Big Stream Processing Systems

Data Systems Group
https://bigdata.cs.ut.ee/

Big Data Management

University of Tartu

Fall 2019

Apache Kafka[1]

An overview

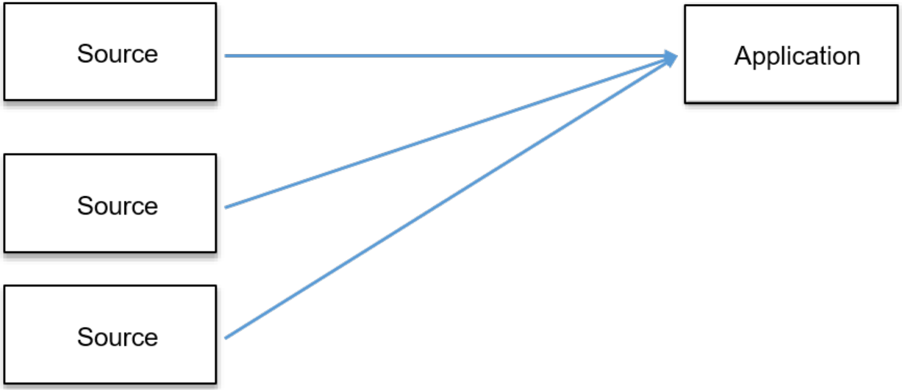
[1] Slides are based on content from Cloudera and Confluent

Motivation

Data pipelines start with a small number of systems to integrates. A single ETL (extract, transform, load) process move data from the source to the interested applications.

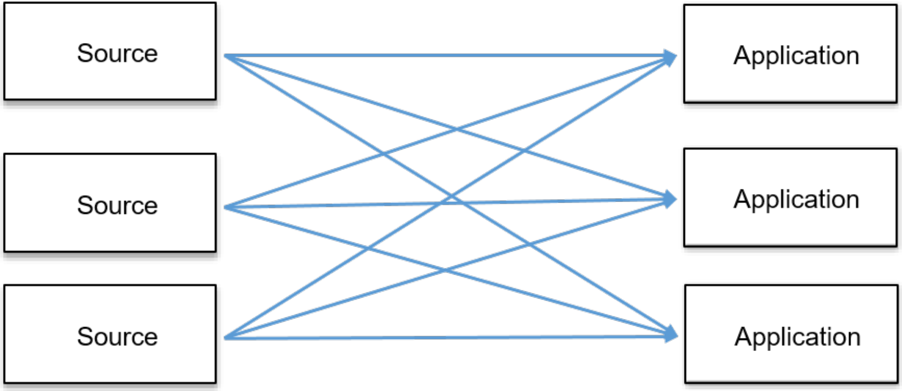
Motivation

But data pipeline grow over time. Adding new system causes the need of new ETL process. The code-base grows together with data formats and services.



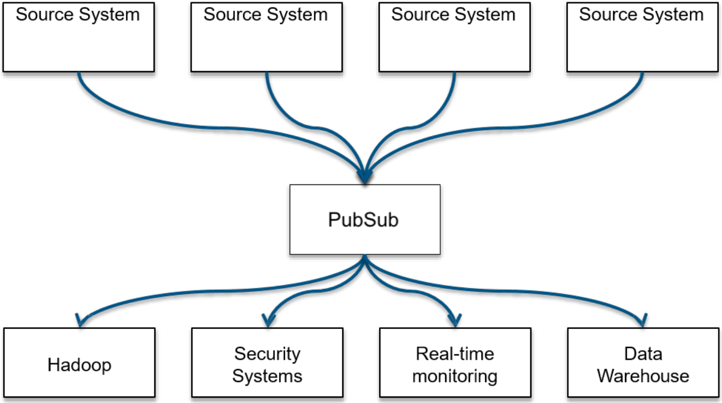
Motivation

Things end up messy when sources and sinks are coupled!



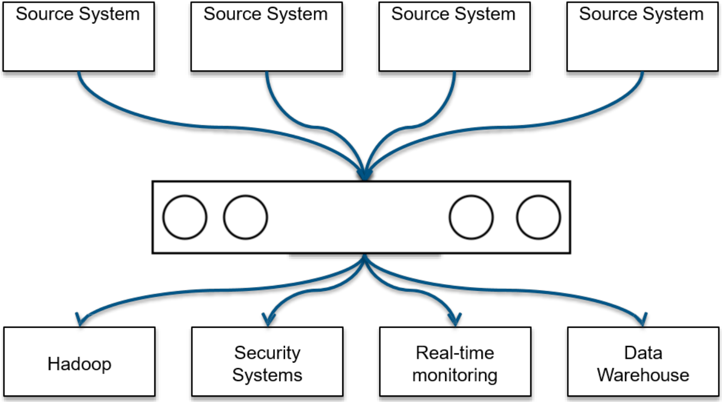
An alternative: Publish/Subscribe

PubSubs decouple data sources and their consumers making communication asynchronous and processing scalable.



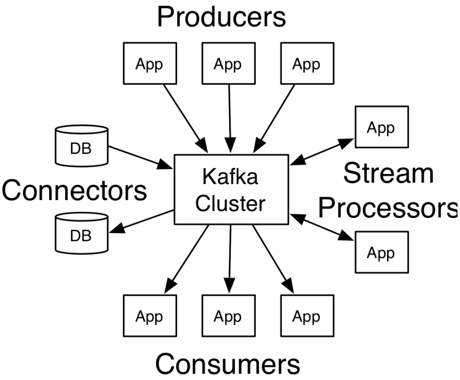
An alternative: Publish/Subscribe

PubSubs organize messages logically so that it is easier for the interested consumers to access.



