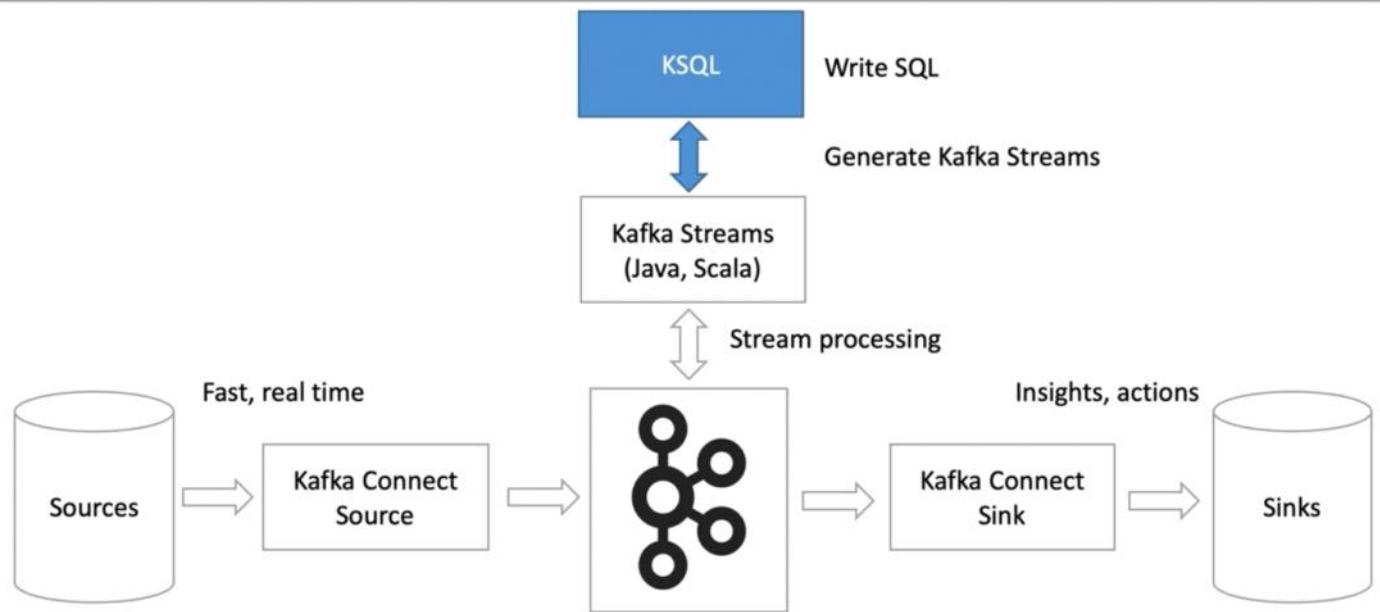




Data Processing Pipelines in Kafka



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

How does KSQL work?



Your computer



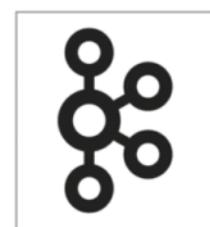
Your computer
Or Remote



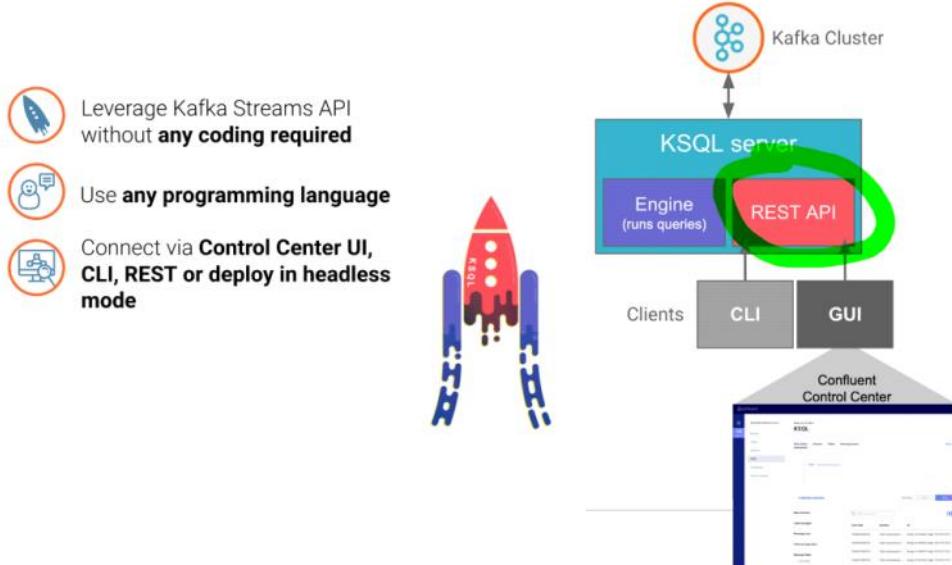
HTTP

HTTP

HTTP



Enable Stream Processing using SQL-like Semantics



KSQL is really two components: the **KSQL Server** and the **KSQL CLI**.

KSQL Server is a pre-built instance of a Kafka Streams application. Rather than coding the streams application in Java, you can develop your processing topology using a higher-level SQL API. The same deployment recommendations for Kafka Streams apply to the instances of the KSQL Server.

Using KSQL to Query & Enrich Data

- **Confluent KSQL** is the streaming SQL engine for Apache Kafka
 - transforms & enriches data in a Kafka cluster
 - does real-time stream processing
 - uses Kafka Streams underneath
- KSQL is an alternative to writing a Java application
- KSQL **STREAM** and **TABLE** abstractions are analogous to Kafka Streams API **KStream** and **KTable**
- KSQL is included in **Confluent Community edition**

KSQL officially launched as part of Confluent Community edition 4.1 and is part of the standard download from that version. Confluent Community edition is free and does not require an enterprise license.

This section is intended to be an introduction to the KSQL. More in-depth discussions and examples can be found in the *Stream Processing with Kafka Streams and KSQL* course, as well as the Confluent Blog and Online Talks: <https://www.confluent.io/blog/> <https://www.confluent.io/online-talks/>

KSQL Deployment Choices



There are two deployment options for KSQL

- Interactive mode
 - KSQL command line “ksql”
 - Confluent Control Center
 - REST API
- “Headless” mode
 - Used for production
 - Scripted only
 - The Headless KSQL server is *not* aware of any streams or tables you defined in other (interactive) KSQL sessions

SQL-like Semantics

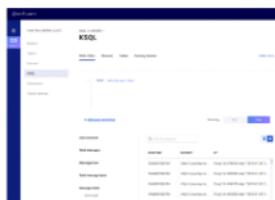
- Identify patterns or anomalies in real-time data, surfaced in milliseconds

```
CREATE TABLE possible_fraud AS
SELECT card_number, count(*)
FROM authorization_attempts
WINDOW TUMBLING (SIZE 5 SECONDS)
GROUP BY card_number
HAVING count(*) > 3;
```

This example creates a new topic called "possible_fraud" by looking at the number of times a field called "card_number" appears in a topic called "authorization_attempts" over 5 second windows. If a "card_number" appears more than 3 times in that window, create a message with the "card_number" as the key and the count as the value in "possible_fraud".

KSQL Can be used Interactively + Programmatically

1 UI



2 CLI



3 REST



4 Headless



The ability to use the KSQL Client from Confluent Control Center was added in CE 5.0.

ksql vs kafka streams

19 декабря 2020 г. 19:08



Apache Kafka® library to write real-time applications and microservices in Java and Scala

```
object FraudFilteringApplication extends App {  
  
    val config = new java.util.Properties  
    config.put(StreamsConfig.APPLICATION_ID_CONFIG, "fraud-filtering-app")  
    config.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "kafka-broker1:9092,kafka-broker2:9092")  
  
    val builder: StreamsBuilder = new StreamsBuilder()  
    val fraudulentPayments: KStream[String, Payment] = builder  
        .stream[String, Payment]("payments-kafka-topic")  
        .filter(_.payment.fraudProbability > 0.8)  
  
    val streams: KafkaStreams = new KafkaStreams(builder.build(), config)  
    streams.start()  
}
```



Confluent KSQL

The streaming SQL engine for Apache Kafka®

```
CREATE STREAM fraudulent_payments AS  
SELECT * FROM payments  
WHERE fraudProbability > 0.8;
```

You write only SQL. No Java, Python, or other boilerplate to wrap around it!

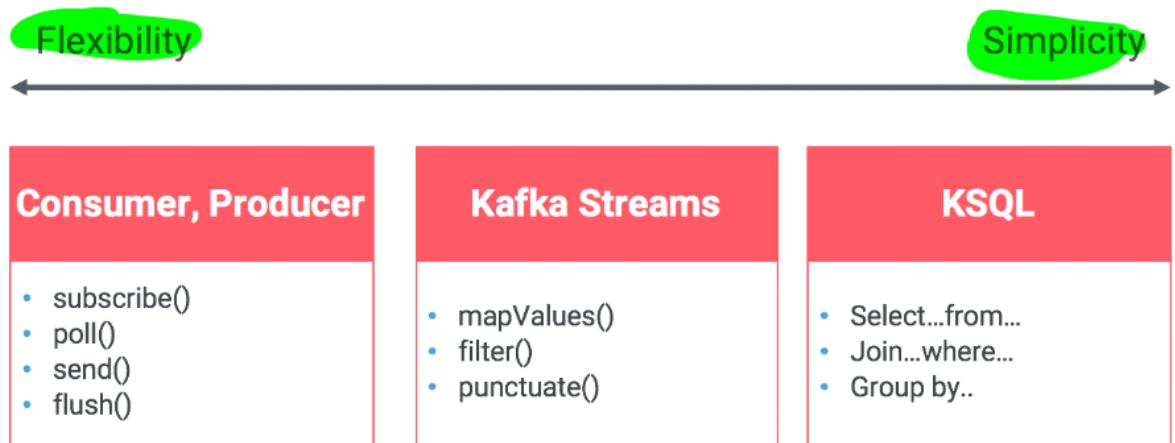
Both, Kafka Streams and KSQL can be used to transform streams of events. KSQL is way more approachable than Kafka streams for a big population of non-Java developer (e.g. data analysts, engineers, business analysts, etc.)

On the slide we see the same result achieved in Java and in KSQL. It is clear that KSQL is free from any boilerplate code. On the other hand we need to admit that Kafka Streams applications offer much more fine grained control to the developer when it comes to stream processing.

Which Approach to Choose?

Comparing KSQL to Kafka Streams to Producer/Consumer

- KSQL is an API around Kafka Streams, and
- Kafka Streams is an API around Producer and Consumer
- Choose the right API for your business



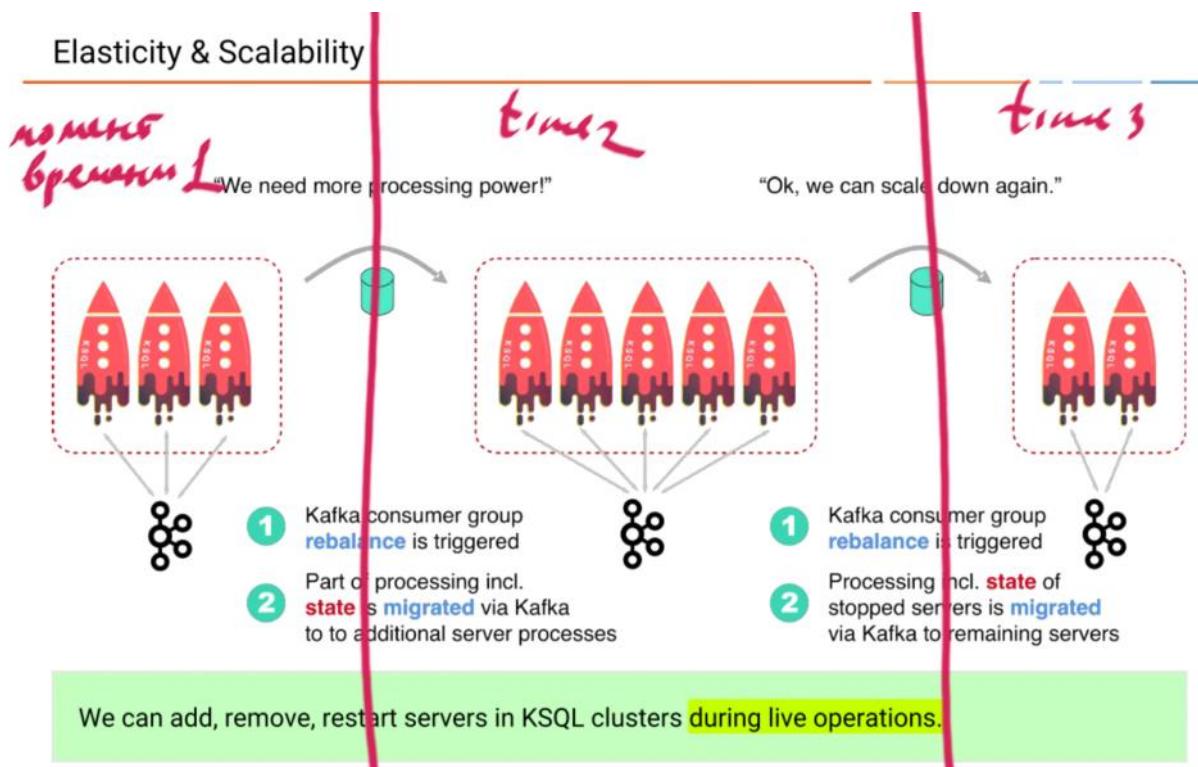
Depending on the level of control required for your environment, you can leverage the appropriate tool.

