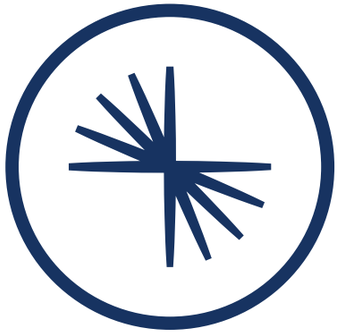
Stream Processing using Apache Kafka® Streams and Confluent ksqlDB

Version 7.0.0-v1.0.1

Table of Contents



CONFLUENT

Global Education

Class Logistics and Overview

Copyright & Trademarks

Prerequisite

This course requires a working knowledge of the Apache Kafka architecture.

New to Kafka? Need a refresher?

Sign up for free ***Confluent Fundamentals for Apache Kafka*** course at https://confluent.io/training

Agenda

1. **Starting with Stream Processing**
   a. Bridging from Fundamentals and core Apache Kafka
   b. Kafka Streams concepts
   c. ksqlDB concepts
2. **Stateful Processing and Advanced Operations**
   a. Time-based processing
   b. Stateful processing
   c. Custom processing
3. **Safely Deploying and Operating Stream Processing**
   a. Testing, Troubleshooting, Monitoring
   b. Deployment
   c. Security

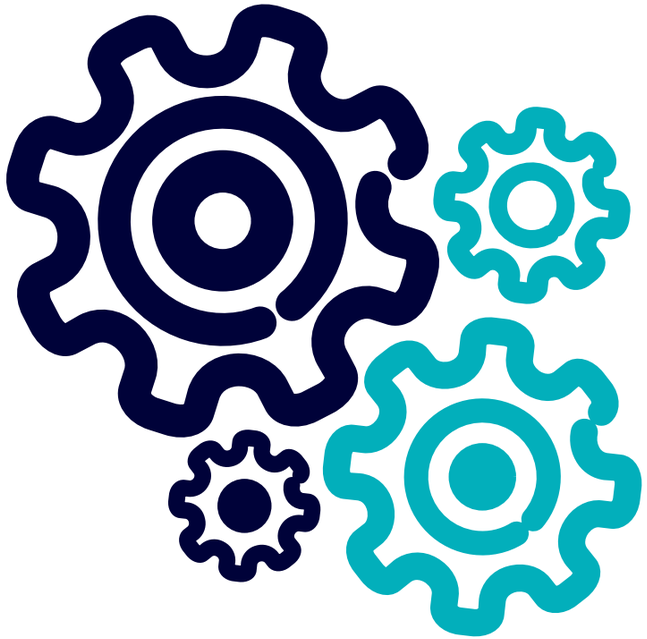
Course Objectives

Upon completion of this course, you should be able to:

- Identify common patterns and use cases for real-time stream processing
- Describe the high-level architecture of Apache Kafka Streams
- Write real-time applications with the Kafka Streams API to filter, transform, enrich, aggregate, and join data streams
- Describe how ksqlDB combines the elastic, fault-tolerant, high-performance stream processing capabilities of Kafka Streams with the simplicity of a SQL-like syntax
- Author ksqlDB queries that showcase their balance of power and simplicity
- Test, secure, deploy, and monitor Kafka Streams applications and ksqlDB queries

Throughout the course, hands-on exercises will reinforce the topics being discussed.

Class Logistics



- Timing
  - Start and end times
  - Can I come in early/stay late?
  - Breaks
  - Lunch
- Physical Class Concerns
  - Restrooms
  - Wi-Fi and other information
  - Emergency procedures
  - Don't leave belongings unattended

No recording, please!

How to get the courseware?

1. Register at **training.confluent.io**
2. Verify your email
3. Log in to **training.confluent.io** and enter your **license activation key**
4. Go to the **Classes** dashboard and select your class

Introductions

- About you:
  - What is your name, your company, and your role?
  - Where are you located (city, timezone)?
  - What is your experience with Kafka?
  - Which other Confluent courses have you attended, if any?
  - Optional talking points:
    - What are some other distributed systems you like to work with?
    - What technology most excited you early in your life?
    - Anything else you want to share?

- About your instructor

Fundamentals Review

Discussion

Question Set 1 [6 mins]

Determine if each statement is true or false and why:

1. All messages in a topic are on the same broker.
2. All messages in a partition are on the same broker.
3. All messages that have the same key will be on the same broker.
4. The more partitions a topic has, the better.

Question Set 2 [3 mins]

Determine the best answer to each question.

1. What are the roles of a producer and a consumer?
2. How is it decided which messages consumers read?
3. Who initiates the reading of messages: consumers or the Kafka cluster?

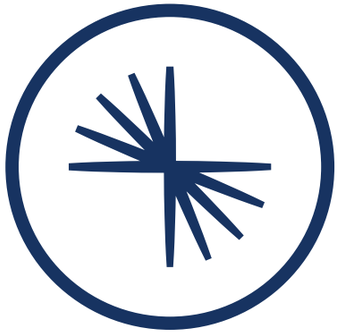
Discussion, Cont'd.

Question 3 [1 min]

Suppose there is a message in our Kafka cluster about my breakfast purchase of $12.73. Consumer c_0 has consumed it to process the charge. Could consumer c_7 consume this same message this afternoon?

Question 4 [2 mins]

Kafka has a transactions API. When we know that all messages that are part of a transaction successfully made it to the cluster, we want to tag those messages as "good." When we know that not all messages in a transaction made it, we want to tag those messages that did make it as "bad." Kafka uses markers in the logs, that are effectively new messages written after existing messages to do this. Why not just put something in the metadata? Why not delete "bad" messages?



CONFLUENT

Global Education

Module Overview

This module contains two lessons:

- What Do You Need to Know about Group Management in Kafka Before Creating Streaming Applications?
- How Can You Leverage Streaming to Transform the Immutable Data in Your Kafka Cluster?

Where this fits in:

- Hard Prerequisite: Fundamentals Course
- Recommended Prerequisite: Core Branch of Developer Course
- Recommended Follow-Up: Working with Kafka Streams

1a: What Do You Need to Know about Group Management in Kafka Before Creating Streaming Applications?

Description

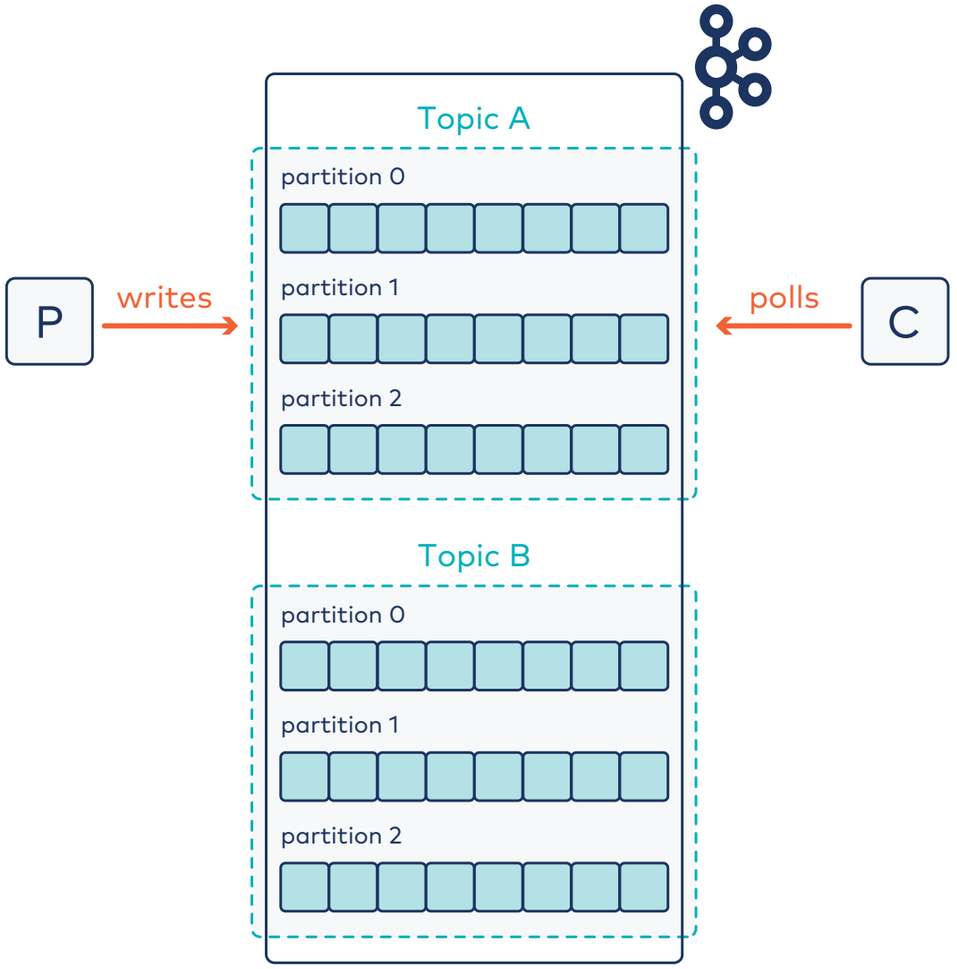
Apache Kafka is the De Facto Standard for Real-Time Event Streaming. The Kafka consumer group protocol allows for hands-off resource management and load balancing. Incremental cooperative rebalancing protocol allows Kafka Streams and ksqlDB application to scale and smoothly handle failures.

What is Apache Kafka?

Kafka is an event streaming platform. Three key benefits are scalability, fault-tolerance, and reliability:

- You can publish and subscribe to events
- Kakfa can store events for as long as you want
- You can process and analyze events

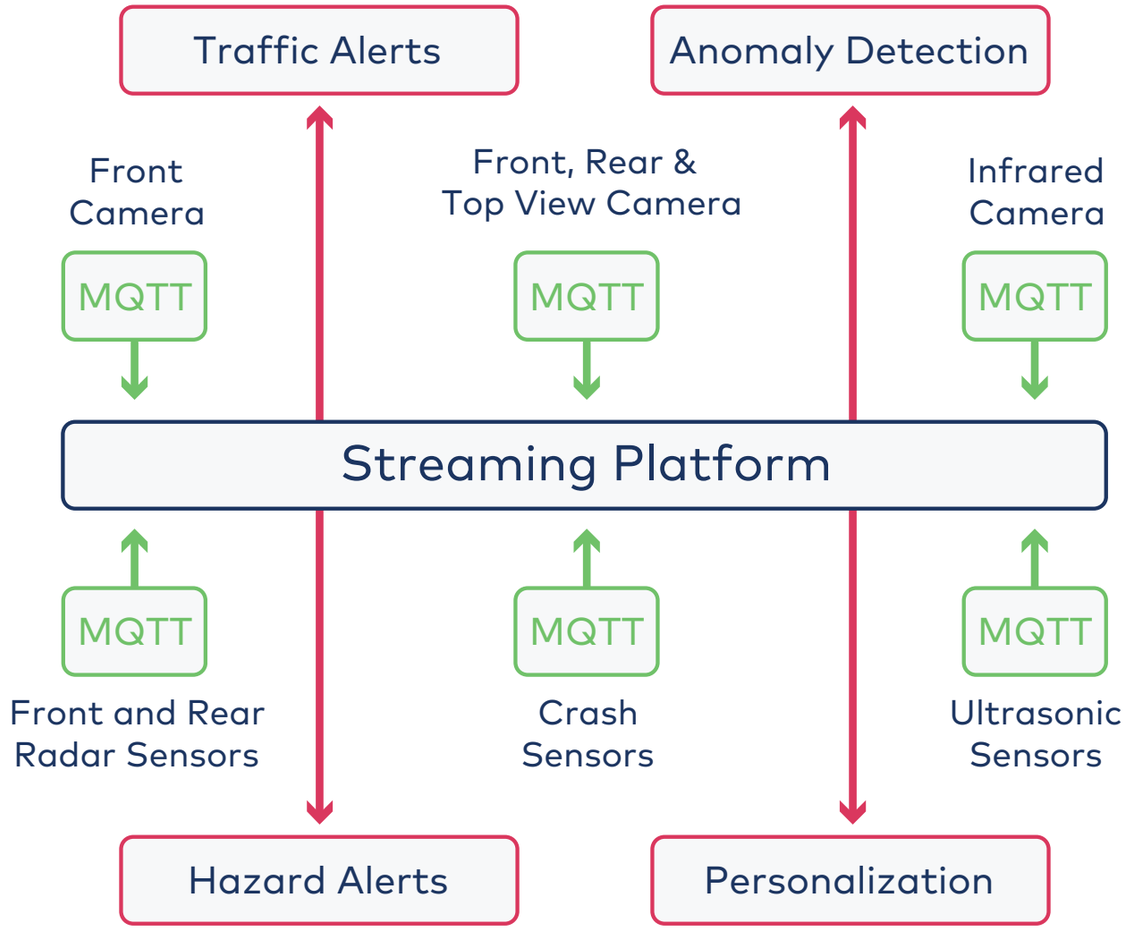
Apache Kafka as a Streaming Platform



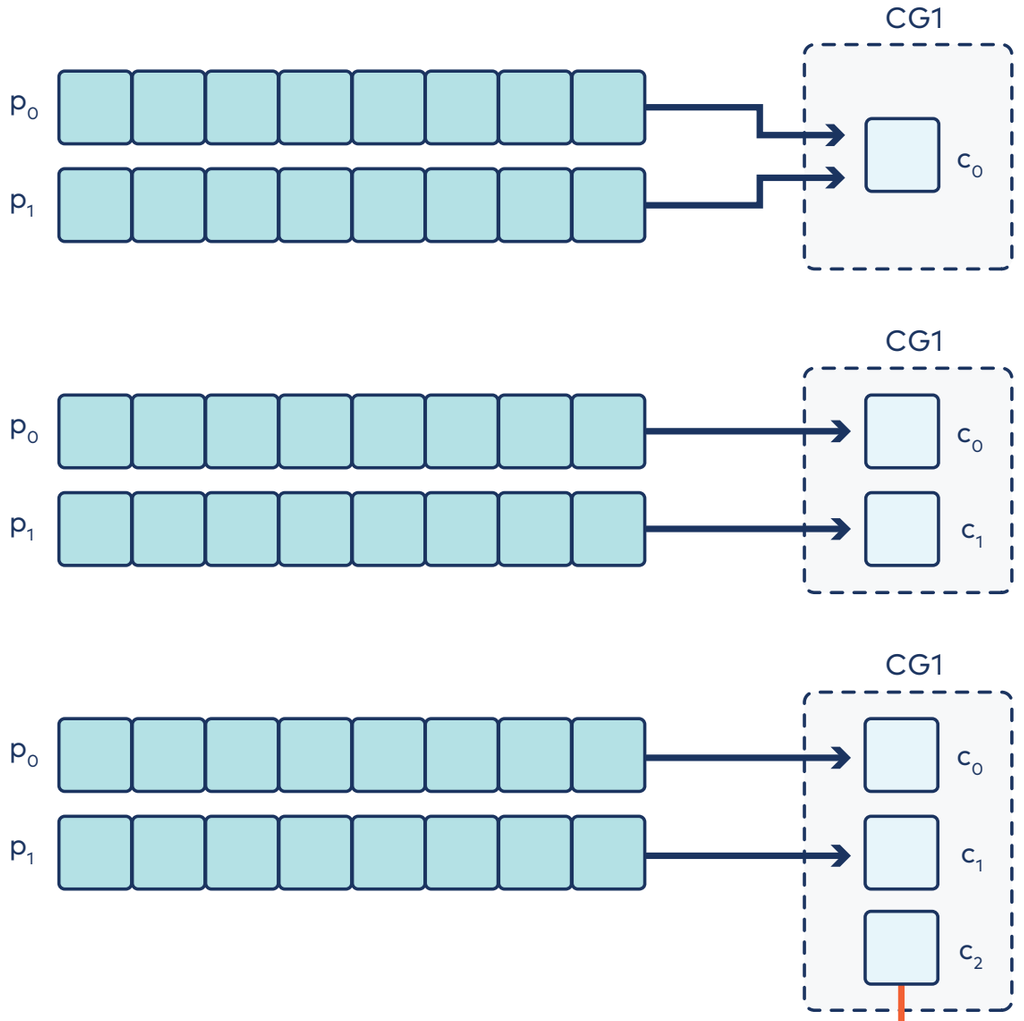
Data in Kafka is immutable

Use Case: Automotive Internet of Things

The Future of the Automotive Industry is a Real-Time Data Cluster



Consumer Groups



Consumer Partition Assignment Strategy

RangeAssignor

Use when joining data from multiple topics (default)

RoundRobin

Use when performing stateless operations on records from many topics

Sticky

RoundRobin with a best effort to maintain assignments across rebalances

CooperativeSticky

Sticky but it uses consecutive rebalances rather than the single stop-the-world used by Sticky

1b: How Can You Leverage Streaming to Transform the Immutable Data in Your Kafka Cluster?

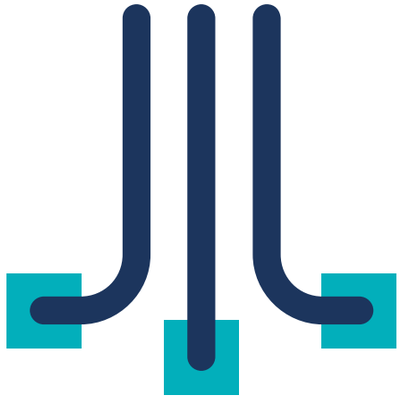
Description

Kafka Streams and Confluent ksqlDB are two of the options to build real-time streaming applications. Kafka Streams is a client library for building applications and microservices, where the input and output data are stored in an Apache Kafka cluster. The Streams processor topology defines the stream processing computational logic for the application.

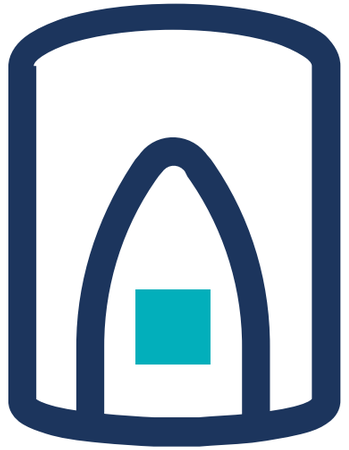
How Do You Process Data in Kafka?

- Data in Kafka is immutable
- But what if you need to transform, enrich the data in Kafka? For example…
  - Filter, merge, group, repartition, etc. (stateless)
  - Aggregate, join, etc. (stateful)

Options for Writing Streaming Applications

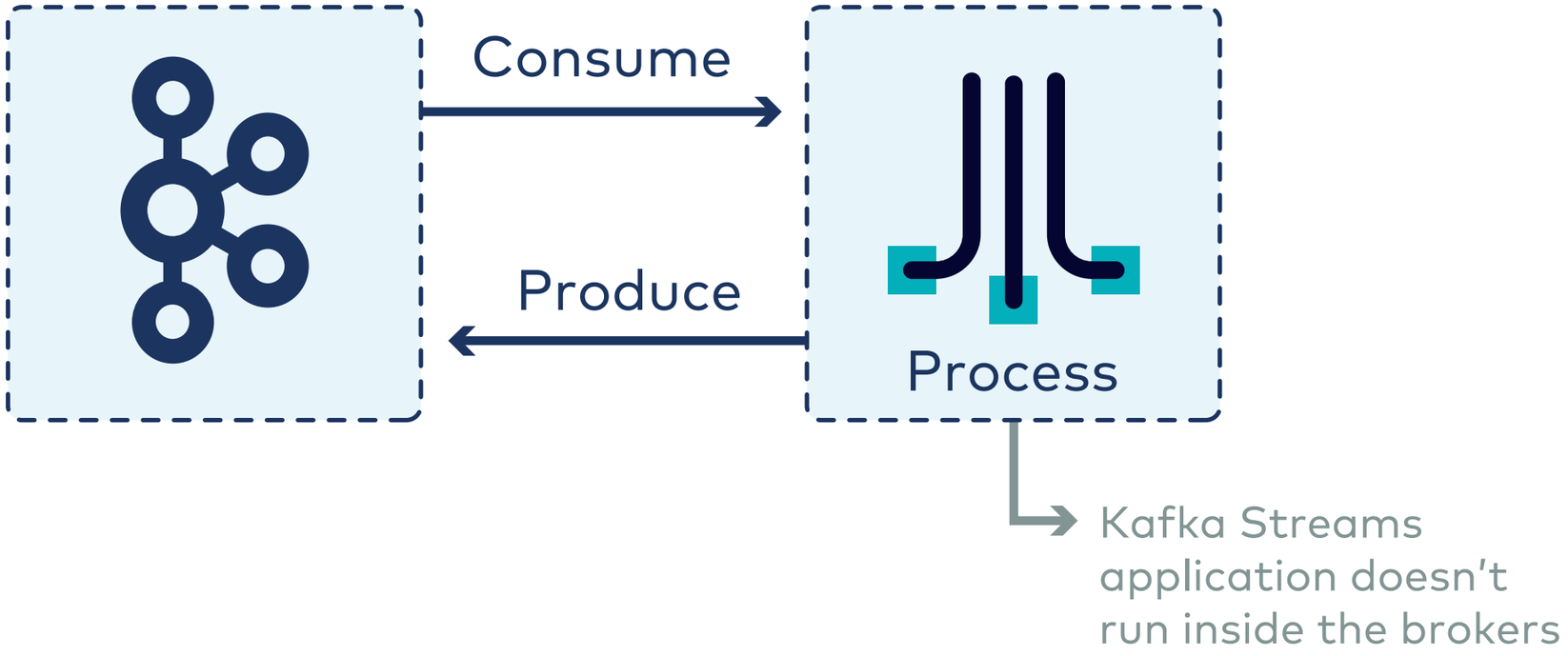


Kafka Streams

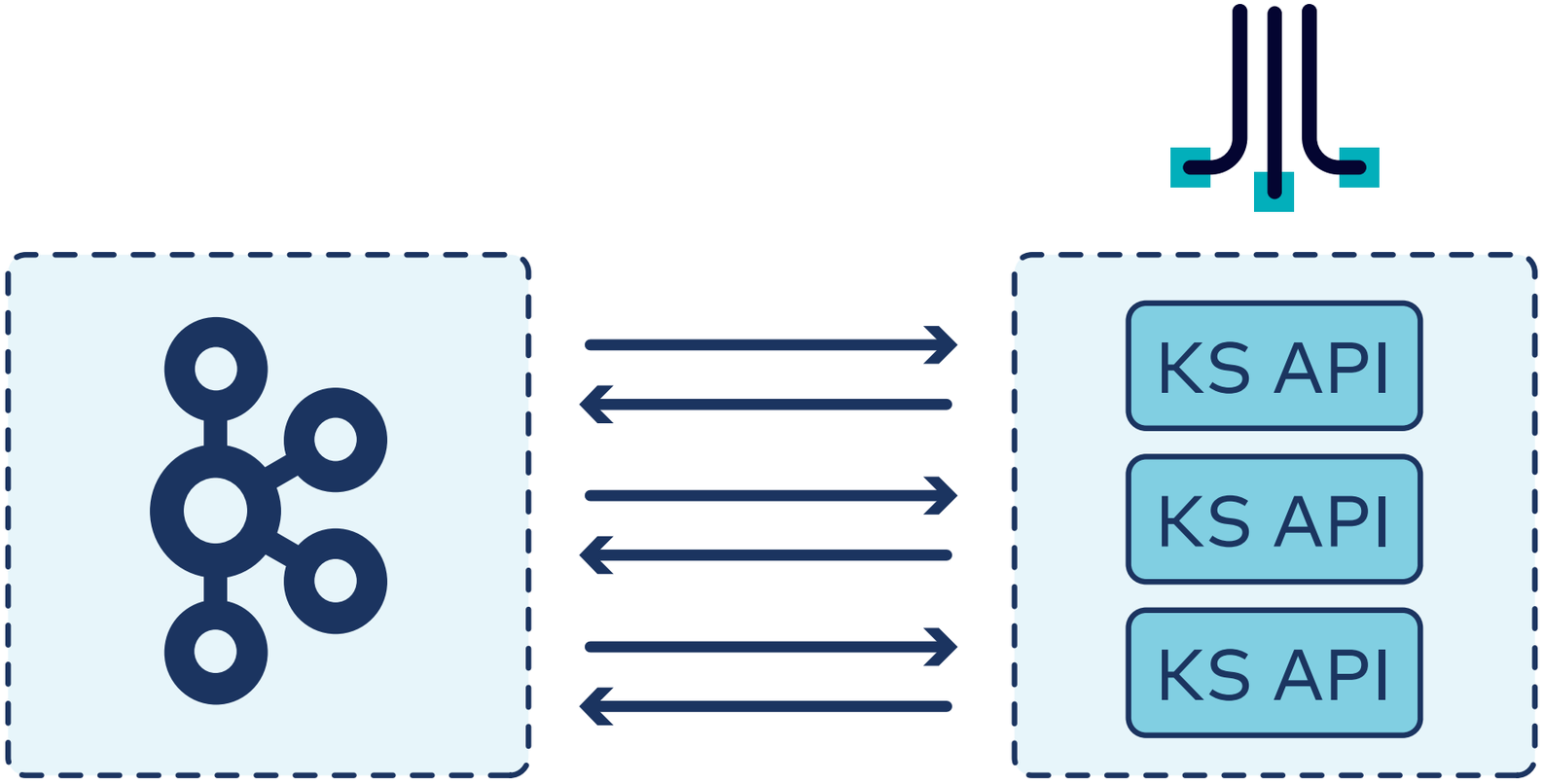


Confluent ksqlDB

Kafka Streams is a Client of Kafka



Same Application, Many Instances



Parallelism Model

Kafka Streams uses the concepts of stream partitions and stream tasks as logical units of its parallelism model.

Links between Kafka Streams and Kafka:

- Each stream partition…
  - is an ordered sequence of data records.
  - maps to a Kafka topic partition.
- A data record in the stream maps to a Kafka message from that topic.
- The keys of data records determine the partitioning of data in both Kafka and Kafka Streams.

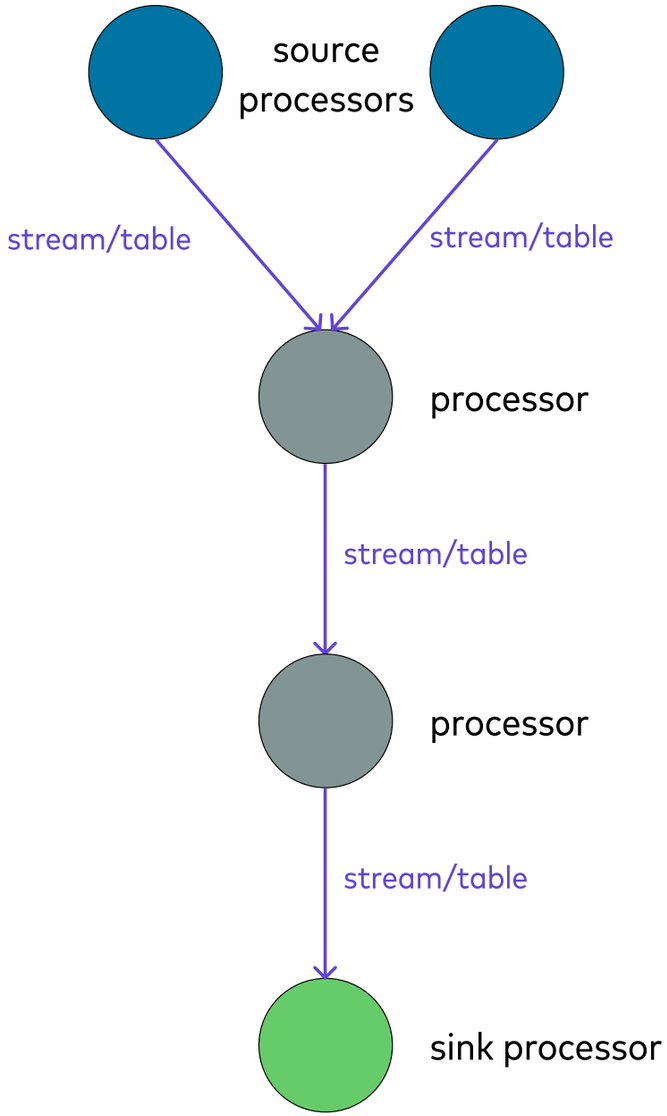
Processor Topology

Processor topology

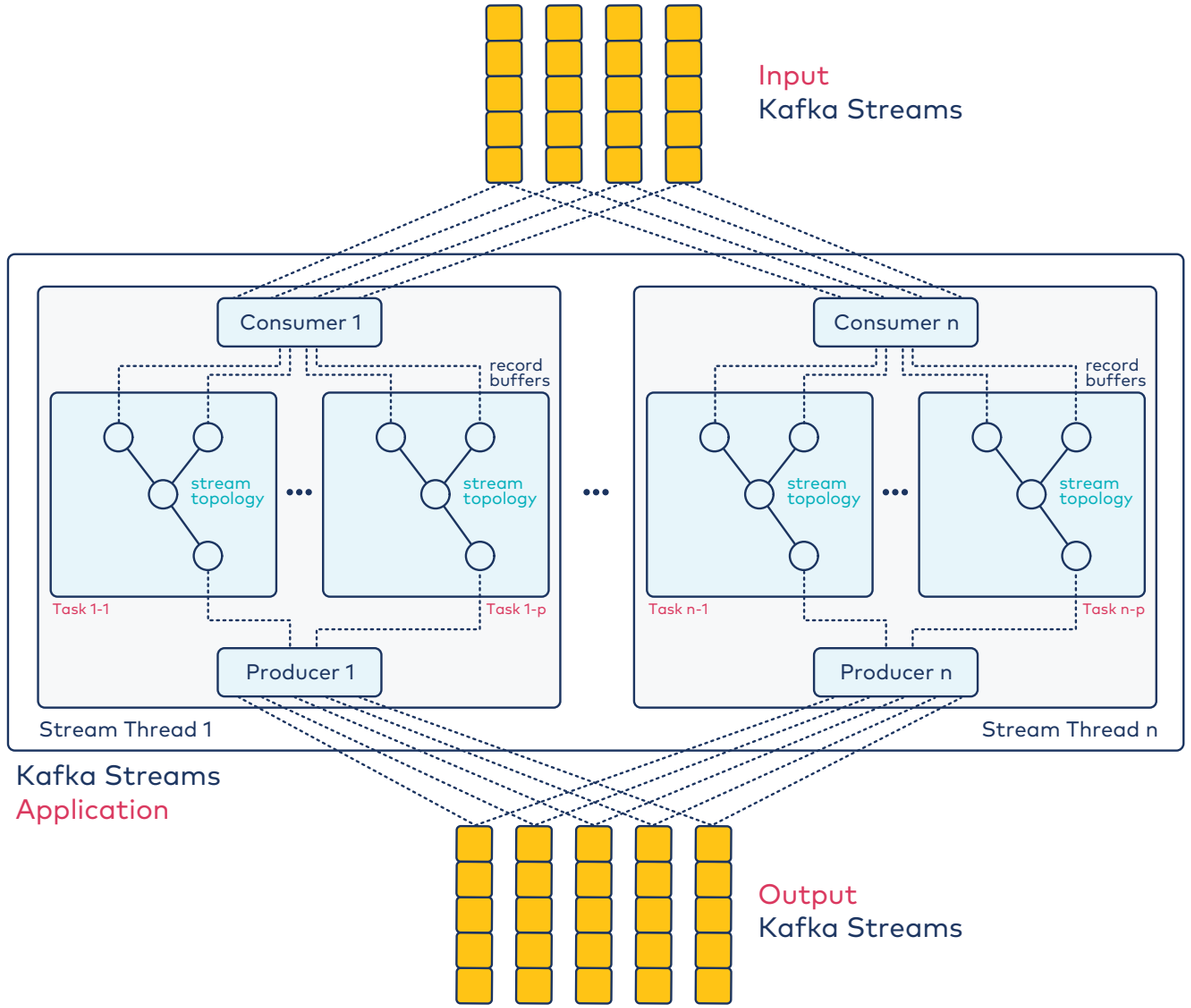
computational logic of the data processing performed by a stream processing application.