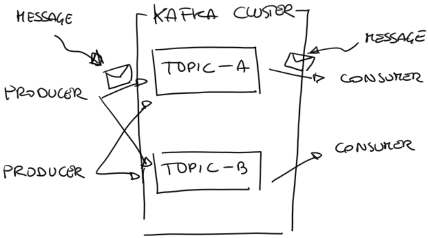
Apache Kafka

Apache Kafka is an horizontally scalable, fault-tolerant, publish-subscribe system. It can process over 1 trillion messages without neglecting durability, i.e., it persists data on disk.

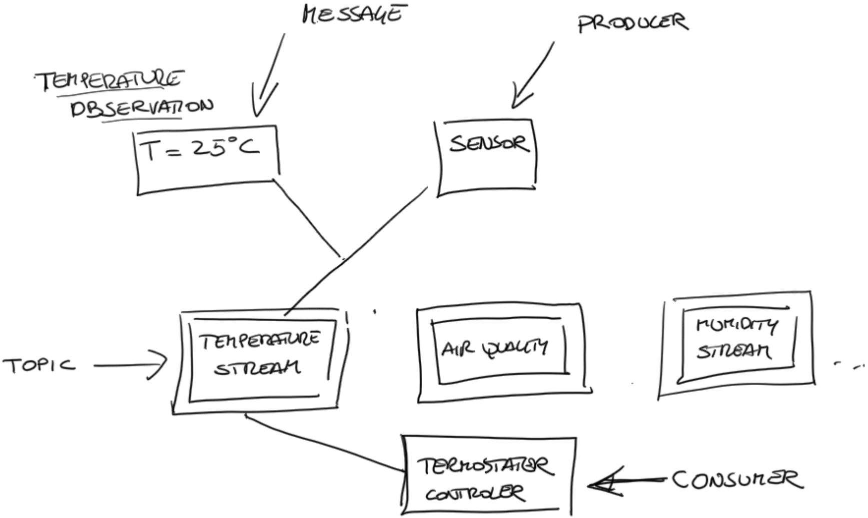


Kafka Conceptual View

- **Messages**, the basic unit in Kafka, are organized in **Topics**
- **Producers** write messages topics
- **Consumers** read messages by from topics

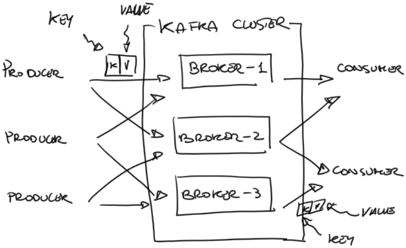


Kafka Conceptual View: Example



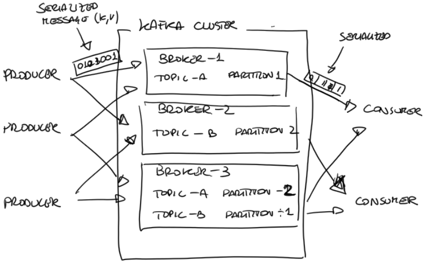
Kafka Logical View

- **Messages** are key-value pairs
- **Brokers** are the main component inside the Kafka Cluster.
- **Producers** write messages to a certain broker
- **Consumers** read messages by from a certain broker

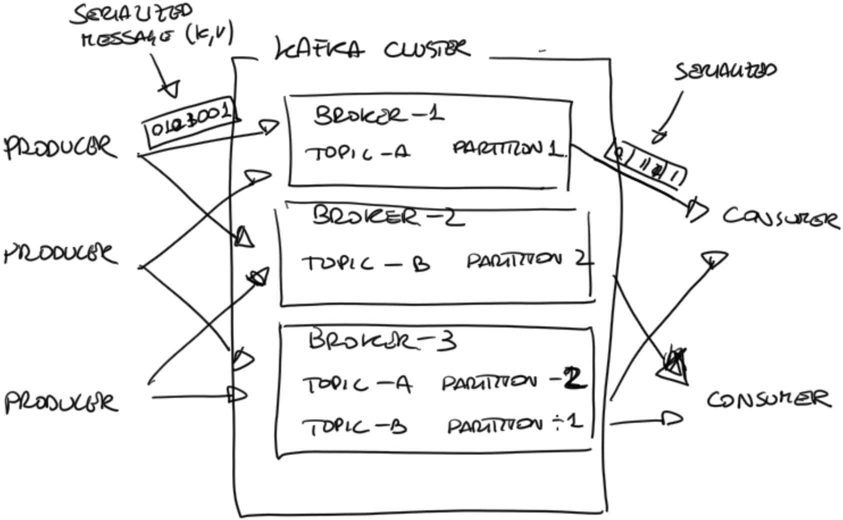


Kafka Physical View

- **Topics** are partitioned across brokers using the message **Key**.
- Typically, **Producers** has the message key to determine the partition. Also they serialize the message
- **Consumers** read messages by from brokers and de-serialize them



Kafka Physical View: Zoom In



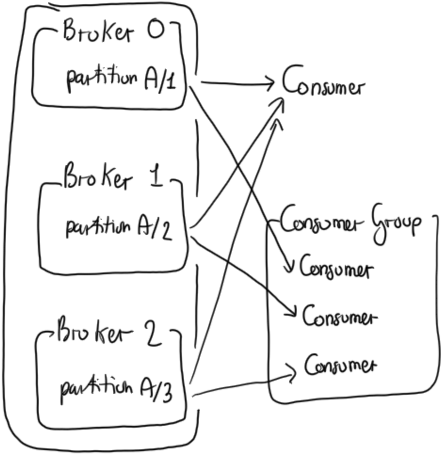
Topics Partitions

Producers shard data over a set of Partitions

- Each Partition contains a subset of the Topic's messages
- Typically, the message key is used to determine which Partition a message is assigned to
- Each Partition is an ordered, immutable log of messages