distributed processing and fault tolerance

19 декабря 2020 г. 20:16

при добавлении нового сервера, новый сервер автоматически подхватит задачи (причем realtaime те во время выполнения уже имеющихся задач)

?? на какой из нод тогда запускать .ksql файл

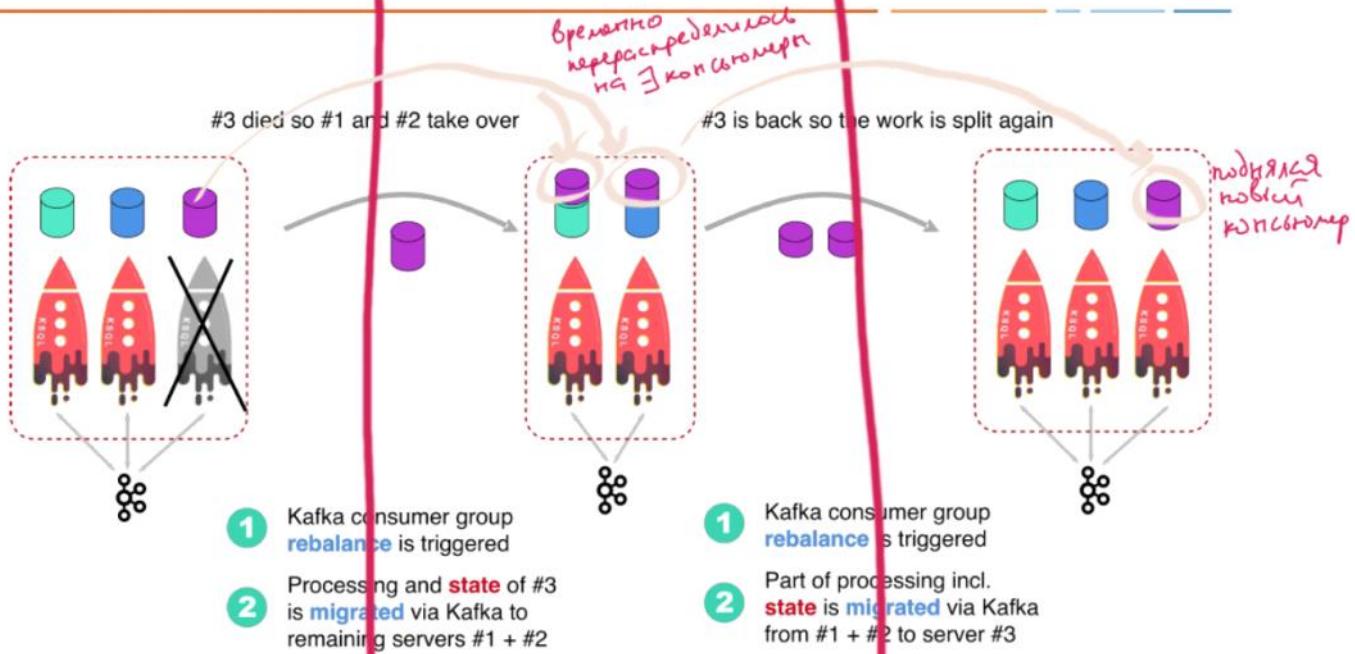


time₁

Failure Tolerance

time₂

time₃



Processing fails over automatically, without data loss or miscomputation.

When one member of a KSQL cluster is failing the mechanism of Kafka Consumer Group is used to automatically trigger a rebalancing of work among the remaining group members.

When a new member is added to the cluster once again the rebalancing happens yet again and the workload is equally distributed to all members of the group.



Scaling Out KSQL

KSQL “runtime” is built on Kafka Streams

- Kafka Streams is a distributed data processing system
- Each instance processes a portion of the data
 - Scale out KSQL cluster by adding new instances
 - Scale in KSQL cluster by removing instances
- If a KSQL instance fails, the remaining servers will take-over and run the failed server’s tasks

ksql.service.id для всех нод должен быть одинаковым

- каждая из ksql нод подсоединенна ко всем kafka нодам

Scaling Out KSQL



Scaling out KSQL server

`bootstrap.server=kb1:9092, kb2:9092, kb3:9092`

`ksql.service.id=myservice`

KSQL Instance 1

KSQL Instance 2

Kafka
Broker 1

Kafka
Broker 2

Kafka
Broker 3

KSQL CLI

Other App

```

# To switch KSQL over to communicating using HTTPS comment out the 'listeners' line above
# uncomment and complete the properties below.
# See: https://docs.confluent.io/current/ksql/docs/installation/server-config/security.html#configuring-ksql-cli-for-https
#
# listeners=https://localhost:8088
# ssl.keystore.location=?
# ssl.keystore.password=?
# ssl.key.password=?

----- Logging config -----

# Automatically create the processing log topic if it does not already exist:
ksql.logging.processing.topic.auto.create=true

# Automatically create a stream within KSQL for the processing log:
ksql.logging.processing.stream.auto.create=true

# Uncomment the following if you wish the errors written to the processing log to include the
# contents of the row that caused the error.
# Note: care should be taken to restrict access to the processing topic if the data KSQL is
# processing contains sensitive information.
#ksql.logging.processing.rows.include=true

----- External service config -----

# The set of Kafka brokers to bootstrap Kafka cluster information from:
bootstrap.servers=localhost:9092

# Uncomment and complete the following to enable KSQL's integration to the Confluent Schema Registry:
#ksql.schema.registry.url=?           I
ksql.service.id=myservicename

```

54,0-1

Property		Default	override	Effective Value
ksql.avro.maps.named		true		
ksql.extension.dir		ext		
ksql.functions.substring.legacy.args		false		
ksql.named.internal.topics		on		
ksql.output.topic.name.prefix				
ksql.persistent.prefix		query_		
ksql.query.persistent.active.limit		2147483647		
ksql.schema.registry.url		http://localhost:8081		
ksql.service.id	SERVER	myservicename	I	
ksql.sink.partitions		4		
ksql.sink.replicas		1		
ksql.sink.window.change.log.additional.retention		1000000		
ksql.statestore.suffix		_ksql_statestore		
ksql.streams.application.id	SERVER	KSQL_REST_SERVER_DEFAULT_APP_ID		
ksql.streams.auto.offset.reset		latest		
ksql.streams.bootstrap.servers	SERVER	localhost:9092		
ksql.streams.cache.max.bytes.buffering	SERVER	10000000		
ksql.streams.commit.interval.ms	SERVER	2000		
ksql.streams.consumer.interceptor.classes	SERVER	io.confluent.monitoring.clients.interceptor.MonitoringConsumerInterceptor		
ksql.streams.default.serialization.exception.handler	SERVER	io.confluent.ksql.errors.LogMetricAndContinueExceptionHandler		
ksql.streams.default.production.exception.handler	SERVER	io.confluent.ksql.errors.ProductionExceptionHandlerUtil\$LogAndFailProductionExceptionHandler		
ksql.streams.num.stream.threads	SERVER	4		
ksql.streams.producer.interceptor.classes	SERVER	io.confluent.monitoring.clients.interceptor.MonitoringProducerInterceptor		
ksql.streams.state.dir	SERVER	/tmp/confluent.yKbx0TN3/ksql-server/data/kafka-streams		
ksql.streams.topology.optimization	SERVER	all		
ksql.transient.prefix		transient_		
ksql.udf.collect.metrics		false		
ksql.udf.enable.security.manager		true		
ksql.udfs.enabled		true		
ksql.windowed.session.key.legacy		false		

файл свойств .properties

- либо задается при старте ksql ноды
- либо подхватится автоматически, если лежит в нужном месте

```

saisbury:ksql$ cd /opt/confluent/etc/ksql
saisbury:ksql$ vi ksql-server.properties

```

KSQL Server Settings

KSQL configuration settings

- Specified at the KSQL server
 - Config file: ksql-server.properties
- Specified for the KSQL session
 - Config file, using the “--config-file” argument
 - Using the "SET" command
- See the configuration settings
 - Using “LIST PROPERTIES”

как задать свойство **перманентно**

- так как если его задать через SET команду в CLI то оно будет действовать **только на период сессии**

Property		Default	override	Effective Value
ksql.avro.maps.named				true
ksql.extension.dir				ext
ksql.functions.substring.legacy.args				false
ksql.named.internal.topics				on
ksql.output.topic.name.prefix				
ksql.persistent.prefix				query_
ksql.query.persistent.active.limit				2147483647
ksql.schema.registry.url				http://localhost:8081
ksql.service.id	SERVER			myservicename
ksql.sink.partitions				4
ksql.sink.replicas				1
ksql.sink.window.change.log.additional.retention				1000000
ksql.statestore.suffix				_ksql_statestore
ksql.streams.application.id	SERVER			KSQL_REST_SERVER_DEFAULT_APP_ID
ksql.streams.auto.offset.reset	SESSION			earliest

RocksDB где реально хранятся данные на диске

Kafka Streams: state stores



- Kafka Streams uses “state stores” for stream processing to store data for stateful operators
 - Windowing, joins or aggregate functions
- These state stores (by default) utilize a RocksDB database
 - Fault-tolerance and recovery for local state stores
 - Storage directory for stateful operations : “ksql.streams.state.dir”

Property		Default	override	Effective Value
ksql.avro.maps.named				true
ksql.extension.dir				ext
ksql.functions.substring.legacy.args				false
ksql.named.internal.topics				on
ksql.output.topic.name.prefix				
ksql.persistent.prefix				query_
ksql.query.persistent.active.limit				2147483647
ksql.schema.registry.url				http://localhost:8081
ksql.service.id	SERVER			myservicename
ksql.sink.partitions				4
ksql.sink.replicas				1
ksql.sink.window.change.log.additional.retention				1000000
ksql.statestore.suffix				_ksql_statestore
ksql.streams.application.id	SERVER			KSQL_REST_SERVER_DEFAULT_APP_ID
ksql.streams.auto.offset.reset				latest
ksql.streams.bootstrap.servers	SERVER			localhost:9092
ksql.streams.cache.max.bytes.buffering	SERVER			10000000
ksql.streams.commit.interval.ms	SERVER			2000
ksql.streams.consumer.interceptor.classes	SERVER			io.confluent.monitoring.clients.interceptor.MonitoringConsumerInterceptor
ksql.streams.default.serialization.exception.handler	SERVER			io.confluent.ksql.errors.LogMetricAndContinueExceptionHandler
ksql.streams.default.production.exception.handler	SERVER			io.confluent.ksql.errors.ProductionExceptionHandlerUtil\$LogAndFailProductionExceptionHandler
ksql.streams.num.stream.threads	SERVER			4
ksql.streams.producer.interceptor.classes	SERVER			io.confluent.monitoring.clients.interceptor.MonitoringProducerInterceptor
ksql.streams.state.dir	SERVER			/tmp/confluent.yKbx0TN3/ksql-server/data/kafka-streams
ksql.streams.topology.optimization	SERVER			all
ksql.transient.prefix				transient

1) CLI

9 декабря 2020 г. 23:38

```
docker-compose exec ksqlcli ksql http://ksqldb-server:8088
```

```
cd C:\Users\vovan\cp-all-in-one\cp-all-in-one  
ed C:\Users\vovan\Downloads\ksqldb-course-master  
docker-compose logs --tail=200 ksqldb-server
```

на самом деле выполнится секция из docker-compose файла

смотрим свойства сервера

Property		Default	override	Effective Value
ksql.avro.maps.named		true		
ksql.extension.dir		ext		
ksql.functions.substring.legacy.args		false		
ksql.named.internal.topics		on		
ksql.output.topic.name.prefix		query_		
ksql.persistent.prefix		2147483647		
ksql.query.persistent.active.limit		http://localhost:8081		
ksql.schema.registry.url		default_		
ksql.service.id		4		
ksql.sink.partitions		1		
ksql.sink.replicas		1000000		
ksql.window.change.log.additional.retention		_ksql_statestore		
ksql.statestore.suffix	SERVER	KSQL_REST_SERVER_DEFAULT_APP_ID		
ksql.streams.application.id		latest		
ksql.streams.auto.offset.reset		localhost:9092		
ksql.streams.bootstrap.servers	SERVER	2000		
ksql.streams.cache.max.bytes.buffering	SERVER	io.confluent.monitoring.clients.interceptor.MonitoringConsumerInterceptor		
ksql.streams.commit.interval.ms	SERVER	io.confluent.ksql.errors.LogMetricAndContinueExceptionHandler		
ksql.streams.consumer.interceptor.classes	SERVER	io.confluent.ksql.errors.ProductionExceptionHandlerUtil\$LogAndFailProductionExceptionHandler		
ksql.streams.default.deserialization.exception.handler	SERVER	4		
ksql.streams.default.production.exception.handler	SERVER	io.confluent.monitoring.clients.interceptor.MonitoringProducerInterceptor		
ksql.streams.num.stream.threads	SERVER	/tmp/confluent.yxB85HXB/ksql-server/data/kafka-streams		
ksql.streams.producer.interceptor.classes	SERVER	all		
ksql.streams.state.dir	SERVER	transient_		
ksql.streams.topology.optimization	SERVER	false		
ksql.transient.prefix		true		
ksql.udf.collect.metrics		true		
ksql.udf.enable.security.manager		false		
ksql.udfs.enabled				
ksql.windowed.session.key.legacy		false		