Details:

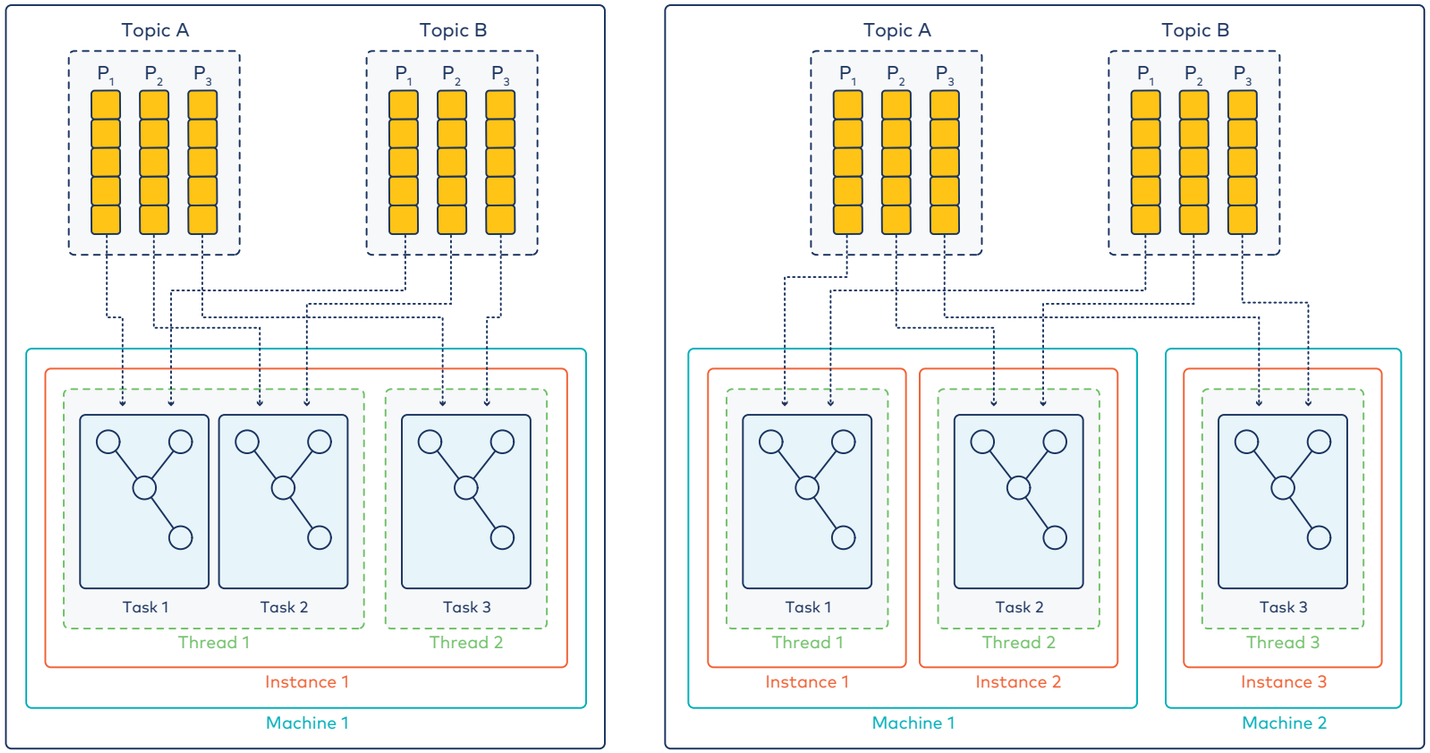
- A topology is a graph of stream processors (nodes) that are connected by streams (edges).
- You can define topologies via the low-level Processor API or via the Kafka Streams DSL.



Streams Architecture - Single Application Instance with Multiple Threads Configured



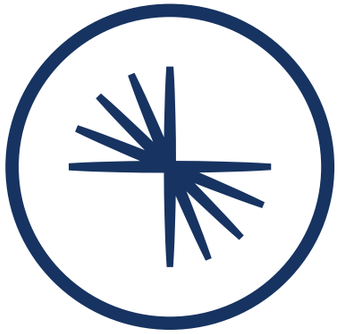
Streams Architecture - Multiple Application Instances on Multiple Machines



Lab: Scaling a Kafka Streams Application

Please work on **Lab 1a: Scaling a Kafka Streams Application**

Refer to the Exercise Guide



CONFLUENT

Global Education

Module Overview

This module contains four lessons:

- What Are the Big-Picture Kafka Streams Concepts?
- How Do You Put Together a Kafka Streams Application?
- What are Some Operations You Can Use To Transform Streams?
- What Changes When Your Stream Processing Needs to Track State?

Where this fits in:

- Hard Prerequisite: Introduction to Kafka Streams
- Recommended Prerequisite: Core Branch of Developer Course
- Recommended Follow-Up: Introduction to ksqlDB

2a: What Are the Big-Picture Kafka Streams Concepts?

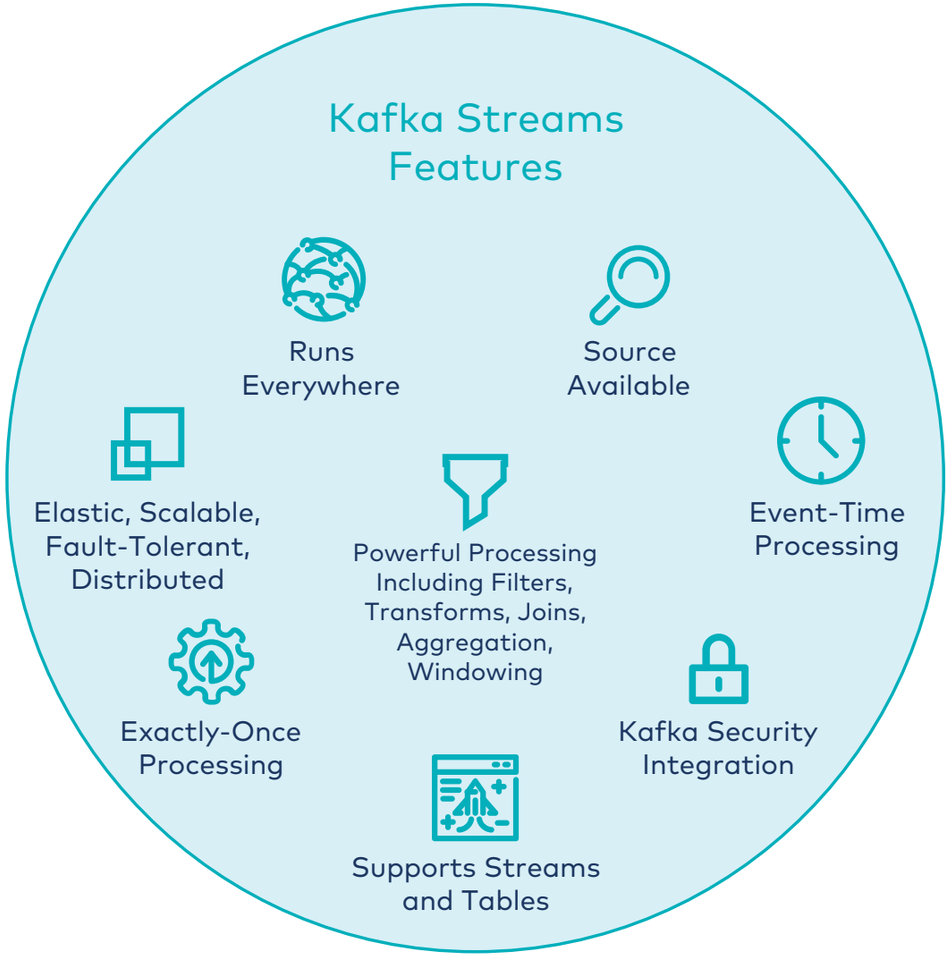
Description

Kafka Streams is a Java library, and it uses the Domain Specific Language to define processor topology. The Kafka Streams DSL is built on top of the Streams Processor API and it has built-in abstractions for streams and tables in form of `KStream`, `KTable`, and `GlobalKTable` objects.

Kafka Streams Applications

- The Streams API of Apache Kafka is available through a Java library which can be used to write a distributed streams application.
- Using the DSL (Domain Specific Language), you can define processor topologies in your application. The steps are:
  - *Consume*: Specify input streams that are read from Kafka topics.
  - *Process*: Compose transformations on these streams.
  - *Produce*: Write the resulting output streams back to Kafka topics.

Properties of Kafka Streams



Kafka Streams Applications Topology

- The logic is defined as a processor topology, the graph of stream processors and streams.
- You can define the processor topology with the Kafka Streams APIs:
  - Kafka Streams DSL
  - Processor API

Kafka Streams DSL

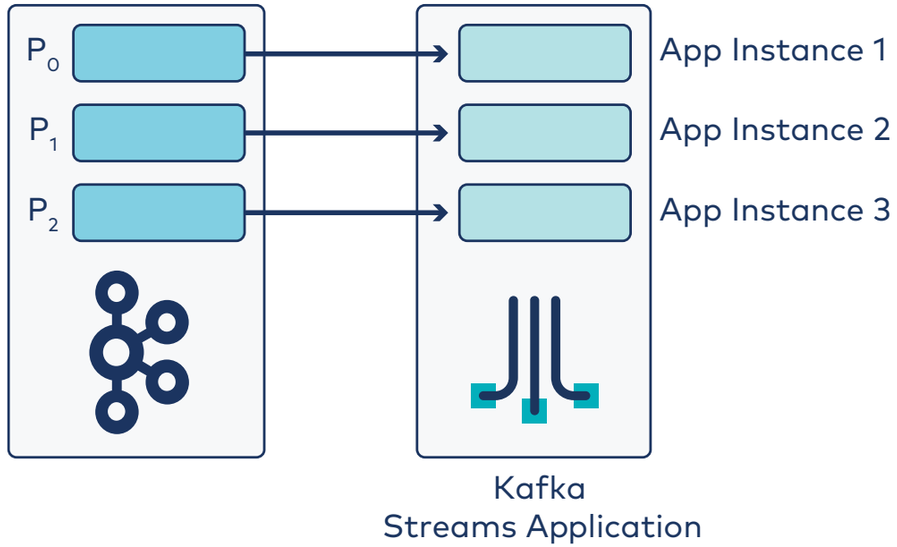
- The Kafka Streams DSL is built on top of the Streams Processor API.
- It has built-in abstractions for streams and tables in form of:
  - `KStream` (stream)
  - `KTable` (table)
    - `GlobalKTable`

Stream and Table Example

Event	Stream	Table	State
Bus XYZ (key) **departed** from NYC (value)	Insert	Insert	Traveling to Chicago
Bus XYZ (key) **arrived** at Chicago (value)	Insert	Update	Waiting for passengers
Bus ABC (key) **departed** from Boston (value)	Insert	Insert	Traveling to Florida
Bus XYZ (key) **departed** from Chicago (value)	Insert	Update	Travelling to Salt Lake City
Bus XYZ (key) **arrived** at Salt Lake City (value)	Insert	Update	Waiting for passengers
Bus XYZ (key) : null (value)	Insert	Delete	Bus is decommissioned
Key (null): arrived at San Francisco	Insert	Ignored	(blank)

KStream and **KTable** Objects

KStream	KTable
Immutable	Mutable
Unbounded	Bounded
Insert (append)	Insert/Update/Delete
Can have many events/key	One event/key
Partitioned	Partitioned
Ordering is guaranteed per partition	Ordering is not guaranteed per partition
Persistent, durable, and fault-tolerant	Persistent, durable, and fault-tolerant

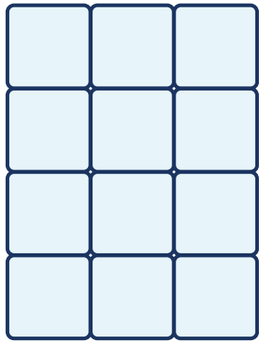


Stream-Table Duality

Relationship between streams and tables:

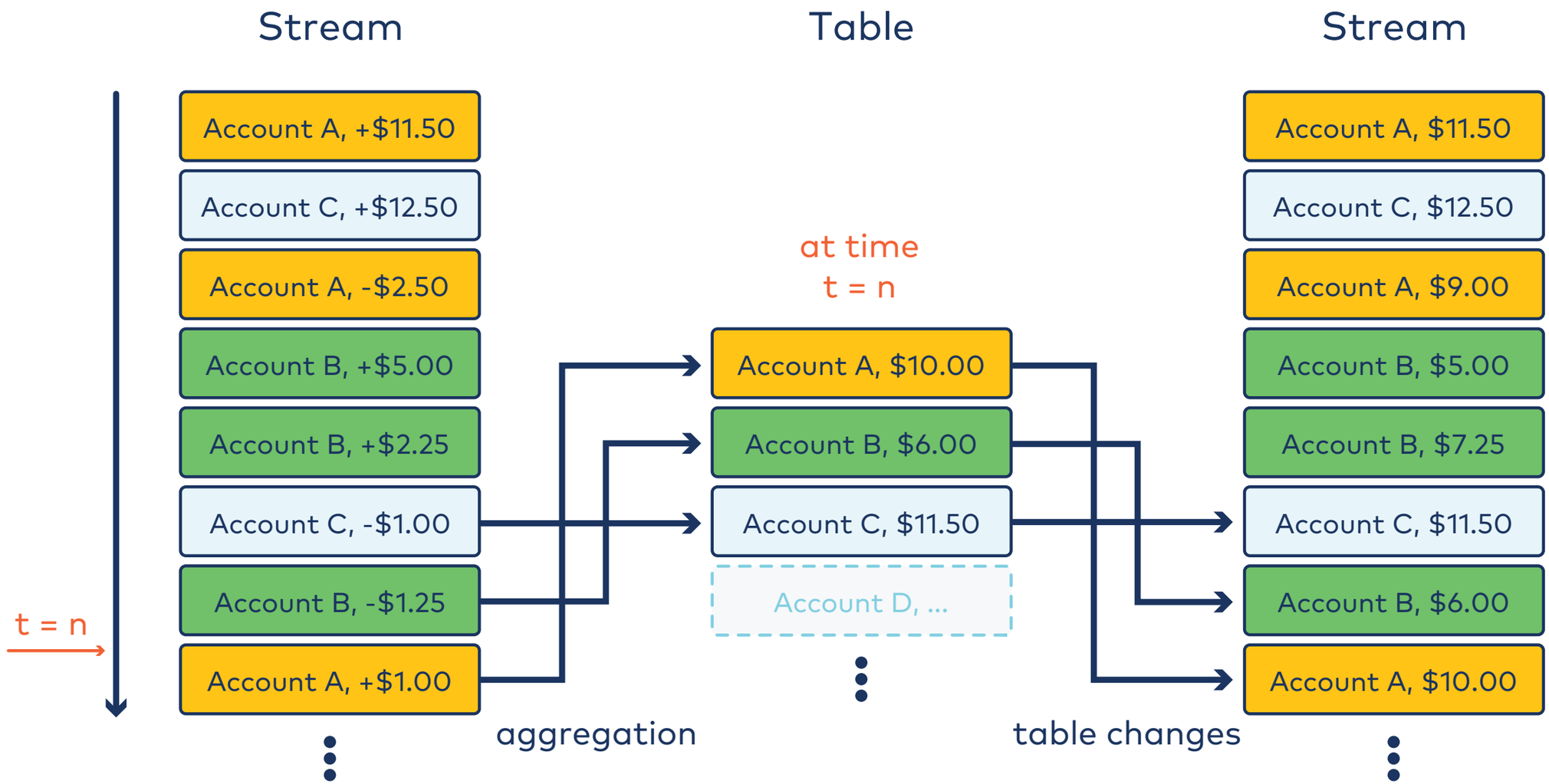
- You can turn a stream into a table by aggregating the stream with operations such as `COUNT()` or `SUM()`
- We can turn a table into a stream by capturing the changes made to the table—inserts, updates, and deletes—into a "change stream."

Streams
record history



Tables
represent state

Stream-Table Duality With Aggregation Example



Activity

Review the scenario on the left and determine the type of object from the right column that is best suited to storing it.

Scenario	Table/Stream
Checking account balance	Table? / Stream?
The past five years of experience for your resume	Table? / Stream?
Sequence of moves in a chess game	Table? / Stream?
State of a chess board at a given time	Table? / Stream?
Count of countries to which you have traveled	Table? / Stream?
Your addresses over the last five years for a visa application	Table? / Stream?
Items you have shopped for online	Table? / Stream?
RSVP responses from guests for a party you are having	Table? / Stream?

2b: How Do You Put Together a Kafka Streams Application?

Description

Any Java application that makes use of the Kafka Streams library is considered a Kafka Streams application. The computational logic of a Kafka Streams application is defined as a processor topology. A Kafka Streams Application written in Java has five clearly identifiable sections.

Kafka Streams Application Anatomy

Imports

```
code goes here
```

```
public class StreamsApp
{
   public static void main(String[] args)
   {
```

Config

```
      //code goes here
```

Streaming topology

```
      //code goes here
```

Shutdown behavior

```
      //code goes here
```

Start app

```
      //code goes here
```

```
   }
}
```

Kafka Streams Application Anatomy - Imports

Libraries available for writing Kafka Streams applications:

Group ID	Artifact ID	Version	Description	Req?
`org.apache.kafka`	`kafka-streams`	7.0.1-ccs	Base library for Kafka Streams	Yes
`org.apache.kafka`	`kafka-streams-scala_2.11`, `kafka-streams-scala_2.12`	7.0.1-ccs	Scala API for Kafka Streams	No
`org.apache.kafka`	`kafka-clients`	7.0.1-ccs	Apache Kafka® client library, contains built-in serializers/deserializers	Yes
`org.apache.avro`	`avro`	1.8.2	Apache Avro library	Avro only
`io.confluent`	`kafka-streams-avro-serde`	7.0.1	Confluent's Avro Serializer/Deserializer	Avro only

Kafka Streams Application Anatomy - Configuration

Imports

Config

```
Properties settings = new Properties();
settings.put(StreamsConfig.APPLICATION_ID_CONFIG,
             "streams-app-1");
settings.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG,
             "kafka-1:9092, kafka-2:9092, kafka-3:9092");
settings.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG,
             Serdes.String().getClass());
settings.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG,
             Serdes.Double().getClass());
// ...
```

Topology

```
//code goes here
```

Shutdown

```
//code goes here
```

Start app

```
//code goes here
```

Kafka Streams Application Anatomy - Topology

Imports

Config

```
// code goes here
```

Streaming topology

```
StreamsBuilder builder = new StreamsBuilder();

KStream<String, Double> temperatures =
   builder.stream("temp-topic");
KStream<String, Double> highTemps =
   temperatures.filter((key, value) -> value > 25);
highTemps.to("high-temps-topic");

Topology topology = builder.build();
```

Shutdown

```
//code goes here
```

Start app

```
//code goes here
```

Kafka Streams Application - Shutdown

Imports	
Config	`// code goes here`
Topology	`// code goes here`
	`KafkaStreams streams = new KafkaStreams(topology, settings);`

Shutdown behavior	`final CountDownLatch latch = new CountDownLatch(1);`
	`Runtime.getRuntime().addShutdownHook(new(Thread(() -> {`
	`  streams.close();`
	`  latch.CountDown();`
	`}));`

Start app	`//code goes here`

Kafka Streams Application - Start Streaming

Imports

Config

```
// code goes here
```

Streaming topology

```
// code goes here
```

```
KafkaStreams streams = new KafkaStreams(topology, settings);
```

Shutdown

```
// code goes here
```

Start app

```
try
{
   streams.start();
   latch.await();
}
catch(final Throwable e) { /* ... */ }
System.exit(0);
```

Summary: Full Program (1)

```
1  // imports
2
3  public class StreamsApp
4  {
5     public static void main(String[] args)
6     {
7        Properties settings = new Properties();
8        settings.put(StreamsConfig.APPLICATION_ID_CONFIG", streams-app-1");
9        settings.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG,
10                    "kafka-1:9092, kafka-2:9092, kafka-3:9092");
11       settings.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass());
12       settings.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.Double().getClass());
13       // ...
14
15       StreamsBuilder builder = new StreamsBuilder();
16
17       KStream<String, Double> temperatures = builder.stream("temp-topic");
18       KStream<String, Double> highTemps = temperatures.filter((key, value) -> value > 25);
19       highTemps.to("high-temps-topic");
```

Summary: Full Program (2)

```
21        Topology topology = builder.build();
22
23        KafkaStreams streams = new KafkaStreams(topology, settings);
24
25        final CountDownLatch latch = new CountDownLatch(1);
26        Runtime.getRuntime().addShutdownHook(new(Thread(() -> {
27           streams.close();
28           latch.CountDown();
29        }));
30
31        try
32        {
33           streams.start();
34           latch.await();
35        }
36        catch(final Throwable e) { /* ... */ }
37        System.exit(0);
38     }
39  }
```

Alternate Serde Configuration

In the running example, we specified default Serdes in the Config.

Alternative: specify Serdes upon each use. Here is what is different:

```
final Serde<String> stringSerde = Serdes.String();
final Serde<Double> doubleSerde = Serdes.Double();
```

Streaming topology

```
StreamsBuilder builder = new StreamsBuilder();

KStream<String, Double> temperatures =
   builder.stream("temp-topic",
                  Consumed.with(stringSerde, doubleSerde));
KStream<String, Double> highTemps =
   temperatures.filter((key, value) -> value > 25);
highTemps.to("high-temps-topic",
             Produced.with(stringSerde, doubleSerde));

Topology topology = builder.build();
```

Lab: Anatomy of a Kafka Streams App

Please work on **Lab 2a: Anatomy of a Kafka Streams App**

Refer to the Exercise Guide

2c: What are Some Operations You Can Use To Transform Streams?

Description

The `KStream` and `KTable` interfaces support stateless and stateful transformations. Stateless transformations `mapValues`, `flatMapValues`, `filter`, etc. Some of the stateless transformations like `map` and `flatMap` mark the stream for repartitioning.

Transforming Data

- Kafka streams supports a number of transformation operation using the objects `KStream` and `KTable`.
- These operations can be translated into one or more connected processors into the underlying processor topology.
  - Some `KStream` transformations may generate one or more `KStream` objects or a `KTable` object.
  - All `KTable` transformation operations can only generate another `KTable`.
  - All of these transformation methods can be chained together to compose a complex processor topology.
  - The transformation operations fall into these categories:
    - Stateless
    - Stateful

Stateless Transforming

Stateless transformations do not require state for processing and hence **do not require a state store** associated with the stream processor.

Stateless Operations - Mapping

key: number
value: object with shape and color

key: color
value: shape

INPUT Transformation OUTPUT

map

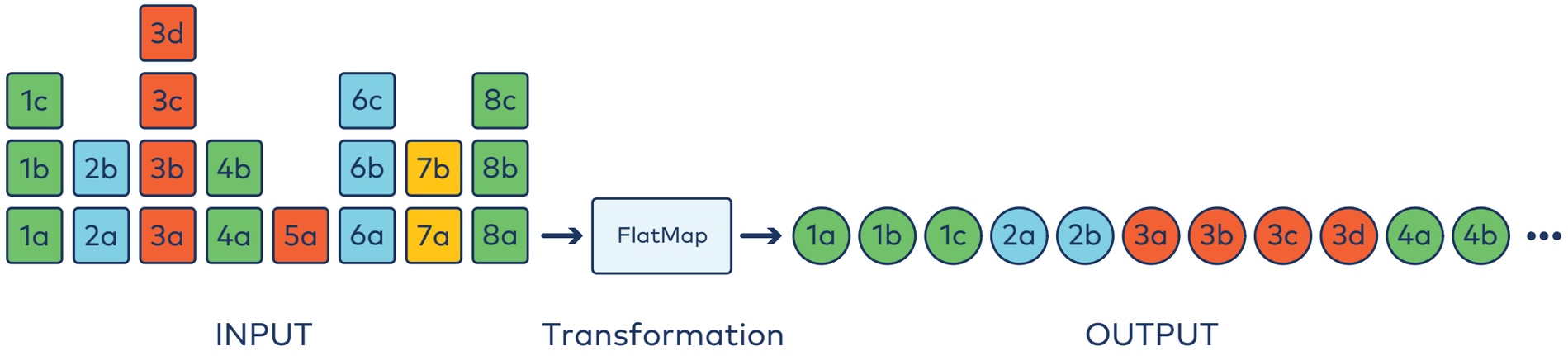
- new key and new value

```
KStream<Color, Shape> result =
    stream.map((key, value) ->
                KeyValue.pair(value.color, value.shape));
```

map values

```
KStream<Integer, String> result =
    stream.mapValues(value -> value.toUpperCase());
```

Stateless Operations - **flatMap**



flat map

- new key and new value

```
KStream<Char, String> words =
    sentences.flatMap((key, value) ->
                        KeyValue.pair(value.substring(0,1),
                                      Arrays.asList(value.split("\\W+"))));
```

flat map values

```
KStream<byte[], String> words =
    sentences.flatMapValues(value -> Arrays.asList(value.split("\\W+")));
```

Stateless Operations - Filtering

key: shape
value: color

INPUT Transformation OUTPUT

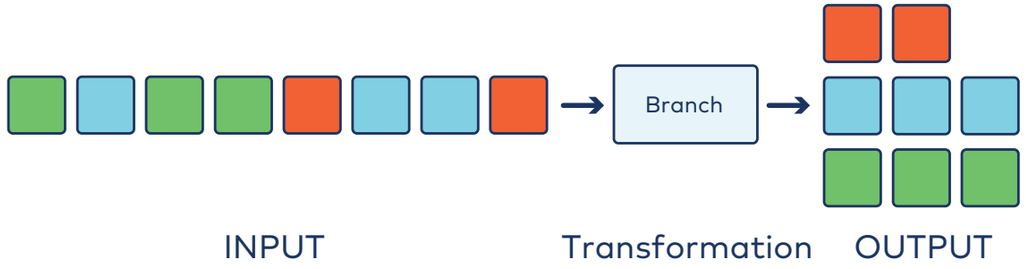
filter

```
KStream<Shape, Color> nonOrangeItems =
    stream.filter((key, value) -> !value.equals("orange"));
```

inverse filter

```
KStream<Shape, Color> nonOrangeItems =
    stream.filterNot((key, value) -> value.equals("orange"));
```

Stateless Operations - Branching



branch

```
KStream<String, Long>[] branches = stream.branch(
    (key, value) -> key.startsWith("A"), /* predicate 1 */
    (key, value) -> key.startsWith("B"), /* predicate 2 */
    (key, value) -> true                 /* predicate 3 */
  );
```

Returns 3 new `KStream`s:

- `branches[0]` contains all records whose keys start with "A"
- `branches[1]` contains all records whose keys start with "B"
- `branches[2]` contains all other records

Activity

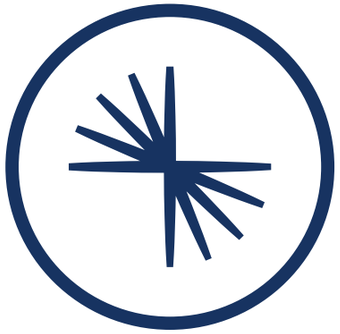
True or false?

1. Aggregation is applied to records of the same key.
2. Grouping is a prerequisite for aggregation.
3. You cannot run `groupBy` on a `KStream` or a `KTable`.

Lab: Working With JSON

Please work on **Lab 2b: Working With JSON**

Refer to the Exercise Guide



CONFLUENT

Global Education

Module Overview

This module contains three lessons:

- What Can ksqlDB Do for You and How Do You Interact with It?
- What Kinds of Queries Can You Write in ksqlDB and How Do They Work?
- How Can You Use ksqlDB to Integrate with Kafka Connect?

Where this fits in:

- Hard Prerequisite: Working with Kafka Streams
- Recommended Prerequisite: Core Branch of Developer Course
- Recommended Follow-Up: Using ksqlDB

3a: What Can ksqlDB Do for You and How Do You Interact with It?

Description

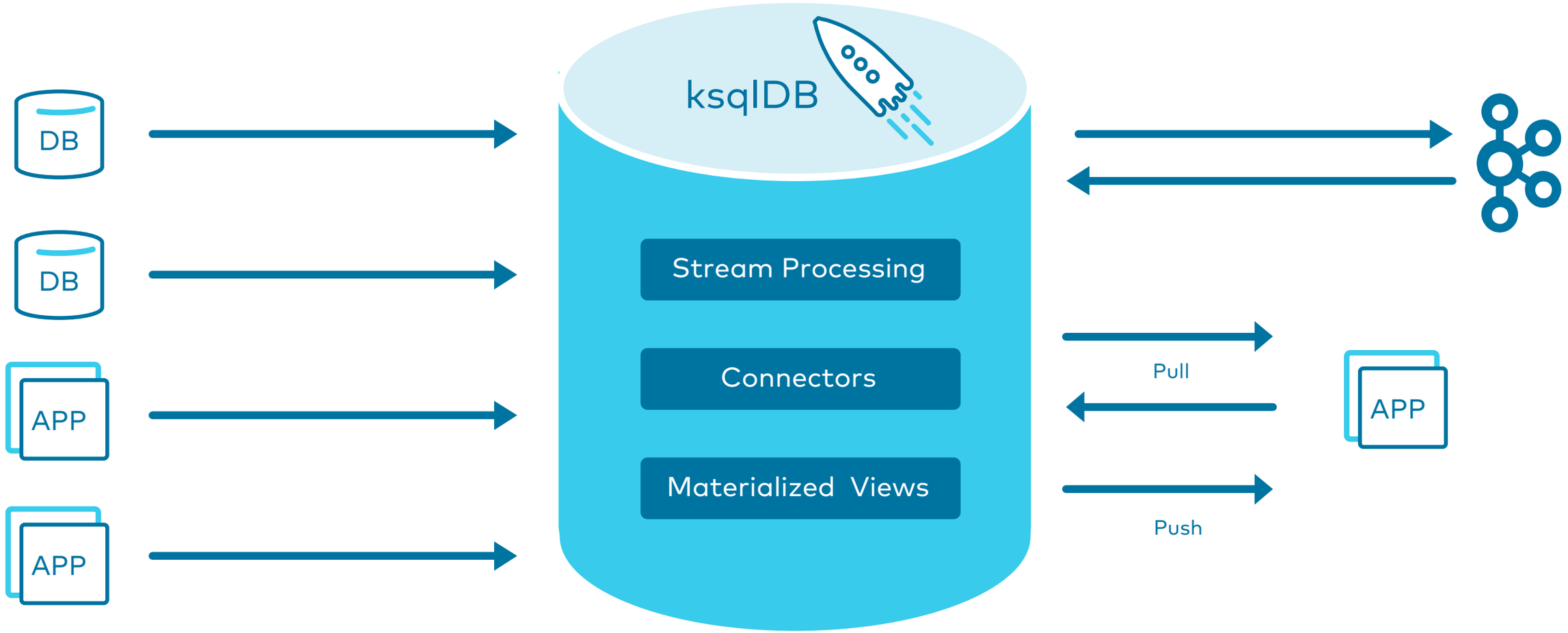
Confluent ksqlDB brings the power of Kafka Streams with a SQL-like syntax. ksqlDB offers a single solution for collecting streams of data, enriching them, and serving queries on new derived streams and tables. ksqlDB can be accessed using a CLI, REST API, and Confluent Control Center.

What is ksqlDB?

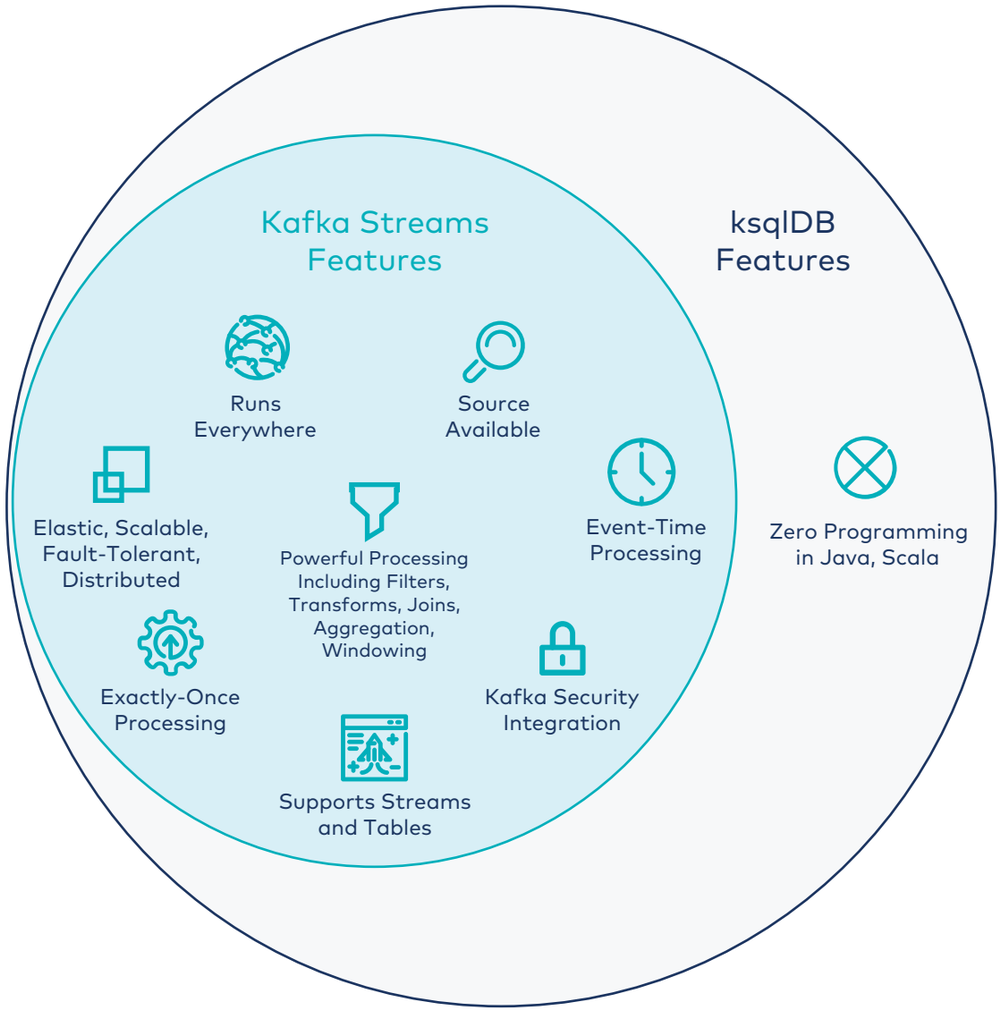
ksqlDB...