Topics Partitions and Distributed Consumption

- Different Consumers can read data from the same Topic
  - By default, each Consumer will receive all the messages in the Topic
- Multiple Consumers can be combined into a Consumer Group
  - Consumer Groups provide scaling capabilities
  - Each Consumer is assigned a subset of Partitions for consumption



Apache Kafka[2]

Internals

[2] Slides are based on content from Cloudera and Confluent

Messages and Metadata

Messages are Key-Value pairs and there is not restriction on what each of them can be.

Additionally, messages are enriched with metadata:

- Offset
- Timestamp
- Compression type
- Magic byte
- Optional message headers API
- Application teams can add custom key-value paired metadata to messages
- Additional fields to support batching, exactly once semantics, replication protocol

Topics Partitions: Physical View

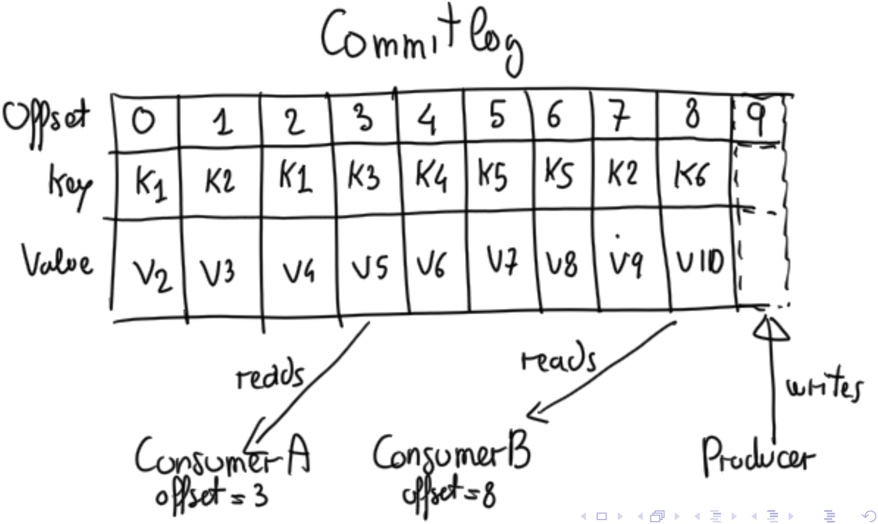
Each Partition is stored on the Broker's disk as one or more log files Each message in the log is identified by its offset number

Commit log

Offset	0	1	2	3	4	5	6	7	8	
key	K_1	K2	K1	K3	K4	K5	K5	K2	K6	
Value	V_2	V3	V4	V5	V6	V7	V8	V9	V10	

Topics Partitions: Physical View

Messages are always appended. Consumers can consume from different offset. Brokers are single thread to guarantee consistency



Topics Partitions: Load Balancing

Producers use a partition strategy to assign each message a partition

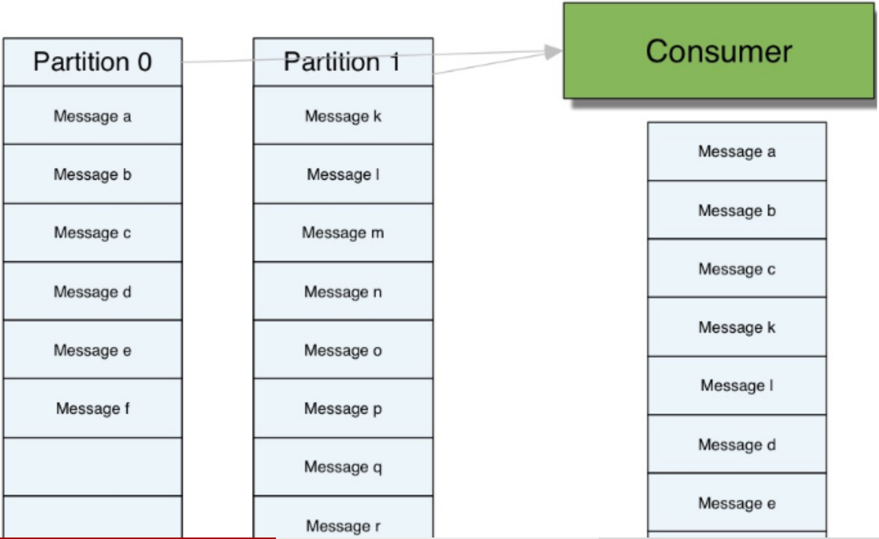
- To ensure load balancing across the Brokers
- To allow user-specified key

You can customize the partition strategy, but!

