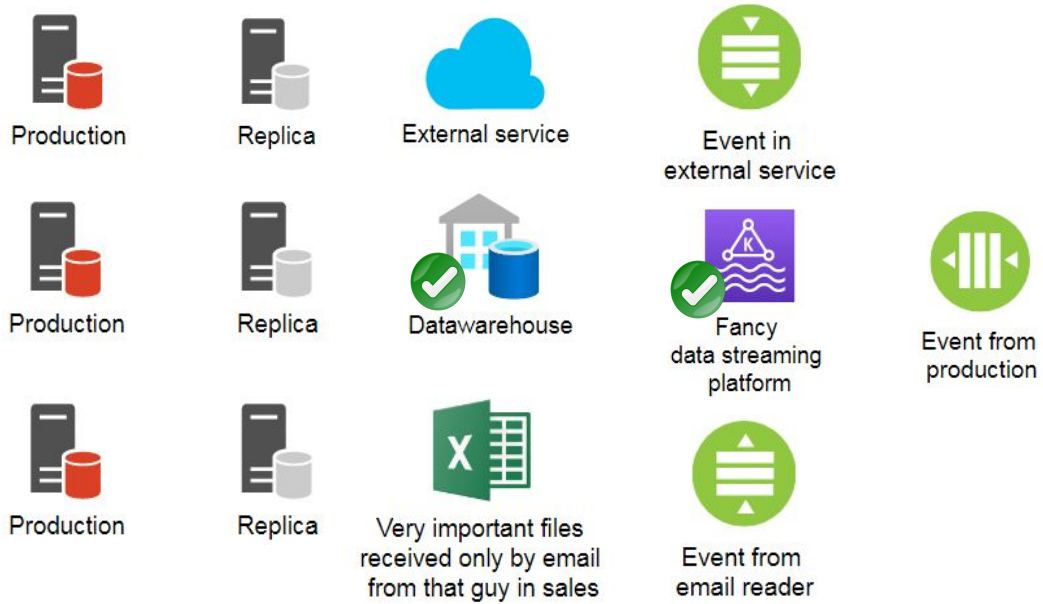
Stream Processing with Kafka and Python

What is it about?

- Context: data engineering
- Goal: make decision process less intuitive and more informed
- Approach: datawarehouse
  - Collect information in one place, process on schedule
  - Make well-informed decisions using all of it
- Approach: streaming
  - Processing as a reaction to an event
  - Make decisions as soon as possible