КОМАНДЫ НАСТРОЙКИ

В качестве примера задания свойства конфигурации рассмотрим, как изменить значение параметра `auto.offset.reset` на `earliest`:

```
SET 'auto.offset.reset'='earliest';
```

2) GUI (Confluent)

19 декабря 2020 г. 19:56

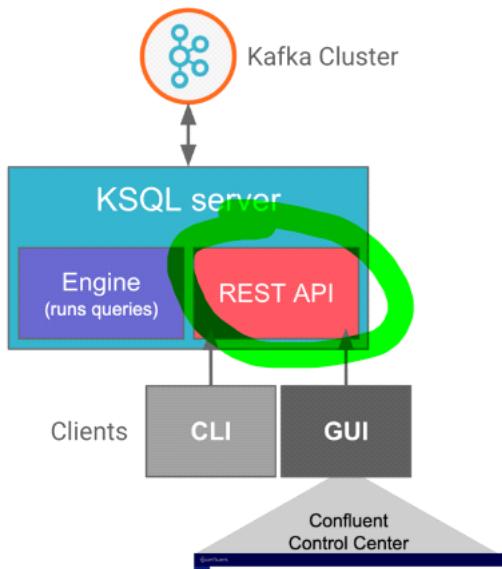
3) REST (ksql PYTHON еще проще чем читать из очереди)

9 декабря 2020 г. 23:16

python-клиент использует REST-апи по HTTP

```
ksql-python.py X
SYSTEMDESIGN > @_KAFKA > _@_OReilly_stephane_maarek > @8@ Apache Kafka
1   from ksql import KSQLAPI
2   # Refer to https://pypi.org/project/ksql/
3
4   client = KSQLAPI('http://localhost:8088')
5   query = client.query('select * from table1')
6   for item in query:
7       print(item)
8
9
10
```

на самом деле REST-арі это основной API для общения с KSQL



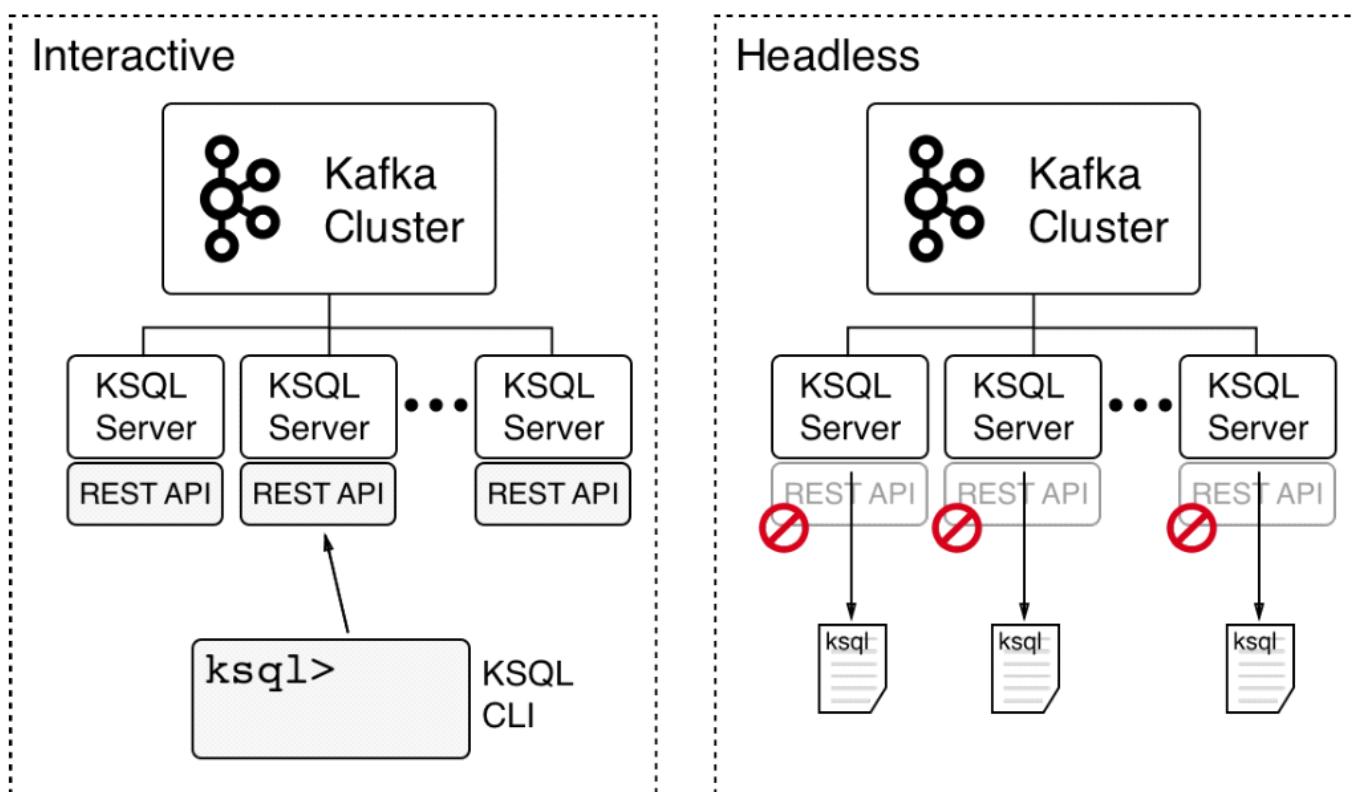
4) headless mode

19 декабря 2020 г. 20:02

в этом режиме REST API выключен (для секьюрности) и сервер выполняет файл-скрипта который лежит на самом сервере

- а также в таком режиме легче прогнозировать производительность так как больше никто извне запросов не выполняет
- лучше для секьюрности

KSQL Server Modes



KSQL can be run in two modes, interactive and headless

- Interactive mode: a user can access a KSQL server being part of a resource group. The access to the KSQL server happens via its REST API
- Headless mode: The KSQL servers are configured to use a **file containing KSQL statements**. In this case the KSQL server automatically disables the REST API and thus cannot be accessed from outside anymore. This is the recommended mode when running in production.

... запуск .ksql файлов (продакшн)

10 декабря 2020 г. 22:23

The screenshot shows a terminal window with a red circle and arrow highlighting the command at the bottom:

```
user_profile.pretty.ksql
1
2 -- This file will generate the user_profile_pretty presentation
3
4
5 -- Change scode default to earliest
6 SET 'auto.offset.reset'='earliest';
7
8 -- Now generate
9 create stream user_profile_pretty as
10 select firstname || ''
11 || ucase( lastname )
12 || '' from || country_code
13 || '' has a rating of ' cast(rating as varchar) || ' stars.
14 || case when rating < 2.5 then 'Poor'
15     when rating between 2.5 and 4.2 then 'Good'
16     else 'Excellent'
17 end as description
18 from userprofile;
19
20
ksql> run script './user_profile.pretty.ksql';
```

особенно это нужно для продакшена (так как CLI отключаем)

KSQL Headless Operations

Headless mode

- Build an application file
 - All the streams we require in a single file
- Shutdown the interactive server
- Start a headless server
 - Pass in application file

--queries-file запуск сервера со стартовым .ksql скриптом (по CLI подключится невозможно)

```
saubury:ksql-course$ /opt/confluent/bin/ksql-server-start /opt/confluent/etc/ksql/ksql-server.properties --queries-file ./where-is-bob.ksq
```

; выражения должны заканчиваться точкой с запятой

KSQL Syntax

- KSQL has similar semantics to SQL
- KSQL statements must be terminated with a semicolon ;
- Multi-line statements

SET 'auto.offset.reset'='earliest'; select * from stream emit changes;

если заново выбрать записи через select* то будет пусто тк записи уже выбраны,

чтобы при каждом запросе select * выбирались все записи нужно выставить настройку
SET 'auto.offset.reset'='earliest';

||| конкатенция строк

```
ksql> select TIMESTAMPTOSTRING(rowtime, 'dd/MMM HH:mm') as createtime, firstname || ' ' || ucase(lastname) as full_name
>from userprofile;
```

я не могу работать прямо с топиком, я должен создать из него STREAM или TABLE

General KSQL Commands

Command	Description
<code>DESCRIBE</code>	List the columns in a stream or table along with their data type and other attributes
<code>CREATE STREAM</code>	Create a new stream with the specified columns and properties
<code>CREATE TABLE</code>	Create a new table with the specified columns and properties
<code>SHOW TOPICS</code>	List the available Topics in the Kafka cluster that KSQL is configured to connect to
<code>SHOW STREAMS</code>	List the defined streams
<code>SHOW TABLES</code>	List the defined tables
<code>DROP</code>	Drop an existing stream or table

persistent SQL и non-persistent SQL

Non-persistent and Persistent Queries

- Non-persistent query
 - The result of a non-persistent query will not be persisted into a Kafka Topic and will only be printed out in the console
- Persistent query
 - The result of a persistent query will be persisted into a Kafka Topic

Command	Description
<code>SELECT</code>	Non-persistent
<code>CREATE STREAM AS SELECT</code>	Persistent
<code>CREATE TABLE AS SELECT</code>	Persistent

`CTRL+C` как остановить non-persistent запрос

`TERMINATE` как остановить persistent выполняющийся запрос -

- но лучше делать более явный `DROP`

`LIMIT` можно просто ограничить число выданных записей

- запрос будет выполняться бесконечно, если его не остановить

- остановка и перезапуск сервера не остановят выполнения запроса

Stopping Queries

- By default, `SELECT` won't stop on its own
 - Since these are never-ending streams of data, there is no inherent "end"
- There are differences in stopping a non-persistent versus persistent query
 - Stop a non-persistent query with `CTRL-C` to the console
 - Stop a persistent query with the `TERMINATE` command
- You may also limit the number of records returned with the `LIMIT` clause

```
1 | TERMINATE query_id;
```

Description

Terminate a persistent query. Persistent queries run continuously until they are explicitly terminated.

- In client-server mode, exiting the CLI doesn't stop persistent queries, because the ksqlDB Server(s) continue to process the queries.

To terminate a non-persistent query, use Ctrl+C in the CLI.

```
-- terminate all queries that output to stream S.  
TERMINATE ALL FOR STREAM S;
```

JSON, AVRO, TEXT commaDelimited как явно задать интерпретацию
содержимого сообщений топика

- но я не могу сам задавать форматы также как в kafka streams

Explore Data in Kafka Topics

- View the content of the Kafka topic `my-pageviews-topic`, which has a JSON payload, in the form of a stream

```
CREATE STREAM pageviews (viewtime BIGINT, user_id VARCHAR, page_id VARCHAR)  
WITH (VALUE_FORMAT = 'JSON', KAFKA_TOPIC = 'my-pageviews-topic');
```

- View the content of the Kafka topic `my-users-topic`, which has a JSON payload, in the form of a changelog

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



you can't query a topic directly in KSQL. You have to define a `STREAM` or `TABLE` against that topic first, which maps a schema on top of the raw binary data stored in the topic.

набор готовых функций

KSQL Functions and Clauses

- KSQL supports many popular scalar functions that have similar meaning to its counterpart in SQL
 - ABS, CEIL, CONCAT, LEN, ROUND, TRIM, SUBSTRING, and many more
 - EXTRACTJSONFIELD: Given a string column in JSON format, extract the field that matches
- You can JOIN streams and tables
- KSQL supports aggregation functions as well
 - COUNT, MAX, MIN, SUM, etc
- Re-key the streams
 - With the PARTITION BY clause, the resulting stream will have a key with the specified column
- Extend KSQL with UDFs and UDAFs

Most of the commands available are the same as are available in standard SQL.

One that is specific to KSQL is the PARTITION BY clause which allows the use of specific fields as keys for the new messages.

