**Lesson Overview**

In this lesson you will learn about the greater Kafka ecosystem and how it can save you time and effort and even allow you to integrate legacy applications with Kafka. Specifically, you’ll see how you can use Kafka Connect to quickly integrate common data sources into Kafka, and move data from Kafka into other data stores. You’ll also see how we can use REST Proxy to consume and produce data to and from Kafka using only an HTTP client.

**Glossary of Key Terms You Will Learn in this Lesson**

* Kafka Connect - A web server and framework for integrating Kafka with external data sources such as SQL databases, log files, and HTTP endpoints.
* JAR - **J**ava **AR**chive. Used to distribute Java code reusably in a library format under a single file.
* Connector - A JAR built on the Kafka Connect framework which integrates to an external system to either source or sink data from Kafka
* Source - A Kafka client putting data into Kafka from an external location, such as a data store
* Sink - A Kafka client removing data from Kafka into an external location, such as a data store
* JDBC - Java Database Connectivity. A Java programming abstraction over SQL database interactions.
* Task - Responsible for actually interacting with and moving data within a Kafka connector. One or more tasks make up a connector.
* Kafka REST Proxy - A web server providing APIs for producing and consuming from Kafka, as well as fetching cluster metadata.

**Kafka Connect - Summary**

In this section you learned that:

* Kafka Connect can be used to handle common and repeated scenarios.
* Kafka Connect is a web-server written in Java and Scala, and runs on the JVM.
* Kafka Connect has a plugin architecture, meaning that you can easily write your own connectors in addition to using the rich open-source ecosystem of connectors.
* By using Kafka Connect, you may be able to isolate your application entirely from integrating with a Kafka client library.

**Optional Further Reading on Kafka Connect**

* [**Overview of Kafka Connect**](https://docs.confluent.io/current/connect/concepts.html)

## Kafka Connect Connector Plugins

Here are some popular Kafka Connect plugins:

* [**Confluent Kafka Connect Plugin Search**](https://www.confluent.io/hub/?utm_medium=sem&utm_source=google&utm_campaign=ch.sem_br.brand_tp.prs_tgt.confluent-brand_mt.mbm_rgn.namer_lng.eng_dv.all&utm_term=%2Bconfluent%20%2Bconnect&creative=357969856550&device=c&placement=&gclid=Cj0KCQjwp5_qBRDBARIsANxdcikzhat8UGdi8TVVfxhSATPhQqLibR81tnJC0lVGsPaRubygAORySDEaAhrbEALw_wcB)
* [**Amazon S3 Connector**](https://www.confluent.io/hub/confluentinc/kafka-connect-s3)
* [**SQL JDBC Connector**](https://www.confluent.io/hub/confluentinc/kafka-connect-jdbc)
* [**HDFS Connector**](https://www.confluent.io/hub/confluentinc/kafka-connect-hdfs)
* [**HTTP Connector**](https://www.confluent.io/hub/confluentinc/kafka-connect-http)

**Kafka Connect Connectors - Summary**

In this section you learned:

* Kafka Connect supports a number of Connectors for common data sources
  + Files on disk
  + Amazon S3 and Google Cloud Storage
  + SQL databases such as MySQL and Postgres
  + HDFS
* Kafka Connect has an extensive REST API for managing and creating Connectors

**Kafka Connect Connectors - Optional Further Research**

* [**Confluent Connector Hub**](https://www.confluent.io/hub/?_ga=2.42557541.1345547963.1563205519-78292278.1561645529)
* [**List of core Connectors included with most distributions**](https://docs.confluent.io/current/connect/managing/connectors.html)
* [**Connect REST API Documentation**](https://docs.confluent.io/current/connect/references/restapi.html)

**JDBC Sinks and Sources**

* JDBC = Java DataBase Connector. The JDBC API is used to abstract the interface to SQL Databases for Java applications. In the case of Kafka Connect, JDBC is used to act as a generic interface to common databases such as MySQL, Postgres, etc.
* JDBC Sinks are a common way to move data into Kafka from existing databases. Once the data is available in Kafka, it can be used in stream processing operations to enrich data or provide insights that may otherwise be missing
* JDBC Sources are a common way to move data out of Kafka to traditional SQL datastores. This is a common way of making stream processing insights available for more ad-hoc or batch querying.