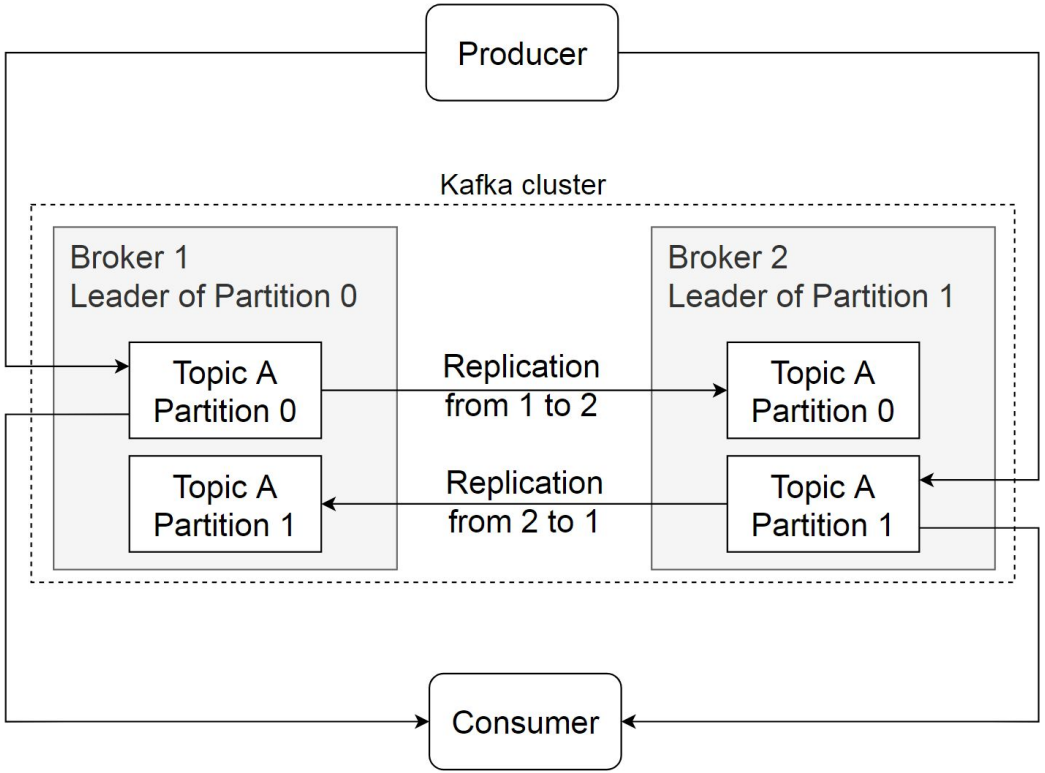
The story of data



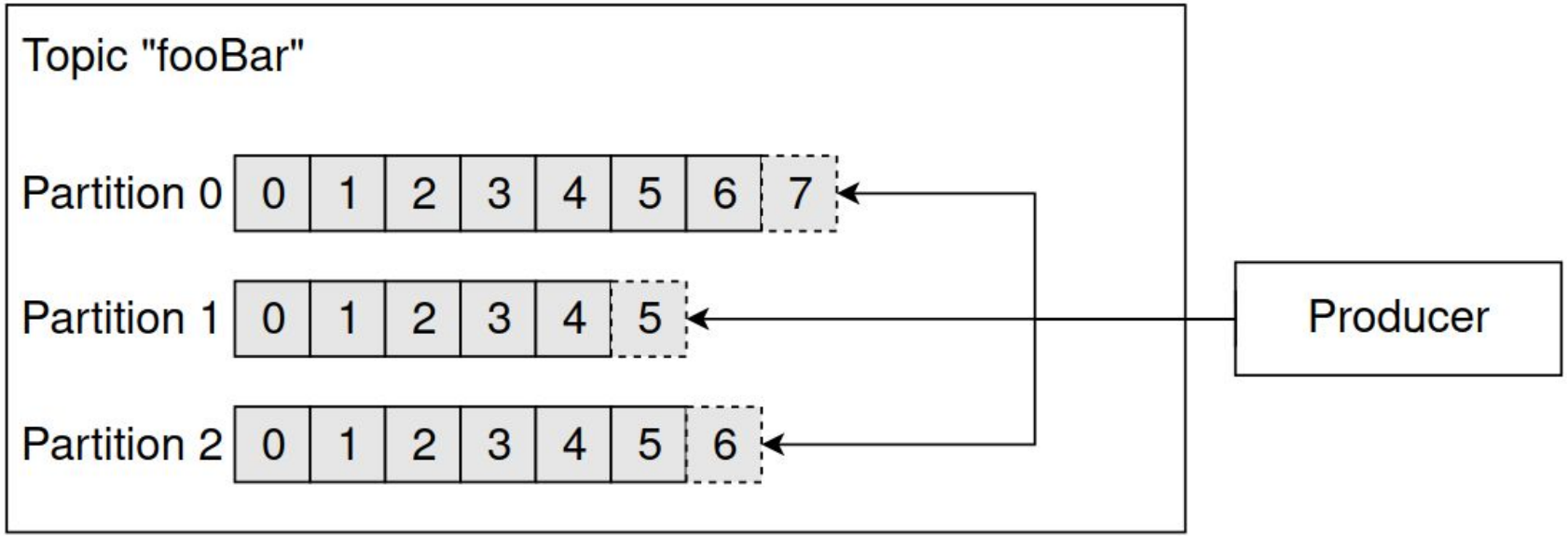
Welcome to Kafka

- Publish/subscribe messaging system
- Unit of data is a *message*
  - Similar to a row in relational databases
- Message belongs to a *topic*
  - Similar to a table in relational databases
- Topic is split in *partitions*

