connect for Kafka contains the following fixed issues.

<table>
<thead>
<tr>
<th>Date resolved</th>
<th>Issue number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2020-08-26</td>
<td>INGEST-17660</td>
<td>Forever retry on incorrectly formatted data</td>
</tr>
<tr>
<td>2020-06-15</td>
<td>INGEST-17659</td>
<td>Offsets not committed when using formatted events</td>
</tr>
</tbody>
</table>

Known issues

Version 2.0.0 of Splunk Connect for Kafka contains the following known issues. If none appear, then no issues are currently known.
Third-party software

Credits

Some of the components included in Splunk Connect for Kafka are licensed under free or open source licenses. We wish to thank the contributors to those projects.

View the licenses associated with each component by selecting a component name on the left.

Apache Commons Logging

https://mvnrepository.com/artifact/commons-logging/commons-logging

Version 1.2

Apache Commons Logging is licensed under the

Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Apache Commons CLI

https://mvnrepository.com/artifact/commons-cli/commons-cli

Version 1.4

Apache Commons BeanUtils is licensed under the

Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.
Apache Commons Lang

Snappy

https://mvnrepository.com/artifact/org.xerial.snappy/snappy-java

Version 1.1.7.3

Snappy is licensed under the

Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

LZ4

https://mvnrepository.com/artifact/org.lz4/lz4-java

LZ4 compression for Java version 1.6.0

LZ4 is licensed under the

Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Apache HttpCore

https://mvnrepository.com/artifact/org.apache.httpcomponents/httpcore

Version 4.4.13

Apache HttpCore is licensed under the

Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at
http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Apache Commons Codec

https://mvnrepository.com/artifact/commons-codec/commons-codec

Apache Commons Codec version 1.14

Apache Commons Codec is licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Apache log4j API


Version 2.13.1

Apache log4j API is licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Apache log4j Core


Version 2.13.2

Apache log4j Core is licensed under the
Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

**Apache HttpClient**

http://hc.apache.org/

Version 4.5.12

Apache Commons BeanUtils is licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

**Apache Kafka Clients**


Version 2.4.1

Apache Kafka Clients is licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

**Apache Kafka Connect API**

https://mvnrepository.com/artifact/org.apache.kafka/connect-api

Version 2.4.1

Apache Kafka Connect API is licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may
obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Zstandard

https://github.com/luben/zstd-jni

Version 1.4.3-1

Zstandard is licensed under the

BSD License

For Zstandard software

Copyright (c) 2016-present, Facebook, Inc. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
* Neither the name Facebook nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Javax WS RS API

https://github.com/eclipse-ee4j/jaxrs-api

Version 2.1.1
Javax WS RS API is licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Simple Logging Facade for Java API

http://www.slf4j.org/

Simple Logging Facade for Java API version 1.7.26

Simple Logging Facade for Java API source code and binaries are distributed under the MIT license.

Copyright (c) 2004-2017 QOS.ch All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.