If the functions provided out of the box do not suit your needs then it is possible to extend KSQL by authoring user defined functions (UDFs) and user defined aggregate functions (UDAFs) in Java.

case statement

```
select firstname
, ucase(lastname)
, countrycode
, rating
, case when rating < 2.5 then 'Poor'
       when rating between 2.5 and 4.2 then 'Good'
       else 'Excellent'
end
from userprofile;
```

* push query

10 декабря 2020 г. 20:52

select * from stream emit changes; выбор записей из стрима/таблицы

Push queries

- **Push** queries – constantly query & output results
 - We'll introduce to **Pull** queries in a later lesson after tables
- Push queries continue to output results until
 - Terminated by the user
 - Exceed LIMIT condition
- Push queries were the default in KSQL 5.3 and earlier
 - 'EMIT CHANGES' indicates a query is a push query.
 - Required in KSQL against ksqlDB 5.4 onwards

```
select name, countrycode  
from users_stream;
```

KSQL 5.3 and earlier

```
select name, countrycode  
from users_stream  
emit changes;
```

KSQL on ksqlDB 5.4 onwards

push query -продолжают выдавать результаты как поток

Push and Pull Queries

Push queries (refresher) : 'EMIT CHANGES'

- Push queries – constantly query & output results
- Continue to output results until
 - terminated by the user
 - exceed LIMIT condition
- Push queries were the default in KSQL 5.3 and earlier

```
vovan@VOV C:\Users\vovan>kafka-console-producer --broker-list localhost:9092 --topic USERS
>vovan,RU
```

```
ksql> select * from users_stream emit changes;
+-----+-----+
| NAME | COUNTRYCODE |
+-----+-----+
| vovan | RU          |
+-----+-----+
Press CTRL-C to interrupt
```

select `rowtime` показать время создания записи

```
ksql> select rowtime, firstname from userprofile;
```

```
ksql> SELECT TIMESTAMPTOSTRING(rowtime, 'dd/MMM HH:mm') as createtime, firstname
>from userprofile;
```

KSQL Syntax Reference — Confluent

<https://docs.confluent.io/current/ksql/docs/developer-guide/syntax-reference.html#scalar-functions>

Product Cloud Developers Blog Docs Download

Scalar functions

Function	Example	Description
ABS	ABS(col1)	The absolute value of a number.
ARRAYCONTAINS	ARRAYCONTAINS(['1', '2', '3'], 3)	Given JSON array and search value, returns true if value is present.
CEIL	CEIL(col1)	The ceiling function rounds up a number to the nearest integer.
CONCAT	CONCAT(col1, '_hello')	Concatenates two strings.
DATETOSTRING	DATETOSTRING(START_DATE, 'yyyy-mm-dd')	Converts a timestamp into a string in the given format. Given a timestamp, two successive example: represents encoding used.

SHOW FUNCTIONS
SHOW TOPICS
SHOW STREAMS
SHOW TABLES
SHOW QUERIES
SHOW PROPERTIES
TERMINATE
Operators
Scalar functions
Aggregate functions
Key Requirements
Message Keys
What To Do If Your Key Is Not Set or Is In A Different Format
Streams
Tables

where условия

```
ksql> select countrycode, countryname from countrytable;
AU | Australia
IN | India
GB | UK
US | United States
```

```
ksql> select countrycode, countryname from countrytable where countrycode='GB' limit 1;
GB | UK
```

Pull queries



- Pull queries - the current state of the system
- Return a result, and terminates
- New from 5.4
- KSQL currently only supports pull queries on aggregate tables
- Must query against rowkey

```
create table countryDrivers as  
select countrycode, count(*)  
from driverLocations  
group by countrycode
```

```
select countrycode, numdrivers  
from countryDrivers  
where rowkey='AU';
```

например для запросов с группировкой возможны только
запросы типа pull query

* group by

10 декабря 2020 г. 21:08

```
kafka-console-producer --broker-list localhost:9092 --topic USERS << EOF
Alice,US
Bob,GB
Carol,AU
Dan,US
EOF
```

```
ksql> select countrycode, count(*) from users_stream group by countrycode;
+-----+-----+
| AU   | 1   |
| US   | 2   |
| GB   | 2   |
+-----+-----+
```

В интерфейсе KStream есть два метода для группировки записей: [GroupByKey](#) и [GroupBy](#).

- Оба возвращают KGroupedTable, так что у вас может появиться закономерный вопрос: в чем же различие между ними и когда использовать какой из них?
- Метод [GroupByKey](#) применяется, когда ключи в KStream уже непустые. А главное, флаг «требует повторного секционирования» никогда не устанавливался.
- Метод [GroupBy](#) предполагает, что вы меняли ключи для группировки, так что флаг повторного секционирования установлен в true. Выполнение после метода [GroupBy](#) соединений, агрегирования и т. п. [приведет к автоматическому повторному секционированию](#).
- Резюме: следует при малейшей возможности использовать [GroupByKey](#), а не [GroupBy](#).

* stream

10 декабря 2020 г. 21:49

при создании стрима автоматически создаётся и топик (если его до этого не было)

create stream s1 with (KAFKA_TOPIC=""); создать стрим из топика

- отличие от топика будет в том что стрим структурирован (то состав данных понятен)

```
CREATE STREAM userprofile (userid INT, firstname VARCHAR, lastname VARCHAR, countrycode VARCHAR, rating DOUBLE) \
WITH (VALUE_FORMAT = 'JSON', KAFKA_TOPIC = 'USERPROFILE');
```

```
kafka-topics --zookeeper localhost:2181 --create --partitions 1 --replication-factor 1 --topic USERPROFILE
```

```
saubury:ksql-udemy$ kafka-console-producer --broker-list localhost:9092 --topic USERPROFILE << EOF
>
> {"userid": 1000, "firstname":"Alison", "lastname":"Smith", "countrycode":"GB", "rating":4.7}
```

```
ksql> CREATE STREAM userprofile (userid INT, firstname VARCHAR, lastname VARCHAR, countrycode VARCHAR, rating DOUBLE) \
>
> WITH (VALUE_FORMAT = 'JSON', KAFKA_TOPIC = 'USERPROFILE');
```

create stream s1 as select * from s2; создание стрима из стрима

```
create stream up_joined as
select up.firstname
+ ' ' + ucase(up.lastname)
+ ' from ' + ct.countryname
+ ' has a rating of ' + cast(rating as varchar) + ' stars.' as description
, up.countrycode
from USERPROFILE up
```

show streams; показать все стримы

```
ksql> show streams;
Stream Name          | Kafka Topic           | Format
-----+-----+-----+-----+-----+-----+-----+
KSQl_PROCESSING_LOG | default_ksql_processing_log | JSON
-----+-----+-----+-----+-----+-----+-----+
ksql> show topics;
Kafka Topic          | Registered | Partitions | Partition Replicas | Consumers | ConsumerGroups
-----+-----+-----+-----+-----+-----+-----+
_confluent-metrics   | false      | 12        | 1                | 0        | 0
_confluent-monitoring | false      | 1          | 1                | 0        | 0
_schemas              | false      | 1          | 1                | 0        | 0
default_ksql_processing_log | true      | 1          | 1                | 0        | 0
-----+-----+-----+-----+-----+-----+-----+
```

describe s1; описать структуру стрима

```
ksql> describe userprofile;
Name          : USERPROFILE
Field        | Type
-----|-----
ROWTIME     | BIGINT      (system)
ROWKEY      | VARCHAR(STRING) (system)
USERID      | INTEGER
FIRSTNAME   | VARCHAR(STRING)
LASTNAME    | VARCHAR(STRING)
COUNTRYCODE | VARCHAR(STRING)
RATING      | DOUBLE
```

describe extended ; описать полностью стрим

```
ksql> describe extended user_profile;
Queries that write into this STREAM
-----
CSAS_USER_PROFILE_PRETTY_B : create stream user_profile_pretty as
select firstname , lastname , rating , countrycode
, from " || countrycode
, " has a rating of ' || cast(rating as varchar) || ' stars.
case when rating <=3.5 then 'Poor'
when rating between 3.5 and 4.2 then 'Good'
else 'Excellent'
end as description
from userprofile;
For query topology and execution plan please run: EXPLAIN <QueryId>
local runtime statistics
-----
messages-per-second: 2.05 total-messages: 204 last-message:
```

drop stream s1; удалить стрим

```
ksql> drop stream if exists users_stream delete topic;
Message
-----
Source USERS_STREAM was dropped. Topic 'USERS' was marked for deletion. Actual deletion and removal from brokers may take some time to complete.
```


Introducing Tables

- A table in Kafka is the state “now”
- A messages for a table
 - *updates* the previous message in the set with the same key
 - *adds a new message* when there is no message with the same key
- Examples:
 - Stock level
 - Web traffic seen in a time period
 - Latest weather for a city

create table t1 with (KAFKA_TOPIC=t1) создание таблицы из топика

```
kafka-topics --zookeeper localhost:2181 --create --partitions 1 --replication-factor 1 --topic COUNTRY-CSV
```

```
>AU:AU,Australia  
>IN:IN,India  
>GB:GB,UK  
>US:US,United States
```

```
ksql> CREATE TABLE COUNTRYTABLE (countrycode VARCHAR, countryname VARCHAR) WITH (KAFKA_TOPIC='COUNTRY-CSV', VALUE_FORMAT='DELIMITED', KEY = 'countrycode');
```

```
ksql> select countrycode, countryname from countrytable;  
AU | Australia  
IN | India  
GB | UK  
US | United States
```

PRIMARY KEY - у таблицы обязательно должен быть первичный ключ для дедупликации

```
ksql> create table weathernow with (kafka_topic='WEATHERREKEYED', value_format='AVRO', key='CITY_NAME');
```

show tables; получить список таблиц

```
ksql> show tables;  
Table Name | Kafka Topic | Format      | Windowed  
-----  
COUNTRYTABLE | COUNTRY-CSV | DELIMITED | false
```

describe t1; получить список колонок таблицы

```
ksql> describe COUNTRYTABLE;  
Name          : COUNTRYTABLE  
Field        | Type  
-----  
ROWTIME      | BIGINT      (system)  
ROWKEY       | VARCHAR(STRING) (system)  
COUNTRYCODE  | VARCHAR(STRING)  
COUNTRYNAME  | VARCHAR(STRING)
```

describe extended ; описать полностью

```
ksql> describe extended COUNTRYTABLE;  
Name          : COUNTRYTABLE  
Type         : TABLE  
Key field    : COUNTRYCODE  
Key format   : STRING  
Timestamp field: not set - using <ROWTIME>  
Value format : DELIMITED  
Kafka topic  : COUNTRY-CSV (partitions: 1, replication: 1)  
  
Field        | Type  
-----  
ROWTIME      | BIGINT      (system)  
ROWKEY       | VARCHAR(STRING) (system)  
COUNTRYCODE  | VARCHAR(STRING)  
COUNTRYNAME  | VARCHAR(STRING)
```

* topic

10 декабря 2020 г. 22:06

print t1 interval 5 ; вывести содержимое топика

<https://docs.ksqldb.io/en/latest/developer-guide/ksqldb-reference/print/>

```
ksql> print 'USERPROFILE' interval 5;
```

Format:JSON

```
{"ROWTIME":1555042029061,"ROWKEY":"1015","userid":"1015","firstname":"Frank","lastname":"Fawcett","countrycode":"US","rating":4.9}
```

```
ksql> print 'db-carusers' from beginning;
```

* join

11 декабря 2020 г. 21:46

[onenote:///C:/Users/trans/Qsync/vova_from_onenote/tf_algonote_v1/SYSTEMDESIGN/KAFKA_KAFKA%20STREAMS.one#join%20operations§ion-id={422214CB-B11E-4F4F-93A0-EF7DDD1C10A7}&page-id={86F85F8C-27B6-49B4-9881-076F84CCA244}&end](oneneote:///C:/Users/trans/Qsync/vova_from_onenote/tf_algonote_v1/SYSTEMDESIGN/KAFKA_KAFKA%20STREAMS.one#join%20operations§ion-id={422214CB-B11E-4F4F-93A0-EF7DDD1C10A7}&page-id={86F85F8C-27B6-49B4-9881-076F84CCA244}&end)

... repartition a stream

11 декабря 2020 г. 21:32

сделаем репартишн стрима для того чтобы его можно было сдюйнить его с другим (у которого другое количество партиций)

```
kafka-topics --zookeeper localhost:2181 --create --partitions 2 --replication-factor 1 --topic DRIVER_PROFILE
```

```
ksql> CREATE STREAM DRIVER_PROFILE (driver_name VARCHAR, countrycode VARCHAR, rating DOUBLE)
> WITH (VALUE_FORMAT = 'JSON', KAFKA_TOPIC = 'DRIVER_PROFILE');
```

topic A с 2мя партициями
stream A

```
kafka-topics --zookeeper localhost:2181 --create --partitions 1 --replication-factor 1 --topic COUNTRY-CSV
```

```
CREATE TABLE COUNTRYTABLE (countrycode VARCHAR PRIMARY KEY, countryname VARCHAR) WITH (KAFKA_TOPIC='COUNTRY-CSV', VALUE_FORMAT='DELIMITED');
```

topic B с 1-ой партицией
stream B

Мы не можем сделать join т.к. число партиций разное

```
ksql> select dp.driver_name, ct.countryname, dp.rating
> from DRIVER_PROFILE dp
> left join COUNTRYTABLE ct on ct.countrycode=dp.countrycode;
```

[partitions=1] partition by как сделать репартишн (на самом деле просто создается еще один стрим с другим количеством партиций)

```
create stream driverprofile_rekeyed with (partitions=1) as select * from DRIVER_PROFILE partition by driver_name;
```

пример StreamPartitioner

The keys are originally null, so distribution is done round-robin, resulting in records with the same ID across different partitions.

Original topic

Repartition topic

Partition 0

Partition 0

(null, {"id": "5", "info": "123"})

("4", {"id": "4", "info": "def"})

(null, {"id": "4", "info": "abc"})

("4", {"id": "4", "info": "abc"})

For repartitioning, set the ID field as the key, and then write the records to a topic.