- it must ensure load balancing across the Brokers too, i.e., hash(key) % number_of_partitions
- if key is not specified, messages are sent to Partitions on a round-robin basis

Important: About Ordering

If there are multiple Partitions, you will not get total ordering across all messages when reading data



Log Retention

- Duration default: messages will be retained for seven days
- Duration is configurable per Broker by setting
  - a time period
  - a size limit
- Topic can override a Broker's retention policy
- When cleaning up a log
  - the default policy is delete
  - An alternate policy is compact

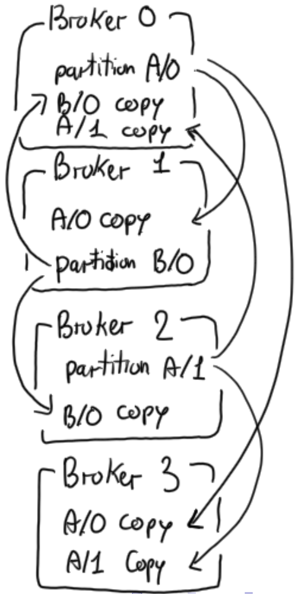
A compacted log retains at least the last known message value for each key within the Partition Before After

Offset	0	1	2	3	4	5	6	7	8
Key	K_1	K2	K1	K3	K4	K5	K5	K2	K6
Value	V_2	V3	V4	V5	V6	V7	V8	V9	V10

K_1	K_3	K_4	K_5	K_2	K_6
V_4	V_5	V_6	V_8	V_9	V_{10}

Fault Tolerance via a Replicated Log

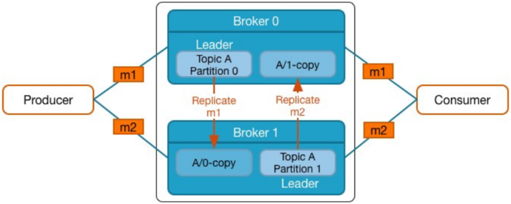
- Kafka maintains replicas of each partition on other Brokers in the cluster
  - Number of replicas is configurable
- One Broker is the leader for that Partition
  - All writes and reads go to and from the leader
  - Other Brokers are followers
- Replication provides fault tolerance in case a Broker goes down



Important: Clients do not Access Followers

It is important to understand that Producers and Consumers only write/read to/fromb the leader

- Replicas only exist to provide reliability in case of Broker failure



- If a leader fails, the Kafka cluster will elect a new leader from among the followers

In the diagram, m1 hashes to Partition 0 and m2 hashes to Partition 1

Delivery Semantics

- At least once
  - Messages are never lost but may be redelivered
- At most once
  - Messages are lost but never redelivered
- Exactly once
  - Messages are delivered once and only once

Zookeeper

- ZooKeeper is a centralized service that stores configurations for distributed applications
- Kafka Brokers use ZooKeeper for a number of important internal features
  - Cluster management
  - Failure detection and recovery
  - Access Control List (ACL) storage

Quiz

Provide the correct relationship - 1:1, 1:N, N:1, or N:N -

- Broker to Partition - ?
- Key to Partition - ?
- Producer to Topic - ?
- Consumer Group to Topic - ?
- Consumer (in a Consumer Group) to Partition - ?

Quiz

Provide the correct relationship - 1:1, 1:N, N:1, or N:N -

- Broker to Partition - N:N
- Key to Partition - N:1
- Producer to Topic - N:N
- Consumer Group to Topic - N:N
- Consumer (in a Consumer Group) to Partition - 1:N

Getting Exactly Once Semantics

- Must consider two components
  - Durability guarantees when publishing a message
  - Durability guarantees when consuming a message
- Producer
  - What happens when a produce request was sent but a network error returned before an ack?
  - Use a single writer per partition and check the latest committed value after network errors
- Consumer
  - Include a unique ID (e.g. UUID) and de-duplicate.
  - Consider storing offsets with data

Apache Kafka[3]

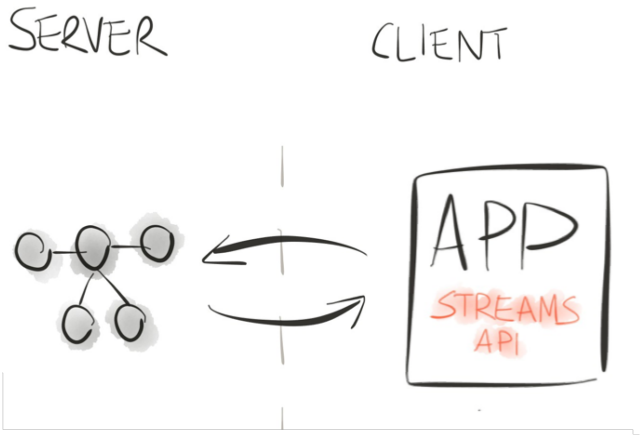
Streams

[3] Slides are based on content from Cloudera and Confluent

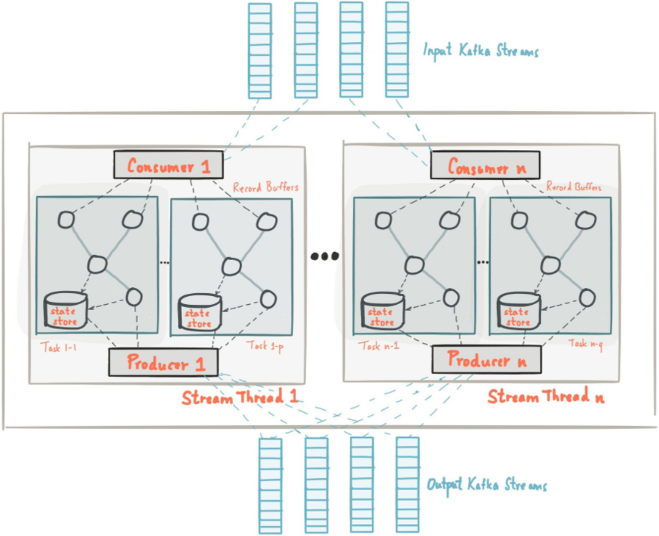
Stream Processing with Kafka

- As of version 0.10.0, Kafka streams library has been added to Kafka distribution
- It is no longer just a distributed message broker
- You can process messages in the different Kafka topics in real-time
- You can produce new messages to (other) topics