**Kafka Key Connectors - Summary**

In this section you learned how to:

* Apply the Kafka Connect FileStream Source connector to push logs into Kafka
* Apply the Kafka Connect JDBC Source connector to push SQL data into Kafka

**Kafka Key Connectors - Optional Further Research**

* [**Kafka FileStream Connector Documentation**](https://docs.confluent.io/current/connect/filestream_connector.html)
* [**Kafka JDBC Source Connector Documentation**](https://docs.confluent.io/current/connect/kafka-connect-jdbc/index.html)

**Summary - Kafka REST Proxy**

In this section we learned that Kafka REST Proxy:

* Is a web server built in Java and Scala that allows any client capable of HTTP to integrate with Kafka
* Allows production and consumption of Kafka data
* Allows read-only operations on administrative information and metadata

**REST Proxy Producer**

* [**POST data to /topics/<topic\_name> to produce data**](https://docs.confluent.io/current/kafka-rest/api.html#post--topics-(string-topic_name))
* The Kafka data may be POSTed in Binary, JSON, or Avro
* When sending Avro data you must always include the schema data as a string
* [***Always check your Content-Type header* to ensure that it is correctly configured**](https://docs.confluent.io/current/kafka-rest/api.html#content-types)
  + Content-Type is in the format application/vnd.kafka[.embedded\_format].[api\_version]+[serialization\_format]
  + embedded\_format is how the data destined for Kafka is formatted. Must be one of binary, json, or avro
  + api\_version is the API version for REST Proxy -- this should always be v2 as of this writing
    - serialization\_format has nothing to do with your Kafka data, this is how the actual data being sent to REST proxy is serialized. Only json is supported for now -- so always set this to json!
* When using REST Proxy, always start by ensuring that the Content-Type is correctly set before running your code. A misconfigured Content-Type can lead to confusing and hard-to-debug errors.

## REST Proxy Consumer

* [**POST to /consumers/<group\_name> to create a consumer group**](https://docs.confluent.io/current/kafka-rest/api.html#post--consumers-(string-group_name))
* [**POST to /consumers/<group\_name>/instances/<instance\_id>/subscriptions to create a subscription**](https://docs.confluent.io/current/kafka-rest/api.html#post--consumers-(string-group_name)-instances-(string-instance)-subscription)
* [**GET from /consumers/<group\_name>/instances/<instance\_id>/records to retrieve records**](https://docs.confluent.io/current/kafka-rest/api.html#get--consumers-(string-group_name)-instances-(string-instance)-records)
* [**Always check your Accept header to ensure that it is correctly configured**](https://docs.confluent.io/current/kafka-rest/api.html#content-types)
* Content-Type is in the format application/vnd.kafka[.embedded\_format].[api\_version]+[serialization\_format]
* embedded\_format is how the data requested from Kafka is formatted. Must be one of binary, json, or avro
* api\_version is the API version for REST Proxy -- this should always be v2 as of writing
* serialization\_format has nothing to do with your Kafka data, this is how the actual data being received from REST proxy is serialized. Only json is supported for now -- so always set this to json!

**Using REST Proxy - Summary**

REST Proxy is a powerful tool for integration applications into Kafka that could not otherwise use it.

REST Proxy offers a comprehensive API for producing and consuming data from Kafka topics, and even provides fine-grained control over consumer groups, offsets, and partitions.

**Keep REST Proxy in Mind!**

Throughout this section you’ve seen how you can easily leverage the HTTP API of REST Proxy to integrate applications directly into your Kafka ecosystem. While the HTTP API may not be as nice as a native Kafka client, it still provides a fairly straightforward path for integration.

As Kafka grows within your organization, you will likely find applications that can’t integrate with Kafka for one reason or another. When that happens, look to REST Proxy!

**Lesson Recap**

In this lesson you learned how to use Kafka Connect to quickly integrate Kafka into a number of your existing data stores and workflows. We went hands-on with the JDBC and FileStream connectors and saw how to configure and deploy them. Next, we saw how REST Proxy can be used to bring Kafka to applications that can’t integrate native clients, but do have REST capabilities.

**Glossary of Key Terms in this Lesson: (same as provided in beginning of lesson)**

* Kafka Connect - A web server and framework for integrating Kafka with external data sources such as SQL databases, log files, and HTTP endpoints.
* JAR - **J**ava **AR**chive. Used to distribute Java code reusably in a library format under a single file.
* Connector - A JAR built on the Kafka Connect framework which integrates to an external system to either source or sink data from Kafka
* Source - A Kafka client putting data into Kafka from an external location, such as a data store
* Sink - A Kafka client remove data from Kafka into an external location, such as a data store
* JDBC - Java Database Connectivity. A Java programming abstraction over SQL database interactions.
* Task - Responsible for actually interacting with and moving data within a Kafka connector. One or more tasks make up a connector.
* Kafka REST Proxy - A web server providing APIs for producing and consuming from Kafka, as well as fetching cluster metadata.