Partition 1

Partition 1

null, {"id": "5", "info": "456"})

("5", {"id": "5", "info": "456"})

(null, {"id": "4", "info": "def"})

("5", {"id": "5", "info": "123"})

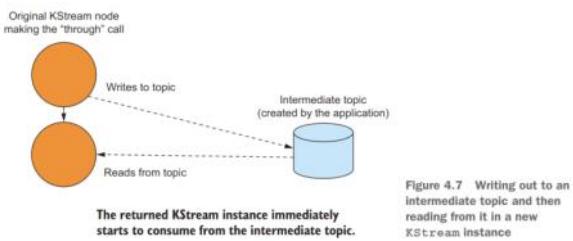
Now with the key populated, all records with the identical ID land the same partition.

Figure 4.6 Repartitioning: changing the original key to move records to a different partition

Повторное секционирование в Kafka Streams легко выполняется с помощью метода KStream.through()

Метод KStream.through() создает промежуточный топик, а текущий экземпляр KStream начинает заносить в него записи

- репартиционирование сопряжено с определенными издержками: дублированием данных и дополнительными вычислительными затратами на обработку. Я бы советовал вам использовать по возможности операции mapValues(), transformValues() или flatMapValues(), поскольку map(), transform() и flatMap() могут приводить к автоматическому повторному секционированию. Лучше применять повторное секционирование умеренно.



Если же вы захотите применить свой подход к секционированию, то можете воспользоваться StreamPartitioner

Listing 4.4 Using the KStream.through method

```
RewardsStreamPartitioner streamPartitioner =  
↳ new RewardsStreamPartitioner();           ↳ Instantiates the concrete  
                                              StreamPartitioner instance  
  
KStream<String, Purchase> transByCustomerStream =  
↳ purchaseKStream.through("customer_transactions",  
                           Produced.with(stringSerde,  
                                         purchaseSerde,  
                                         streamPartitioner));    ↳ Creates a new  
                                              KStream with the  
                                              KStream.through  
                                              method
```

Listing 4.5 RewardsStreamPartitioner

```
public class RewardsStreamPartitioner implements  
↳ StreamPartitioner<String, Purchase> {  
  
    @Override  
    public Integer partition(String key,  
                            Purchase value,  
                            int numPartitions) {  
        return value.getCustomerId().hashCode() % numPartitions;  
    }  
}
```

StreamPartitioner когда для репартиционирования требуется применить какое-либо сочетание ключа и значения

- В этом примере повторное секционирование производится только по ключам. Но бывают случаи, когда использовать ключи нежелательно или требуется применить какое-либо сочетание ключа и значения. В подобных случаях можно воспользоваться интерфейсом StreamPartitioner<K, V>, как вы видели в листинге 4.5 в пункте «Использование StreamPartitioner».

REPARTITIONING HAS A COST

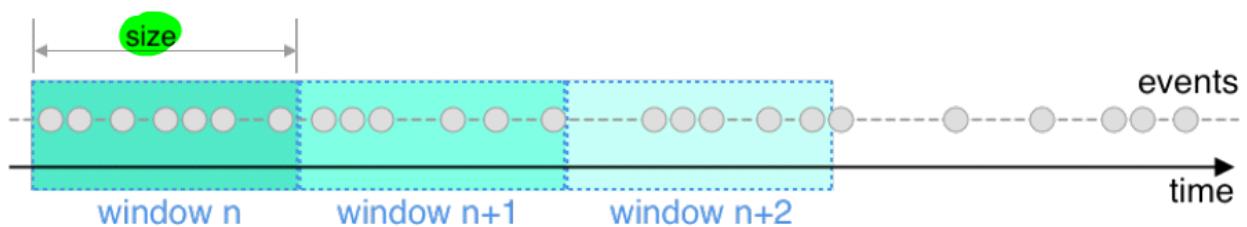
- Повторное секционирование требует затрат — дополнительных затрат ресурсов на создание промежуточных топиков, сохранение дублирующихся данных в еще одном топике; оно также означает повышение задержки вследствие записи и чтения из этого топика. Кроме того, при необходимости выполнить соединение более чем по одному аспекту или измерению нужно организовать соединения цепочкой, отобразить записи с новыми ключами и снова провести процесс повторного секционирования.

В Kafka Streams существует три типа окон

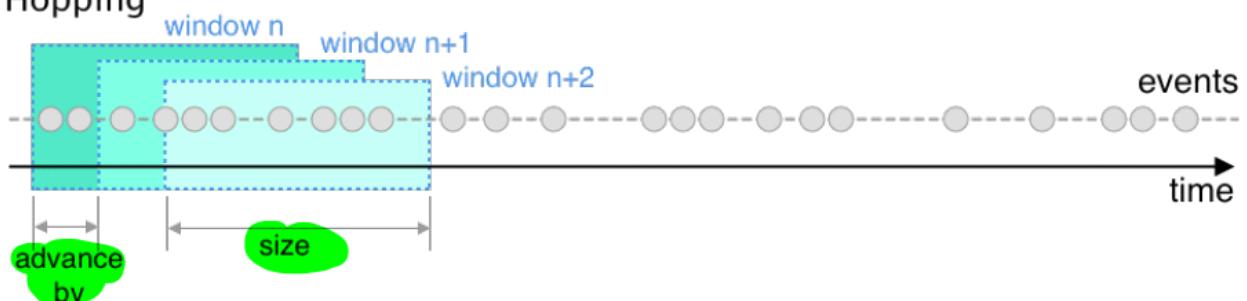
все типы окон основываются на метках даты/времени записей, а не на системном времени

Windows in KSQL

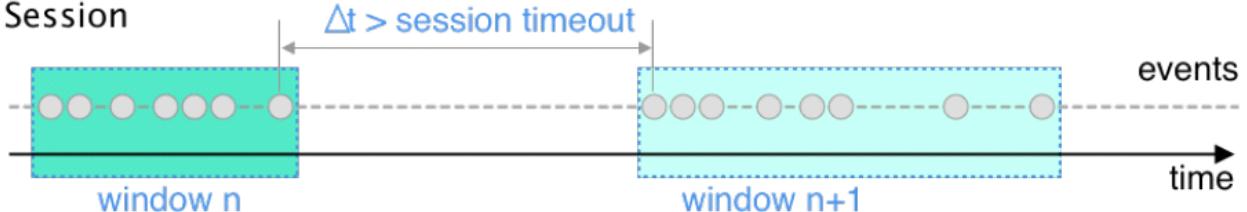
Tumbling



Hopping



Session



- Use the **WINDOW** clause to group input records for aggregations or joins into
 - **TUMBLING**: fixed-sized, non-overlapping windows based on the records' timestamps
 - **HOPPING**: fixed-sized, (possibly) overlapping windows based on the records' timestamps
 - **SESSION**: sessions with a period of activity and gaps in between
- Specify window size, i.e., period of time

окна добавляют еще один уровень группировки в запросы с группировкой

Persistent Query With Windowing Example

- Create a persistent query that counts data in a tumbling window.

```
CREATE TABLE pageviews_regions AS
    SELECT gender, regionid , COUNT(*) AS num_users
    FROM pageviews
    WINDOW TUMBLING (size 30 second)
    GROUP BY gender, regionid
    HAVING COUNT(*) > 1;
```

Windows in KSQL

Streaming has time based records

- In KSQL, time boundaries are named windows
- Windows can apply to streams and tables
- There are 3 ways to define time windows in KSQL
 - Tumbling - Fixed-duration time window, with no overlaps
 - Hopping - Fixed-duration, overlapping time window
 - Session - Not fixed, rather based on durations of activity data separated by gaps of inactivity.

Within our windows

- We can aggregate – e.g., count(*)
- We can group
- Worth knowing
 - COLLECT_LIST
 - TOPK
 - WindowStart() and WindowEnd()

окна определяются на основе timestamp сообщений

window пример

```
ksql> select data_source, city_name, count(*)  
>from rr_world  
>window tumbling (size 60 seconds)  
>group by data_source, city_name;  
Americas | San Jose | 1  
Europe | London | 3  
Americas | Fresno | 1  
Europe | London | 4  
Americas | San Francisco | 2  
Europe | Birmingham | 1  
Europe | Manchester | 1  
Americas | San Francisco | 1  
Americas | San Francisco | 1  
Europe | London | 1
```

T.R. ненадало 6 пагине окна
но 30 секунд настільки дуже коротко

Tumbling windows

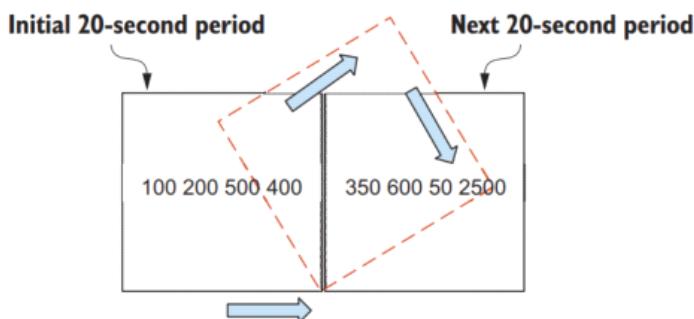
22 февраля 2021 г. 21:56

Tumbling windows

- «Кувыркающиеся» (tumbling) окна захватывают события, попадающие в определенный промежуток времени. Представьте себе, что вам нужно захватывать все биржевые транзакции какой-то компании каждые 20 секунд, так что вы собираете все события за этот промежуток времени. По окончании 20-секундного интервала окно «кувыркается» и переходит на новый 20-секундный интервал наблюдения. Рисунок 5.14 иллюстрирует эту ситуацию.
- «Падающее» окно. Имеет постоянную длину, и каждое событие принадлежит только одному окну. Например, для одноминутных окон все события с отметками времени между 10:03:00 и 10:03:59 попадают в одно окно, события между 10:04:00 и 10:04:59 — в следующее и т. д. Чтобы реализовать одноминутное «падающее» окно, можно взять временную метку каждого события и, округляя ее до ближайшей минуты, определить, какому окну принадлежит данное событие.

tumbling означает что квадрат кувыркается

The current time period "tumbles" (represented by the dashed box)
into the next time period completely with no overlap.



The box on the left is the first 20-second window. After 20 seconds, it “tumbles” over or updates to capture events in a new 20-second period.

There is no overlapping of events. The first event window contains [100, 200, 500, 400] and the second event window contains [350, 600, 50, 2500].

Figure 5.14 Tumbling windows reset after a fixed period.

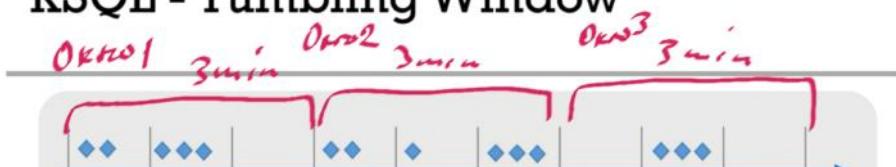
Listing 5.6 Using tumbling windows to count user transactions

```
KTable<Windowed<TransactionSummary>, Long> customerTransactionCounts =  
    builder.stream(STOCK_TRANSACTIONS_TOPIC, Consumed.with(stringSerde,  
                                              transactionSerde)  
    .withOffsetResetPolicy(LATEST)  
    .groupByKey(transaction) -> TransactionSummary.from(transaction),  
    Serialized.with(transactionKeySerde, transactionSerde)  
    .windowedBy(TimeWindows.of(twentySeconds)).count();  
    Specifies a tumbling window of 20 seconds
```

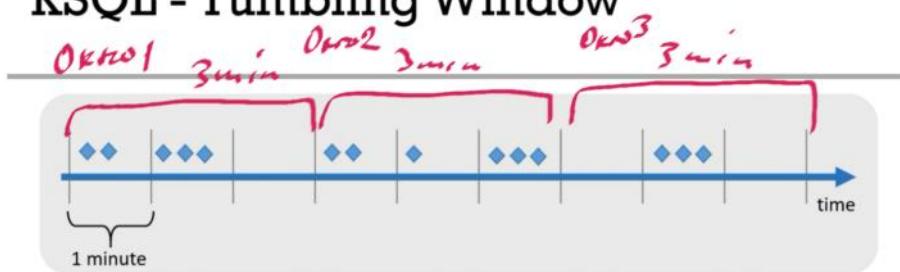
если от источника данных события приходят неупорядоченными во времени то я могу их упорядочить в соответствии с реальным временем их возникновения (если он правильно указан в timestamp)

(grace period - я могу сам явно задать, 1 сутки по дефолту)

KSQL - Tumbling Window



KSQL - Tumbling Window



Tumbling Window

- Fixed-duration time window (e.g., 3 minutes)
- No overlaps

hopping window

22 февраля 2021 г. 21:59

Sliding/hopping window - одно событие может попасть в несколько окон

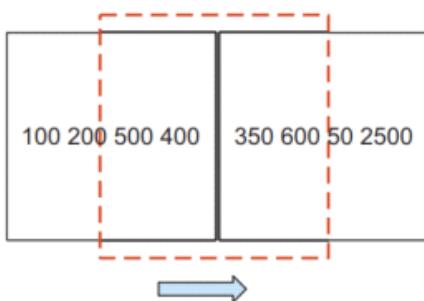
KSQL – Hopping Window



Hopping Window

- Fixed-duration time window (e.g., 3 minutes)
- A “hop” interval (e.g., 1 minute)
- Overlaps

-Скользящие/«прыгающие» (sliding/hopping) окна похожи на «кувыркающиеся», но с небольшим отличием. Скользящие окна не ждут окончания интервала времени перед созданием нового окна для обработки недавних событий. Они запускают новые вычисления после интервала ожидания, меньшего чем длительность окна.