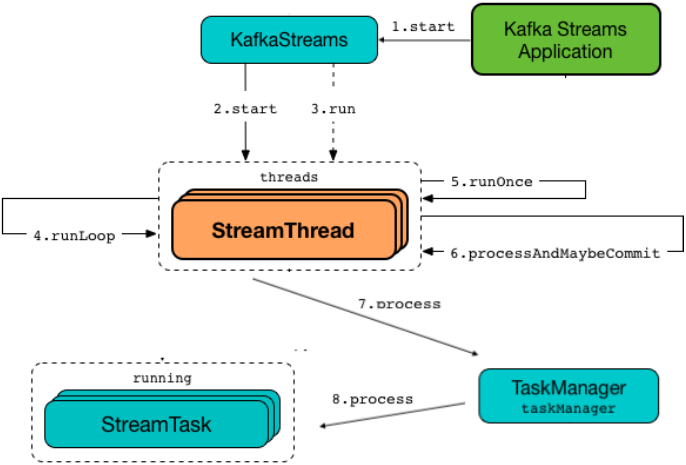
Kafka Streams Library



Kafka Streams: Closer Look



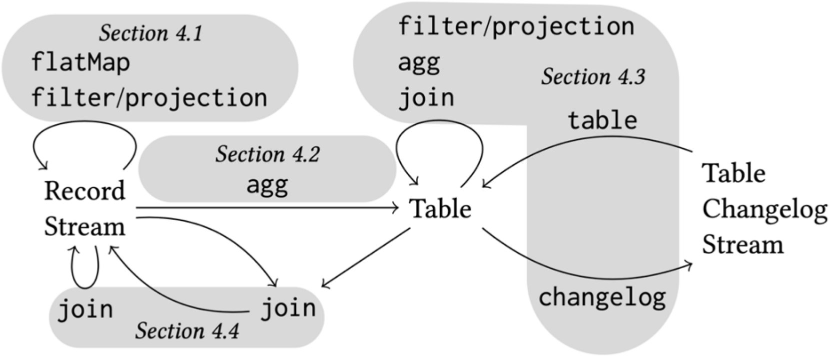
Kafka Streams: Even Closer Look



Kafka Streams: How

- Stream Application is the main abstraction for the User to interact
- Data streams are elicited from topics
- Stream Threads are stream processor threads (a Java Thread) that runs the main record processing loop when started
- Stream Task are build upon Producer and Consumer APIs

Kafka Streams Model[4]



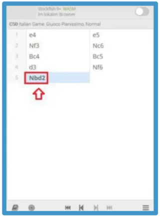
[4] Sax, Matthias J., et al. "Streams and tables: Two sides of the same coin." Proceedings of the International Workshop on Real-Time Business Intelligence and Analytics. 2018.

Stream-Table Duality[5]

- Streams carry individual stateless events
  - An individual event can represent a state change, e.g., for a table
- Tables can be used as a state at a certain time (snapshot)
  - Accumulation of the individual events (stateful)

[5] Matthias J. Sax, Guozhang Wang, Matthias Weidlich, Johann-Christoph Freytag. *Streams and Tables: Two Sides of the Same Coin*. BIRTE 2018: 1:1-1:10

Stream-Table: Chess Analogy

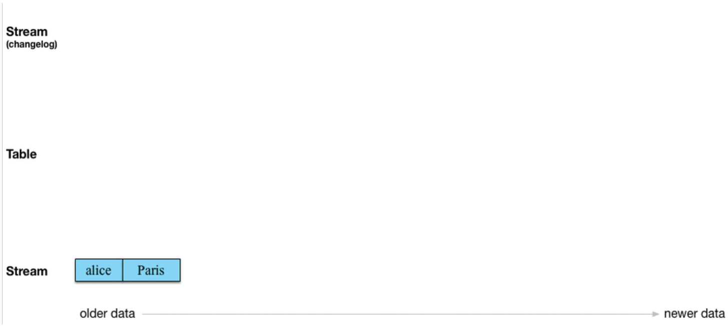


Streams record History
"The sequence of moves."

Tables represent State
"The state of the board at last move."