- Is an event-streaming database
- Is built on top of Apache Kafka and Kafka Streams
- Uses SQL-like syntax to build applications that respond to Data in Motion
- Drafts materialized views
- Receives real-time push updates or pulls current state on demand

Simplify Your Stream Processing Architecture



Why ksqlDB?



ksqlDB vs. Kafka Streams

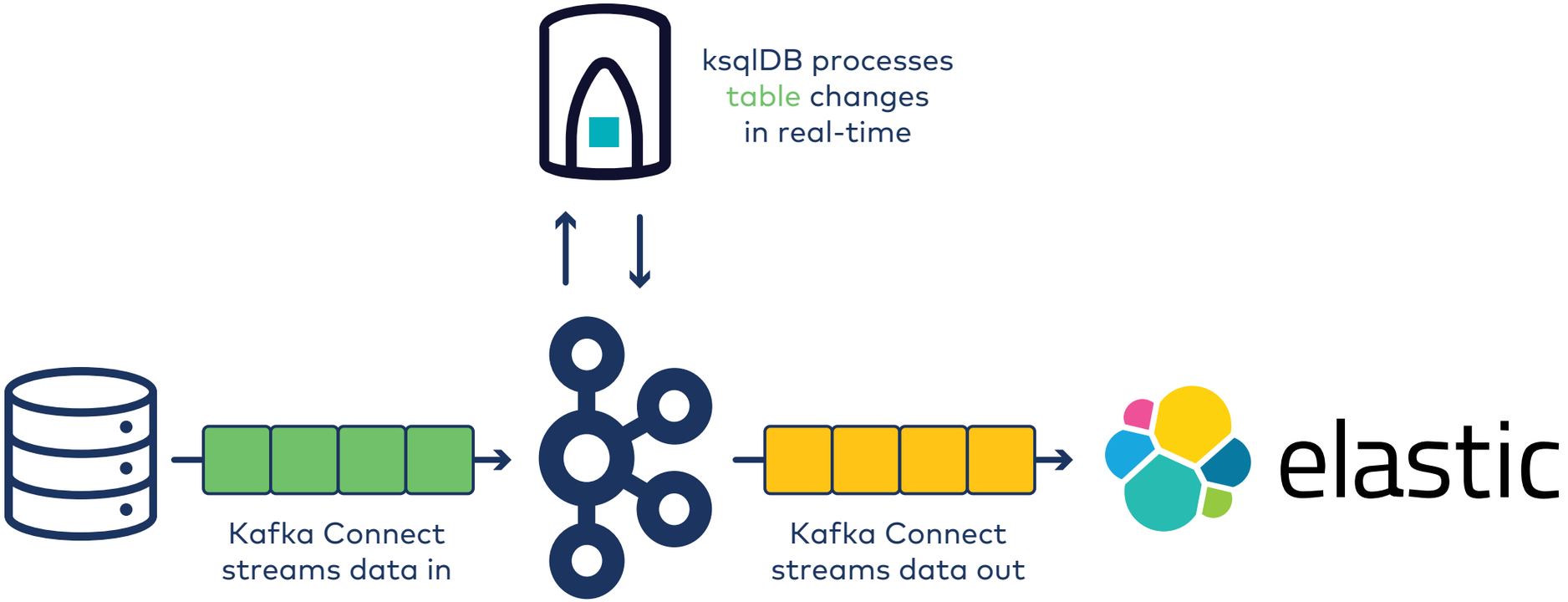
Differences	ksqlDB	Kafka Streams
You write…	SQL statements	JVM applications
Graphical UI?	Yes, in Confluent Control Center and Confluent Cloud	No
Console?	Yes	No
Data formats	Avro, Protobuf, JSON, JSON_SR, CSV	Any data format, including Avro, JSON, CSV, Protobuf, XML
REST API included?	Yes	No, but you can implement your own
Runtime included?	Yes, the ksqlDB server	Applications run as standard JVM processes

SQL-like Semantics - Streaming ETL Sample Use Case

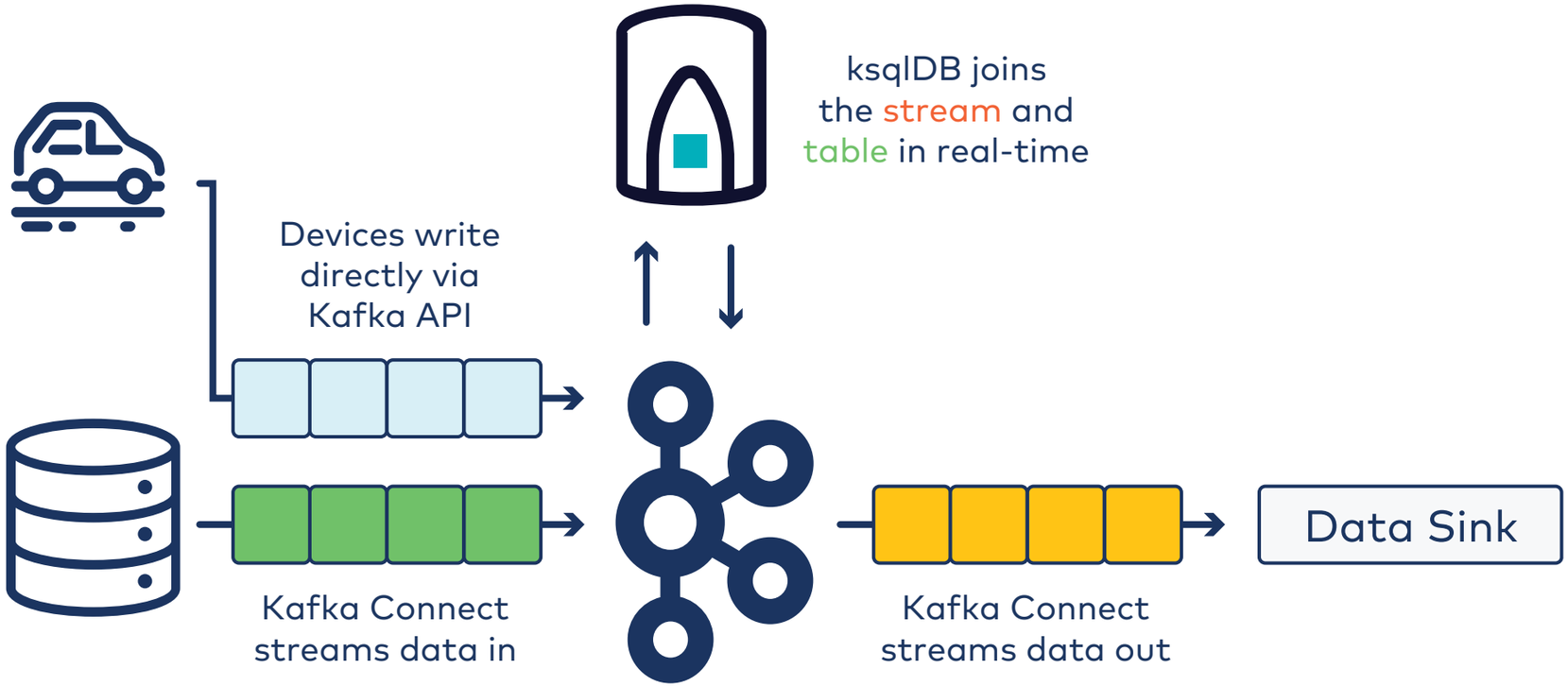
Filter, cleanse, process data while it is moving...

```
CREATE STREAM clicks_from_vip_users AS
   SELECT user_id, u.country, page, action
   FROM   clickstream c
          LEFT JOIN users u
          ON c.user_id = u.user_id
   WHERE  u.level = 'Platinum'
   EMIT CHANGES;
```

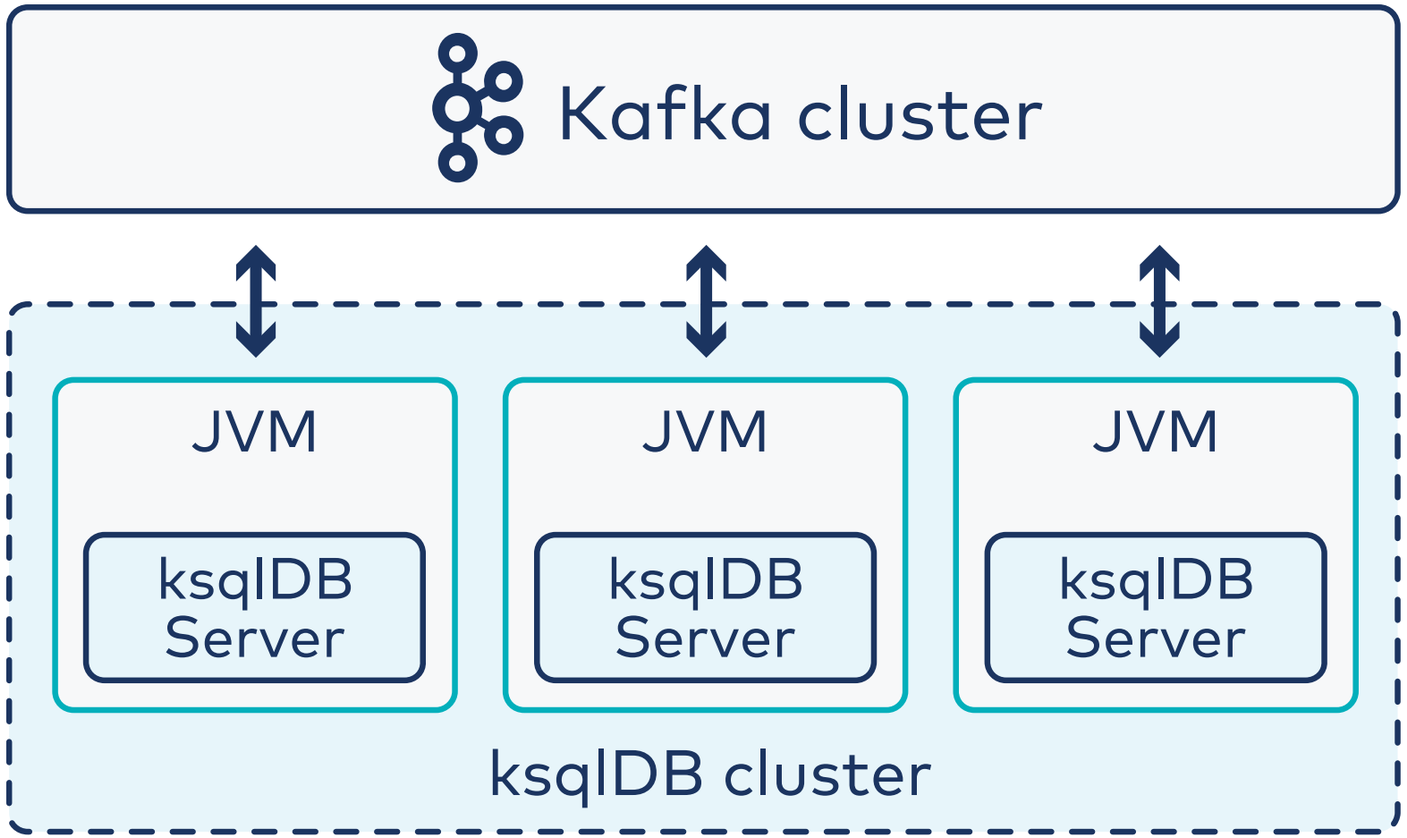
Sample Use Cases - Real-time Data Enrichment



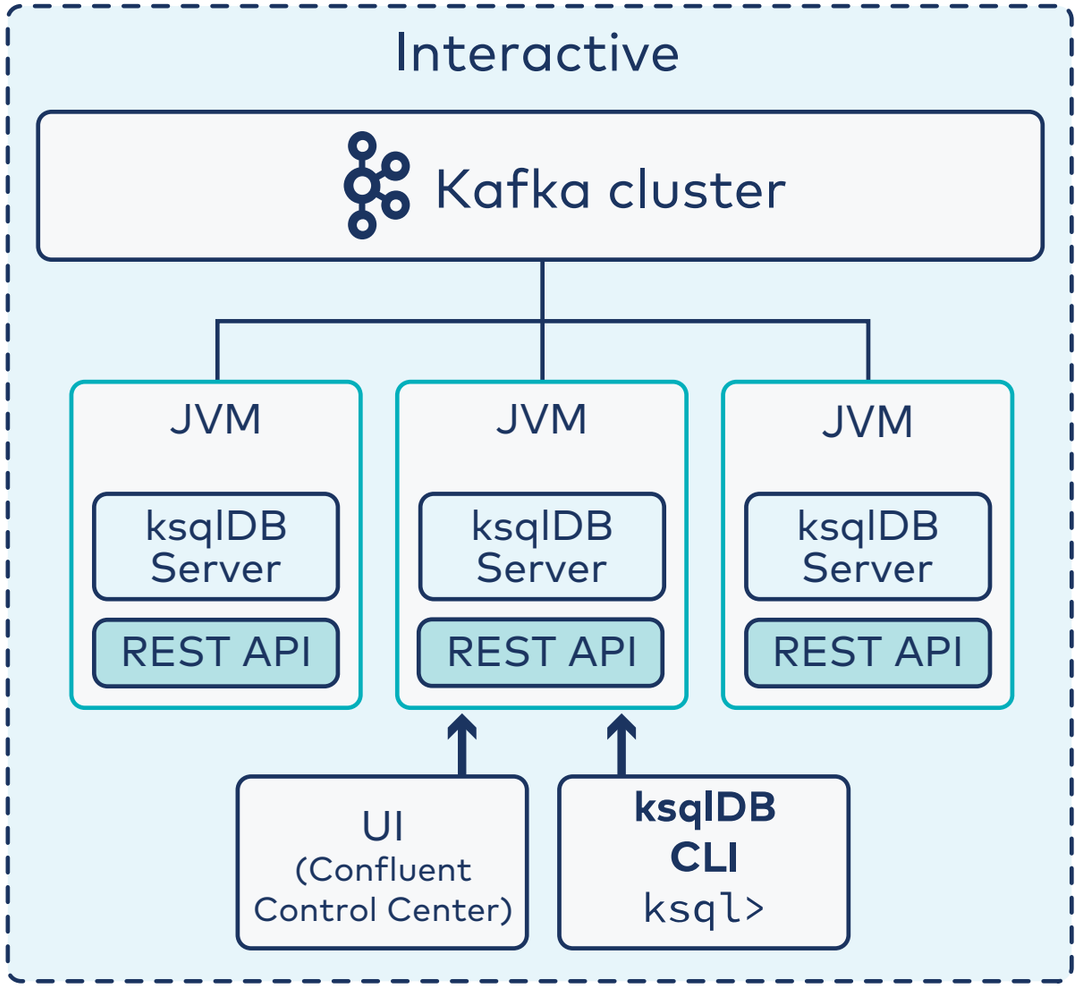
Sample Use Cases - Real-time Data Enrichment



ksqlDB Architecture



ksqlDB Deployment Mode: Interactive

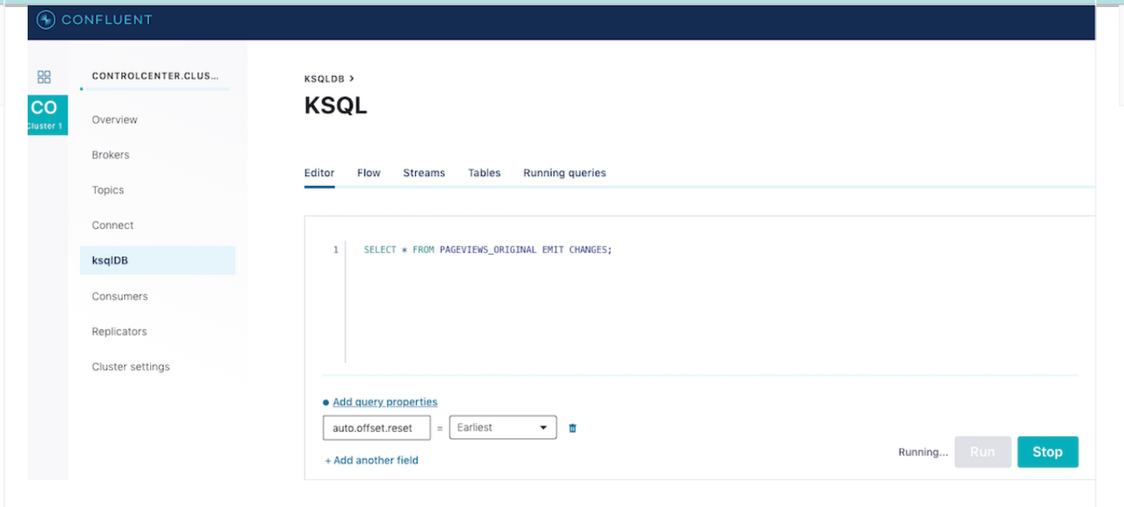


Interactive ksqlDB Usage

1. CLI

```
ksql >
```

2. GUI



3. REST API

```
POST /query
```

ksqlDB - CLI


```
Copyright 2017-2021 Confluent Inc.

CLI v0.17.0, Server v0.17.0 located at http://ksqldb-server:8088
Server Status: RUNNING

Having trouble? Type 'help' (case-insensitive) for a rundown of how things work!

ksql>
```

ksqlDB CLI Output - Tabular

Tabular output is the typical output for ksqlDB...

```
ksql> OUTPUT TABULAR;
Output format set to TABULAR
ksql> SELECT rowtime,userid, pageid, regionid, gender FROM pageviews_female_1 EMIT CHANGES LIMIT 5;
+------------------+------------------+------------------+------------------+------------------+
|ROWTIME           |USERID            |PAGEID            |REGIONID          |GENDER            |
+------------------+------------------+------------------+------------------+------------------+
|1649928071166     |User_1            |Page_92           |Region_1          |FEMALE            |
|1649928071166     |User_1            |Page_54           |Region_1          |FEMALE            |
|1649928071166     |User_1            |Page_11           |Region_1          |FEMALE            |
|1649928071166     |User_1            |Page_75           |Region_1          |FEMALE            |
|1649928071166     |User_1            |Page_45           |Region_1          |FEMALE            |
Limit Reached
Query terminated
```

But what if you want to customize...

ksqlDB CLI Output - Adjusting Column Width

```
ksql> SET CLI COLUMN-WIDTH 13;
ksql> SELECT rowtime,userid, pageid, regionid, gender FROM pageviews_female_1 EMIT CHANGES LIMIT 5;
+-------------+-------------+-------------+-------------+-------------+
|ROWTIME      |USERID       |PAGEID       |REGIONID     |GENDER       |
+-------------+-------------+-------------+-------------+-------------+
|1649928174292|User_4       |Page_84      |Region_1     |FEMALE       |
|1649928174292|User_4       |Page_62      |Region_1     |FEMALE       |
|1649928174292|User_4       |Page_93      |Region_1     |FEMALE       |
|1649928174292|User_4       |Page_65      |Region_1     |FEMALE       |
|1649928174292|User_4       |Page_16      |Region_1     |FEMALE       |
Limit Reached
Query terminated
```

ksqlDB CLI Output - JSON

```
ksql> OUTPUT JSON;
Output format set to JSON
ksql> SELECT rowtime,userid, pageid, regionid, gender FROM pageviews_female_1 EMIT CHANGES LIMIT 3;
{
  "queryId" : "none",
  "schema" : "`ROWTIME` BIGINT, `USERID` STRING, `PAGEID` STRING, `REGIONID` STRING, `GENDER` STRING"
}
{
  "columns" : [ 1649928217394, "User_8", "Page_76", "Region_1", "FEMALE" ]
}
{
  "columns" : [ 1649928217394, "User_2", "Page_73", "Region_1", "FEMALE" ]
}
{
  "columns" : [ 1649928217394, "User_2", "Page_20", "Region_1", "FEMALE" ]
}
Limit Reached
Query terminated
```

Interacting with ksqlDB - Confluent Control Center

ksqlDB REST API - Sample Request

```
POST /ksql HTTP/1.1
Accept: application/vnd.ksql.v1+json
Content-Type: application/vnd.ksql.v1+json

{
  "ksql": "CREATE STREAM pageviews_home
            AS SELECT *
            FROM pageviews_original
            WHERE pageid='home';
          CREATE STREAM pageviews_alice
            AS SELECT *
            FROM pageviews_original
            WHERE userid='alice';",
  "streamsProperties": {
    "ksql.streams.auto.offset.reset": "earliest"
  }
}
```

ksqlDB REST API - Sample Response

```
HTTP/1.1 200 OK
Content-Type: application/vnd.ksql.v1+json
[ { "statementText":"CREATE STREAM pageviews_home ...
    "commandId":"stream/PAGEVIEWS_HOME/create",
    "commandStatus": {
      "status":"SUCCESS",
      "message":"Stream created and running"
    },
    "commandSequenceNumber":10
  }, {
    "statementText":"CREATE STREAM pageviews_alice ...
    "commandId":"stream/PAGEVIEWS_ALICE/create",
    "commandStatus": {
      "status":"SUCCESS",
      "message":"Stream created and running"
    },
    "commandSequenceNumber":11 } ]
```

ksqlDB - Java Client

```
KsqlObject row = new KsqlObject()
    .put("PERSON", "robin")
    .put("LOCATION", "Manchester");

//insert data into stream
client.insertInto("MOVEMENTS", row).get();

//read from stream
client.streamQuery("SELECT * FROM PERSON_STATS EMIT CHANGES;")
      .thenAccept(streamedQueryResult -> {
        RowSubscriber subscriber = new RowSubscriber();
        streamedQueryResult.subscribe(subscriber);
       }).exceptionally(e -> {
          System.out.println("Request failed: " + e);
          return null;
       });
```

ksqlDB Query vs. Kafka Streams Code

ksqlDB code:

```
CREATE STREAM fraudulent_payments AS
    SELECT fraudProbability(data)
    FROM payments
    WHERE fraudProbability(data) > 0.8
    EMIT CHANGES;
```

Partial Scala code (more in your guide):

```
val builder: StreamsBuilder = new StreamsBuilder()
val fraudulentPayments: KStream[String, Payment] = builder
  .stream[String, Payment]("payments")
  .filter((_ ,payment) => payment.fraudProbability > 0.8)
fraudulentPayments.to("fraudulent-payments-topic")

val config = new java.util.Properties
config.put(StreamsConfig.APPLICATION_ID_CONFIG,
           "fraud-filtering-app")
config.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG,
           "kafka-broker1:9092")

val streams: KafkaStreams = new KafkaStreams(builder.build(),
                                             config)
streams.start()
```

ksqlDB and Licensing

- The **Standalone ksqlDB distribution** is free to use and the source is available on GitHub. This edition is licensed under the Confluent Community License and is not commercially supported by Confluent.
- Each new version of the **Confluent Platform distribution** wraps a specific, previously shipped version of the standalone distribution. Enterprise support is available through a subscription to Confluent Platform.

Activity

Answer for yourself, "Why ksqlDB?" Where will you use it in your job?

Think about this and let's share a few ideas with the group.

Lab: Introduction to ksqlDB

Please work on **Lab 3a: Introduction to ksqlDB**

Refer to the Exercise Guide

Lab: Using the ksqlDB REST API

Please work on **Lab 3b: Using the ksqlDB REST API**

Refer to the Exercise Guide

3b: What Kinds of Queries Can You Write in ksqlDB and How Do They Work?

Description

ksqlDB supports two categories of statements: Data Definition Language (DDL) and Data Manipulation Language (DML). The ksqlDB query lifecycle has seven steps, and includes the creation of the logical and physical plan.

ksqlDB Language

ksqlDB supports two categories of statements:

Data Definition Language (DDL)	Data Manipulation Language (DML)
Statements include: • `CREATE STREAM` • `CREATE TABLE` • `DROP STREAM` • `DROP TABLE` • `CREATE STREAM AS SELECT` (CSAS) • `CREATE TABLE AS SELECT` (CTAS)	Statements include: • `SELECT` • `INSERT INTO` • `INSERT INTO VALUES` • `CREATE STREAM AS SELECT` (CSAS) • `CREATE TABLE AS SELECT` (CTAS)

ksqlDB Queries

Pull Query

takes a snapshot of the data in a ksqlDB Table and returns a single result or result set

Push Query

continuous query - as new data arrive, they are examined and added to the results of the query

Persistent Query

server-side queries that run indefinitely processing rows of events. You issue persistent queries by deriving new streams and new tables from existing streams or tables

ksqlDB Query Lifecycle

ksqlDB statements and queries have a lifecycle with the following steps:

1. You **register a ksqlDB stream or table** from an existing Kafka topic with a DDL statement, like `CREATE STREAM <my-stream> WITH <topic-name>`.
2. You **express your app** by using a SQL statement, like `CREATE TABLE AS SELECT FROM <my-stream>`.
3. ksqlDB **parses your statement** into an abstract syntax tree (AST).
4. ksqlDB uses the AST and **creates the logical plan** for your statement.
5. ksqlDB uses the logical plan and **creates the physical plan** for your statement.
6. ksqlDB **generates and runs the Kafka Streams application**.
7. You **manage the application** as a `STREAM` or `TABLE` with its corresponding persistent query.

Registering a ksqlDB Stream or Table

Create a stream from the Kafka topic `authorizations`, which has a JSON payload:

```
CREATE STREAM authorization_attempts
  (card_number VARCHAR, attemptTime BIGINT, ...)
  WITH (kafka_topic='authorizations', value_format='JSON');
```

Create a table from the Kafka topic `my-users-topic`, which has a JSON payload:

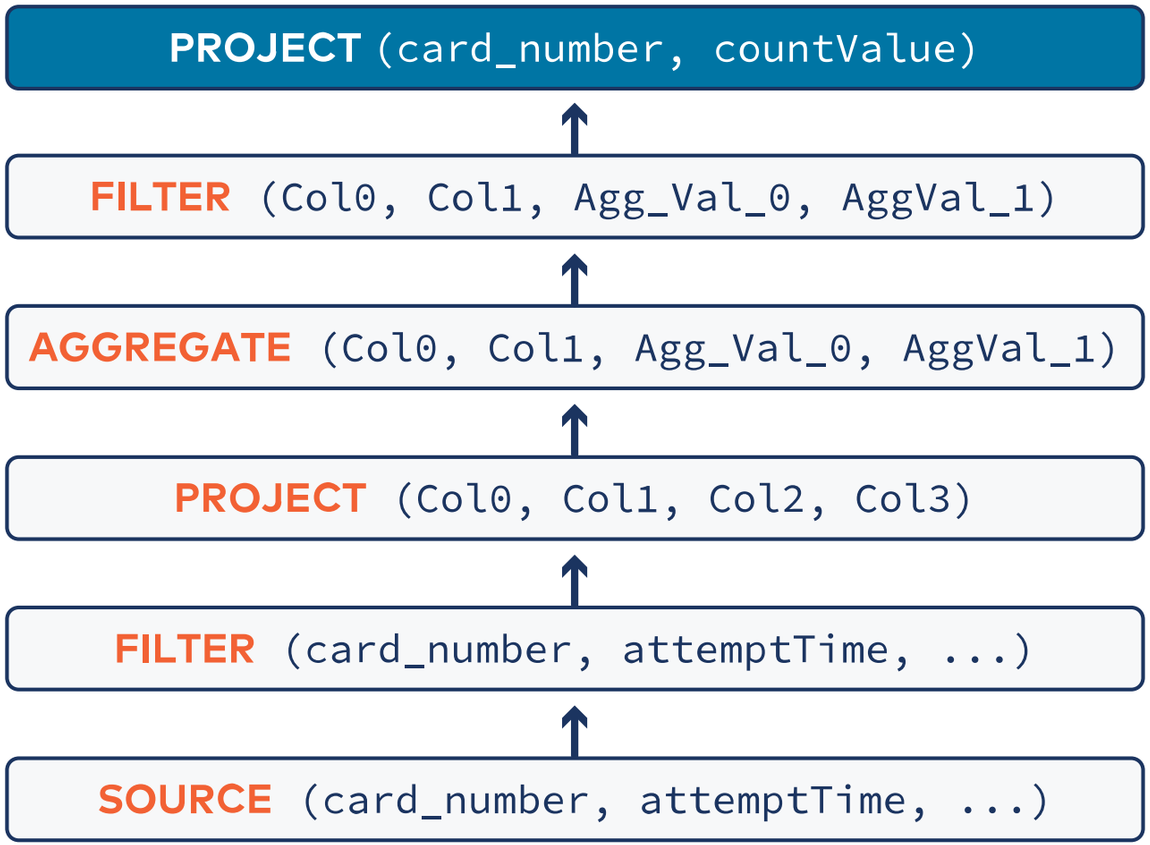
```
CREATE TABLE users (usertimestamp BIGINT, user_id VARCHAR PRIMARY KEY,
                            gender VARCHAR, region_id VARCHAR)
  WITH (VALUE_FORMAT = 'JSON', KAFKA_TOPIC = 'my-users-topic');
```

Express an Application as a SQL Statement

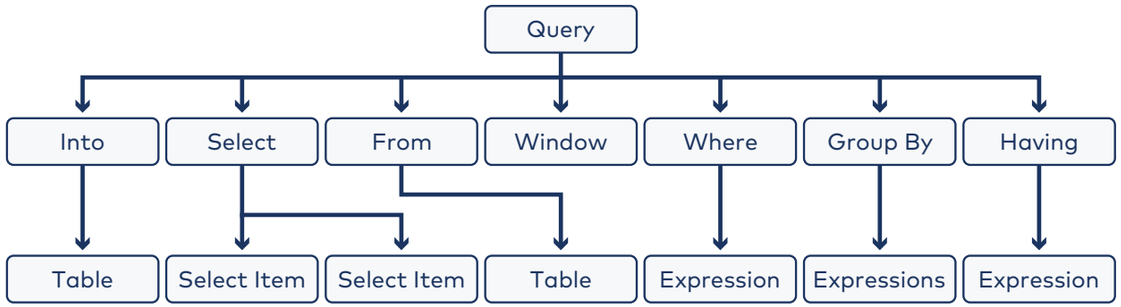
The following DML statement creates a `possible_fraud` table from the `authorization_attempts` stream:

```
CREATE TABLE possible_fraud AS
  SELECT card_number, count(*)
  FROM authorization_attempts
  WINDOW TUMBLING (SIZE 5 SECONDS)
  WHERE region = 'west'
  GROUP BY card_number
  HAVING count(*) > 3
  EMIT CHANGES;
```

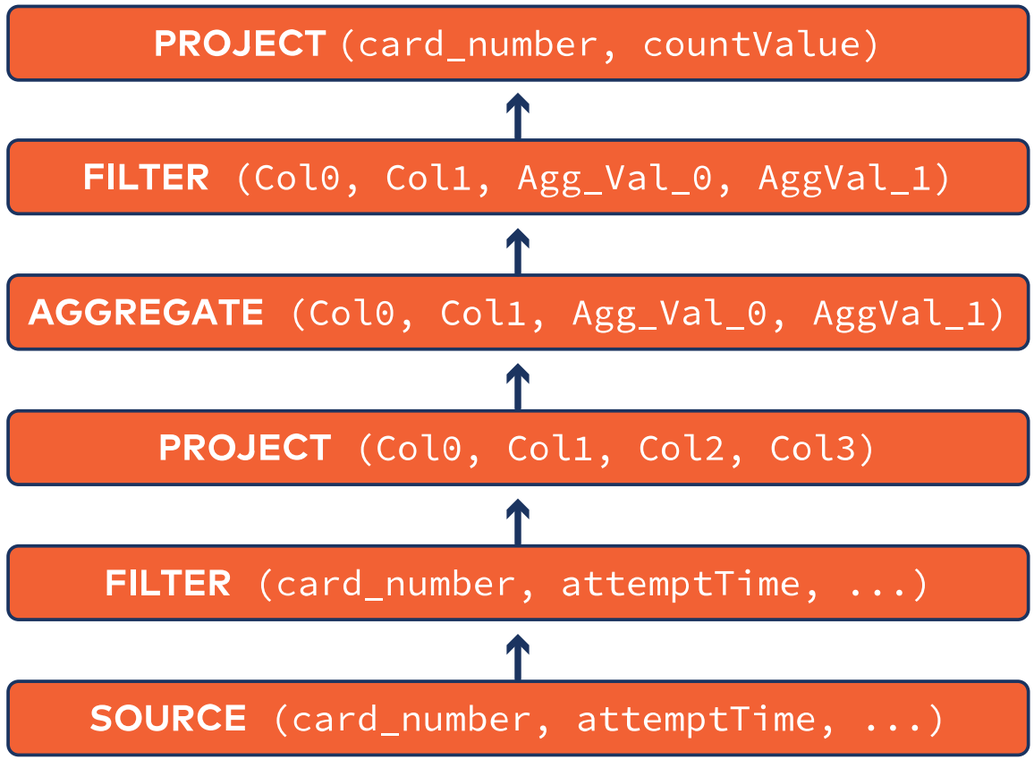
ksqlDB Creates the Logical Plan




```
CREATE TABLE possible_fraud AS
   SELECT card_number, count(*)
   FROM authorization_attempts
   WINDOW TUMBLING (SIZE 5 SECONDS)
   WHERE region = 'west'
   GROUP BY card_number
   HAVING count(*) > 3
   EMIT CHANGES;
```