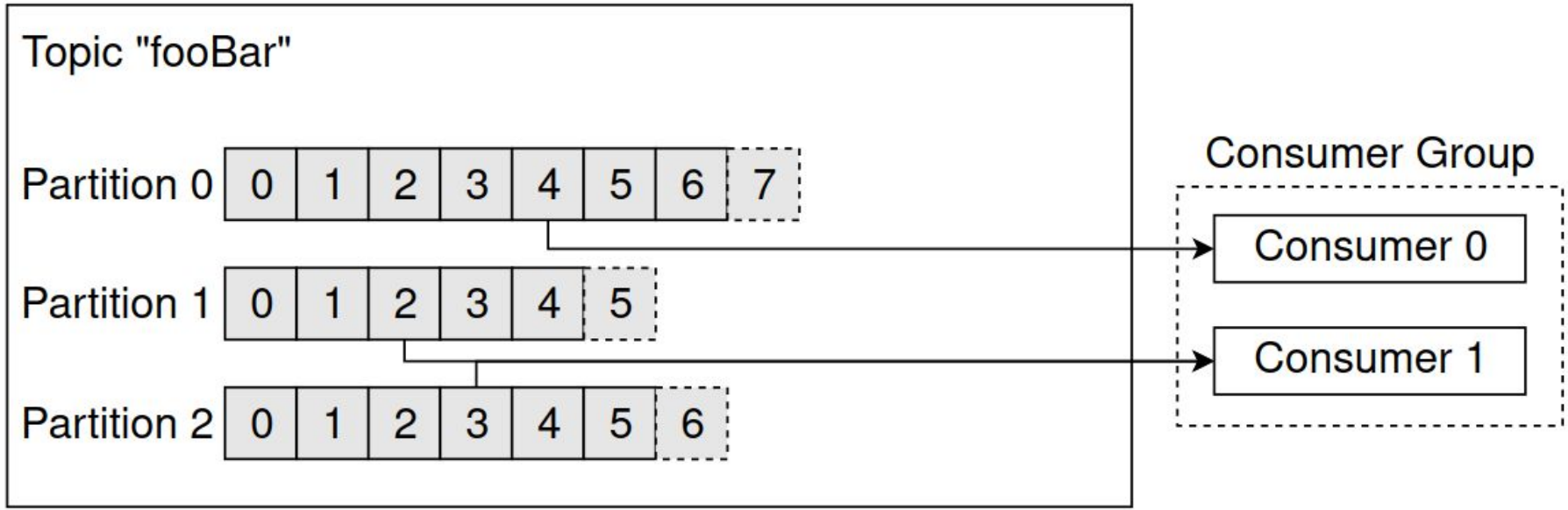
Why Kafka is called distributed?