The box on the left is the first 20-second window, but the window “slides” over or updates after 5 seconds to start a new window. Now you see an overlapping of events. Window 1 contains [100, 200, 500, 400], window 2 contains [500, 400, 350, 600], and window 3 is [350, 600, 50, 2500].

Figure 5.15 Sliding windows update frequently and may contain overlapping data.

Listing 5.7 Specifying hopping windows to count transactions

```
KTable<Windowed<TransactionSummary>, Long> customerTransactionCounts =  
    builder.stream(STOCK_TRANSACTIONS_TOPIC, Consumed.with(stringSerde,  
    transactionSerde)  
    .withOffsetResetPolicy(LATEST))  
    .groupBy((noKey, transaction) -> TransactionSummary.from(transaction),  
    Serialized.with(transactionKeySerde, transactionSerde))  
    .windowedBy(TimeWindows.of(twentySeconds)  
    .advanceBy(fiveSeconds).until(fifteenMinutes)).count();
```

Uses a hopping window of 20 seconds, advancing every 5 seconds

«Кувыркающееся» окно можно преобразовать в «прыгающее» с помощью добавления вызова метода `advanceBy()`.

окна могут перекрываться для обеспечения некоторого сглаживания

- «Прыгающее» окно. Тоже имеет фиксированную длину.
- Однако такие окна могут перекрываться для обеспечения некоторого сглаживания.
- Например, 5-минутное окно с размером «прыжка» в 1 минуту будет содержать события между 10:03:00 и 10:07:59, в следующем окне будут отображаться события между 10:04:00 и 10:08:59 и т. д.
- Чтобы реализовать «прыгающее» окно, нужно сначала вычислить 1-минутные «падающие» окна, а затем объединить несколько соседних окон.

sliding window

22 февраля 2021 г. 22:00

- Скользящее окно. Содержит все события, происходящие с некоторыми интервалами между собой.
- Например, 5-минутное скользящее окно будет охватывать события с 10:03:39 по 10:08:12, поскольку между ними прошло менее 5 минут (обратите внимание, что в случае «падающих» и «прыгающих» 5-минутных окон с их фиксированными границами эти два события не попали бы в одно окно).
- Скользящее окно может быть реализовано путем хранения буфера событий, отсортированных по времени, и удаления из окна старых событий, когда их время истекает.

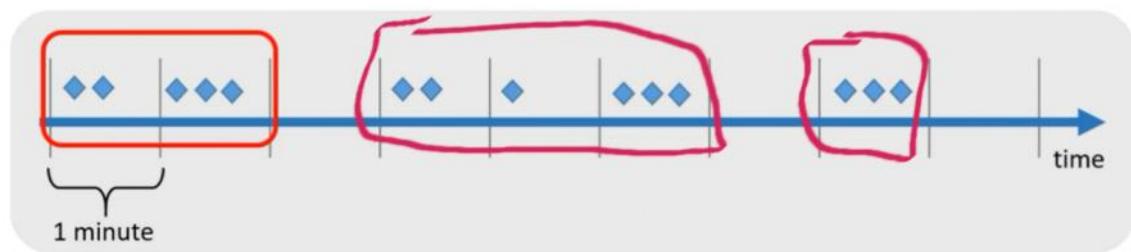
_____session window

22 февраля 2021 г. 21:56

Что такое session window вообще нет параметров

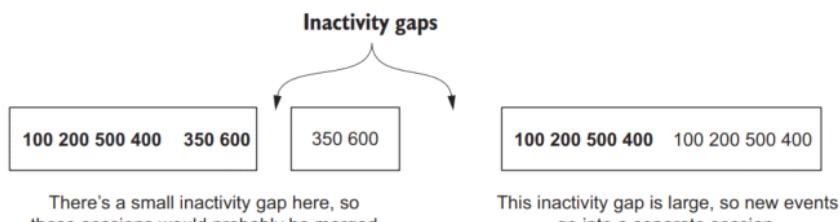
- используется для web click, web blog analysis, активность покупателя на сайте
- это как бы clustering events
- граничения сеансовых связанны с действиями пользователей — длительность сеанса (-ов) определяется исключительно тем, насколько активно ведет себя пользователь
- Сеансовые окна сильно отличаются от всех остальных типов окон. Они ограничиваются не столько по времени, сколько активностью пользователя (или активностью той сущности, которую вы хотели бы отслеживать). Сеансовые окна разграничиваются периодами бездействия
- Сеансовые окна основываются на действиях пользователей, но применяют метки даты/времени из записей для определения того, к какому сеансу относится запись.

KSQL – Session Window



Session Window

- No fixed duration
- Windows based on durations of activity
- Windows separated by gaps of inactivity



Session windows are different because they aren't strictly bound by time but represent periods of activity. Specified inactivity gaps demarcate the sessions.

Figure 5.12 Session windows separated by a small inactivity gap are combined to form a new, larger session.

- С помощью вызова `windowedBy(SessionWindows.with(twentySeconds).until(fifteenMinutes))` мы создаем сеансовое окно с интервалом бездействия 20 секунд и интервалом сохранения 15 минут. Интервал бездействия 20 секунд означает, что приложение будет включать любую запись, которая поступит

в пределах 20 секунд от окончания или начала текущего сеанса в текущий (активный) сеанс.

- Если входящая запись выходит за пределы интервала бездействия (с любой из сторон от метки даты/времени), то приложение создает новый сеанс. Интервал сохранения означает поддержание сеанса в течение определенного времени и допускает запоздавшие данные, которые выходят за период бездействия сеанса, но все еще могут быть присоединены. Кроме того, начало и конец нового сеанса, получившегося в результате объединения, соответствуют самой ранней и самой поздней метке даты/времени.



Figure 5.13 Creating session windows with inactivity periods and retention

Table 5.1 Sessioning table with a 20-second inactivity gap

Arrival order	Key	Timestamp
1	{123-345-654,FFBE}	00:00:00
2	{123-345-654,FFBE}	00:00:15
3	{123-345-654,FFBE}	00:00:50
4	{123-345-654,FFBE}	00:00:05

Окно сессии. В отличие от других типов у окна сессии нет фиксированной продолжительности

- Вместо этого оно определяется как множество всех событий для одного и того же пользователя, которые происходят близко друг к другу во времени.
- Окно заканчивается, когда пользователь какое-то время неактивен (например, если событий не поступало в течение 30 минут).
- Разделение на сессии является обычным требованием при анализе сайта (см. пункт «GROUP BY» подраздела «Объединение и группировка на этапе сжатия» раздела 10.2)

Merging streams: INSERT INTO

- Combine streams together
 - INSERT INTO *some-stream* SELECT from *another-stream*
- Must be identical schema
- You can combine 2 .. or more!
- Tip: you can a pseudo-column to help with “origin”

```
ksql> create stream rr_americas_raw with (kafka_topic='riderequest-america', value_format='avro');
```

```
ksql> create stream rr_europe_raw with (kafka_topic='riderequest-europe', value_format='avro');
```

```
create stream rr_world as select 'Europe' as data_source, * from rr_europe_raw;
```

```
insert into rr_world      select 'Americas' as data_source, * from rr_americas_raw;
```

1555394779102 ride_943 Europe 1555394779101 51.04756939680488 1.374105749437196 ride_943 Ivan London
1555394779501 ride_389 Americas 1555394779501 41.684222683253786 -102.1560170434821 ride_389 Walter Los Angeles
1555394780855 ride_784 Europe 1555394780855 51.196283097585535 1.2963681273548069 ride_784 Eve London
1555394782833 ride_517 Americas 1555394782833 41.3110361451861 -101.12891027872641 ride_517 Ted San Francisco
1555394784737 ride_764 Europe 1555394784737 53.21082671821005 0.04754362599113371 ride_764 Eve London
1555394786872 ride_336 Americas 1555394786871 42.465219296113496 -118.2035508189732 ride_336 Wendy San Francisco



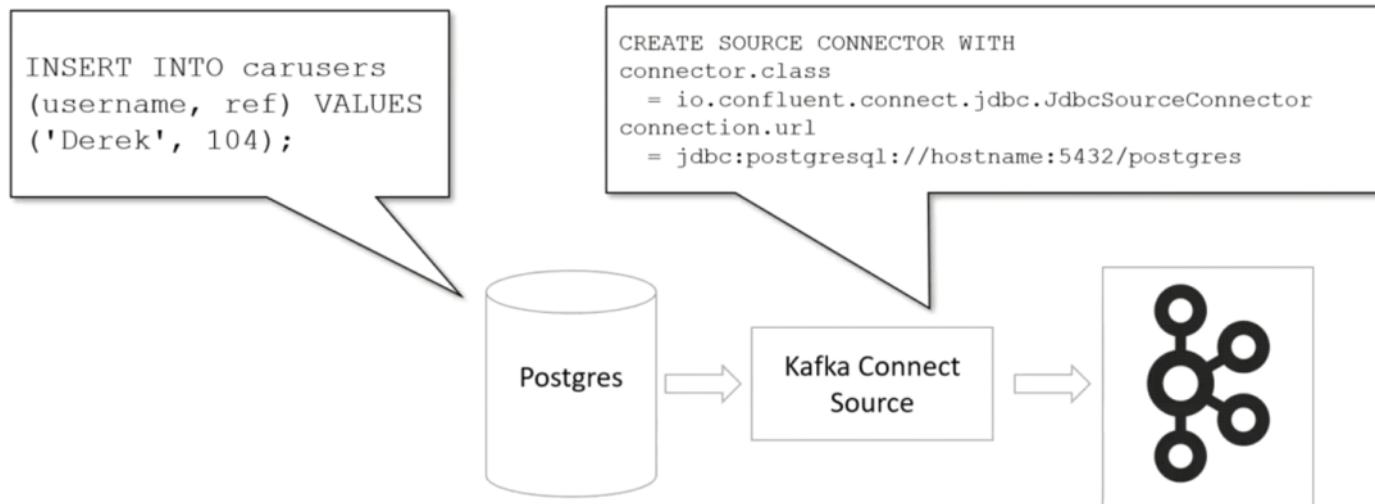
Datagen – Generating a stream

- KSQL Datagen
 - ksql-datagen command-line tool
 - Generate random test data
 - Use provided custom schema
 - We'll be generating USERPROFILE
- Options to generate a stream of
 - Formats : CSV, JSON, AVRO
 - Rate
 - Maximum number to produce

Userid	First Name	Last Name	Country	Rating
1000	Alice	Smith	US	3.4
1001	Bob	Jones	GB	2.2
1002	Carol	Jones	AU	4.9

```
ksql-datagen schema=./riderequest-europe.avro format=avro topic=riderequest-europe key=rideid maxInterval=5000 iterations=100
```

Kafka Connect with ksqlDB



```
ksql> CREATE SOURCE CONNECTOR `postgres-jdbc-source` WITH(
>   "connector.class"='io.confluent.connect.jdbc.JdbcSourceConnector',
>   "connection.url"='jdbc:postgresql://postgres:5432/postgres',
>   "mode"='incrementing',
>   "incrementing.column.name"='ref',
>   "table.whitelist"='carusers',
>   "connection.password"='password',
>   "connection.user"='postgres',
>   "topic.prefix"='db-',
>   "key"='username');
```

```
INSERT INTO carusers (username) VALUES ('Charlie');
saubury:ksql-course$ docker-compose exec postgres psql -U postgres -f /postgres-setup.sql
CREATE TABLE
INSERT 0 1
INSERT 0 1
INSERT 0 1
saubury:ksql-course$ docker-compose exec postgres psql -U postgres -c "select * from carusers;"
```

username	ref
Alice	1
Bob	2
Charlie	3

(3 rows)

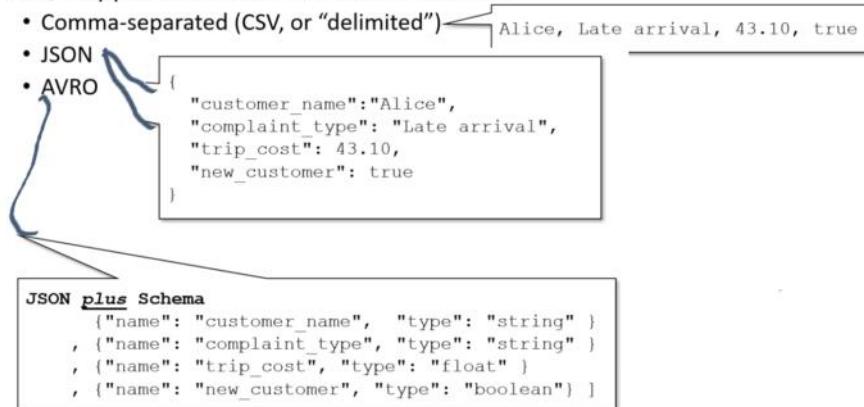
при создании коннектора автоматически создаётся
одноименный(с приставкой db-) топик

```
ksql> print 'db-carusers' from beginning;
Format:AVRO
11/30/19 10:14:48 AM UTC, Alice, {"username": "Alice", "ref": 1}
11/30/19 10:14:48 AM UTC, Bob, {"username": "Bob", "ref": 2}
11/30/19 10:14:48 AM UTC, Charlie, {"username": "Charlie", "ref": 3}
11/30/19 10:16:17 AM UTC, Derek, {"username": "Derek", "ref": 4}
```


Data Formats



- Data comes in different formats.
- KSQL supports three serialization mechanisms



json пример

```
saubury:ksql-udemy$ kafka-console-producer --broker-list localhost:9092 --topic COMPLAINTS_JSON
>{"customer_name": "Alice, Bob and Carole", "complaint_type": "Bad driver", "trip_cost": 22.40, "new_customer": true}
>

ksql> CREATE STREAM complaints_json (customer_name VARCHAR, complaint_type VARCHAR, trip_cost DOUBLE, new_customer BOOLEAN) \
> WITH (VALUE_FORMAT = 'JSON', KAFKA_TOPIC = 'COMPLAINTS_JSON');

Message
-----
Stream created
-----
ksql> select * from complaints_json;
1555159165377 | null | Alice, Bob and Carole | Bad driver | 22.4 | true
```

если передать неправильный формат (например одного из полей), то в логах ksql можно увидеть ошибку

вложенный многоуровневый json

Nested JSON



- KSQL supports both flat and *nested* data structures.