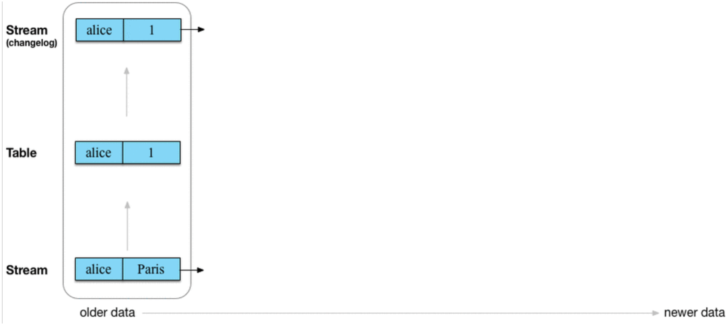
Source: Micheal Noll

Example



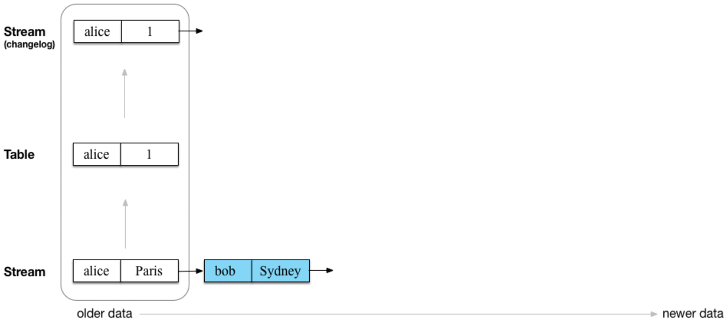
Source: Micheal Noll

Example



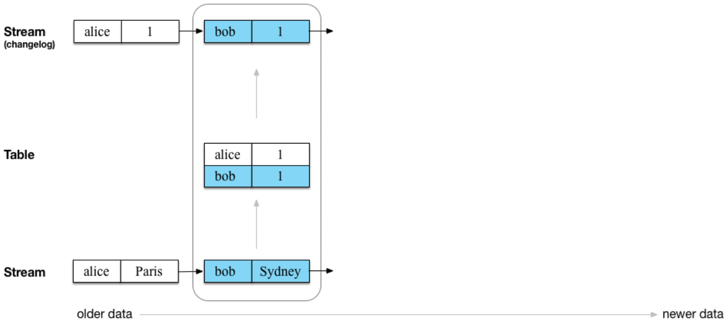
Source: Micheal Noll

Example



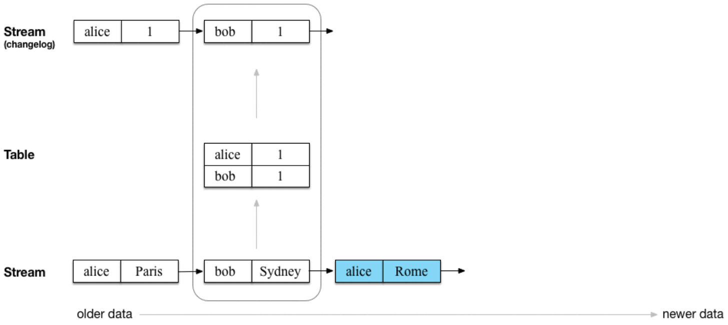
Source: Micheal Noll

Example



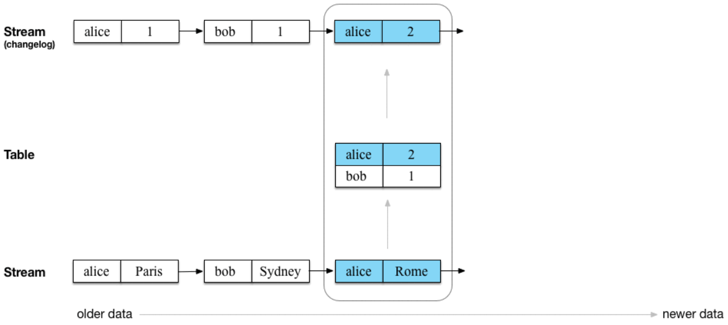
Source: Micheal Noll

Example



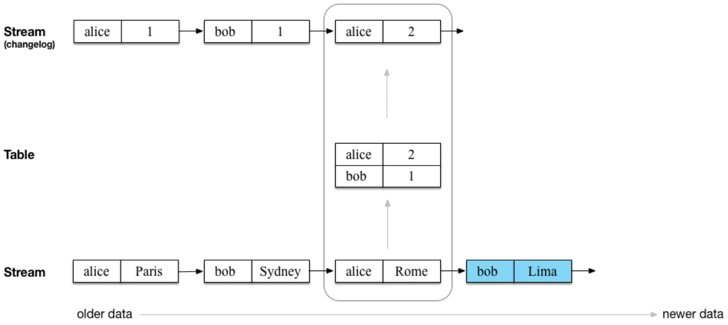
Source: Micheal Noll

Example



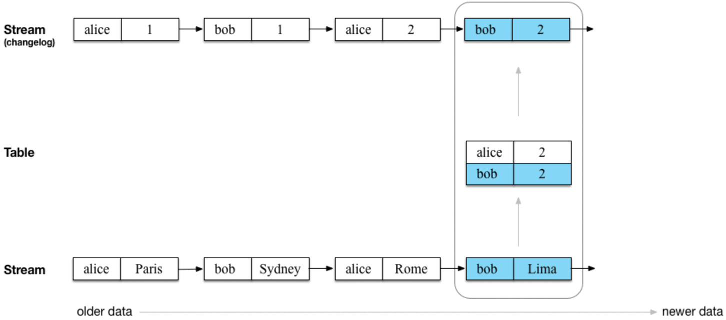
Source: Micheal Noll

Example



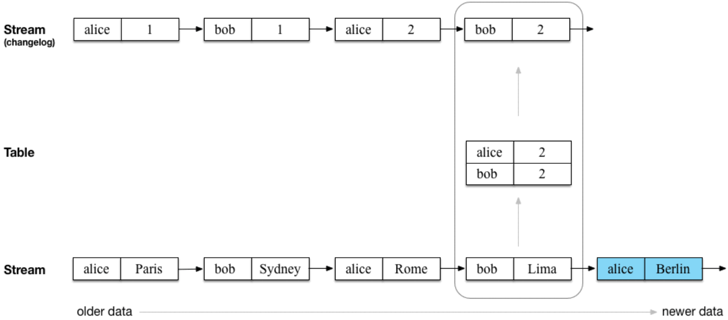
Source: Micheal Noll

Example



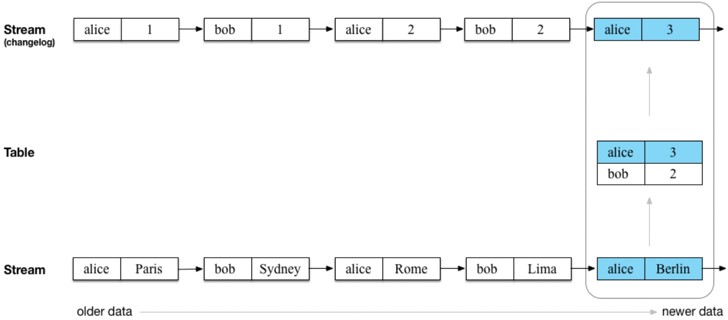
Source: Micheal Noll

Example



Source: Micheal Noll

Example

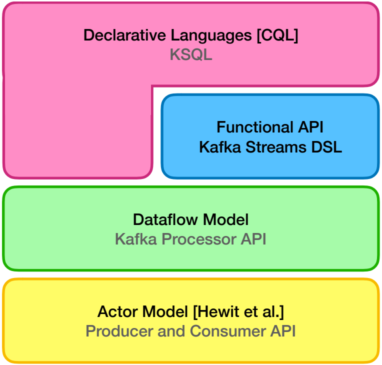


Source: Micheal Noll

Programming with Streams

Stream processing frameworks hide execution details from the programmers, and manage them in the background.

There are different abstraction levels that a programmer can use to express streaming computations.



Kafka Streams: APIs

Kafka allows writing streaming programs at different levels of abstraction:

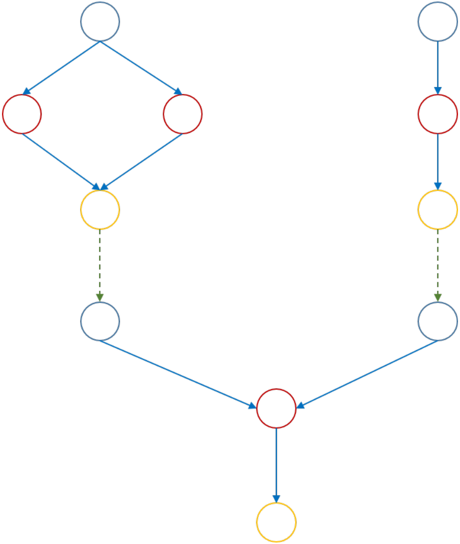
- Kafka Processor API builds directly on top of Apache Kafka. It allows defining operators and organize them in topologies, i.e., Direct-Acyclic Graphs (DAG).
- Kafka Streams DSL builds on Kafka Processor API. It implements a number of functional operations and let the programmers design data pipeline that are automatically translated into DAGs.
- KSQL is yet another level on top of Kafka Streams. It provides a SQL-based declarative syntax that allows defining simple ETL (Extract, Transform,Load) jobs on top of Kafka Streams DSL.

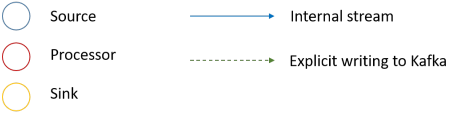
KStream/KTable