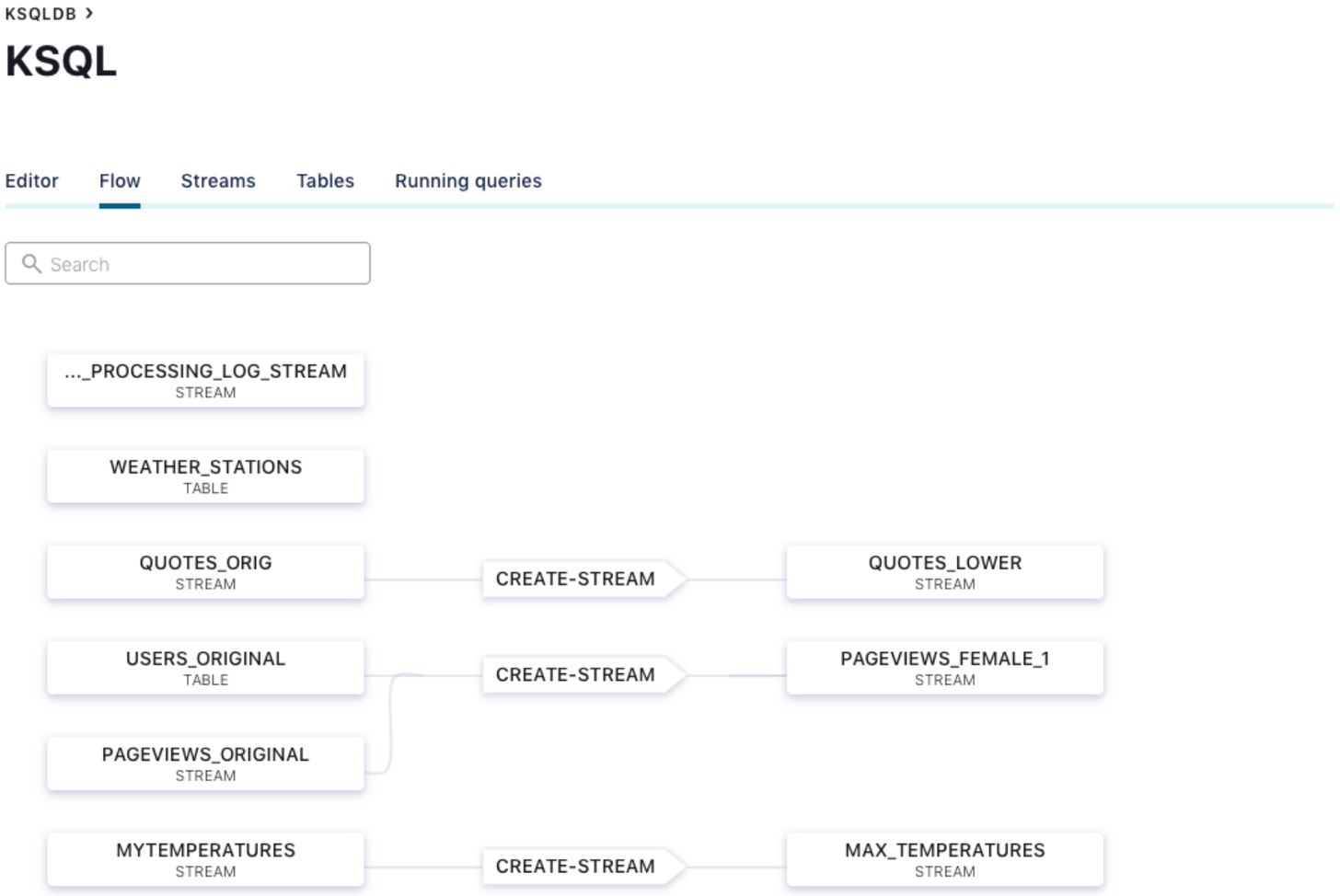
ksqlDB Creates the Physical Plan




```
CREATE TABLE possible_fraud AS
   SELECT card_number, count(*)
   FROM authorization_attempts
   WINDOW TUMBLING (SIZE 5 SECONDS)
   WHERE region = 'west'
   GROUP BY card_number
   HAVING count(*) > 3
   EMIT CHANGES;
```

```
outStream = inStream.filter(...)
                    .select(...)
                    .rekey(...)...
                    .groupBy(...)...
                    .aggregate(...)...
                    .filter(...)
                    .select(...);
```

Exploring Data Flow with Confluent Control Center - Overall Flow



Exploring Data Flow with Confluent Control Center - Stream Join

CREATE STREAM

```
1  CREATE STREAM PAGEVIEWS_FEMALE_1 WITH (KAFKA_TOPIC='PAGEVIE
2    USERS_ORIGINAL.USERID USERID,
3    PAGEVIEWS_ORIGINAL.PAGEID PAGEID,
4    USERS_ORIGINAL.REGIONID REGIONID,
5    USERS_ORIGINAL.GENDER GENDER
6  FROM PAGEVIEWS_ORIGINAL PAGEVIEWS_ORIGINAL
7  LEFT OUTER JOIN USERS_ORIGINAL USERS_ORIGINAL ON ((PAGEVIEW
8  WHERE ((USERS_ORIGINAL.GENDER = 'FEMALE') AND (USERS_ORIGIN
9  EMIT CHANGES;
```

PAGEVIEWS_FEMALE_1

Data structure	STREAM
Total rows	995527
Updates /sec	0

Messages Schema

Filter by keyword

ROWTIME	ROWKEY	USERID	P
1589922329099	User_4	User_4	P
1589922329099	User_4	User_4	P
1589922329099	User_4	User_4	P

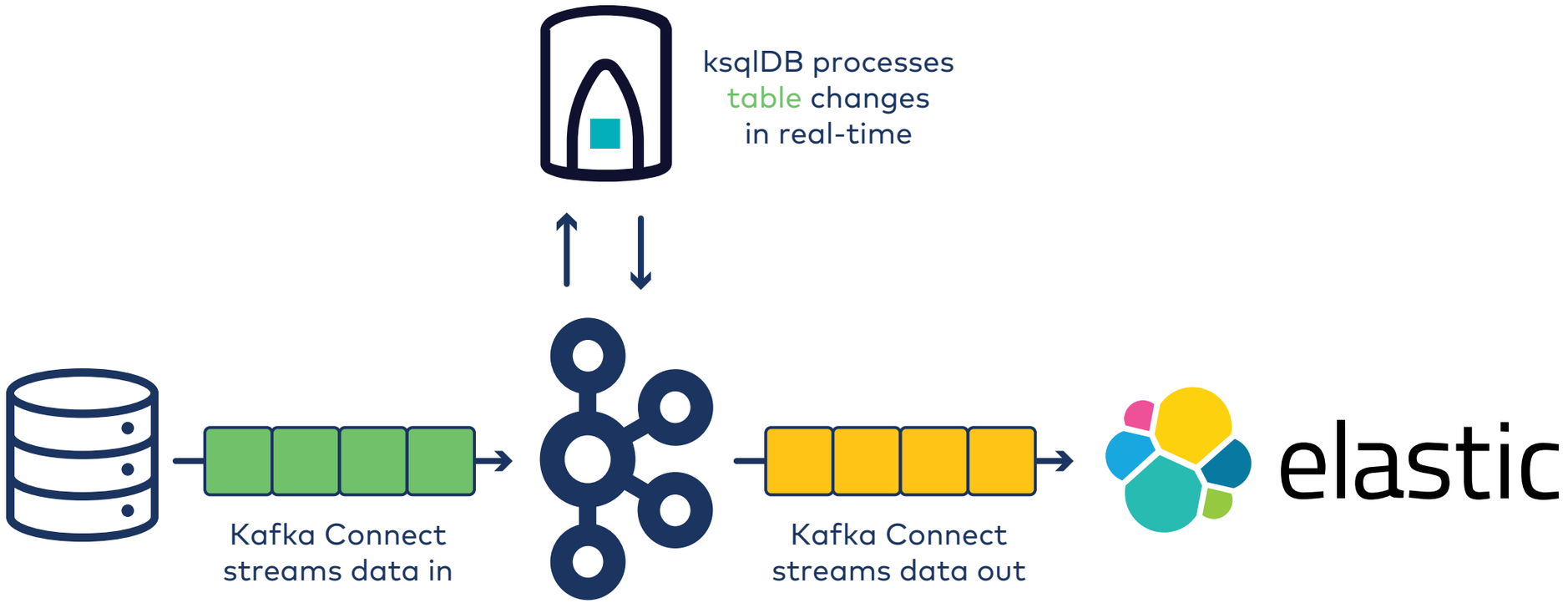
3c: How Can You Use ksqlDB to Integrate with Kafka Connect?

Description

ksqlDB supports running connectors directly on its servers as compared to needing to run a separate Kafka Connect cluster for capturing events. Also, ksqlDB can run pre-built connectors in embedded mode.

Connect Integration

Manage Source and Sink Connectors Directly Through ksqlDB



Managing Connectors from ksqlDB CLI or Confluent Control Center

Explore:

- See a list of all connectors

```
SHOW CONNECTORS
LIST CONNECTORS
```

- View detailed info about a connector

```
DESCRIBE CONNECTOR
```

Manage:

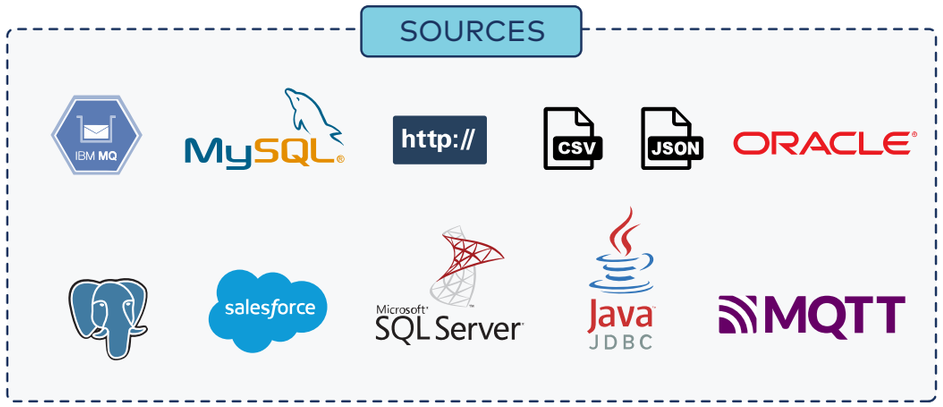
- Create connectors

```
CREATE CONNECTOR
```

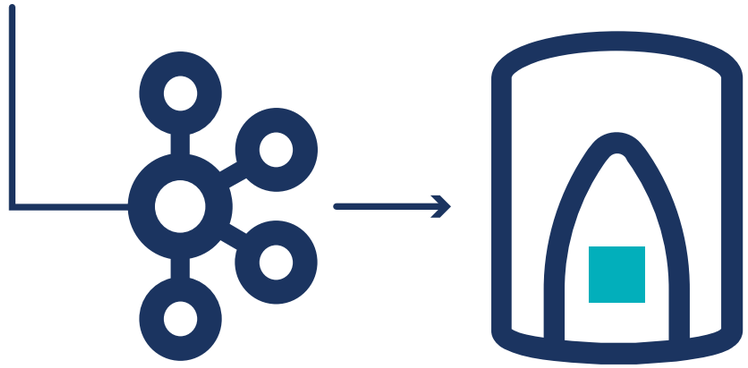
- Remove connectors

```
DROP CONNECTOR
```

Creating a Source Connector

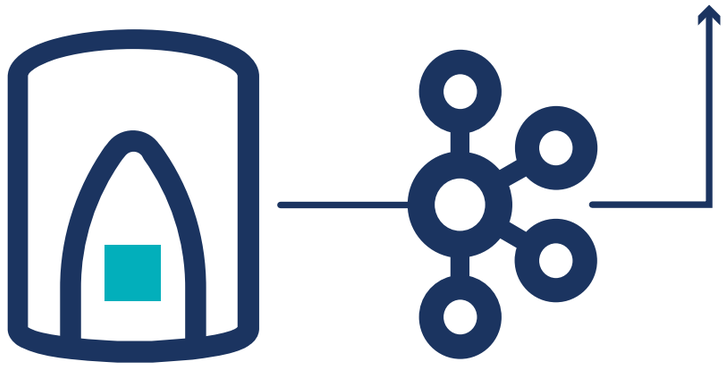



```
CREATE SOURCE CONNECTOR JDBC_SOURCE_EX WITH(
    "connector.class"=
        'io.confluent.connect.jdbc.JdbcSourceConnector',
    "connection.url"=
        'jdbc:postgresql://localhost:5432/my.db',
    "mode"='bulk',
    "topic.prefix"='jdbc-',
    "table.whitelist"='users',
    "key"='username');
```



Creating a Sink Connector

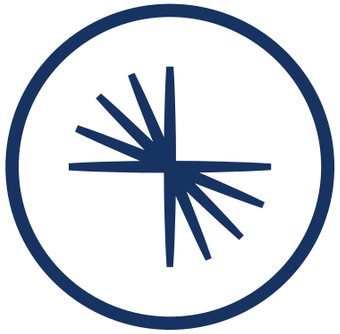
```
CREATE SINK CONNECTOR ELASTIC_SINK_EX WITH(
    "connector.class"=
       'ElasticsearchSinkConnector',
    "connection.url"=
       'http://elasticsearch:9200',
    "topics"='orders');
```



Lab: Creating Connectors With ksqlDB

Please work on **Lab 3c: Creating Connectors With ksqlDB**

Refer to the Exercise Guide



CONFLUENT

Global Education

Module Overview

This module contains two lessons:

- What Syntax Details Do You Need to Know About ksqlDB SQL?
- What Are Some Examples of How To Use ksqlDB for Manipulating Data?

Where this fits in:

- Hard Prerequisite: Introduction to ksqlDB
- Recommended Follow-Up: Time and Windowing, then any of the intermediate modules that build on it

4a: What Syntax Details Do You Need to Know About ksqlDB SQL?

Description

We explore data definition, message format, data types, and importance of using a key to make it easy to use the ksqlDB SQL language.

Data Definition

Stream:

- A stream is a partitioned, immutable, append-only collection that represents a series of historical facts
- To create a stream, use the `CREATE STREAM` command

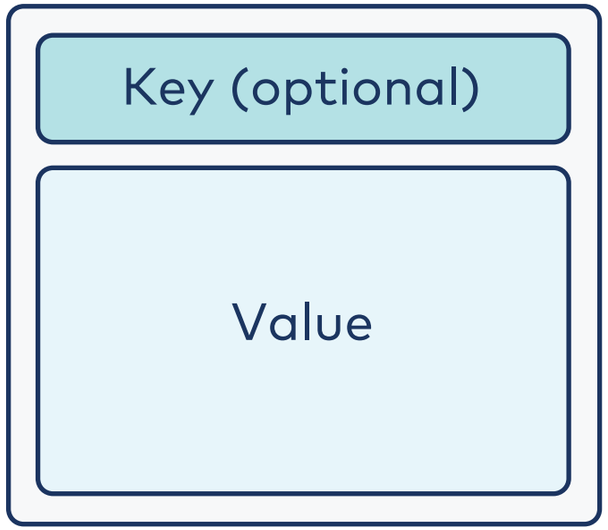
```
CREATE STREAM location_changes (
    k1 VARCHAR KEY,
    v1 VARCHAR
) WITH (
    kafka_topic = 'location_changes',
    value_format = 'json'
);
```

Table:

- A table is a mutable, partitioned collection that models change over time
- A table represents what is true as of "now"
- Tables work by leveraging the keys of each row
- To create a table, use the `CREATE TABLE` command

```
CREATE TABLE current_location (
    person VARCHAR PRIMARY KEY,
    location VARCHAR
) WITH (
    kafka_topic = 'current_location',
    partitions = 3,
    value_format = 'json'
);
```

Message



Messages
(or Event, or Record)

Key Format: →

Value Format: →

- KAFKA
- DELIMITED
- JSON
- JSON_SR
- AVRO
- PROTOBUF

If a column is declared in a schema, but no attribute is present in the underlying Kafka record, the value for the row's column is populated as `null`.

Data Types

Valid Column Data Types:

- `BOOLEAN`
- `INTEGER`
- `BIGINT`
- `DOUBLE`
- `VARCHAR` (or `STRING`)
- `DECIMAL`
- `ARRAY<ArrayType>`*
- `MAP<VARCHAR, ValueType>`*
- `STRUCT<FieldName FieldType, …>`*

*Only available for advanced formats. See guide for more.

Exploring Data

How:

- Use **Confluent Control Center**
- Use **ksqlDB CLI**

What:

- `SHOW TOPICS`

```
SHOW TOPICS;
```

- `PRINT <topic name>`

```
PRINT 'my-topic' FROM BEGINNING;
```

- Simple `SELECT` statements

```
SELECT page, user_id, status, bytes
    FROM clickstream
    WHERE user_agent LIKE 'Mozilla/5.0%'
    EMIT CHANGES;
```

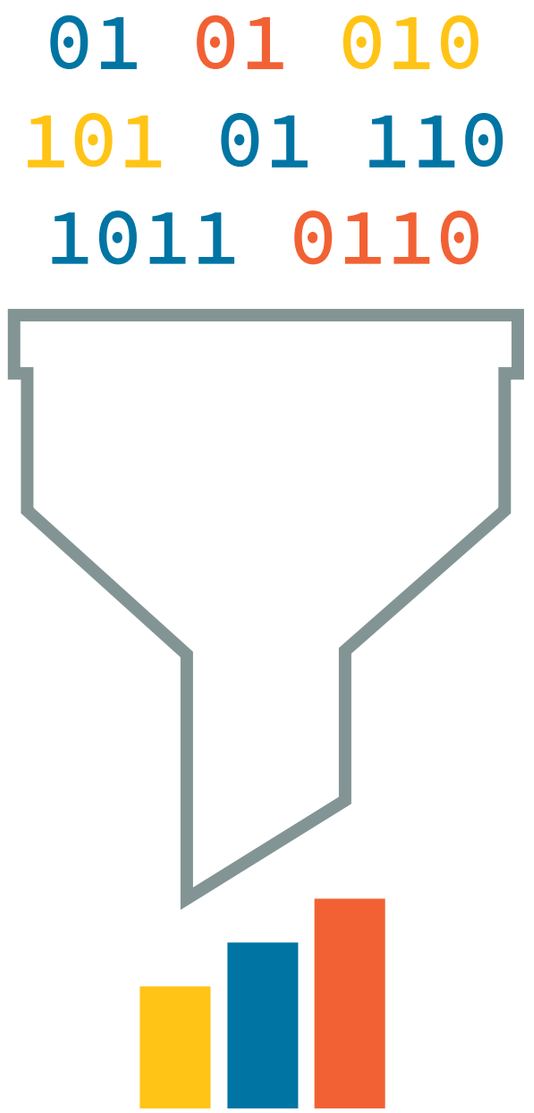
4b: What Are Some Examples of How To Use ksqlDB for Manipulating Data?

Description

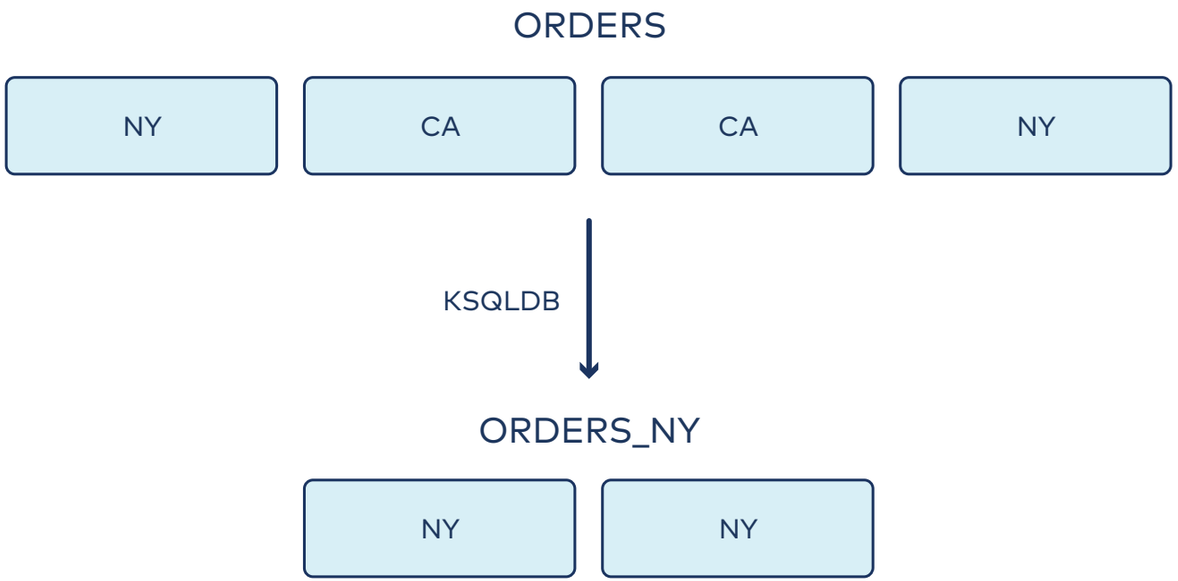
ksqlDB lets you build real-time applications with the same ease and familiarity as building traditional apps on a relational database—all through a familiar, lightweight SQL syntax. Some out-of-the-box functions are scalar (`FILTER`), aggregation (`COUNT`, `MAX`, `MIN`), and table (`EXPLODE`).

Data Filtering

- `WHERE` clause
- `AND`, `OR`, `NOT`
- Support for `LIKE`
- No subqueries



Data Filtering




```
CREATE STREAM ORDERS_NY AS
SELECT *
FROM ORDERS
WHERE ADDRESS->STATE='New York';
```

Data Manipulation - Scalar Functions

Functions:

- Math: `ABS`, `CEIL`, `FLOOR`, `RANDOM`, `ROUND`, `EXP`, `LN`, `SQRT`
- Text: `CONCAT`, `LCASE`, `SUBSTRING`, `TRIM`, `UCASE`, `INITCAP`, `REPLACE`
- Conversion: `CAST`, `CONVERT_TZ`
- Json: `EXTRACTJSONFIELD`
- `ARRAY_CONTAINS`

Examples:

- `CONVERT_TZ(col1, 'from_timezone', 'to_timezone')`
- `FROM_UNIXTIME(milliseconds)`
- `EXTRACTJSONFIELD(message, '$.log.cloud')`
- `ARRAY_CONTAINS('[1, 2, 3]', 3)`

Complete List: https://docs.ksqldb.io/en/latest/developer-guide/ksqldb-reference/scalar-functions/

Data Manipulation - CASE

`CASE` can be used for:

- Data cleansing
- Deriving new columns
- Bucketing data
- Selectively masking data
- Generating values for missing attributes
- Generating conditional aggregates
- Traffic routing

Data Manipulation - CASE - Deriving New Columns

Given this input data:

```
SELECT SKU, PRODUCT FROM PRODUCTS;
```

```
H1235 | Toaster
H1425 | Kettle
F0192 | Banana
F1723 | Apple
x1234 | Cat
```

This query with `CASE`:

```
SELECT SKU,
       CASE
          WHEN SKU LIKE 'H%' THEN 'Homewares'
          WHEN SKU LIKE 'F%' THEN 'Food'
          ELSE 'Unknown'
       END AS DEPARTMENT,
       PRODUCT  FROM PRODUCTS
       EMIT CHANGES;
```

Returns:

```
SKU    DEPARTMENT   PRODUCT
H1235 | Homewares | Toaster
H1425 | Homewares | Kettle
F0192 | Food      | Banana
F1723 | Food      | Apple
x1234 | Unknown   | Cat
```

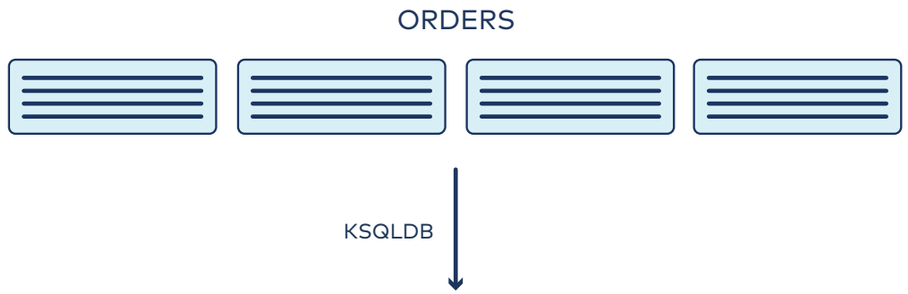
Transforming Data with ksqlDB (1)


```
{
  "ordertime": 1560070133853,
  "orderid": 67,
  "itemid": "Item_9",
  "orderunits": 5,
  "address":
  {
    "street": "243 Utah Way",
    "city": "Orange",
    "state": "California"
  }
}
```

CONVERT THIS TIMESTAMP TO HUMAN-READABLE FORMAT

DROP THESE ADDRESS FIELDS

Transforming Data with ksqlDB (2)



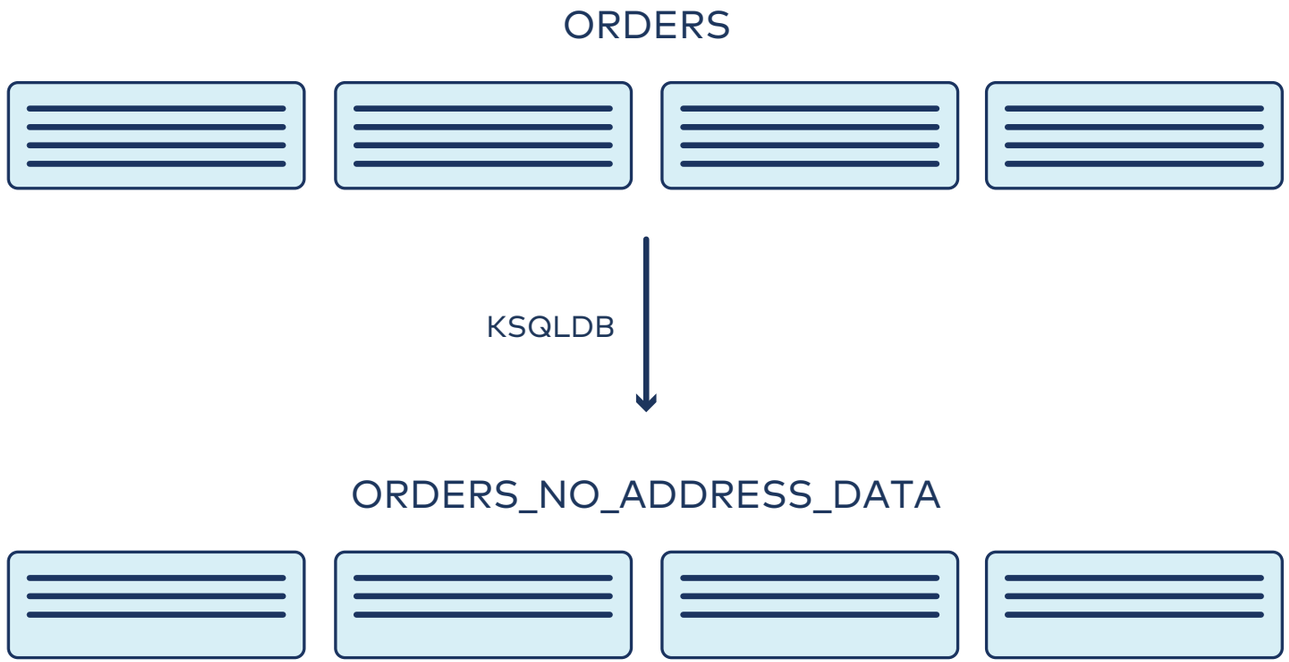
`ORDERS`:

```
{
  "ordertime": 1560070133853,
  "orderid": 67,
  "itemid": "Item_9",
  "orderunits": 5,
  "address":
  {
    "street": "243 Utah Way",
    "city": "Orange",
    "state": "California"
  }
}
```

ksqlDB code:

```
CREATE STREAM ORDERS_NO_ADDRESS_DATA AS
SELECT ORDERTIME, ORDERID, ITEMID, ORDERUNITS
FROM ORDERS;
```

Transforming Data with ksqlDB (3)



Transforming Data with ksqlDB (4)

ORDERS:

```
{
  "ordertime": 1560070133853,
  "orderid": 67,
  "itemid": "Item_9",
  "orderunits": 5,
  "address":
  {
    "street": "243 Utah Way",
    "city": "Orange",
    "state": "California"
  }
}
```

ksqlDB code:

```
CREATE STREAM
ORDERS_NO_ADDRESS_DATA AS
SELECT
   TIMESTAMPTOSTRING(ROWTIME,
      'yyyy-MM-dd HH:mm:ss')
   AS ORDER_TIMESTAMP,
   ORDERID, ITEMID, ORDERUNITS
FROM ORDERS;
```

ORDERS_NO_ADDRESS_DATA:

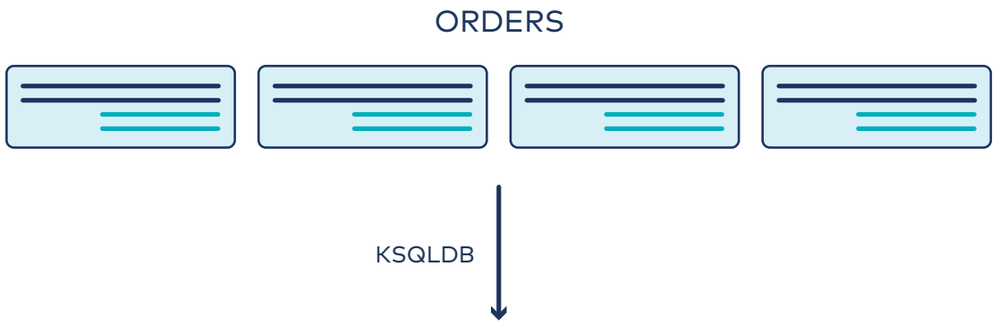
```
{
  "order_ts":
     "2020-02-14 15:10:58",
  "orderid": 67,
  "itemid": "Item_9",
  "orderunits": 5
}
```

Schema Manipulation - Flatten Records (1)

ORDERS:

```
{
  "ordertime": 1560070133853,
  "orderid": 67,
  "itemid": "Item_9",
  "orderunits": 5,
  "address":
  {
    "street": "243 Utah Way",
    "city": "Orange",
    "state": "California"
  }
}
```

Message Transformation - Flatten Records (2)