Nested JSON



- KSQL supports both flat and *nested* data structures.
- KSQL supports the STRUCT data type to access nested data structures.

KSQL Nested JSON Syntax

```
city STRUCT <
    name VARCHAR,
    country VARCHAR,
    latitude DOUBLE,
    longitude DOUBLE
>
```

```
{
  "city": {
    "name": "Sydney",
    "country": "AU",
    "latitude": -33.8688,
    "longitude": 151.2093
  },
  "description": "light rain",
  "clouds": 92,
  "deg": 26,
  "humidity": 94,
  "pressure": 1025.12,
  "rain": 1.25
}
```

```
(1) demo-weather.json •
1  {
2    "city": {
3      "name": "Sydney",
4      "country": "AU",
5      "latitude": -33.8688,
6      "longitude": 151.2093
7    },
8    "description": "light rain",
9    "clouds": 92,
10   "deg": 26,
11   "humidity": 94,
12   "pressure": 1025.12,
13   "rain": 1.25
14 }
15 {
16   "city": {
17     "name": "Seattle",
18     "country": "US",
19     "latitude": 47.6062,
20     "longitude": -122.3321
21   },
22   "description": "heavy rain",
23   "clouds": 92,
24   "deg": 19,
25   "humidity": 94,
26   "pressure": 1025.12,
27   "rain": 7
28 }
29 {
30   "city": {
31     "name": "San Francisco",
32     "country": "US",
33     "latitude": 37.7749,
34     "longitude": -122.4194
35   },
36   "description": "drizzle"
37 }
```

```

ksql> CREATE STREAM weather
>     (city STRUCT <name VARCHAR, country VARCHAR, latitude DOUBLE, longitude DOUBLE>,
>      description VARCHAR,
>      clouds BIGINT,
>      deg BIGINT,
>      humidity BIGINT,
>      pressure DOUBLE,
>      rain DOUBLE)
> WITH (KAFKA_TOPIC='WEATHERNESTED', VALUE_FORMAT='JSON');

ksql> select * from weather;
1555376517662 | null | {NAME=Sydney, COUNTRY=AU, LATITUDE=-33.8688, LONGITUDE=151.2093} | light rain | 92 | 26 | 94 | 1025.12 | 1.25
1555376517672 | null | {NAME=Seattle, COUNTRY=US, LATITUDE=47.6062, LONGITUDE=-122.3321} | heavy rain | 92 | 19 | 94 | 1025.12 | 7.0
1555376517672 | null | {NAME=San Francisco, COUNTRY=US, LATITUDE=37.7749, LONGITUDE=-122.4194} | fog | 92 | 19 | 94 | 1025.12 | 10.0
1555376517672 | null | {NAME=San Jose, COUNTRY=US, LATITUDE=37.3382, LONGITUDE=-121.8863} | light rain | 92 | 23 | 94 | 1025.12 | 3.0
1555376517672 | null | {NAME=Fresno, COUNTRY=US, LATITUDE=36.7378, LONGITUDE=-119.7871} | heavy rain | 92 | 22 | 94 | 1025.12 | 6.0
1555376517673 | null | {NAME=Los Angeles, COUNTRY=US, LATITUDE=34.0522, LONGITUDE=-118.2437} | haze | 92 | 19 | 94 | 1025.12 | 2.0
1555376517673 | null | {NAME=San Diego, COUNTRY=US, LATITUDE=32.7157, LONGITUDE=-117.1611} | fog | 92 | 19 | 94 | 1025.12 | 2.0
1555376517673 | null | {NAME=Birmingham, COUNTRY=UK, LATITUDE=52.4862, LONGITUDE=-1.8904} | light rain | 92 | 26 | 94 | 1025.12 | 4.0
1555376517673 | null | {NAME=London, COUNTRY=GB, LATITUDE=51.5074, LONGITUDE=-0.1278} | heavy rain | 92 | 19 | 94 | 1025.12 | 8.0
1555376517673 | null | {NAME=Manchester, COUNTRY=GB, LATITUDE=53.4808, LONGITUDE=-2.2426} | fog | 92 | 26 | 94 | 1025.12 | 3.0
1555376517674 | null | {NAME=Bristol, COUNTRY=GB, LATITUDE=51.4545, LONGITUDE=-2.5879} | light rain | 92 | 19 | 94 | 1025.12 | 3.0
1555376517674 | null | {NAME=Newcastle, COUNTRY=GB, LATITUDE=54.9783, LONGITUDE=-1.6178} | heavy rain | 92 | 19 | 94 | 1025.12 | 12.0
1555376517674 | null | {NAME=Liverpool, COUNTRY=GB, LATITUDE=53.4084, LONGITUDE=-2.9916} | haze | 92 | 23 | 94 | 1025.12 | 3.0

```

извлекем поля из структуры city

```

ksql> SELECT city->name AS city_name, city->country AS city_country, city->latitude as latitude, city->longitude as longitude, description
, rain from weather;
Sydney | AU | -33.8688 | 151.2093 | light rain | 1.25
Seattle | US | 47.6062 | -122.3321 | heavy rain | 7.0
San Francisco | US | 37.7749 | -122.4194 | fog | 10.0
San Jose | US | 37.3382 | -121.8863 | light rain | 3.0
Fresno | US | 36.7378 | -119.7871 | heavy rain | 6.0
Los Angeles | US | 34.0522 | -118.2437 | haze | 2.0

```

avro пример

KSQL сервер будет обращаться к SchemaRegistry для того чтобы получить/передать схему данных

сами данные юзер передает и получает в формате JSON, несмотря на то что внутри они передаются бинарно

```

saubury:ksql-udemy$ kafka-console-producer --broker-list localhost:9092 --topic COMPLAINTS_AVRO \
> --property value.schema='
> {
>   "type": "record",
>   "name": "myrecord",
>   "fields": [
>     {"name": "customer_name", "type": "string" }
>     , {"name": "complaint_type", "type": "string" }
>     , {"name": "trip_cost", "type": "float" }
>     , {"name": "new_customer", "type": "boolean"}
>   ]
> }
> '
{"customer_name":"Carol", "complaint_type":"Late arrival", "trip_cost": 19.6, "new_customer": false}

```

Схема в виде JSON

Координаты
в виде JSON

```

ksql> print 'COMPLAINTS_AVRO' from beginning;
Format:AVRO
4/14/19 9:38:35 PM AEST, null, {"customer_name": "Carol", "complaint_type": "Late arrival", "trip_cost": 19.6, "new_customer": false}
^CTopic printing ceased
ksql> create stream complaints_avro with (kafka_topic='COMPLAINTS_AVRO', value_format='AVRO');

ksql> select * from complaints_avro;
1555241915802 | null | Carol | Late arrival | 19.600000381469727 | false

```

Координаты в виде таблицы

СХЕМУ МОЖНО ПОСМОТРЕТЬ ЧЕРЕЗ CONFLUENT CONTROL CENTER

The screenshot shows the Confluent Control Center interface. In the top navigation bar, the URL is visible: `hostname:9021/cntrls/kafka_ksql/MQfa/ksql0009fbq/management/topics/COMPLAINTS_AVRO/schema/value`. The main area displays the schema for the `COMPLAINTS_AVRO` topic. On the left, there's a sidebar with tabs for `Brokers`, `Topics` (which is selected), `Connect`, `KSQL`, `Consumers`, and `Cluster settings`. The right panel shows the schema details for the `Value` tab of the `COMPLAINTS_AVRO` topic. The schema is presented as a JSON object:

```

1 | {
2 |   "type": "record",
3 |   "name": "myrecord",
4 |   "fields": [
5 |     {"name": "customer_name", "type": "string" }
6 |     , {"name": "complaint_type", "type": "string" }
7 |     , {"name": "trip_cost", "type": "float" }
8 |     , {"name": "new_customer", "type": "boolean"}
9 |   ]
10 |
11 }
12
13
14
15
16
17
18
19
20
21
22

```

SCHEMA EVOLUTION IN CONFLUENT CONTROL CENTER

The screenshot shows the Confluent Control Center interface. On the left, there's a sidebar with navigation links: 'Brokers', 'Topics' (which is selected), 'Connect', 'KSQL', 'Consumers', and 'Cluster settings'. The main area has tabs for 'Overview', 'Messages', 'Schema', and 'Configuration'. Below these tabs, there's a checkbox labeled 'Turn on version diff.' A dropdown menu shows 'version 1' and 'version 2 (current)'. The central part of the screen displays two JSON schema snippets side-by-side, with a yellow circle highlighting the dropdown menu.

```
version 1
1 {
2   "fields": [
3     {
4       "name": "customer_name",
5       "type": "string"
6     },
7     {
8       "name": "complaint_type",
9       "type": "string"
10    },
11    {
12      "name": "trip_cost",
13      "type": "float"
14    },
15    {
16      "name": "new_customer",
17      "type": "boolean"
18    }
19  ],
20  "name": "myrecord",
21  "type": "record"
22 }
```

```
version 2 (current)
1 {
2   "fields": [
3     {
4       "name": "customer_name",
5       "type": "string"
6     },
7     {
8       "name": "complaint_type",
9       "type": "string"
10    },
11    {
12      "name": "trip_cost",
13      "type": "float"
14    },
15    {
16      "name": "new_customer",
17      "type": "boolean"
18    },
19    {
20      "default": 1,
21      "name": "number_of_rides",
22      "type": "int"
23    }
24  ],
25  "name": "myrecord",
26  "type": "record"
27 }
```

.. поменять формат данных стрима

11 декабря 2020 г. 21:19

(value_format='AVRO') создадим новый стрим с форматом данных из старого стрима

```
ksql> list streams;  
  
Stream Name | Kafka Topic | Format  
-----|-----|-----  
WEATHER | WEATHERNESTED | JSON  
KSQl_PROCESSING_LOG | default_ksq1_processing_log | JSON  
COMPLAINTS_AVRO | COMPLAINTS_AVRO | AVRO  
COMPLAINTS_AVRO_V2 | COMPLAINTS_AVRO | AVRO  
WEATHERRAW | WEATHERRAW | AVRO  
  
ksql> create stream weatherraw with (value_format='AVRO') as SELECT city->name AS city_name, city->country AS city_country, city->latitude  
as latitude, city->longitude as longitude, description, rain from weather ;
```

...поменять первичный ключ стрима (rekeyed table)

11 декабря 2020 г. 21:22

partition by как поменять PK

```
ksql> select rowkey, city_name from weatherraw;
null | Sydney
null | San Jose
null | San Francisco
null | Seattle
null | Fresno
null | Birmingham
```

no key!

```
ksql> create stream weatherrekeyed as select * from weatherraw partition by city_name;
```

Field	Type
ROWTIME	BIGINT (system)
ROWKEY	VARCHAR(STRING) (system)
CITY_NAME	VARCHAR(STRING)
CITY_COUNTRY	VARCHAR(STRING)
LATITUDE	DOUBLE
LONGITUDE	DOUBLE
DESCRIPTION	VARCHAR(STRING)
RAIN	DOUBLE

```
ksql> select rowkey, city_name from weatherrekeyed;
San Diego | San Diego
Fresno | Fresno
Seattle | Seattle
Los Angeles | Los Angeles
Manchester | Manchester
San Francisco | San Francisco
Bristol | Bristol
Birmingham | Birmingham
Sydney | Sydney
London | London
```



User Defined Functions (UDF)

KSQL has a programming API for building your own functions

- User Defined Functions (**UDF**) - A scalar function - for an input parameter return one output value
- User Defined Aggregate Functions (**UDAF**) - An aggregate function - for many input rows return one output value
- Implemented as custom jars
- Jars copied to the “ext” directory of the KSQL server

пример

We'll be building a UDF called “TAXI_WAIT” to calculate wait time (in minutes) based on distance to travel & weather conditions