- KStream
  - Record stream
  - Each record describes an event in the real world
  - Example: click stream
- KTable
  - Changelog stream
  - Each record describes a change to a previous record
  - Example: position report stream

Processor Topology

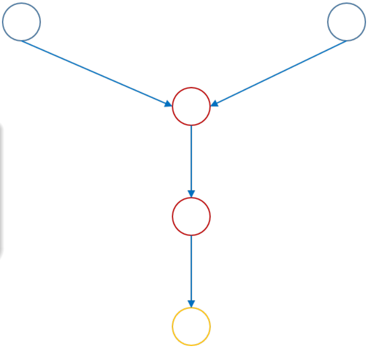
- Close idea to Storm Topology
  - DAG in General
- Several topologies can be linked together
  - Achievable via writing back to Kafka





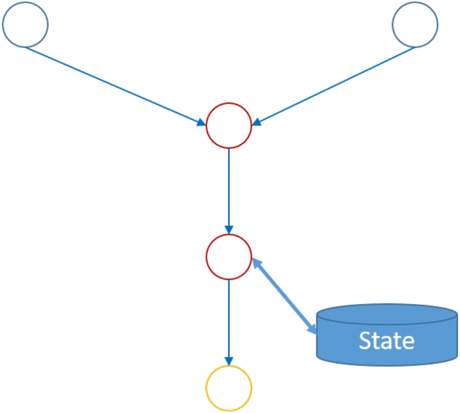
Kafka Streams DSL

```
KStream<..> stream1 = builder.stream("topic1");
KStream<..> stream2 = builder.stream("topic2");
KStream<..> joined = stream1.leftJoin(stream2, ...);
KTable<..> aggregated = joined.aggregateByKey(...);
aggregated.to("topic3");
```



Stateful Processing

- Stateful processors
  - Windowing
  - Joins
  - Aggregation
- Kafka provides a configurable local state store
  - Memory
  - Disk



Notions of Time

- Recall we have
  - Event time: when the data was actually generated
  - Processing time: when the data was received/processed by the system
- Kafka provides a uniform `Timestamp Extractor`
  - Based on Kafka configuration `log.message.timestamp.type`, Kafka streams will read either the ingestion or the event time (default)
  - You can still create your own extractor

Windowing

- Kafka Streams supports time-based windows only
    - **Tumbling**
    - Sliding (called hopping)
    - Session

```
KStream<String, GenericRecord> pageViews = ...;
// Count page views per window, per user, with tumbling windows of size 5 minutes
KTable<Windowed<String>, Long> windowedPageViewCounts = pageViews
.groupByKey(Grouped.with(Serdes.String(), genericAvroSerde))
.windowedBy(TimeWindows.of(Duration.ofMinutes(5))) .count();
```