Writing to Kafka

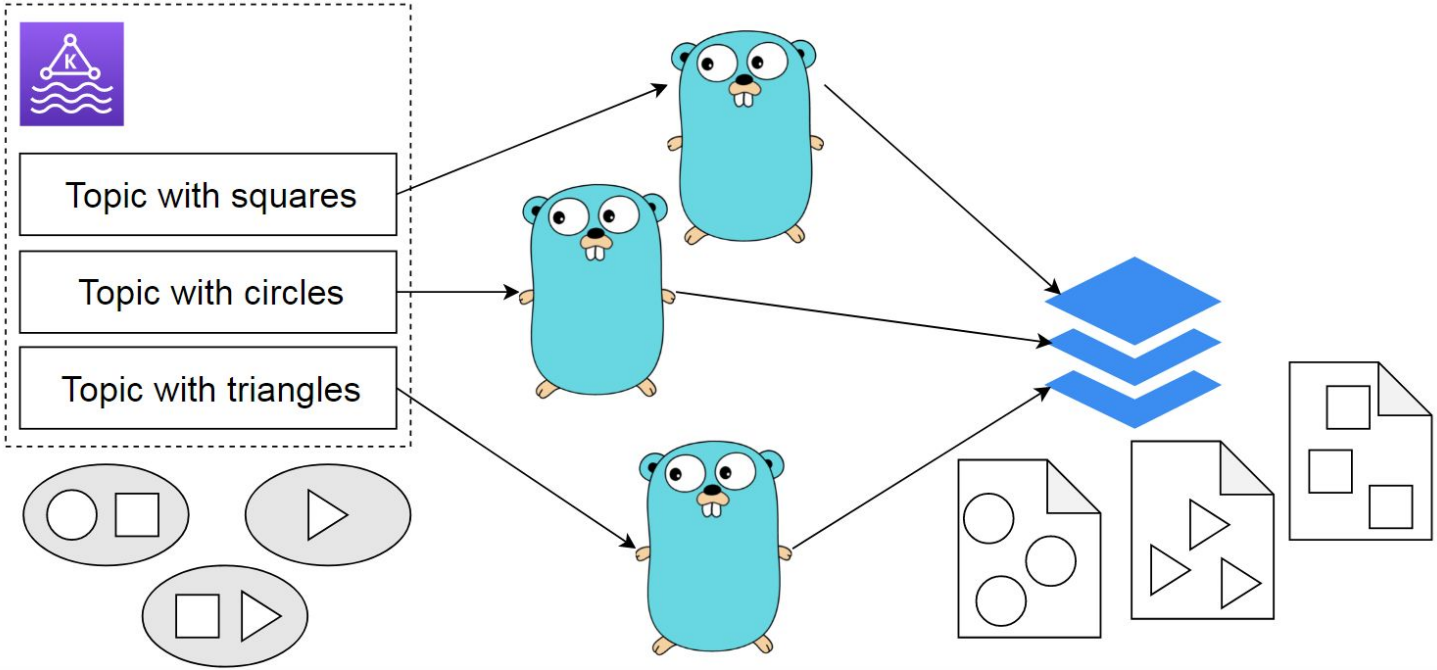


Reading from Kafka



How to write messages to files

and keep your mental health in shape



Configuration: topic

- `num.partitions`
  - Decide basing on future throughput vs consumer capacity
- `log.retention.ms` and `log.retention.bytes`
- `log.segment.bytes` and `log.segment.ms`
  - Retention happens per segments
- `message.max.bytes`
  - Defaults to 1 Mb; beware of performance impact
  - Adjust `fetch.message.max.bytes` and `replica.fetch.max.bytes`

Configuration: producer

- `acks` configures desired delivery guarantee
  - `0` means fire and forget
  - `1` means leader replica received the message
  - `all` means all in-sync replicas received the message
- `batch.size` configures a tradeoff between latency and throughput
  - Large batches increase maximum of messages per second
  - Small batches decrease latency for a single message
- `linger.ms` is the amount of time to wait before sending the batch
  - Defaults to 0; use higher value to increase throughput

Configuration: consumer

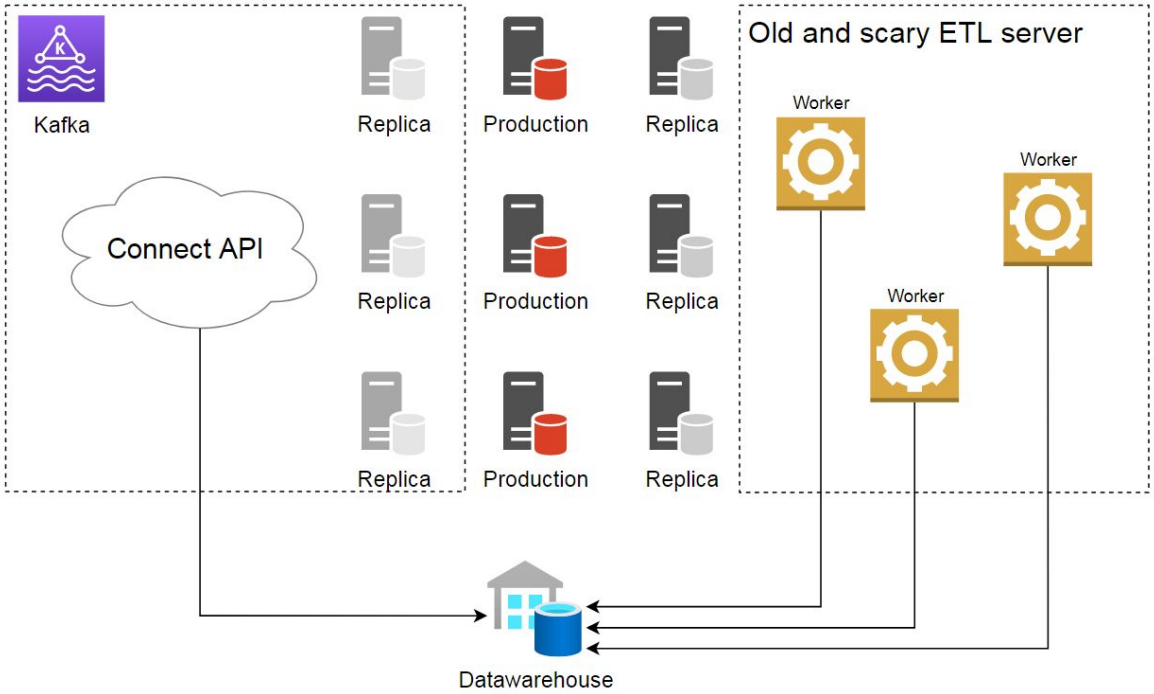
- `auto.offset.reset` controls handling of a new partition
  - `latest` is the default option to read from newest records
  - `earliest` means reading from the oldest records
- `enable.auto.commit` can be `true` or `false`
- `partition.assignment.strategy` defines how partitions are assigned to consumers
  - Range strategy is the default one
  - RoundRobin strategy provides more evenly distributed assignments
  - Custom implementations are possible