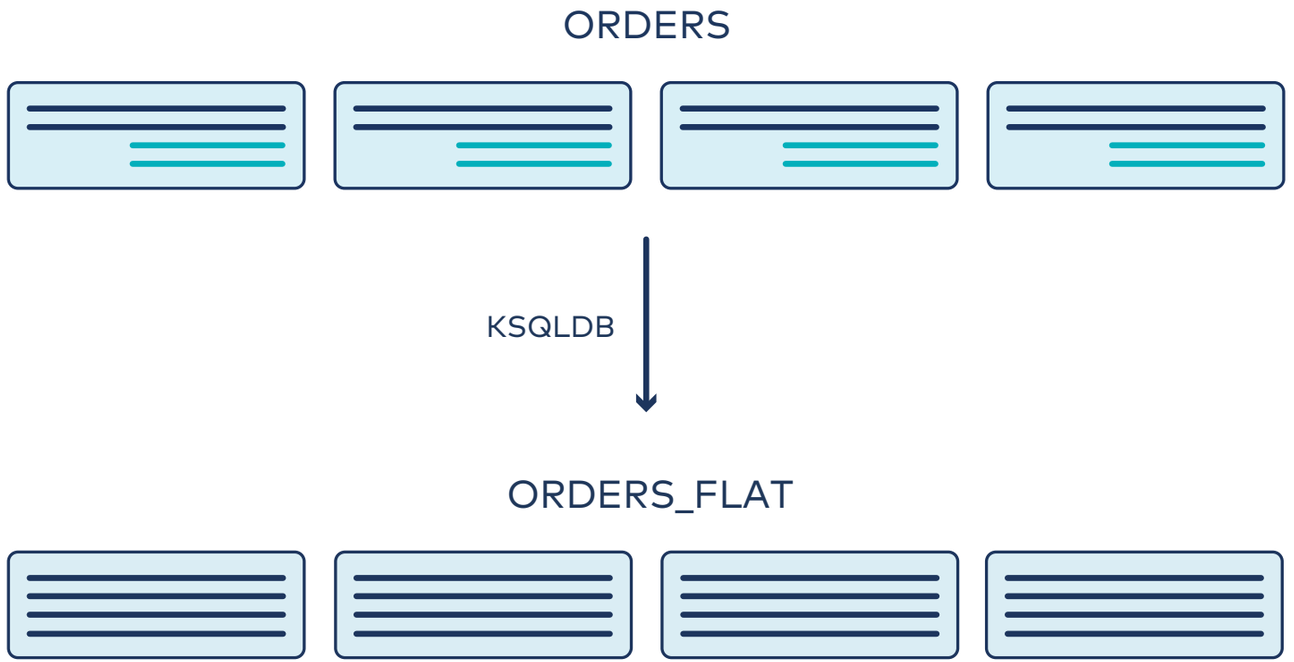
ORDERS:

```
{
  ...
  "address":
  {
    "street": "243 Utah Way",
    "city": "Orange",
    "state": "California"
  }
}
```

ksqlDB code:

```
CREATE STREAM ORDERS_FLAT AS
  SELECT [...]
    ADDRESS->STREET AS ADDRESS_STREET,
    ADDRESS-> CITY AS ADDRESS_CITY,
    ADDRESS-> STATE AS ADDRESS_STATE
  FROM ORDERS;
```

Message Transformation - Flatten Records (3)



Message Transformation - Flatten Records (4)

ORDERS:

```
{
  "ordertime": 1560070133853,
  "orderid": 67,
  "itemid": "Item_9",
  "orderunits": 5,
  "address":
  {
    "street": "243 Utah Way",
    "city": "Orange",
    "state": "California"
  }
}
```

ksqlDB code:

```
CREATE STREAM ORDERS_FLAT AS
  SELECT […]
    ADDRESS->STREET
       AS ADDRESS_STREET,
    ADDRESS-> CITY
       AS ADDRESS_CITY,
    ADDRESS-> STATE
       AS ADDRESS_STATE
  FROM ORDERS;
```

ORDERS_FLAT:

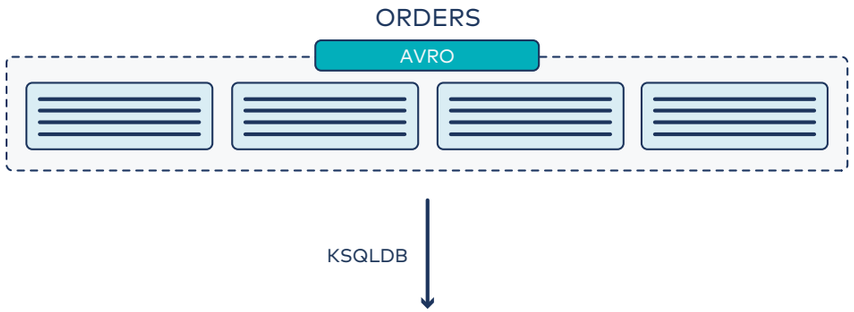
```
{
  "ordertime": 1560070133853,
  "orderid": 67,
  "itemid": "Item_9",
  "orderunits": 5,
  "address_street":
     "243 Utah Way",
  "address_city": "Orange",
  "address_state": "California"
}
```

Converting Data Formats with ksqlDB (1)

ORDERS:

```
{
  "ordertime": 1560070133853,
  "orderid": 67,
  "itemid": "Item_9",
  "orderunits": 5,
  "address":
  {
    "street": "243 Utah Way",
    "city": "Orange",
    "state": "California"
  }
}
```

Converting Data Formats with ksqlDB (2)



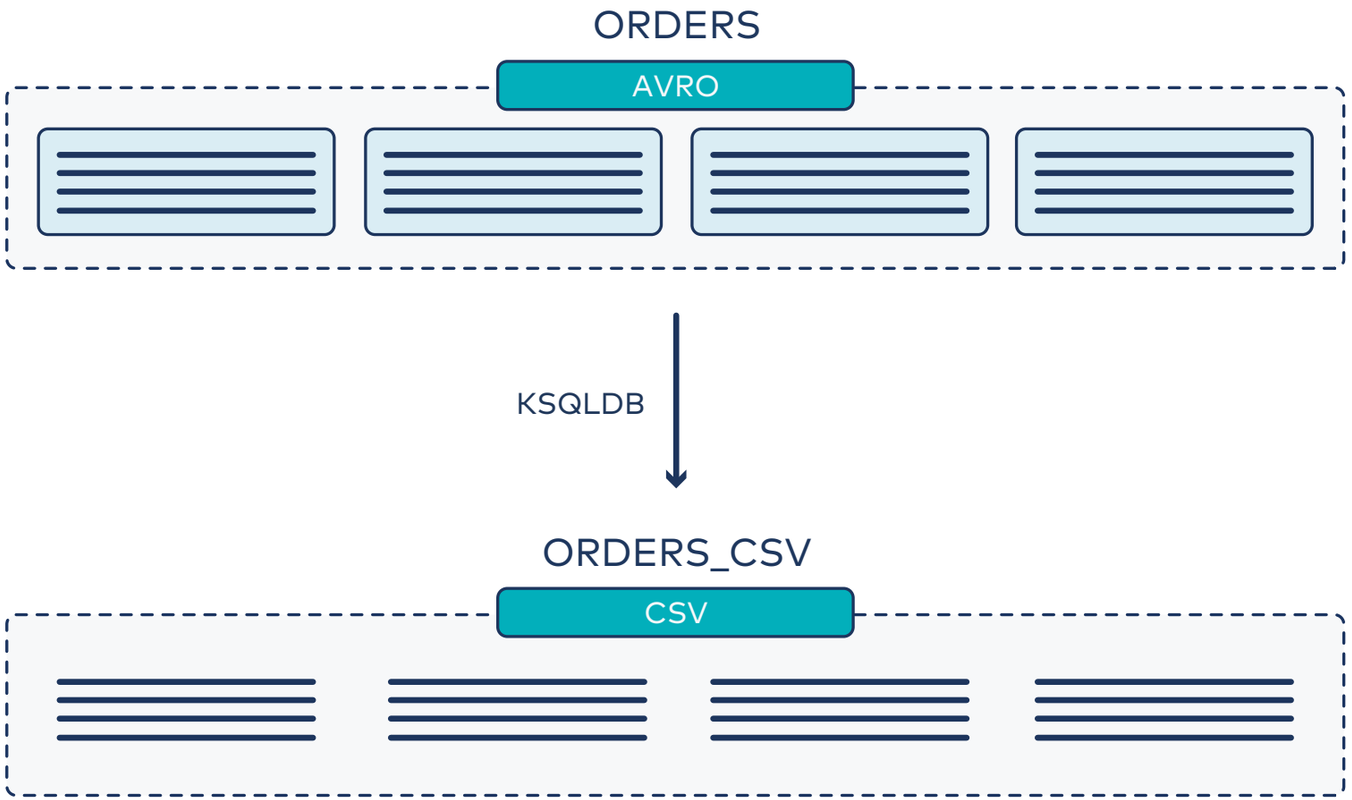
ORDERS:

```
{
  "ordertime": 1560070133853,
  "orderid": 67,
  "itemid": "Item_9",
  "orderunits": 5,
  "address":
  {
    "street": "243 Utah Way",
    "city": "Orange",
    "state": "California"
  }
}
```

ksqlDB code:

```
CREATE STREAM ORDERS_CSV
  WITH (VALUE_FORMAT='DELIMITED') AS
  SELECT * FROM ORDERS_FLAT;
```

Converting Data Formats with ksqlDB (3)



Converting Data Formats with ksqlDB (4)

ORDERS:

```
{
  "ordertime": 1560070133853,
  "orderid": 67,
  "itemid": "Item_9",
  "orderunits": 5,
  "address":
  {
    "street": "243 Utah Way",
    "city": "Orange",
    "state": "California"
  }
}
```

ksqlDB code:

```
CREATE STREAM ORDERS_CSV
  WITH (VALUE_FORMAT='DELIMITED')
AS
  SELECT * FROM ORDERS_FLAT;
```

ORDERS_CSV:

1560045914101,24644,Item_0,1,43078 Dexter Pass,Port Washington,New York

1560047305664,24643,Item_29,3,209 Monterey Pass,Chula Vista,California

1560057079799,24642,Item_38,18,3 Autumn Leaf Plaza,San Diego,California

1560088652051,24647,Item_6,6,82893 Arkansas Center,El Paso,Texas

1560105559145,24648,Item_0,12,45896 Warner Parkway,South Lake Tahoe,California

1560108336441,24646,Item_33,4,272 Heffernan Way,El Paso,Texas

1560123862235,24641,Item_15,16,0 Dorton Circle,Brooklyn,New York

1560124799853,24645,Item_12,1,71 Knutson Parkway,Dallas,Texas

Data Manipulation - Table Functions: **EXPLODE**

A table function returns a set of zero or more rows.

```
Given this input data:
           {sensor_id:12345 readings: [23, 56, 3, 76, 75]}
           {sensor_id:54321 readings: [12, 65, 38]}
```

```
SELECT sensor_id, EXPLODE(readings) AS reading FROM batched_readings;
```

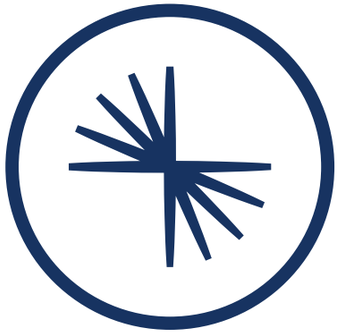
returns:

```
{sensor_id:12345 reading: 23}
{sensor_id:12345 reading: 56}
{sensor_id:12345 reading: 3}
{sensor_id:12345 reading: 76}
{sensor_id:12345 reading: 75}
{sensor_id:54321 reading: 12}
{sensor_id:54321 reading: 65}
{sensor_id:54321 reading: 38}
```

Lab: Using ksqlDB

Please work on **Lab 4a: Using ksqlDB**

Refer to the Exercise Guide



CONFLUENT

Global Education

Module Overview

This module contains three lessons:

- How Does Time Work in Stream Processing?
- How Can You Divide up Streams into Time Windows?
- How Can You Make Windows Handle Late-Arriving Events and Limit Their Output?

Where this fits in:

- Hard Prerequisite: Working with Kafka Streams, Using ksqlDB
- Recommended Follow-Up: Aggregations, Joins, and/or Custom Processing

5a: How Does Time Work in Stream Processing?

Description

The notion of time has a critical aspect in stream processing, and how it is modeled and integrated. When working with stream processing, it is important to understand the concept of time.

5b: How Can You Divide up Streams into Time Windows?

Description

Windowing lets you control how to group records that have the same key for stateful operations like aggregations or joins into windows. Windows are tracked per record key. Tumbling, hopping, and session are commonly used windows.

Types of Windows

Windowing allows you to group records by the same key for stateful operations.

Types of windows:

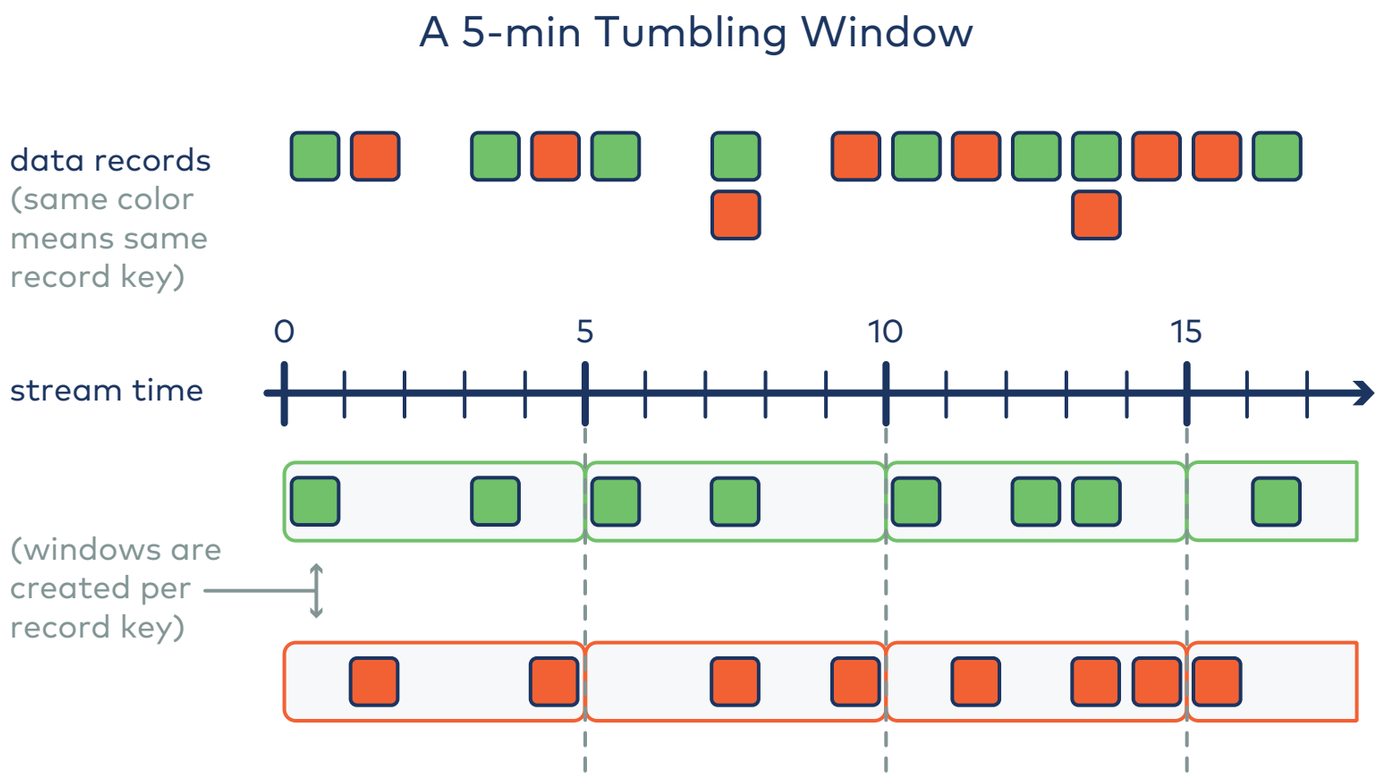
- Tumbling
- Hopping
- Sliding
- Session

Differences Between the Window Types

Window name	Behavior	Short description
Tumbling	Time-based	Fixed-size, non-overlapping, gap-less windows
Hopping	Time-based	Fixed-size, overlapping windows
Sliding	Time-based	Fixed-size, overlapping windows that work on differences between record timestamps
Session	Session-based	Dynamically-sized, non-overlapping, data-driven windows

Tumbling Windows

- Fixed-size
- Non-overlapping, gap-less
- Defined by a single property: the window's size
- Aligned to the epoch
- Each window is inclusive of the lower bound and exclusive of the upper bound



Tumbling Window Code Example

Kafka Streams

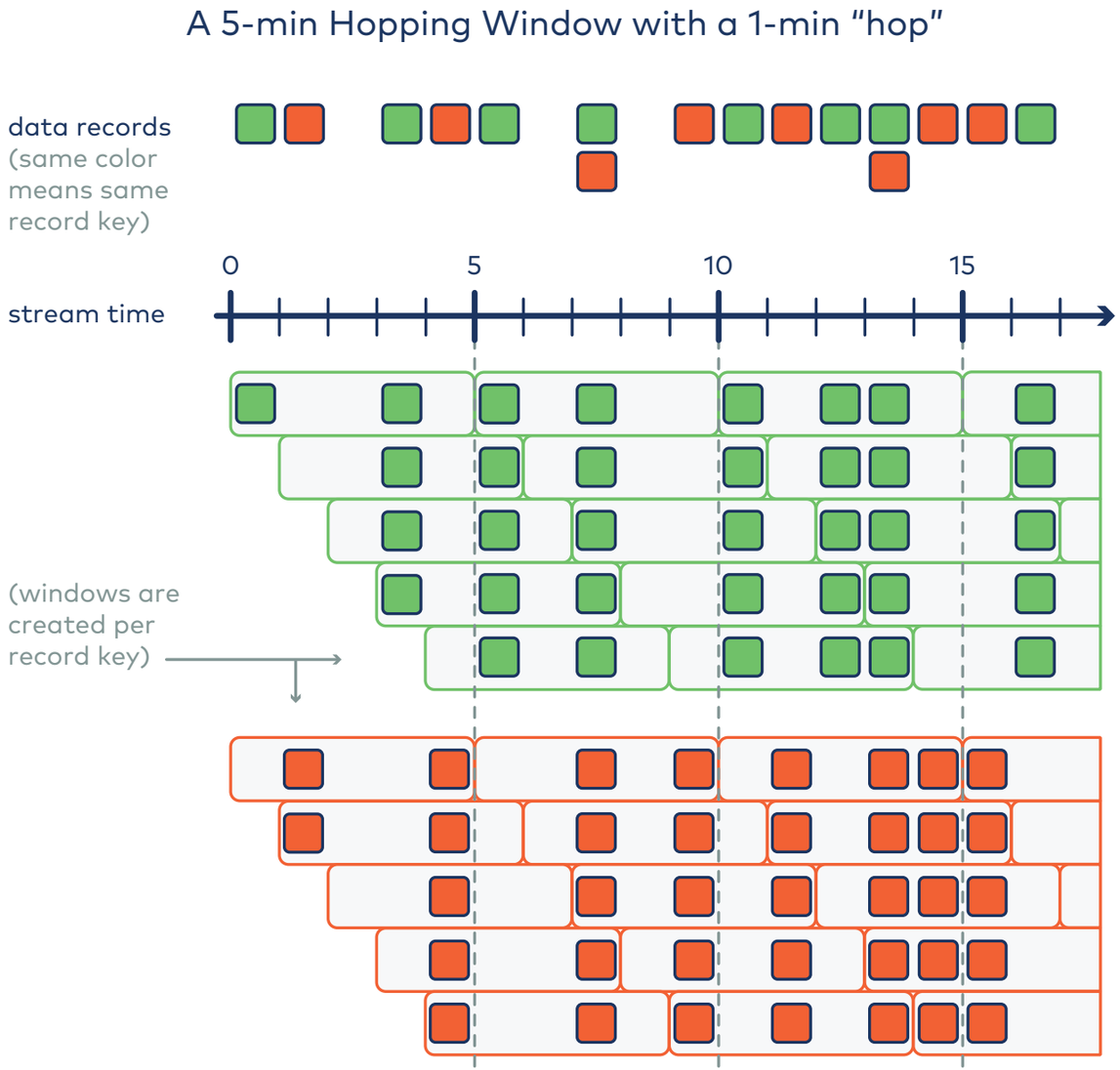
```
builder.<String, Rating>stream(ratingTopic)
            .map((key, rating)
                -> new KeyValue<>(rating.getTitle(), rating))
            .groupByKey()
            .windowedBy(
                TimeWindows.of(Duration.ofMinutes(10)))
            .count()
            .toStream()
            .map((Windowed<String> key, Long count)
                 -> new KeyValue<>(key.key(),
                                   count.toString()))
            .to(ratingCountTopic,
                Produced.with(Serdes.String(),
                              Serdes.String()));
```

ksqlDB

```
CREATE TABLE rating_count
    WITH (kafka_topic='rating_count')
AS
    SELECT title,
           COUNT(*) AS rating_count,
           WINDOWSTART AS window_start,
           WINDOWEND AS window_end
    FROM ratings
    WINDOW TUMBLING (SIZE 6 HOURS)
    GROUP BY title;
```

Hopping Windows

- Fixed-sized
- Overlapping windows
- Defined by two properties: the window's size and its advance interval
- Aligned to the epoch
- Each window is inclusive of the lower bound and exclusive of the upper bound



Hopping Window Code Example

Kafka Streams

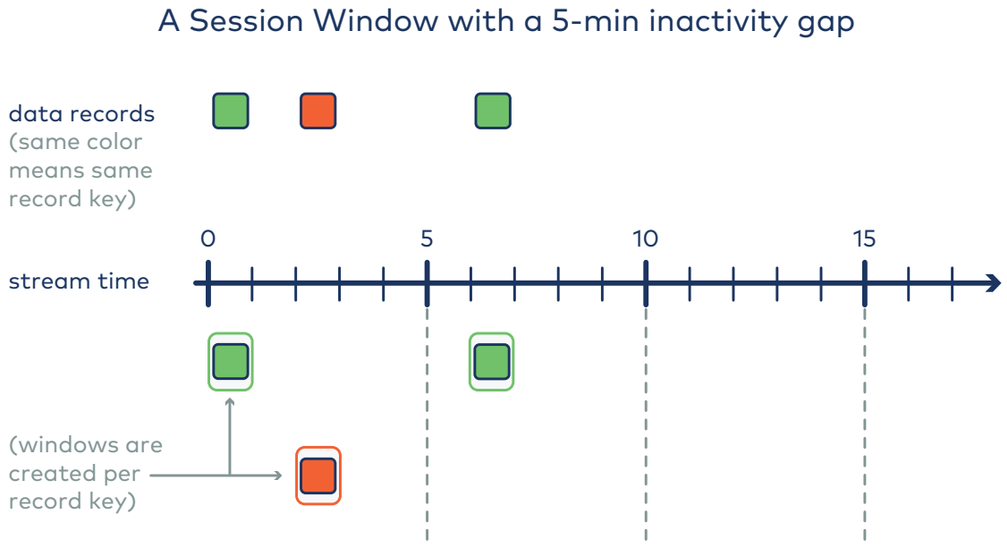
```
KStream<String, GenericRecord> pageViews = ...;

KTable<Windowed<String>, Long> windowedPageViewCounts;

windowedPageViewCounts = pageViews
    .groupByKey(Grouped.with(Serdes.String(),
                genericAvroSerde))
    .windowedBy(TimeWindows.of(
                Duration.ofMinutes(5).
                advanceBy(Duration.ofMinutes(1))))
    .count()
```

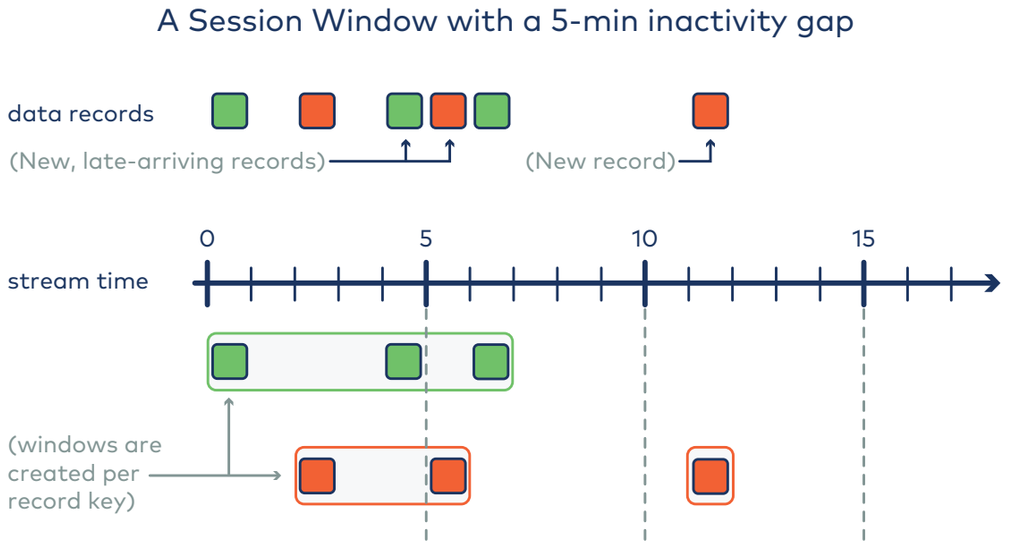
ksqlDB

```
SELECT
    ID,
    TIMESTAMPTOSTRING(WINDOWSTART, 'HH:mm:ss',
                      'UTC') AS START_PERIOD,
    TIMESTAMPTOSTRING(WINDOWEND, 'HH:mm:ss',
                      'UTC') AS END_PERIOD,
    SUM(READING)/COUNT(READING) AS AVG_READING
  FROM TEMPERATURE_READINGS
  WINDOW HOPPING (SIZE 10 MINUTES,
                  ADVANCE BY 5 MINUTES)
  GROUP BY ID
  HAVING SUM(READING)/COUNT(READING) < 45
  EMIT CHANGES
  LIMIT 3;
```

Session Windows

- Session windows are used to aggregate key-based events, sessions.
- All windows are tracked independently across keys, e.g., windows of different keys typically have different start and end times.
- Window sizes vary. Even windows for the same key typically have different sizes based on activity and idleness.

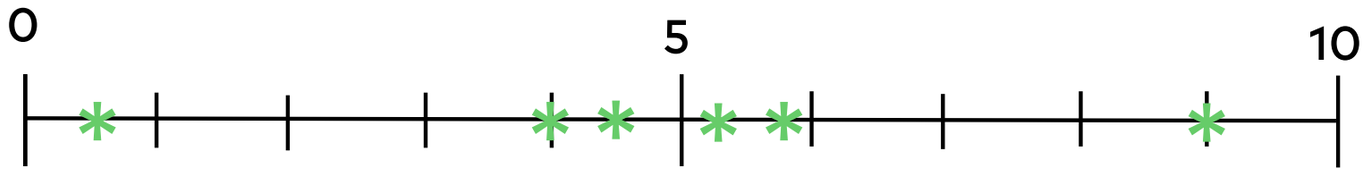




Activity

Discuss with a partner or small group:

Consider this timeline, where each * represents an event:



Then:

1. Suppose we had tumbling windows of size 5. What events would be in each window?
2. Suppose instead we had hopping windows of size 5, advance by 3. What events would be in each window?

5c: How Can You Make Windows Handle Late-Arriving Events and Limit Their Output?

Description

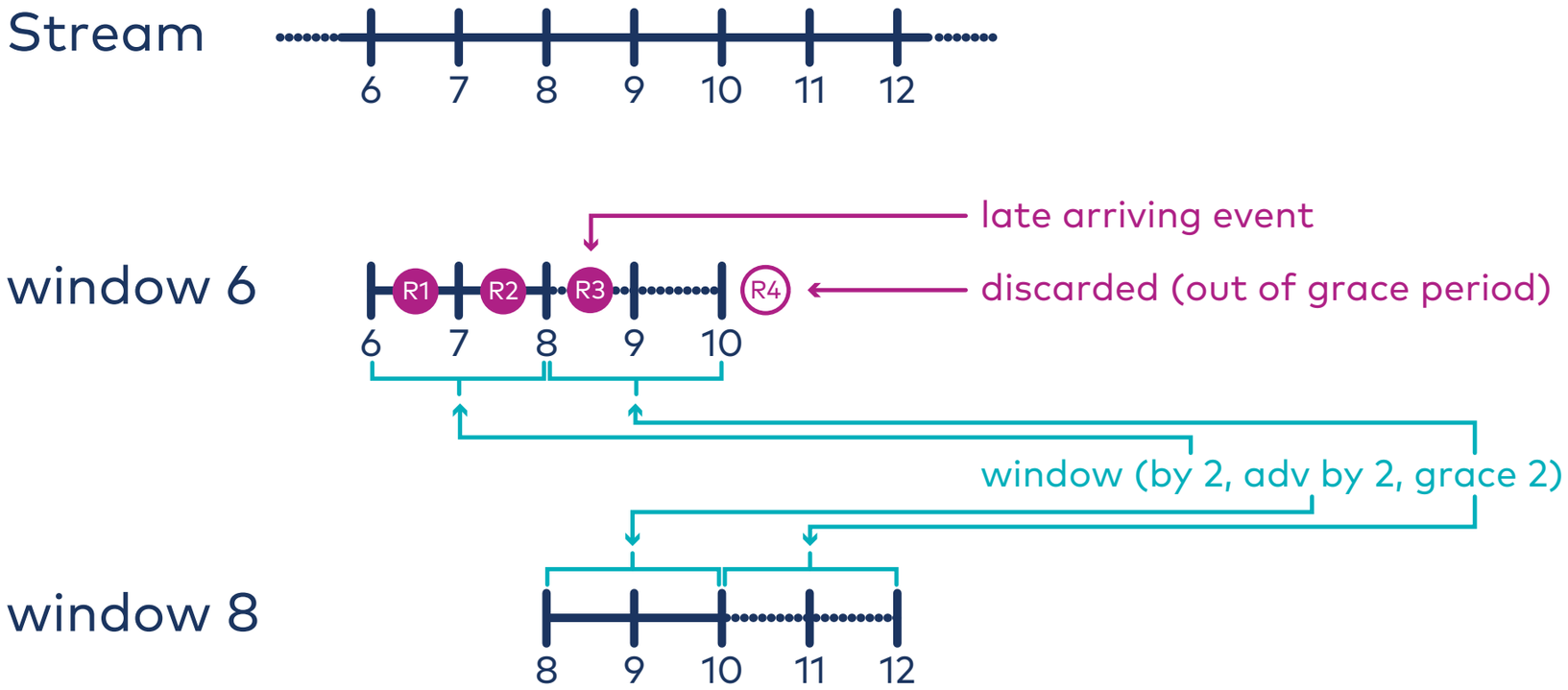
In Kafka Streams, late-arriving records can be handled by configuring a grace period. A grace period is an extension to the size of a window, and it allows events with timestamps greater than the window-end (but less than the window-end plus the grace period) to be included in the windowed calculation.

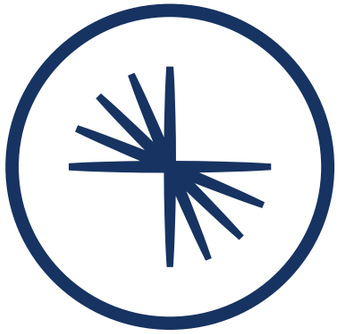
Grace Period

- An extension to the size of a window.
- Allows events with timestamps greater than the window-end (but less than the window-end plus the grace period) to be included in the windowed calculation.
- A record is discarded if it arrived after a grace period of a window is over, i.e., `record.ts > window-end-time + grace-period`.
- Tumbling, hopping, and sliding windows use the concept of grace period.
- The grace period supersedes retention time.

The default grace period is 24 hours.

Grace Period Explained





CONFLUENT

Global Education

Module Overview

This module contains three lessons:

- How Do You Aggregate Data in Kafka Streams?
- What If You Want to Window Your Aggregations?
- How Do You Aggregate Data in ksqlDB?

Where this fits in:

- Hard Prerequisite: Time and Windowing
- Recommended Follow-Up: Joins and/or Custom Processing

Aggregation Overview

Aggregations...

- Are key-based operations
- Are performed on windowed or non-windowed data
- Require a state store
- Use a windowing state store to collect the latest aggregation results per window behind the scenes

6a: How Do You Aggregate Data in Kafka Streams?

Description

Aggregations are key-based operations, meaning they always operate over records values of the same key. When aggregating a `KTable`, updates require us to subtract an old value before adding a new value as compared to stream aggregation, which does not have the notion of a subtractor.

Stateful Operations - toTable

Create Stream

```
StreamsBuilder builder = new StreamsBuilder();
KStream<byte[], String> stream = builder.stream(topicName);
```

Stream to Table

```
KTable<byte[], String> table = stream.toTable();
```

or

```
KTable<byte[], String> table =
    stream.toTable(Materialized.as("table-store-name"));
```

Aggregating a KStream

```
final StreamsBuilder builder = new StreamsBuilder();

// Create stream with default serdes and then group by key
KGroupedStream<String, String> groupedStream =
        builder.stream("input-topic").groupByKey();

KTable<String, Long> aggregatedStream = groupedStream.aggregate(
    () -> 0L, /* initializer */
    (aggKey, newValue, aggValue) -> aggValue + newValue.length(), /* adder */
    Materialized
        .as("aggregated-stream-store")                /* state store name */
        .withValueSerde(Serdes.Long());       /* serde for aggregate value */
)
```

Stateful Operations: count

count

```
1 KTable<String, Long> aggregatedStream = groupedStream.count();
2
3 KTable<String, Long> aggregatedTable = groupedTable.count();
```

Stateful Operations: reduce

reduce

```
KTable<byte[], Integer> aggregatedStream = groupedStream.reduce(
    (aggValue, newValue) -> aggValue + newValue);

KTable<byte[], Integer> aggregatedTable = groupedTable.reduce(
    (aggValue, newValue) -> aggValue + newValue, /* adder */
    (aggValue, oldValue) -> aggValue - oldValue /* subtractor */);
```

6b: What If You Want to Window Your Aggregations?