`TAXI_WAIT(weather, dist)`

- Write the code in Java
- Compile
- Add to KSQL Server

```

1 package com.vsimon.kafka.streams;
2
3 import io.confluent.ksql.function.udf.Udf;
4 import io.confluent.ksql.function.udf.UdfDescription;
5 @UdfDescription(name = "taxi_wait", description = "Return expected wait time in minutes")
6
7 public class TaxiWait {
8
9     @Udf(description = "Given weather and distance return expected wait time in minutes")
10    public double taxi_wait(final String weather_description, final double dist) {
11
12        double ret = -1;
13        double weather_factor = 1;
14
15        switch(weather_description) {
16
17            case "light rain":
18                weather_factor = 2;
19                break;
20
21            case "heavy rain":
22                weather_factor = 4;
23                break;
24
25            case "fog":
26                weather_factor = 6;
27                break;
28
29            case "haze":
30                weather_factor = 1.5;
31                break;
32
33            case "SUNNY":
34                weather_factor = 1;
35                break;
36        }
37
38        return dist * weather_factor / 50.0;
39    }
}

```

просто положим jar с функцией в папку на сервере (для этого
надо остановить сервер и поднять заново)

Property	Default	override	Effective Value
ksql.udf.classpath.enabled			true
ksql.extension.dir			ext +
ksql.functions.substring.legacy.args			false

```

sabury:java$ mkdir /opt/confluent/ext
sabury:java$ cp target/ksql-udf-taxi-1.0.jar /opt/confluent/ext

```

проверим что наша функция появилась в списке функций

```

ksql> list functions;

ksql> DESCRIBE FUNCTION TAXI_WAIT;

Name      : TAXI_WAIT
Overview  : Return expected wait time in minutes
Type      : scalar
Jar       : /opt/confluent/ext/ksql-udf-taxi-1.0.jar
Variations :

Variation   : TAXI_WAIT(VARCHAR, DOUBLE)
Returns     : DOUBLE
Description : Given weather and distance return expected wait time in minutes

```

Using our UDF: TAXI_WAIT



- We have built our Java UDF “TAXI_WAIT”
- Let’s use it to calculate wait time
 - Based on distance to travel & weather conditions

```
SELECT TAXI_WAIT (weather, dist)  
FROM some-stream
```

UDAFs user defined aggregate functions

19 декабря 2020 г. 21:00

user defined functions (**UDFs**) and user defined aggregate functions (**UDAFs**) in Java.

Query Explain Plans

Show the query plan of a query

- Find the “id” of a running query
 - CTAS_xxx – create table as select
 - CSAS_xxx – create stream as select
- Display the execution plan for a SQL expression
 - Show the execution plan
 - Runtime information and metrics
 - Visualizing the plan

show queries;

explain <some-id>

```
ksql> show queries;

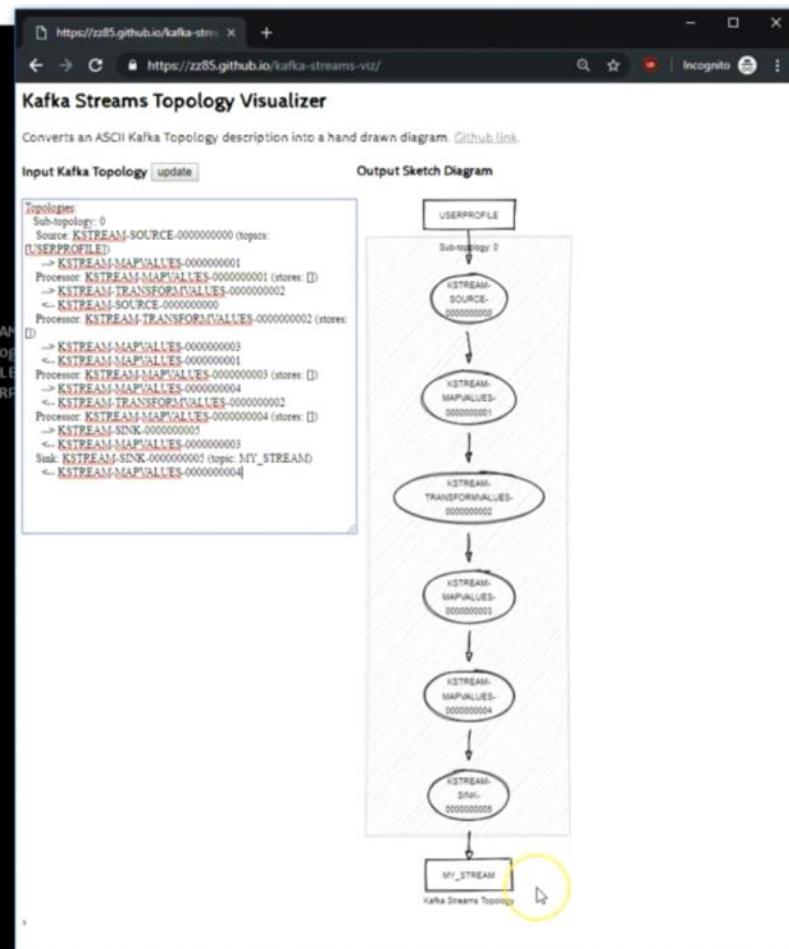
Query ID      | Kafka Topic | Query String
-----+-----+-----+
CSAS_MY_STREAM_2 | MY_STREAM    | create stream my_stream
as select firstname
from userprofile;
+
For detailed information on a Query run: EXPLAIN <Query ID>;
ksql> explain CSAS_MY_STREAM_2
```

```
ec2-user@ip-172-31-6-63:~/gk/ksql-course
```

```
ROWTIME | BIGINT      (system)
ROWKEY  | VARCHAR(STRING) (system)
FIRSTNAME | VARCHAR(STRING)

Sources that this query reads from:
For sink description please run: DESCRIBE [EXTENDED] <SinkId>
Execution plan
> [ SINK ] | Schema: [FIRSTNAME : VARCHAR] | Logger: CSAS_MY_STREAM
    > [ PROJECT ] | Schema: [FIRSTNAME : VARCHAR] | Log
        > [ SOURCE ] | Schema: [USERPROFILE
PROFILE.LASTNAME : VARCHAR, USERPROFILE.COUNTRYCODE : VARCHAR, USERF
Processing topology
Topologies:
Sub-topology: 0
Source: KSTREAM-SOURCE-0000000000 (topics: [USERPROFILE])
--> KSTREAM-MAPVALUES-0000000001
Processor: KSTREAM-MAPVALUES-0000000001 (stores: [])
--> KSTREAM-TRANSFORMVALUES-0000000002
<- KSTREAM-SOURCE-0000000000
Processor: KSTREAM-TRANSFORMVALUES-0000000002 (stores: [])
--> KSTREAM-MAPVALUES-0000000003
<- KSTREAM-MAPVALUES-0000000001
Processor: KSTREAM-MAPVALUES-0000000003 (stores: [])
--> KSTREAM-MAPVALUES-0000000004
<- KSTREAM-TRANSFORMVALUES-0000000002
Processor: KSTREAM-MAPVALUES-0000000004 (stores: [])
--> KSTREAM-SINK-0000000005
<- KSTREAM-MAPVALUES-0000000003
Sink: KSTREAM-SINK-0000000005 (topic: MY_STREAM)
--> KSTREAM-MAPVALUES-0000000004

ksql> []
```



Stream Processing Cookbook

12 декабря 2020 г. 11:41

<https://www.confluent.io/stream-processing-cookbook/>

The screenshot shows the Stream Processing Cookbook page on the Confluent website. The header includes the Confluent logo, navigation links for Product, Cloud, Developers, Blog, and Docs, a prominent orange "Download" button, a search icon, and a language dropdown set to English. Below the header, there are three main cards:

- Data Masking** (STREAMING ETL): Mask streaming data from an external source.
- Data Filtering** (STREAMING ETL): Filter streaming data from an external source.
- Processing Syslog** (ANOMALY DETECTION): Enrich and filter syslog data with machine learning models.

local vs distributed mode

2 февраля 2021 г. 20:52

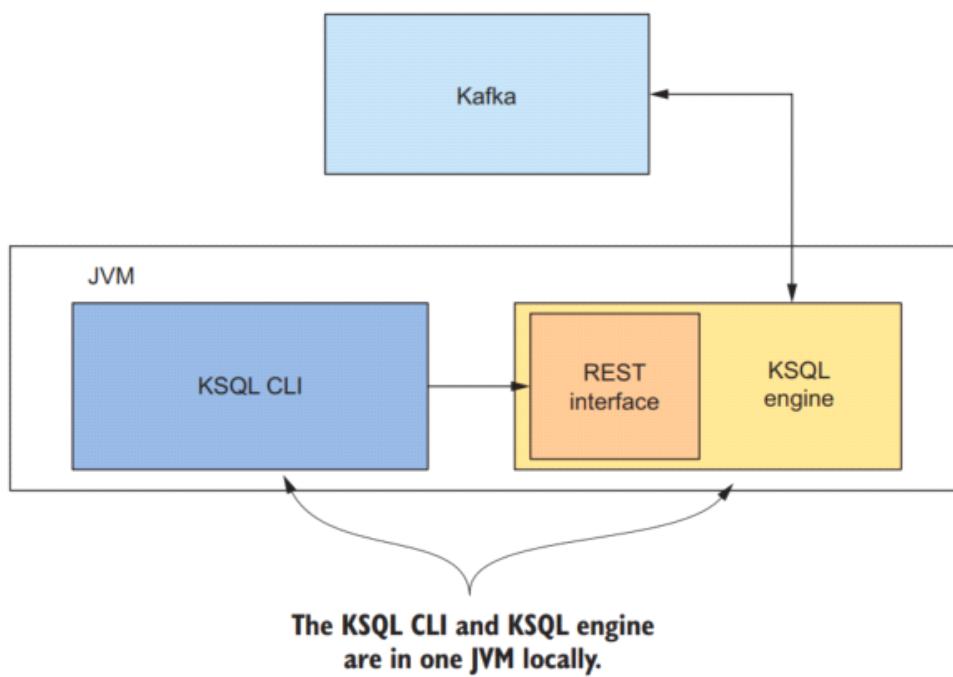


Figure 9.10 KSQL in local mode

Главное то, что, хотя вы подключаетесь явным образом к одному конкретному удаленному серверу KSQL, все относящиеся к одному кластеру Kafka серверы возьмут на себя часть работы по выполнению полученного запроса.

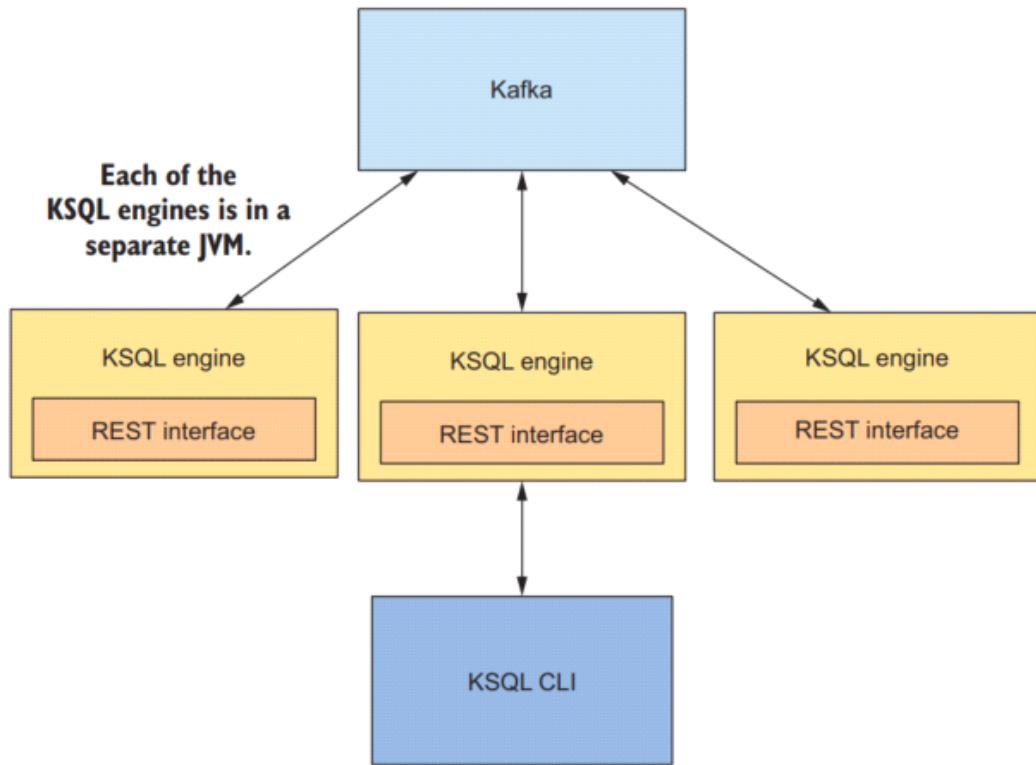


Figure 9.11 KSQL in distributed mode

при необходимости в дополнительных вычислительных мощностях можно просто запустить еще один сервер KSQL, даже и прямо во время работы

аметим, что для выполнения запросов сервера KSQL используют Kafka Streams. Это значит, что при необходимости в дополнительных вычислительных мощностях можно просто запустить еще один сервер KSQL, даже и прямо во время работы (подобно запуску дополнительных экземпляров приложения Kafka Streams). Обратное также справедливо: в случае лишних вычислительных мощностей можно остановить произвольное число серверов KSQL при условии, что хотя бы один останется в рабочем состоянии (иначе ваши запросы перестанут выполняться!).