Windowing

- Kafka Streams supports time-based windows only
  - Tumbling
  - **Sliding (called hopping)**
  - Session

```
KStream<String, GenericRecord> pageViews = ...;
// Count page views per window, per user, with hopping windows of size 5 minutes
// that advance every 1 minute
KTable<Windowed<String>, Long> windowedPageViewCounts = pageViews
.groupByKey(Grouped.with(Serdes.String(), genericAvroSerde))
.windowedBy(TimeWindows.of(Duration.ofMinutes(5).advanceBy(Duration.ofMinutes(1)))).count()
```

Windowing

- Kafka Streams supports time-based windows only
  - Tumbling
  - Sliding (called hopping)
  - **Session**

```
KStream<String, GenericRecord> pageViews = ...;
// Count page views per session, per user, with session windows that have
// an inactivity gap of 5 minutes
KTable<Windowed<String>, Long> sessionizedPageViewCounts = pageViews
.groupByKey(Grouped.with(Serdes.String(), genericAvroSerde))
.windowedBy(SessionWindows.with(Duration.ofMinutes(5))).count();
```

Late Arrival

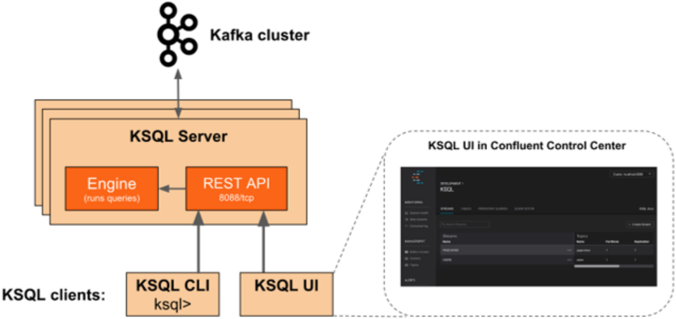
- By default, Kafka streams emits an updated result for each newly arriving record
  - No need for special handling for late arrival, simply a new result is emitted
  - Window retention is a configuration parameter in Kafka, default is one day
  - No watermark support
- Some times, you need to receive one result at the end of the window
  - You can do that in your code

```
KGroupedStream<UserId, Event> grouped = ...;
grouped.windowedBy(TimeWindows.of(Duration.ofHours(1)).grace(Duration.ofMinutes(10)))
.count()
.suppress(Suppressed.untilWindowCloses(unbounded()))
.filter((windowedUserId, count) -> count < 3)
.toStream()
.foreach((windowedUserId, count) -> sendAlert(windowedUserId.window(),
windowedUserId.key(), count));
```

KSQL

- Brings SQL support to Kafka Streams
- Streaming ETL
  - DDL
  - Querying
  - Link streams to tables
    - Don't confuse it with KStream and KTable

KSQL

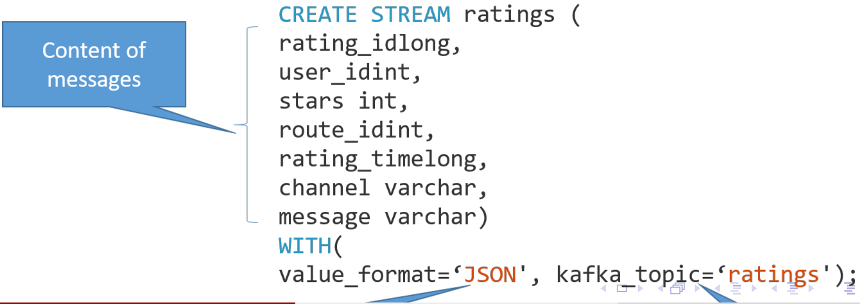


Kafka cluster
KSQL Server
Engine
(runs queries)
REST API
8088/tcp
KSQL clients:
KSQL CLI
ksql>
KSQL UI
KSQL UI in Confluent Control Center

Create Stream

Stream is KSQL's wrapper for the data in a Kafka topic

```
CREATE STREAM ratings (
rating_idlong ,
user_idint ,
stars int ,
route_idint ,
rating_timelong ,
channel varchar ,
message varchar)
WITH(
value_format='JSON', kafka_topic='ratings');
```



Selecting From the Stream

```sql
SELECT *
FROM ratings
WHERE stars <= 2
AND lcase(channel) LIKE '%ios%'
AND user_id > 0
LIMIT 10;
```

Selecting From the Stream

```sql
SELECT *
FROM ratings
WHERE stars <= 2
AND lcase(channel) LIKE '%ios%'
AND user_id > 0
LIMIT 10;
```

We can derive another stream based on the query result

```sql
CREATE STREAM poor_ratings AS
SELECT *
FROM ratings
WHERE stars <= 2
AND lcase(channel) LIKE '%ios%';
```

Create Table

```
CREATE TABLE users (
uidint,
name varchar,
elite varchar)
WITH(
Key= 'uid',
value_format='JSON', kafka_topic='mysql-users');
```

Enrich Stream with Table Data

```sql
CREATE STREAM vip_poor_ratings AS
SELECT uid, name, elite,
 stars, route_id, rating_time, message
FROM poor_ratingsr LEFT JOIN users u ON r.user_id= u.uid
WHERE u.elite= 'P';
```

Aggregation and Windowing

```sql
SELECT uid, name, count(*) as rating_count
FROM vip_poor_ratings
WINDOW TUMBLING(size 2 minutes)
GROUPBY uid, name;
```

The End


Thank You
Mahalo
Kiitos
Tack
Toda
Thanks
Grazie
Obrigado
Takk
Merci
Gracias