Description

We delve deeper into an aggregation of data performed over time using various windowed aggregations and Kafka Streams.

Windowed Aggregation - Using a 5-Minute Tumbling Window

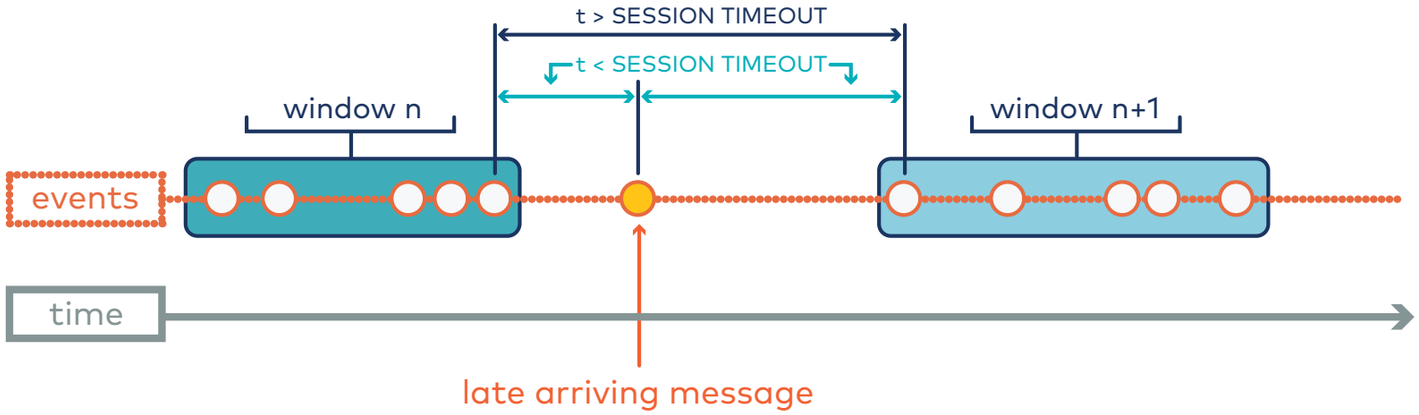
```
1 KTable<Windowed<String>, Long> timeWindowedAggregatedStream = groupedStream
2     .windowedBy(TimeWindows.of(Duration.ofMinutes(5)))
3     .aggregate(
4       () -> 0L, /* initializer */
5       (aggKey, newValue, aggValue) -> aggValue + newValue, /* adder */
6       Materialized.<String, Long, WindowStore<Bytes, byte[]>>
7         as("time-windowed-aggregated-stream-store") /* state store name */
8         .withValueSerde(Serdes.Long())); /* serde for aggregate value */
```

Windowed Aggregation - Using a 5-Minute Hopping Window

```
1 KTable<Windowed<String>, Long> timeWindowedAggregatedStream = groupedStream
2     .windowedBy(TimeWindows.of(
3                 Duration.ofMinutes(5)).advanceBy(Duration.ofSeconds(10)))
4     .aggregate(
5       () -> 0L, /* initializer */
6       (aggKey, newValue, aggValue) -> aggValue + newValue, /* adder */
7       Materialized.<String, Long, WindowStore<Bytes, byte[]>>
8         as("time-windowed-aggregated-stream-store") /* state store name */
9         .withValueSerde(Serdes.Long())); /* serde for aggregate value */
```

Windowed Aggregation - Using a 5-Minute Session Window

```
KTable<Windowed<String>, Long> sessionizedAggregatedStream = groupedStream
    .windowedBy(SessionWindows.with(Duration.ofMinutes(5))
    .aggregate( () -> 0L, /* initializer */
        (aggKey, newValue, aggValue) -> aggValue + newValue, /* adder */
        (aggKey, leftAggValue, rightAggValue)
                -> leftAggValue + rightAggValue, /* session merger */
        Materialized.<String, Long, SessionStore<Bytes, byte[]>>
            as("sessionized-aggregated-stream-store") /* state store name */
            .withValueSerde(Serdes.Long())); /* serde for aggregate value */
```



Adding Windowing to **reduce**

reduce
windowed

```
KTable<Windowed<String>, Integer> timeWindowedAggregatedStream = groupedStream
    .windowedBy(TimeWindows.of(Duration.ofMinutes(5)))
    .reduce((aggValue, newValue) -> aggValue + newValue);

KTable<Windowed<String>, Integer> sessionAggregatedStream = groupedStream
    .windowedBy(SessionWindows.with(Duration.ofMinutes(5)))
    .reduce((aggValue, newValue) -> aggValue + newValue);
```

6c: How Do You Aggregate Data in ksqlDB?

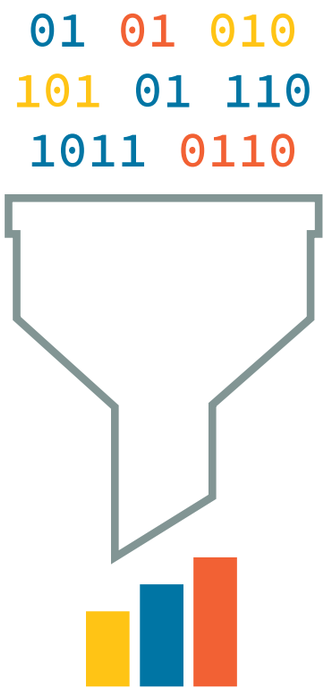
Description

ksqlDB supports several aggregate functions, like `COUNT`, `MAX`, `MIN`, and `SUM`. ksqlDB can also be used to aggregate data over tumbling, hopping, and session windows.

Data Aggregation

```
SELECT
    s.name,
    MIN(t.temperature) AS t_min,
    MAX(t.temperature) AS t_max
FROM
    temperatures t
    LEFT JOIN stations s
    ON t.stationid = s.stationid
GROUP BY s.name
```

- `COUNT`
- `COUNT_DISTINCT`
- `MAX`, `MIN`
- `AVG`
- `SUM`
- `TOPK`
- `TOPKDISTINCT`
- `COLLECT_LIST`
- `COLLECT_SET`



Windowed Aggregation

Aggregate data to identify patterns or anomalies in real-time...

```
1 CREATE TABLE possible_fraud AS
2    SELECT card_number, count(*)
3    FROM authorization_attempts
4    WINDOW HOPPING (SIZE 30 SECONDS, ADVANCE BY 1 SECOND)
5    GROUP BY card_number
6    HAVING count(*) > 3
7    EMIT CHANGES;
```

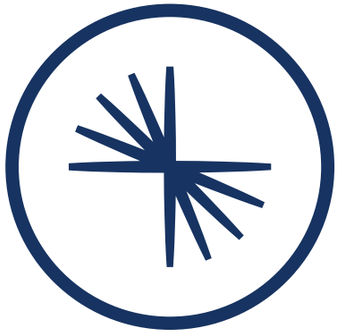
Notes:

- Line 2 aggregates data
- Line 4 makes it per 30-second window

Lab: Windowing & Aggregation

Please work on **Lab 6a: Windowing & Aggregation**

Refer to the Exercise Guide



CONFLUENT

Global Education

Module Overview

This module contains two lessons:

- How Can You Join Data Across Stream Processing Entities?
- How Can You Join Data With Foreign Keys?

Where this fits in:

- Hard Prerequisite: Time and Windowing
- Recommended Follow-Up: Aggregations and/or Custom Processing

7a: How Can You Join Data Across Stream Processing Entities?

Description

Kafka Streams and ksqlDB allow you to merge streams of events in real time. They support inner, left, and outer joins. For joining, input data must be co-partitioned. Join operations can be windowed or non-windowed.

Joins using Kafka Streams

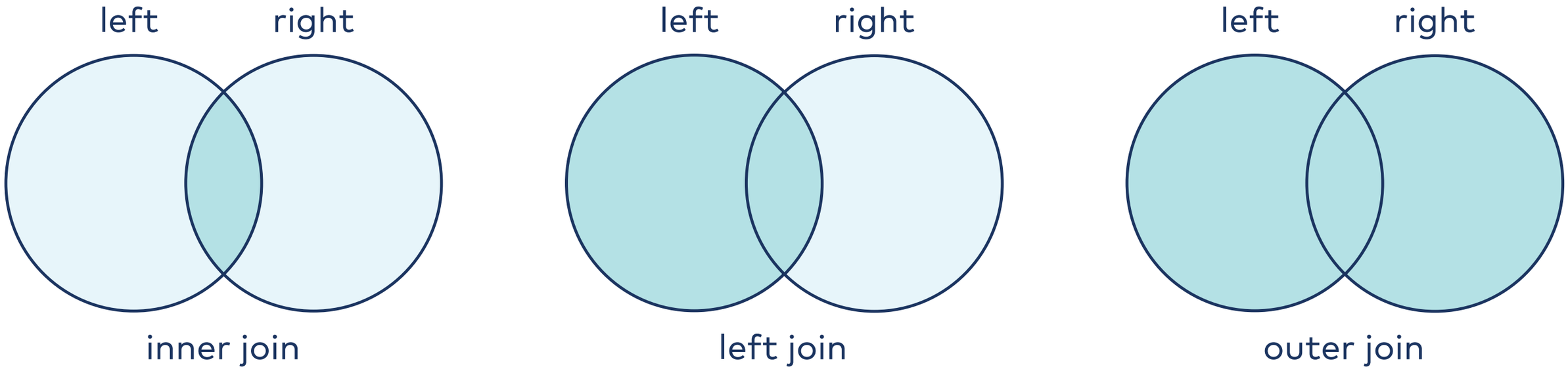
Join operands	Type	(INNER) JOIN	LEFT JOIN	OUTER JOIN
`KStream`, `KStream` → `KStream`	windowed	✓	✓	✓
`KTable`, `KTable` → `KTable`	*non* -windowed	✓	✓	✓
`KStream`, `KTable` → `KStream`	*non*-windowed	✓	✓	✗
`KStream`, `GlobalKTable` → `KStream`	*non*-windowed	✓	✓	✗
`KTable`, `GlobalKTable` → ?	✗	✗	✗	✗

Data needs to be co-partitioned

Joins using ksqlDB

Join operands	Type	(INNER) JOIN	LEFT JOIN	OUTER JOIN
`STREAM`, `STREAM` → `STREAM`	windowed			
`TABLE`, `TABLE` → `TABLE`	*non*-windowed			
`STREAM`, `TABLE` → `STREAM`	*non*-windowed			

Visualizing Joins in Kafka Streams & ksqlDB



Stream — Stream
Table — Table
Stream — Table
KStream — GlobalKTable (supported in Kafka Streams only)

Stream — Stream
Table — Table

Join Operations in Code

Kafka Streams

```
KStream<String, String> joined = left.join(right,
    (leftValue, rightValue) ->
       "left=" + leftValue + ", right=" + rightValue,
    JoinWindows.of(Duration.ofMinutes(5)),
    Joined.with(
      Serdes.String(),    /* key */
      Serdes.Long(),      /* left value */
      Serdes.Double())    /* right value */
  );
```

ksqlDB

```
CREATE STREAM enriched_payments AS
   SELECT payment_id, u.country, total
   FROM   payments_stream p
          LEFT JOIN users_table u
          ON p.user_id = u.user_id
```

Join Requirements

Input data must be co-partitioned when joining.

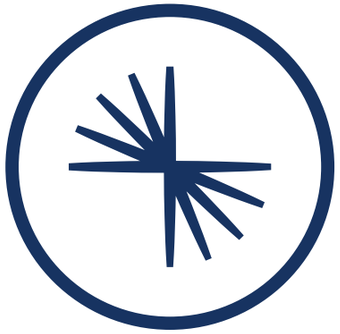
The requirements for data co-partitioning are:

- The input topics of the join (left side and right side) must have the same number of partitions.
- All applications that write to the input topics must have the same partitioning strategy.
- The input topics use the same set of keys.

Lab: Joining Two Streams

Please work on **Lab 7a: Joining Two Streams**

Refer to the Exercise Guide



CONFLUENT

Global Education

Module Overview

This module contains two lessons:

- How Do You Leverage the Processor API for Low-Level Processing?
- How Can You Add Your Own Functions to ksqlDB?

Where this fits in:

- Hard Prerequisite: Working with Kafka Streams, Using ksqlDB
- Recommended Prerequisite: Time and Windowing
- Recommended Follow-Up: Aggregations and/or Joins

8a: How Do You Leverage the Processor API for Low-Level Processing?

Description

The Processor API is a low-level API which allows you to customize and implement special logic that is not available in the DSL. The Kafka Streams DSL is built on the Processor API.

What is the Processor API?

The Kafka Streams DSL is built on top of the Streams Processor API.

What is the Processor API (PAPI)?

- The PAPI allows you to:
  - Define a custom processor
  - Connect processors
  - Interact with the state stores

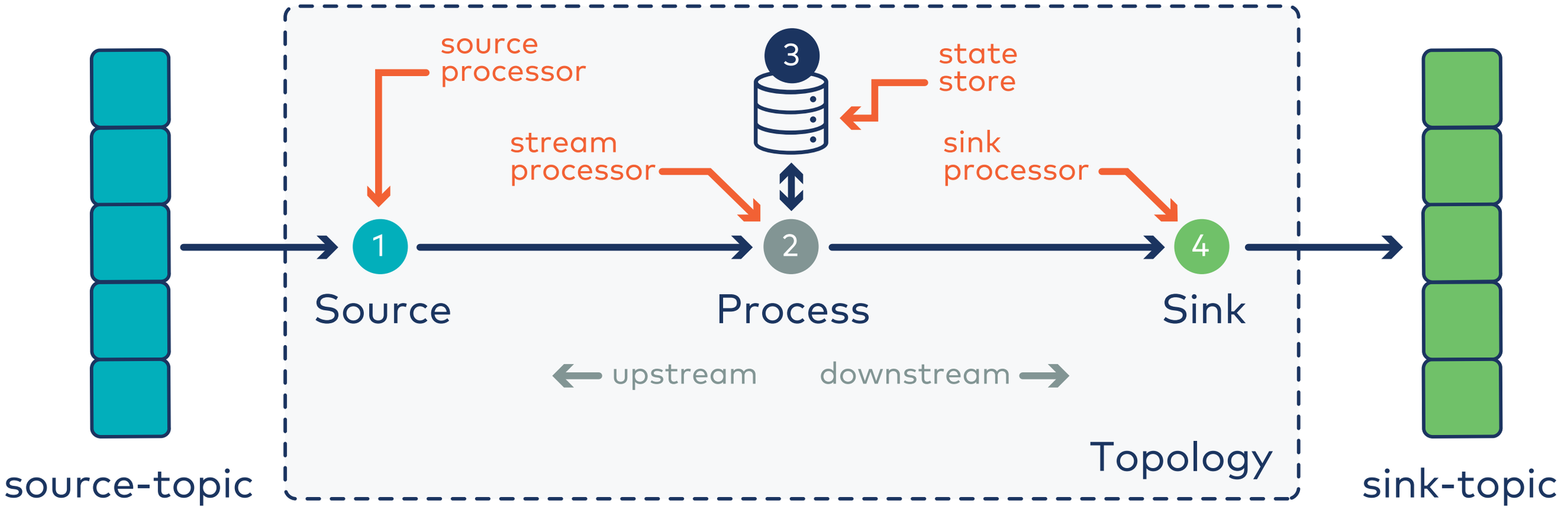
The Processor API can be used to implement both stateless and stateful operations.

Where Can the PAPI be Useful?

- Customization
- Combining ease-of-use with full flexibility where needed

Processor API

```
builder.addSource("Source", "source-topic")
       .addProcessor("Process", () -> WordCountProcessor(), "Source")
       .addStateStore(countStoreBuilder, "Process")
       .addSink("Sink", "sink-topic", "Process");
```



Lab: Using the Processor API

Please work on **Lab 8a: Using the Processor API**

Refer to the Exercise Guide

8b: How Can You Add Your Own Functions to ksqlDB?

Description

User-defined functions enable you to extend ksqlDB's suite of built-in functions using Java hooks. Once a UDF has been created, at start up time, ksqlDB scans the jar files in its extensions directory looking for classes with UDF annotated. Each function that is found is parsed and, if successful, loaded into ksqlDB.

Functions

ksqlDB comes with many built-in functions:

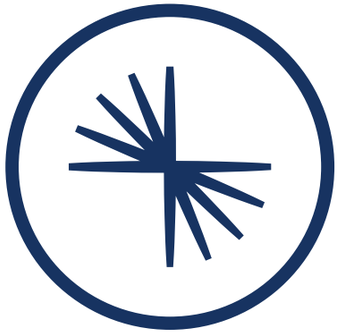
Class	Examples
Aggregate Functions	`AVG`, `COUNT`, `MAX`, `MIN`, `SUM`, ...
Scalar Functions	`ABS`, `CAST`, `EXP`, `GEO_DISTANCE`
Table Functions	`CUBE_EXPLODE`, `EXPLODE`

User-Defined Functions

- But what if you have a specific use-case for which function is not available out of the box in ksqlDB?
- User-defined functions let you add new functions using Java hooks.

How to Create a User-Defined Function

1. Set up a Java project.
2. Implement the classes:
   - Scalar function (UDF)
   - Tabular function (UDTF)
   - Aggregation function (UDAF)
3. Add the uberjar to ksqlDB server.
4. Invoke the functions.



CONFLUENT

Global Education

Module Overview

This module contains three lessons:

- How Should You Test Streaming Applications?
- How Can You Monitor Streaming Applications?
- How Should You Troubleshoot Streaming Applications?

Where this fits in:

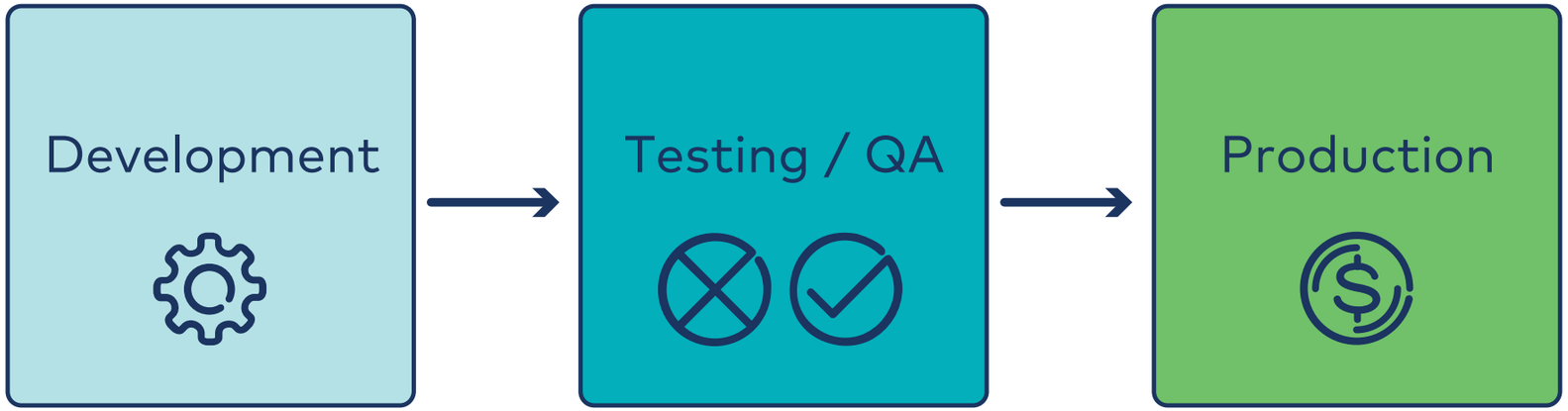
- Hard Prerequisite: Introduction to Kafka Streams, Introduction to ksqlDB
- Recommended Prerequisite: Working with Kafka Streams, Using ksqlDB
- Recommended Follow-Up: Either of Deployment or Security

9a: How Should You Test Streaming Applications?

Description

How can you ensure your Kafka Streams application is working as expected? Learn different types of testing and how to test applications.

Why Test?

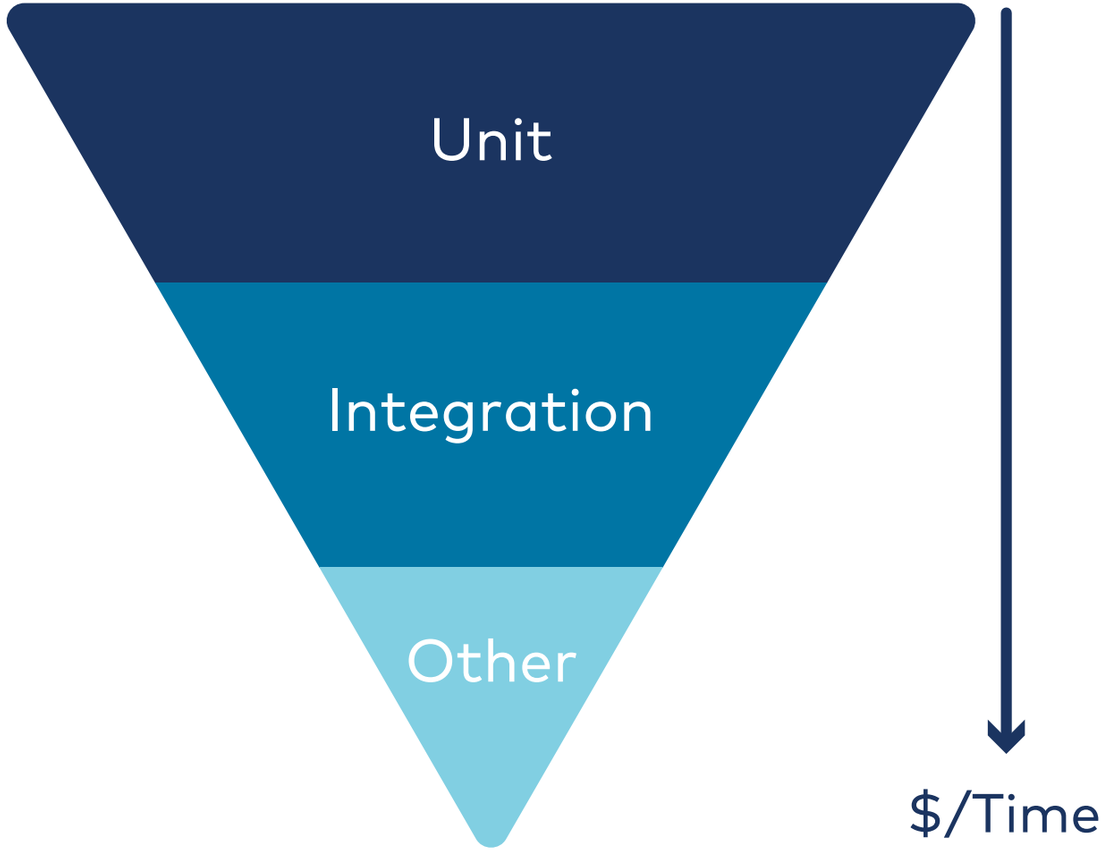


Types of Testing

Unit testing: Testing actual behavior of one component against intended behavior of the API.

Integration testing: Several pieces are tested working in conjunction.

Other testing: Performance, soak, chaos testing: For optimizing your client applications, ensuring long-running code, and resilience against failures.



Generating the Test Data

Use data that is as close to realistic as possible for testing. You could:

- Write your own Kafka client application.
- Use the Datagen Connector.

Test Utilities

	ksqlDB	Kafka Streams	JVM Producer & Consumer	librdkafka Producer & Consumer
Unit Testing	ksql-test-runner	TopologyTestDriver	MockProducer, MockConsumer	rdkafka_mock
Integration Testing	Testcontainers	Testcontainers	Testcontainers	trivup
	Confluent Cloud	Confluent Cloud	Confluent Cloud	Confluent Cloud

Lab: Building Unit Tests

Please work on **Lab 9a: Building Unit Tests**

Refer to the Exercise Guide

Lab: Integration Tests Using Embedded Kafka

Please work on **Lab 9b: Integration Tests Using Embedded Kafka**

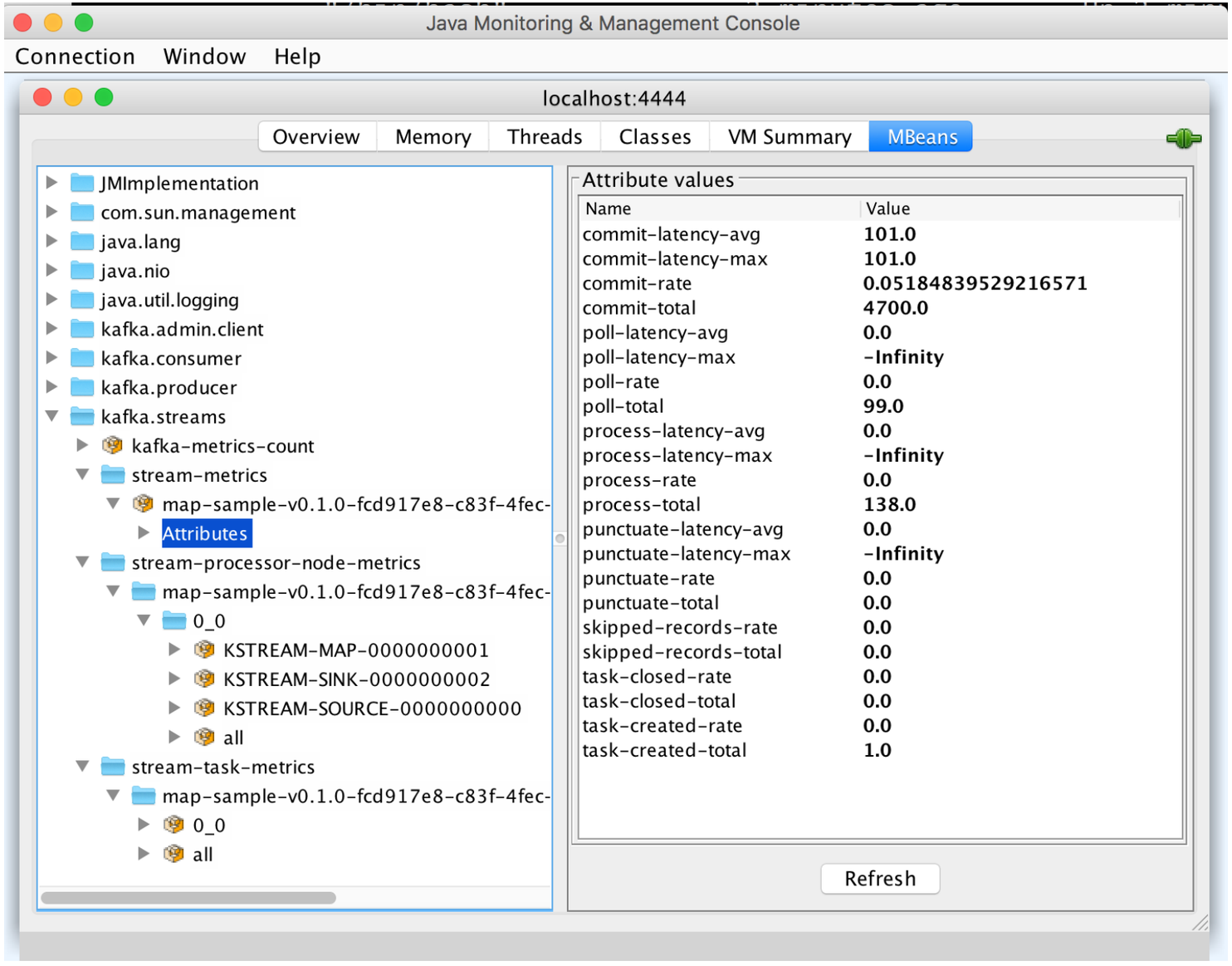
Refer to the Exercise Guide

9b: How Can You Monitor Streaming Applications?

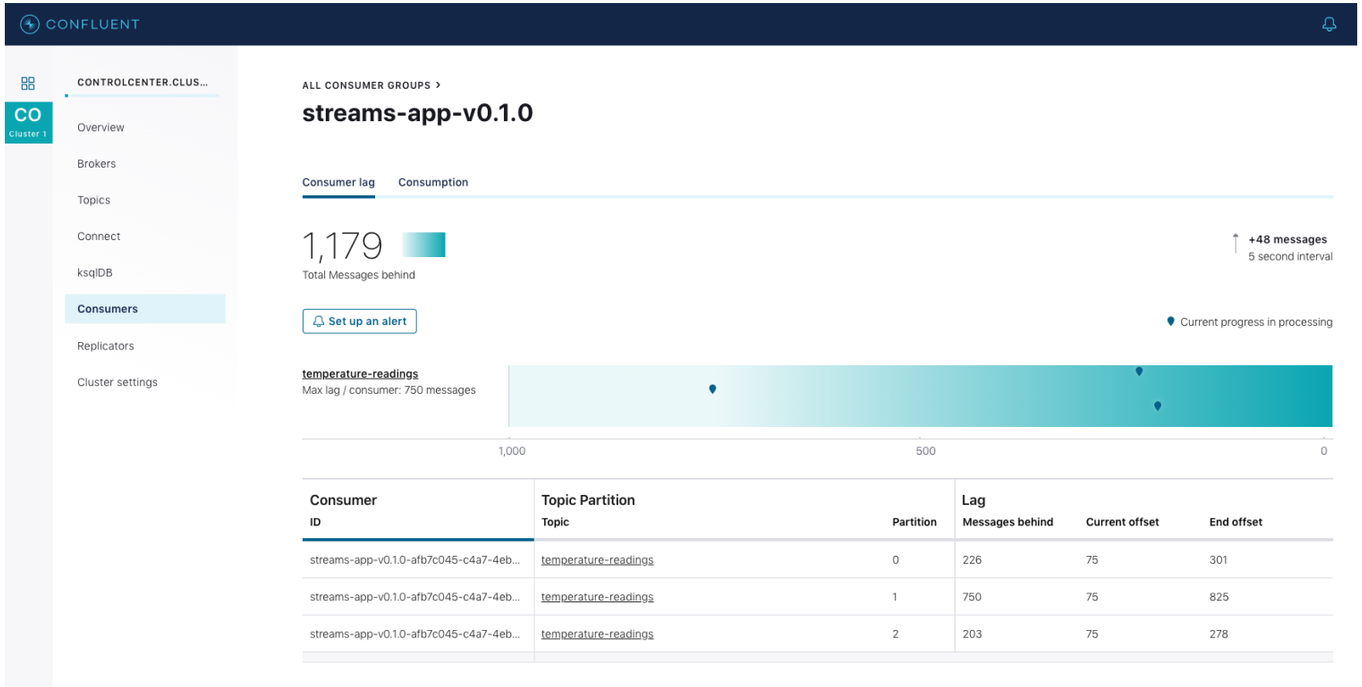
Description

Once a Kafka Streams application runs in production, monitoring it is of the utmost importance. The Kafka Streams library reports a variety of metrics through JMX. Confluent Control Center is one of the ideal tools to use to monitor.

Using JMX-Based Monitoring



Confluent Control Center - Monitoring Interceptors



- Set `producer.interceptor.classes` equal to:
  `io.confluent.monitoring.clients.interceptor.MonitoringProducerInterceptor`
- Set `consumer.interceptor.classes` equal to:
  `io.confluent.monitoring.clients.interceptor.MonitoringConsumerInterceptor`

ksqlDB Built-in Metrics

Metric type	Description
All persistent queries	Describe the full set of persistent queries on a given server
Persistent query status	Describe the health of each persistent query
Persistent query production	Describe the producer activity of each persistent query
Persistent query consumption	Describe the consumer activity of each persistent query
Pull queries	Describe the activity of pull queries on each server
User-defined functions	Describe the activity of user-defined functions, both in-built and custom added

Lab: Getting Metrics From a Kafka Streams Application

Please work on **Lab 9c: Getting Metrics From a Kafka Streams Application**

Refer to the Exercise Guide

Lab: Using JConsole to Monitor a Streams App

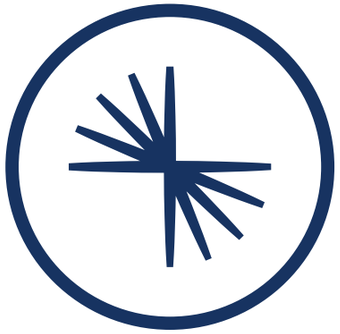
Please work on **Lab 9d: Using JConsole to Monitor a Streams App**

Refer to the Exercise Guide

Lab: Monitoring a Kafka Streams App in Confluent Control Center

Please work on **Lab 9e: Monitoring a Kafka Streams App in Confluent Control Center**

Refer to the Exercise Guide



CONFLUENT

Global Education

Module Overview

This module contains five lessons:

- How Can You Leverage Parallelism in Stream Processing?
- What if You Need to Adjust Processing Power in Your Stream Processing Deployment?
- How Can I Make Your Stream Processing Deal with Failures?
- What Are Some Guidelines for Sizing Your Stream Processing Deployment?
- What Configurations Should You Set for Kafka Streams and ksqlDB?

