**KSQL Architecture**

KSQL is a Java application built on top of the Kafka Streams Java stream processing library. KSQL is a web-server with a REST API that accepts incoming or preconfigured requests containing SQL-like commands. These commands are translated by the KSQL server into the equivalent Kafka Streams application and then executed.

Users can interact with KSQL via a REST API, its dedicated CLI, or predefined SQL files.

**KSQL vs. Traditional Frameworks**

**Pros**

* It is often simpler to use KSQL and SQL than to build and deploy an entire application
* KSQL is typically a better fit for rapid experimentation and exploration than a full stream processing application
* KSQL doesn’t require a particular programming language, like Python for Faust, or Java for Kafka Streams
* KSQL already comes bundled with standard logs, metrics, and tooling for you to use, so you don’t have to build it yourself

**Cons**

* SQL does not always best capture certain types of data manipulation or remapping scenarios
* Can’t just use whatever other libraries you want like you can with Faust
  + However, KSQL does allow User Defined Functions (UDFs), written in Java

**Turning Kafka Topics into Tables and Streams**

* Every KSQL Table or Stream is derived from an underlying Kafka Topic
* Use the SHOW TOPICS command in KSQL CLI to see all available topics

**Stream Creation**

* Creating Streams from an underlying topic requires you to specify column names and their types
* You must also specify the serialization format as one of JSON, AVRO, or DELIMITED (csv)
* You must also specify the underlying topic name
* You may create a stream from another existing stream with CREATE STREAM <stream\_name> AS SELECT …
* [**KSQL Create Stream Documentation**](https://docs.confluent.io/current/ksql/docs/developer-guide/syntax-reference.html#create-stream)
* [**KSQL Create Stream from SELECT documentation**](https://docs.confluent.io/current/ksql/docs/developer-guide/syntax-reference.html#create-stream-as-select)

## Table Creation

* Creating Tables from an underlying topic requires you to specify column names and their types
  + You must also specify the serialization format as one of JSON, AVRO, or DELIMITED (csv)
  + You must also specify the underlying topic name
  + You may create a table from another existing stream or table with CREATE STREAM <stream\_name> AS SELECT …
  + [**KSQL Key Requirements Documentation**](https://docs.confluent.io/current/ksql/docs/developer-guide/syntax-reference.html#ksql-key-requirements)
  + [**KSQL Create Table Documentation**](https://docs.confluent.io/current/ksql/docs/developer-guide/syntax-reference.html#create-table)
  + [**KSQL Create Table from SELECT documentation**](https://docs.confluent.io/current/ksql/docs/developer-guide/syntax-reference.html#create-table-as-select)

**Querying Syntax**

* SELECT statements may be run in KSQL CLI, but as soon as the session is terminated, so too is the data calculation.
* Use CREATE STREAM <stream\_name> AS SELECT… and CREATE TABLE <table\_name> AS SELECT … to persist your queries for long-term usage
* [**KSQL Querying Syntax Documentation**](https://docs.confluent.io/current/ksql/docs/developer-guide/syntax-reference.html#select)
* [**See the KSQL documentation for a list of all Scalar functions supported for querying**](https://docs.confluent.io/current/ksql/docs/developer-guide/syntax-reference.html#scalar-functions)

**Demo: Querying**

**Important note:**

Before you execute anything in this workspace, or anytime after you've refreshed, or woken up, or reset data, or used the "Get New Content" button in this workspace, you'll need to run this command in the terminal:

**Hopping and Tumbling Windowing**

* [**KSQL supports Tumbling windows with the WINDOW TUMBLING (SIZE <duration>) syntax**](https://docs.confluent.io/current/ksql/docs/developer-guide/aggregate-streaming-data.html#aggregate-records-over-a-tumbling-window)
* [**KSQL supports Hopping windows with the WINDOW HOPPING (SIZE <duration>, ADVANCE BY <interval>) syntax**](https://docs.confluent.io/current/ksql/docs/developer-guide/aggregate-streaming-data.html#aggregate-records-over-a-hopping-window)

## Session Windowing - Key Points

* Keeps track of differences between the time a key was last seen and the current key arrival time.
* If the difference between the time a key was last seen and the current key arrival time, for two records with the same key, is larger than the session window length defined, a new window is started.
* If the difference between the time a key was last seen and the current key arrival time, for two records with the same key, is less than the session window length, the record is added to the current window, and the session expiration time is started anew.
  + Session expiration denotes that the full session period begins again
* [**KSQL Session window documentation**](https://docs.confluent.io/current/ksql/docs/concepts/time-and-windows-in-ksql-queries.html#session-window)

**KSQL Aggregations**

* [**Use GROUP BY to create aggregations in KSQL**](https://docs.confluent.io/current/ksql/docs/developer-guide/aggregate-streaming-data.html)
* GROUP BY always creates a KSQL Table
* [**KSQL supports aggregations like COUNT, MAX, MIN, SUM, TOPK, HISTOGRAM and more**](https://docs.confluent.io/current/ksql/docs/developer-guide/syntax-reference.html#aggregate-functions)

## KSQL JOINs

* [**KSQL supports Stream to Stream, Stream to Table, and Table to Table JOINs**](https://docs.confluent.io/current/ksql/docs/developer-guide/join-streams-and-tables.html#join-event-streams-with-ksql)
* [**Limitations on the kind of JOINs supported exist for each of the types of JOINs**](https://docs.confluent.io/current/ksql/docs/developer-guide/join-streams-and-tables.html#join-capabilities)
* [**Stream to Stream JOINs may be windowed**](https://docs.confluent.io/current/ksql/docs/developer-guide/join-streams-and-tables.html#joins-and-windows)
* [**JOINed entities must be co-partitioned**](https://docs.confluent.io/current/ksql/docs/developer-guide/join-streams-and-tables.html#join-requirements)
* [**JOINed data must share the same KEY in KSQL as used for the Kafka record**](https://docs.confluent.io/current/ksql/docs/developer-guide/join-streams-and-tables.html#join-requirements)

**Lesson Summary**

In this lesson you learned:

* How KSQL is an excellent tool for rapid iteration and deployment of Stream Processing applications
* That KSQL can GROUP BY aggregate data
* That KSQL supports HOPPING, TUMBLING, and SESSION windows
* That KSQL supports JOIN operations to merge two or more streams

**Glossary of Key Terms for this Lesson (same as glossary at beginning of lesson):**

* Kafka Streams - A Java library for constructing stream processing applications. KSQL translates SQL statements to Kafka Streams applications.
* User Defined Function (UDF) - An extension to the SQL capabilities of KSQL written by the user. For KSQL, these are written in Java.
* Key (KSQL) - Data which uniquely identifies the value contained in this data message relative to other pieces of data in the stream. For example, a user\_id may uniquely identify a user object.
* Session Windowing (KSQL) - A system that keeps track of when the last time a particular key was seen. When a new record with the same key arrives, the difference between the timestamps is calculated. If the difference is larger than a defined *session window*, then a new window is started for that session. If the difference is less than the defined session window, the new record is added to the existing window.