Where this fits in:

- Hard Prerequisite: Introduction to Kafka Streams, Introduction to ksqlDB
- Recommended Prerequisite: Working with Kafka Streams, Using ksqlDB
- Recommended Follow-Up: Either other module in this branch

10a: How Can You Leverage Parallelism in Stream Processing?

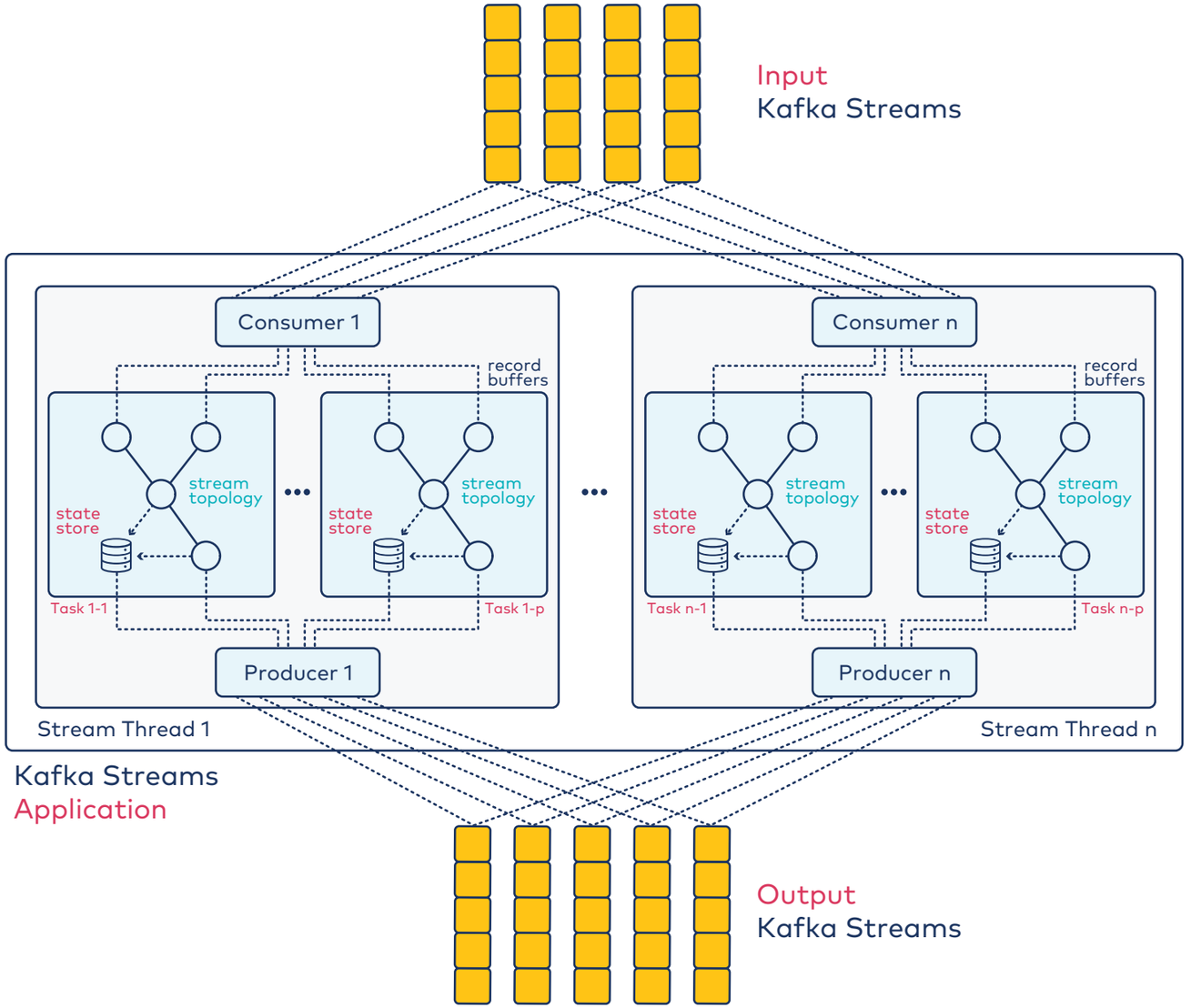
Description

Kafka Streams uses the Apache Kafka producer and consumer APIs, and leverages the native capabilities of Kafka to offer data parallelism, distributed coordination, fault tolerance, and operational simplicity. In Kafka Streams, the basic unit of parallelism is a stream task. So, to improve the parallelism, increase the number of partitions for the input topics which will automatically lead to a proportional increase in the number of tasks.

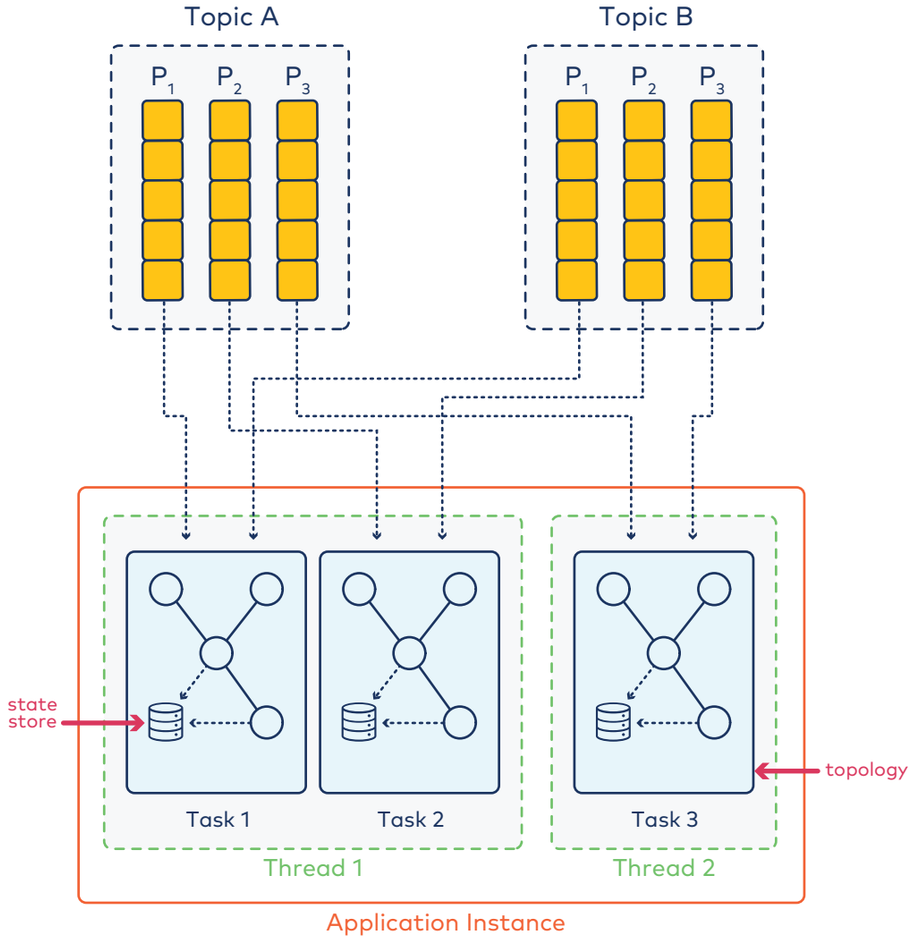
Deployment Concepts

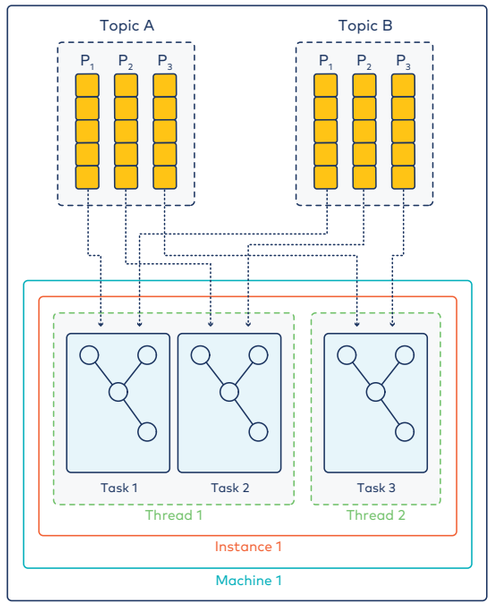
- Kafka Streams uses Kafka's **Producer and Consumer APIs**
- Unit of parallelism is a **Task**
- Task **Placement** matters
- Load Balancing is **automatic**

Made of Consumers and Producers

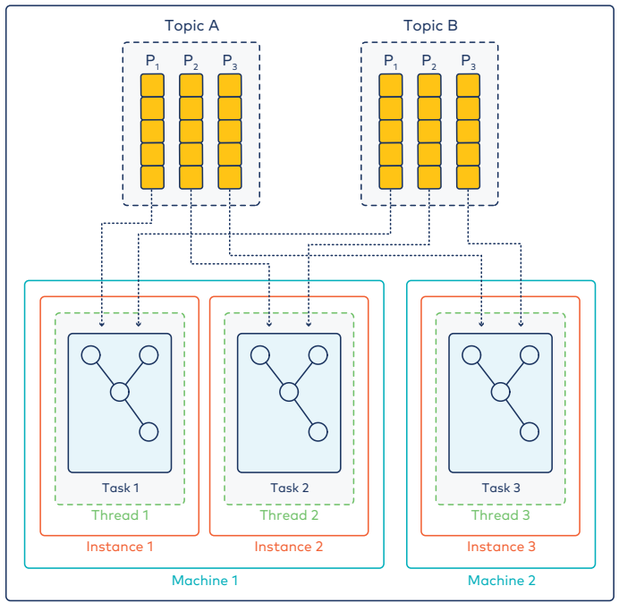


Task - Unit of Parallelism



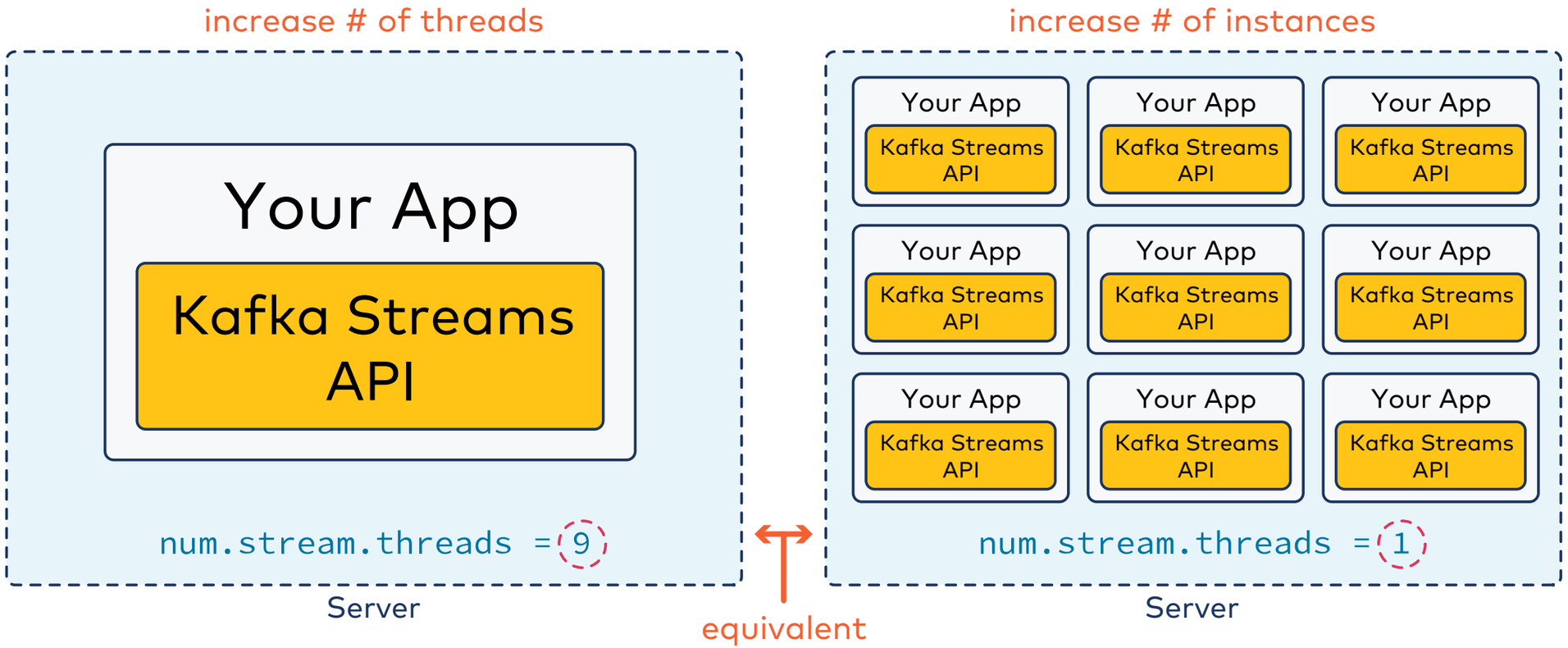


Single Server

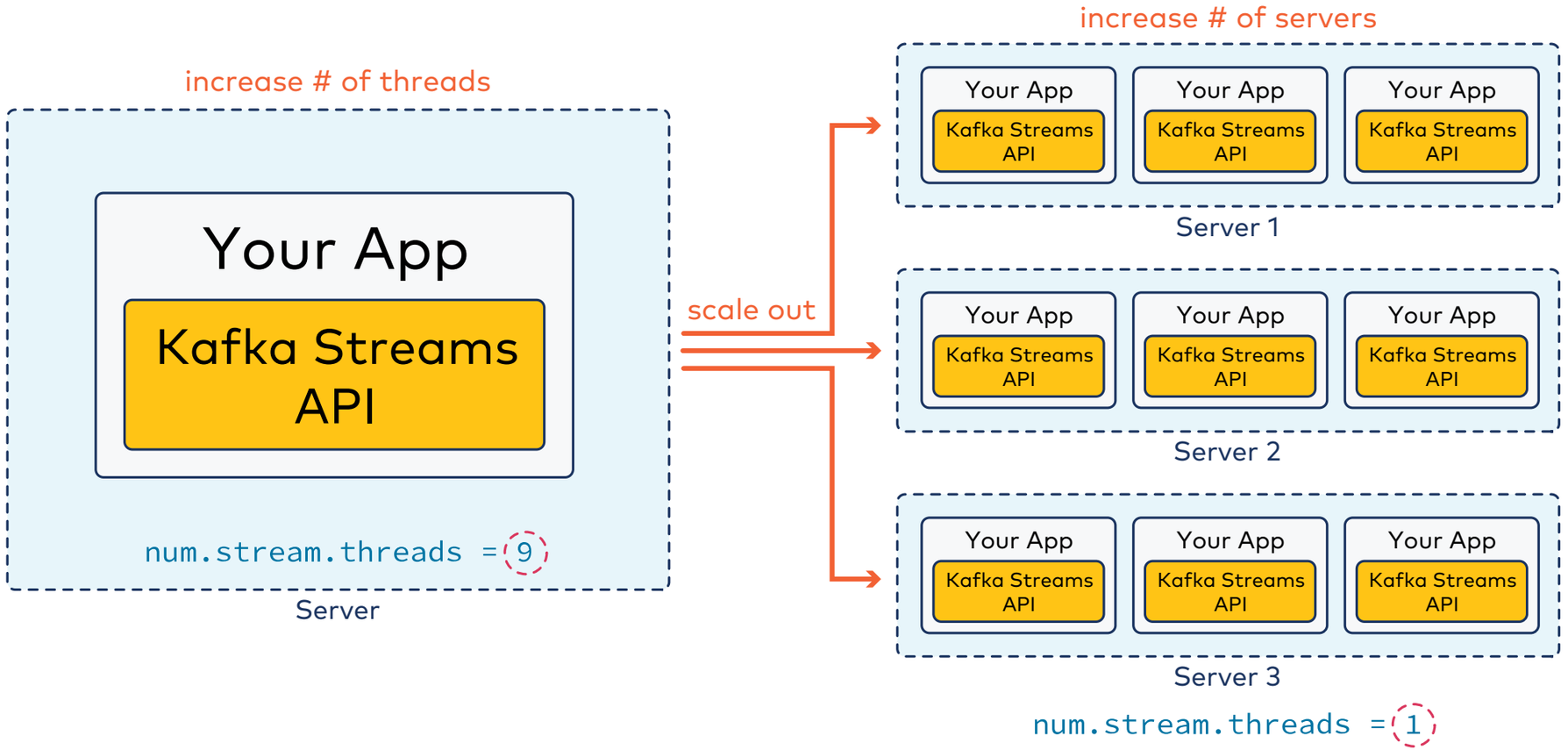


Multiple Servers

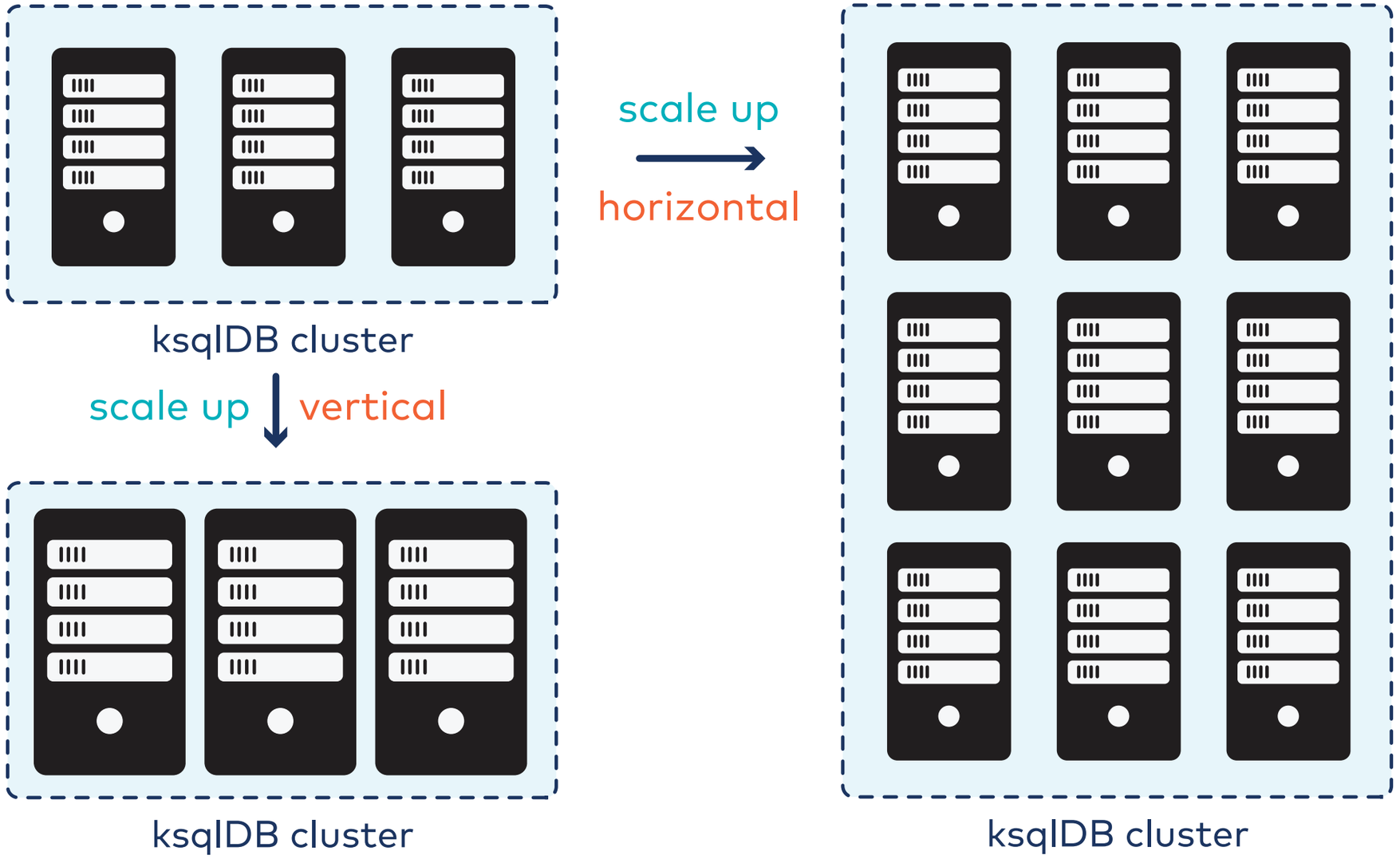
Task Placement - Scale Up



Task Placement - Scale Out



Scaling ksqlDB

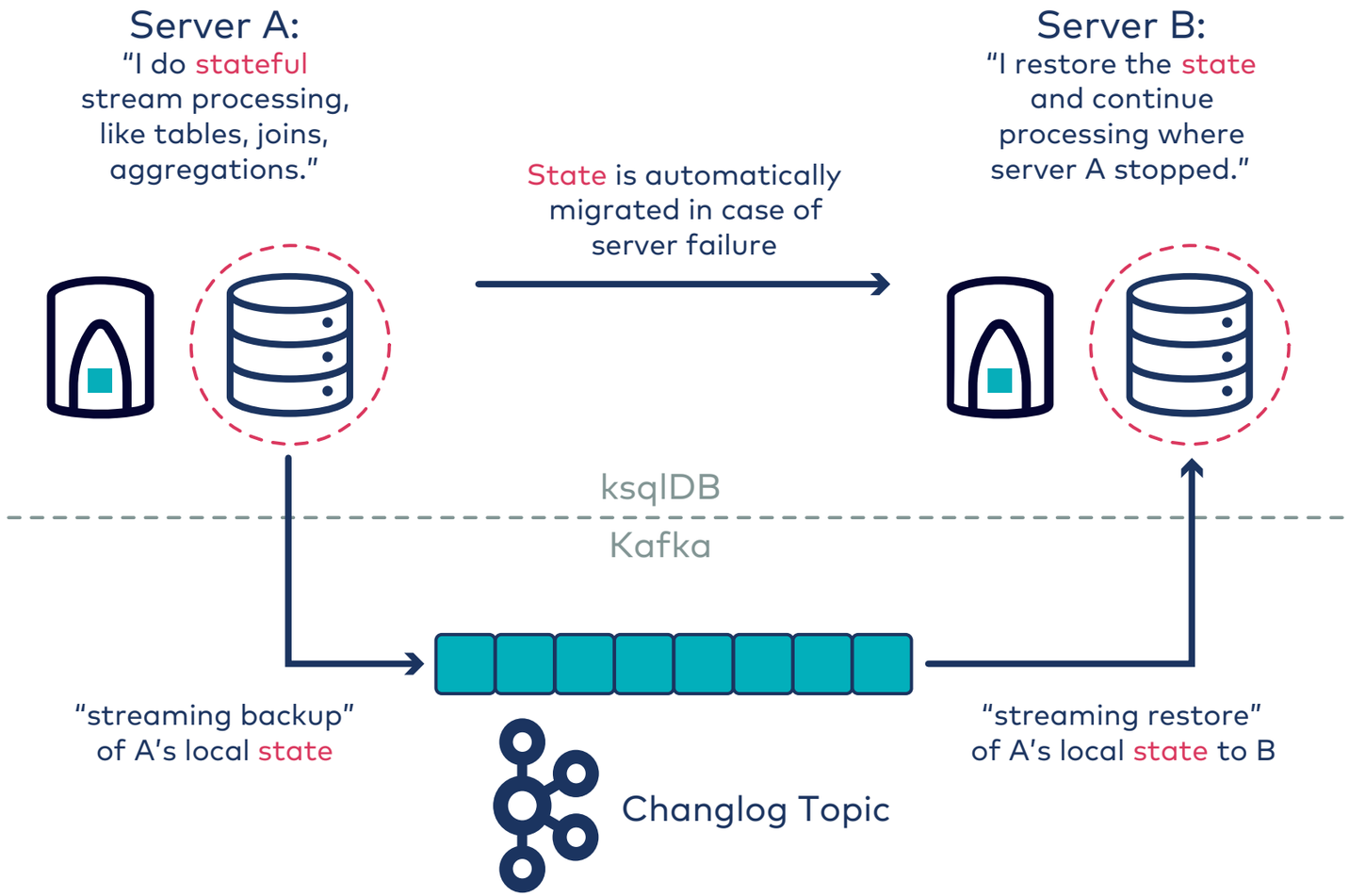


10c: How Can I Make Your Stream Processing Deal With Failures?

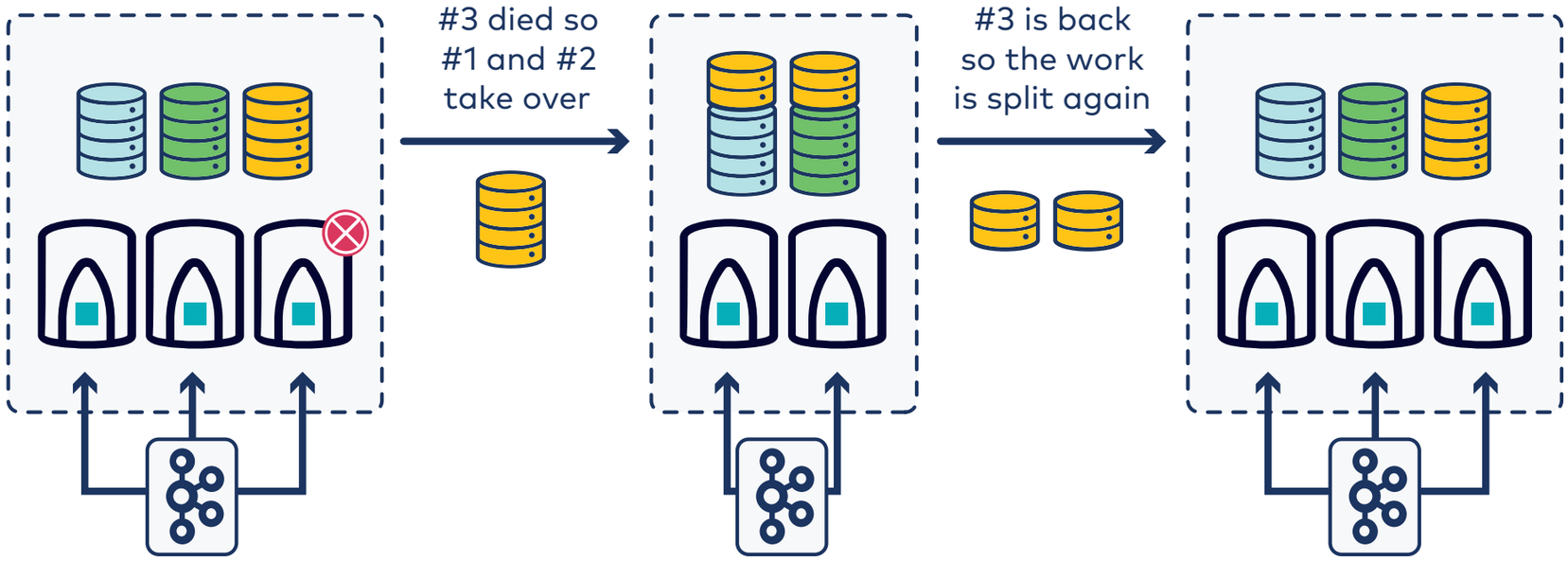
Description

Kafka Streams uses the Kafka Group Coordination Protocol which provides automatic fault tolerance and load sharing. Configure `num.standby.replicas` to be 1 or greater to reduce the recovery time during the fault.

Fault Tolerance Powered by Kafka



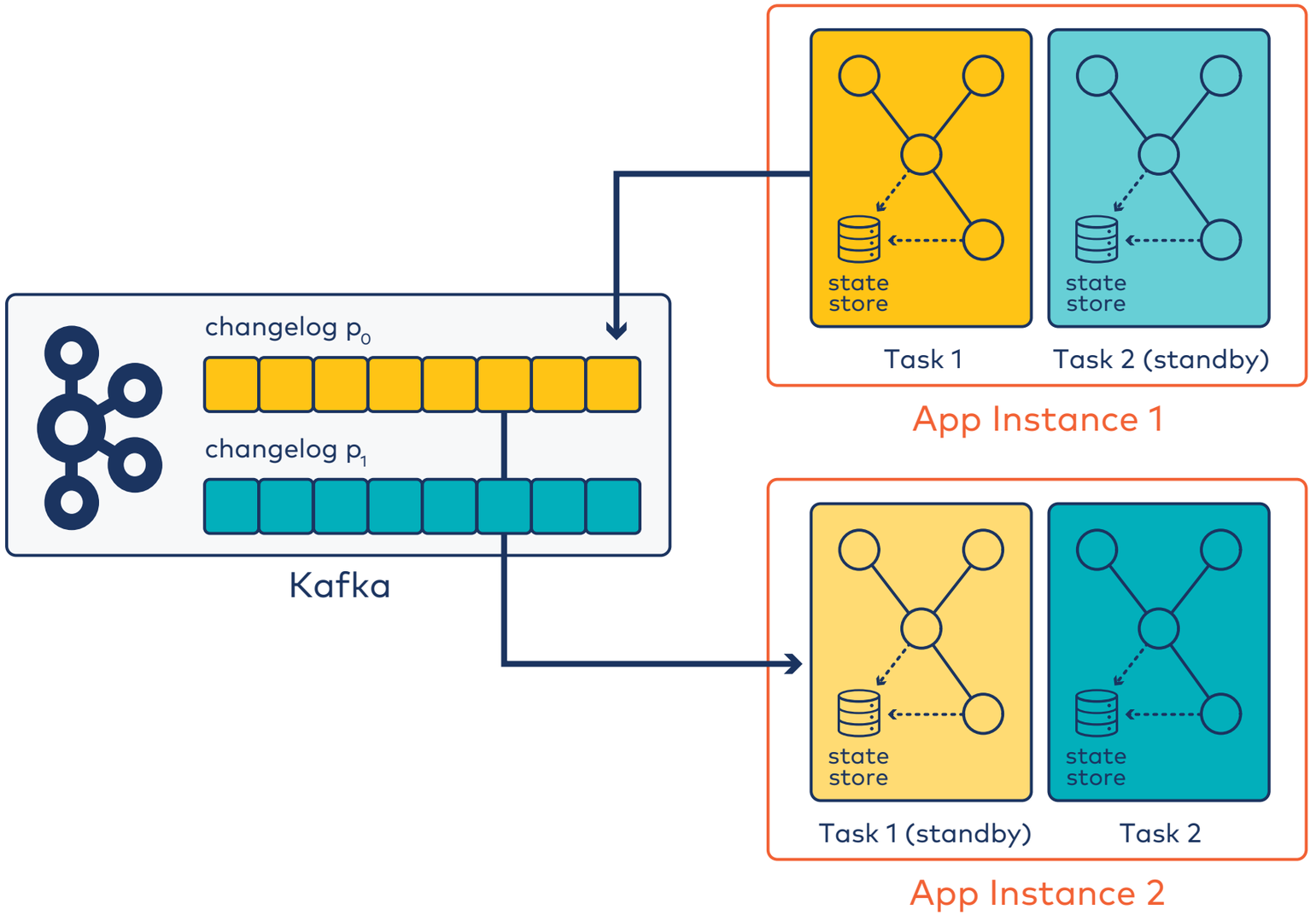
Fault Tolerance Powered by Kafka

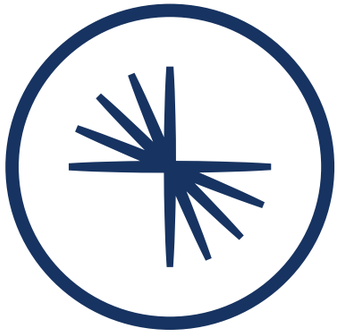


1. Kafka consumer group rebalance is triggered
2. Processing and state of #3 is migrated via Kafka to remaining servers #1 + #2

1. Kafka consumer group rebalance is triggered
2. Part of processing incl. state is migrated via Kafka from #1 + #2 to server #3

Standby Replicas





CONFLUENT

Global Education

Module Overview

This module contains one lesson:

- How Do You Secure Your Stream Processing?

Where this fits in:

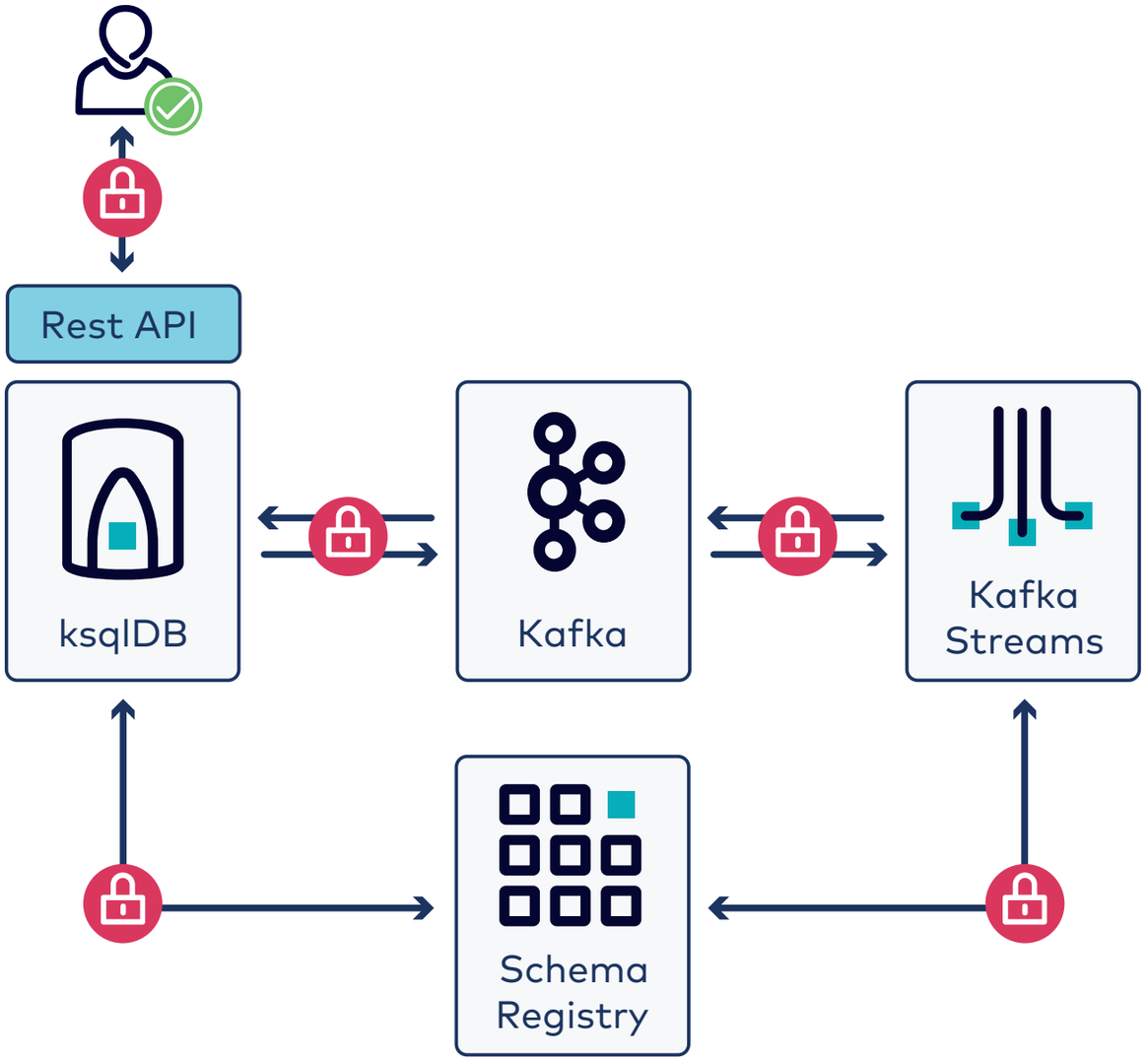
- Hard Prerequisite: Introduction to Kafka Streams, Introduction to ksqlDB
- Recommended Prerequisite: Working with Kafka Streams, Using ksqlDB
- Recommended Follow-Up: Either of Deployment or Testing, Troubleshooting, and Monitoring

11a: How Do You Secure Your Stream Processing?

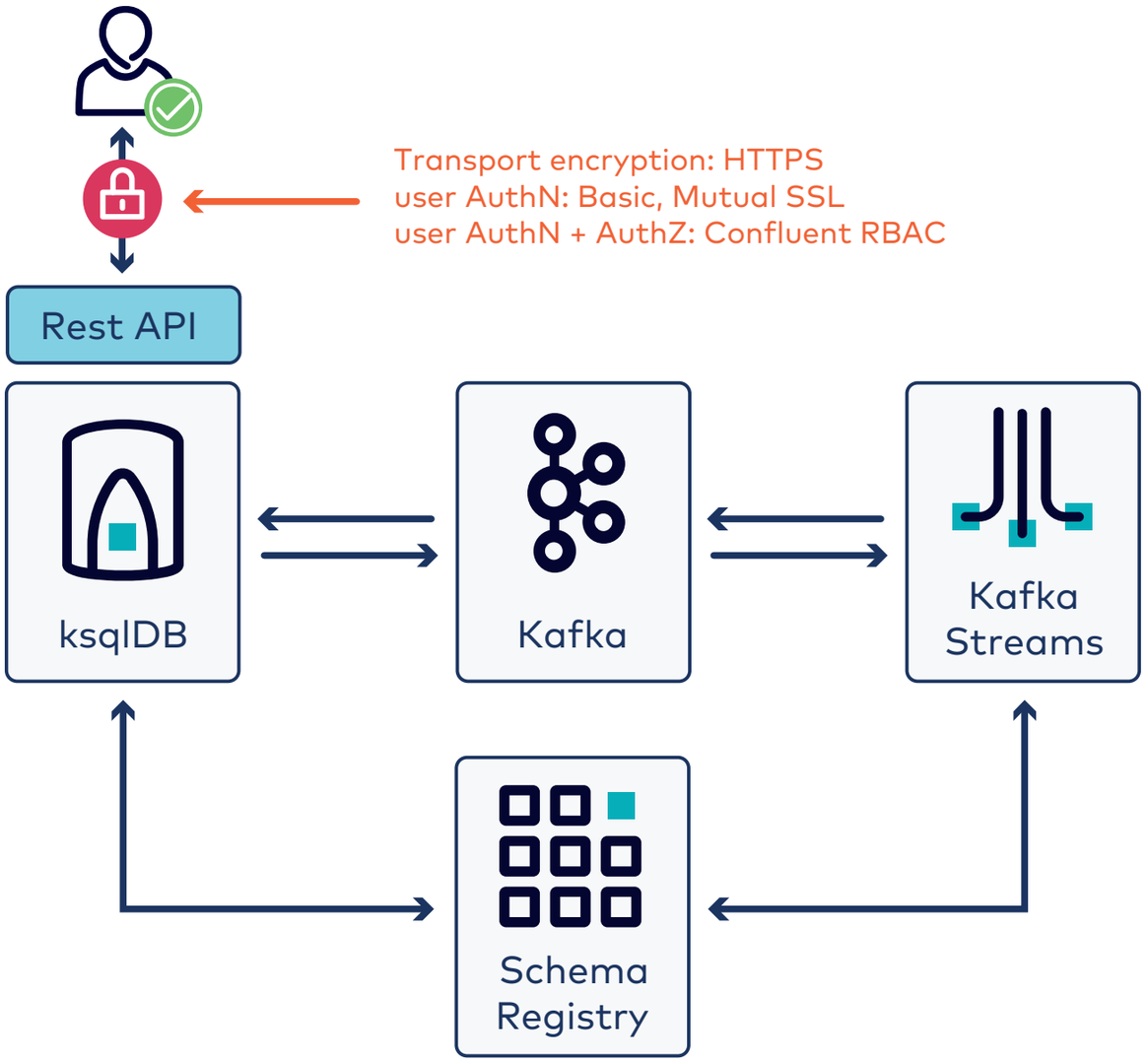
Description

How do you ensure only verified entities can access your Kafka Streams and ksqlDB applications? This lesson explores what you need to know to secure access to your ksqlDB cluster and Kafka Streams applications.

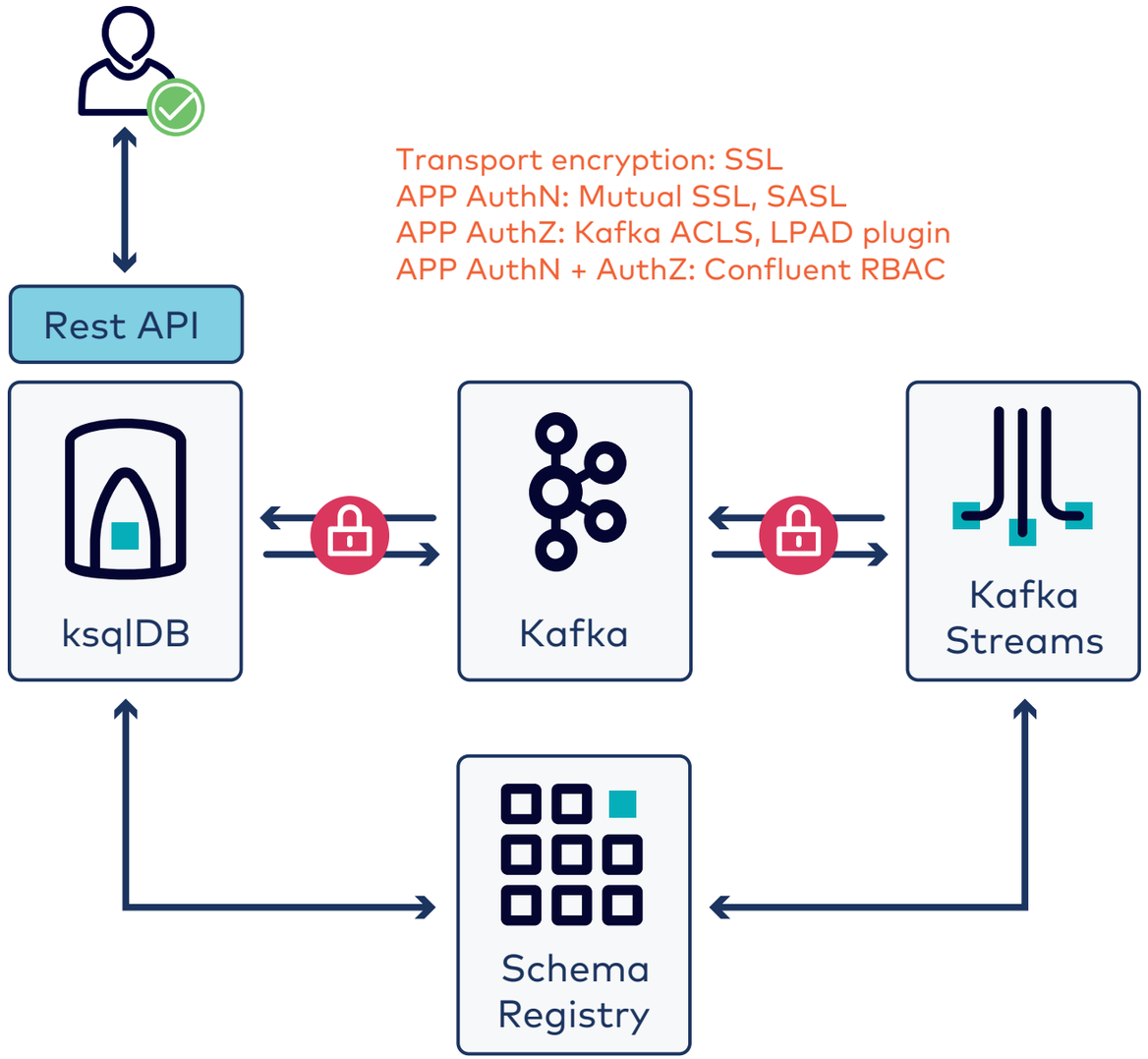
Security Overview



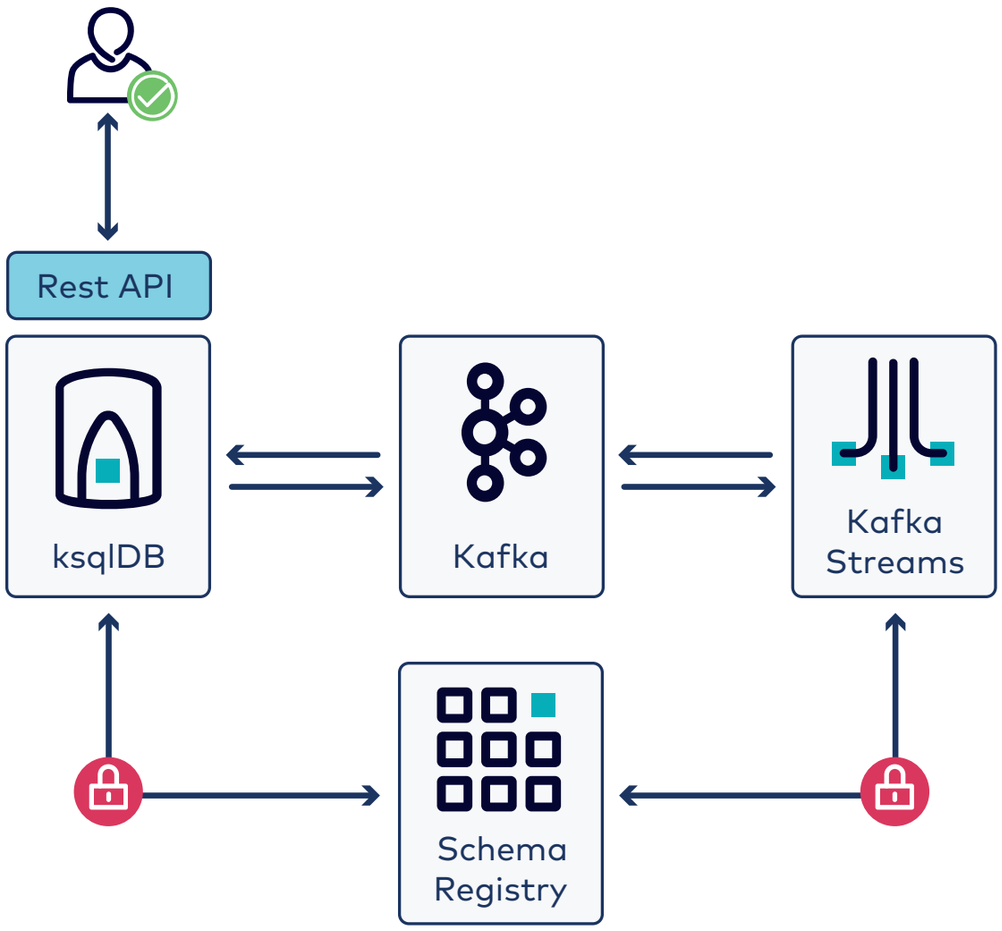
Security Overview — ksqlDB REST API Users



Security Overview — Connecting to Kafka



Security Overview — Schema Registry



Transport encryption: HTTPS
AuthN: Basic, Mutual SSL
AuthZ: Schema Registry ACL Plugin
AuthN + AuthZ: Confluent RBAC

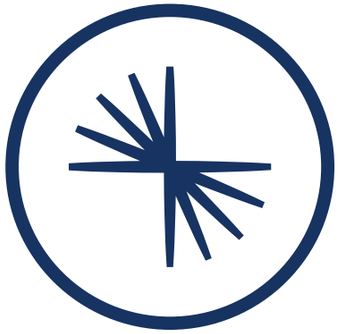
Access Control Lists (ACLs)

- The `DESCRIBE_CONFIGS` operation on the CLUSTER resource type.
- The `CREATE` operation on the CLUSTER resource type.
- The `DESCRIBE`, `READ`, `WRITE`, and `DELETE` operations on all TOPIC resource types.
- The `DESCRIBE` and `READ` operations on all GROUP resource types.

Lab: Securing a Kafka Streams Application

Please work on **Lab 11a: Securing a Kafka Streams Application**

Refer to the Exercise Guide



CONFLUENT

Global Education

Course Contents

Now that you have completed this course, you should have the skills to:

- Identify common patterns and use cases for real-time stream processing
- Describe the high-level architecture of Apache Kafka Streams
- Write real-time applications with the Kafka Streams API to filter, transform, enrich, aggregate, and join data streams
- Describe how ksqlDB combines the elastic, fault-tolerant, high-performance stream processing capabilities of Kafka Streams with the simplicity of a SQL-like syntax
- Author ksqlDB queries that showcase their balance of power and simplicity
- Test, secure, deploy, and monitor Kafka Streams applications and ksqlDB queries

Other Confluent Training Courses

- Confluent Developer Skills for Building Apache Kafka®
- Apache Kafka® Administration by Confluent
- Confluent Advanced Skills for Optimizing Apache Kafka®
- Managing Data in Motion with Confluent Cloud

For more details, see https://confluent.io/training

Confluent Certified Developer for Apache Kafka

Duration: 90 minutes

Qualifications: Solid understanding of Apache Kafka and Confluent products, and 6-to-9 months hands-on experience

Availability: Live, online, 24-hours a day!

Cost: $150

Register online: `www.confluent.io/certification`

Confluent Certified Administrator for Apache Kafka

Duration: 90 minutes

Qualifications: Solid work foundation in Confluent products and 6-to-9 months hands-on experience

Availability: Live, online, 24-hours per day!

Cost: $150

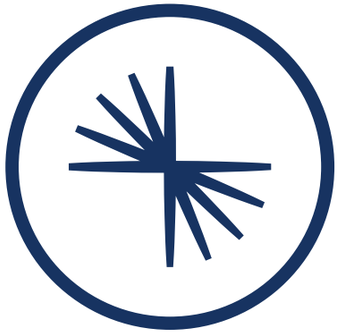
Register online: `www.confluent.io/certification`

We Appreciate Your Feedback!

- Please complete the course survey now.
- For additional feedback, email `training-admin@confluent.io`.

Thank You!

Additional Problems to Solve



CONFLUENT

Global Education

Overview

This section contains a few additional problems problems to be solved that will reinforce the concepts in this course.

Some of these problems were originally written as warm-up problems for instructor-led training for this course. Your instructor may or may not choose to incorporate some or all of these problems in class; you may find them to provide additional enrichment in any case. Some other problems originally created as warm-up problems have been adapted into activities in the content of this version of this course.

Some other problems were written as "food for thought" extra problems, not necessarily intended to be used in the flow of a class, but intended to give curious students additional problems to think about.

Problem A: Getting Started with Stream Concepts

Suppose we are working with a Kafka cluster that has topic `purchases_topic`, which has these key-value pairs: (*a*, 15), (*b*, 52), (*b*, 32), (*c*, 2), (*a*, 21), (*c*, 71). Keys can be interpreted as an ID of a user, and values can be interpreted as how much a purchase costs, rounded to the nearest dollar. Then…

a. If we create a stream from this topic with the current data, what is in the stream?
b. If we treat that stream as a table, what is in the table?
c. How could you interpret the meaning of what is the table?

Problem B: DSL

Problem 1

Look at this code, reformatted from the slides in the submodule "Anatomy of a Kafka Streams App":

```
1  Properties settings;
2  Serde<String> stringSerde;
3  Serde<Double> doubleSerde;
4  StreamsBuilder builder;
5  KStream<String, Double> temps;
6  KStream<String, Double> highTemps;
7  Topology topology;
8  KafkaStreams streams;
9
10 // ...
11
12 stringSerde = serdes.String();
13 doubleSerde = serdes.Double();
14
15 builder = new StreamsBuilder();
```

```
16
17 temps = builder.stream("temp-topic",
18                         Consumed.with(stringSerde, doubleSerde));
19
20 highTemps = temps.filter((key, value) -> value > 25);
21
22 highTemps.to("high-temp-topic",
23              Produced.with(stringSerde, doubleSerde));
24
25 topology = builder.build();
26
27 streams = new KafkaStreams(topology, settings);
28 streams.start();
29
30 //...
```

Consider these two lines of code:

a. `highTemps = temps.filter((key, value) → value > 25);`

b. `streams.start()`

Which executes first? Explain.

Problem 2

Suppose you have a stream of events whose keys are account numbers and whose values are delimited text listings of transactions for the corresponding account for a month at a time. Your goal is to create a stream where keys are account numbers and values are *individual* transactions parsed from the input stream. You plan to use the Kafka Streams DSL to do this.

a. Would you alter the existing stream or create a new stream? Why?
b. What DSL operation would be ideal to achieve this task? Explain.
c. What DSL operation would achieve this task, but be a poor choice? Explain.

Problem C: Aggregating a KTable - Demographic Data

Consider the Step by Step KTable aggregation example on the Slide "Aggregating a KTable - Step by Step." This problem is in the same vein, but a second example for you to work out. The end goal here would be to calculate the average age of users by postal code. Here is our problem setup to do this...

- Inputs will be tuples: (user ID, (postal code, age in years))
- State will be a key-value pair, where the keys are postal codes and values are, in turn, pairs of (total age of users in postal code, number of users in postal code).
- Beyond the problem at hand, one would simply do a final division step for each state element.

Fill in a copy of this table, step by step, as modeled in the slide, but for the scenario described above:

Time-stamp	Input Record	Inter-preted As	Grouping	Initializer	Adder	Subtractor	Changed State
1	(a, (16802, 20))						
2	(b, (16802, 18))						
3	(c, (16803, 70))						
4	(d, (16801, 35))						

Time-stamp	Input Record	Inter-preted As	Grouping	Initializer	Adder	Subtractor	Changed State
5	(a, (16801, 20))						
6	(e, (16802, 19))						
7	(b, null)						

Problem D: Windowing

Problem 1

Let's pretend that, for whatever reason, we are required to choose only some ksqlDB features to keep and cannot keep them all. We are only permitted to use one of tumbling or hopping windows, but not both. Which would you pick and why?

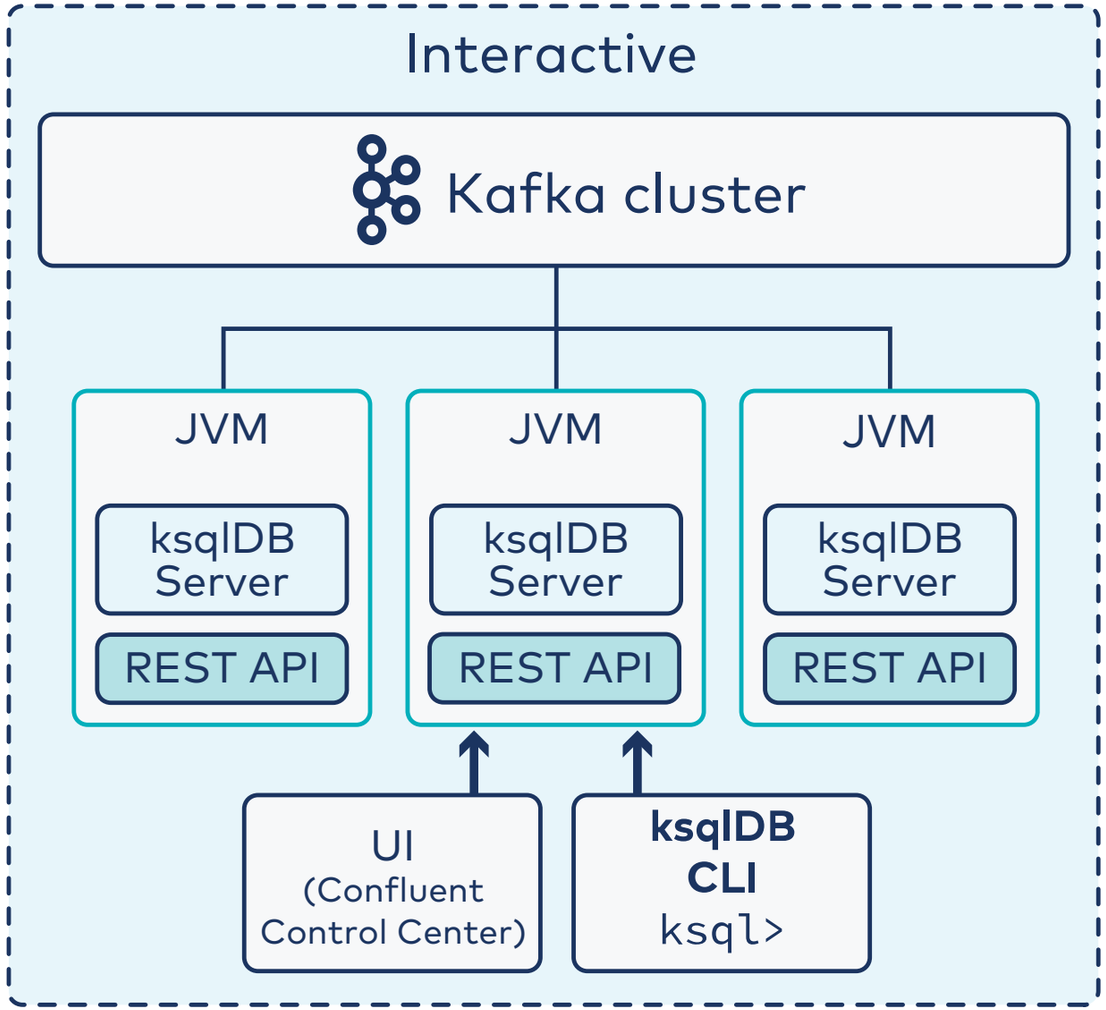
Problem 2

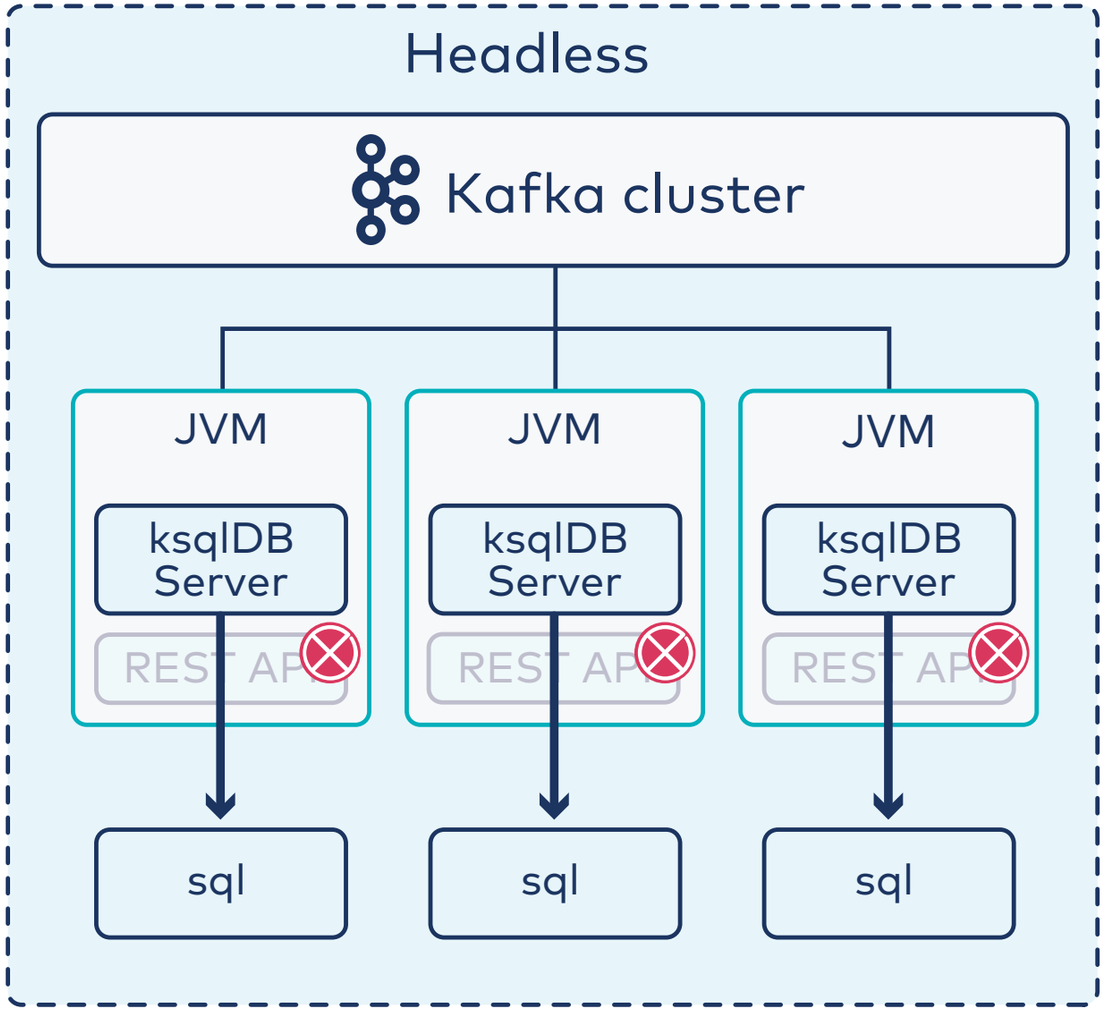
Suppose we have a timeline of click events that happen at the following times: 1, 2, 4, 7, 8, 9, 12, 14, 17, 19. List the times of events included in each window if the windowing mode is...

1. Tumbling with size 5
2. Hopping with size 5 and "advance by" 3
3. Sliding with size 5

Problem E: Deployment Modes

There are two deployment modes for ksqlDB:





You want to run ksqlDB SQL queries at the command line. Should/could the ksqlDB server(s) be run in interactive mode or headless mode? Could both work? Neither? Does it matter? Why or why not?

Problem F: Aggregating Where the Adder Isn't Adding; Reduce

On the Slide titled "Aggregating a KStream," we talk about using `aggregate()` to add up the lengths of some strings. In that example, our adder is literally an adder, but it doesn't need to be.

If you come from a programming or CS background, you may recall in your first programming class learning how to solve some of the classic problems you can solve with loops - like sums and counts. Extreme values fit into the same group. Read the problems below and see if you can solve them:

a. Using `aggregate`, compute the maximum value in `inputStream` (assuming it is as on the referenced slide, a stream with `Long` values. (Hint: the ternary conditional operator is your friend.)
b. Look a few slides ahead and look up the documentation for `reduce()` and use it to solve the same problem as (a).
c. We were able to use both `aggregate()` and `reduce()` to solve this problem. Why? What is a characteristic of that, if changed, would make one or the other not suitable?

Problem G: Repartitioning Streams

Suppose we have a Kafka topic *t* with numeric keys and values that are tuples with a character and a numeric value. Suppose *t* has two partitions:

p_0:
- [2, (*a*, 10)]
- [2, (*b*, 6)]
- [4, (*a*, 7)]
- [4, (*a*, 25)]

p_1:
- [1, (*b*, 3)]
- [3, (*a*, 7)]

Under the hood, streams are partitioned. Consider the following:

a. Suppose we've initialized and built a stream `tStream` from *t*. What do you expect the partitioning for the resulting stream to be?

b. Suppose this code runs:

```
tStream.map((key, value) -> KeyValue.pair(value.getLetter()),
                                          new Pair(key,
                                                   value.getNumber()))
```

What do you expect the partitioning for the intermediate result to be?

c. Suppose we add the following on to the previous code:

```
        .filter((key, value) -> (value.number <= 10))
```

What do you expect the partitioning for the intermediate result to be?

d. Now suppose we further add on to the previous code:

```
        .groupByKey()
```

What do you expect the partitioning for the intermediate result to be?

e. At some point among the above operations, under the hood, information is produced to a new Kafka topic and a subtopology reads it back to the Streams app. When do you expect that to happen?

f. Finally, suppose the code in (b) instead used `mapValues` and made the output value the letter part of the input value. How would your answers to (b)-(e) change? Why?

Problem H: Using the Branch Operation

Consider the module 2 slide "Stateless Operations - branch." Think of a practical application, ideally in your company's context, where you could leverage this. Draw the processor topology.

Problem I: Challenges With ksqlDB Queries

In Chicago, the cornerstone of the public transit system is href="the 'L' (so-called because it's the eLevated railway in many places). It's organized into several color-coded train lines. Suppose you're advising the transit authority on using ksqlDB to get some information on ridership to help decide where it can have a positive impact by renovating a station or stations.

One other bit of background is that the city is organized on a grid system, where any point in the city can be expressed by a north-south coordinate and an east-west coordinate relative to the city center, e.g., 3600 N, 940 W. So, a table of station info would likely have a field for each of those coordinates. As for other field names in streams and tables, as well as the streams and tables themselves, make up something sensible. So then:

a. Write a ksqlDB SQL query to determine the 3 busiest stations based on ridership per station such that...
   - you consider only red line stations

- you consider only between 3500 S and 4000 N
- you consider only between 7 a.m. and 10 a.m....
- ...but only on weekdays
- ...and not holidays
- you leave out stations that have been renovated in the last decade
- you get a report for every 20 minutes

b. If you were actually writing such a query on using the ksqlDB CLI set up with the right streams and tables populated with data, how would you go about it to minimize errors?

c. Even better, describe how you would refine the above to
   - not just do red line but get a report for each line...
   - ...with just the top 3 per line
   - ...and do this in a way you can see all trends, e.g., don't leave out the 20-minute window from 8:46 to 9:06 that has the biggest ridership peak that would be lost in both a window from 8:40 to 9:00 and from 9:00 to 9:20

For example, for each window, you'd get the top 3 stations for red, top 3 for purple, top 3 for blue, etc.

d. Chicago coordinates are such that every 800 in a coordinate, in general, is one mile. Suppose you wanted your report to include an "as the crow flies" distance from the city center of every station too. Could you achieve this using ksqlDB SQL? If so, how? If not, why not?