Extending Kafka: Kafka Connect

- Producer/Consumer APIs require programming
  - Connecting Kafka with typical data sources should be easier
- Kafka Connect is a tool for connecting Kafka with other systems
  - Supports two types of connectors: **source** and **sink**
  - Runs in distributed or standalone mode
  - Uses Kafka itself as a backend to keep state
  - Provides REST API to create connectors

Creating of near real time datawarehouse

or how to trick your DBA with a fake replication server



Extending Kafka: Kafka Streams

- Once you want to do data processing, you need to consume it first
  - Do you?
  - What if all data you need is *already in Kafka*?
- Kafka Streams allows to run data processing inside Kafka
- Write Streams app in Java and let Kafka handle operational load

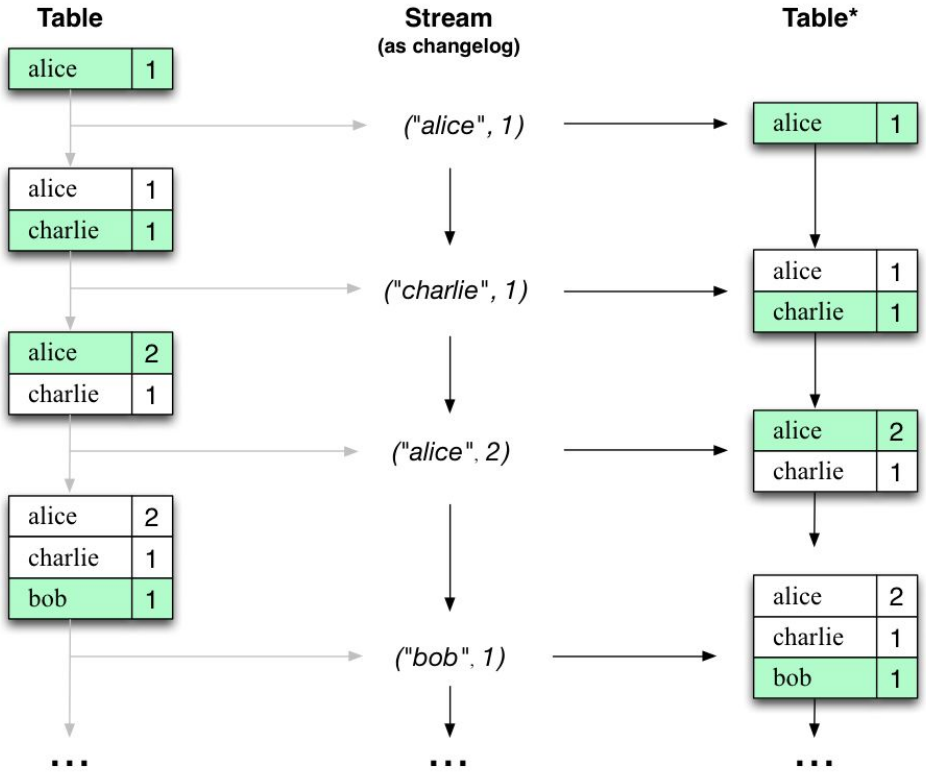
Kafka Streams: Define time

- Event time
  - Created “at the source”
- Log append or ingestion time
  - When message is written into a topic partition
- Processing time
  - When message is being consumed by Streams app
  - It can be milliseconds or hours and days greater than event time
  - Not really reliable

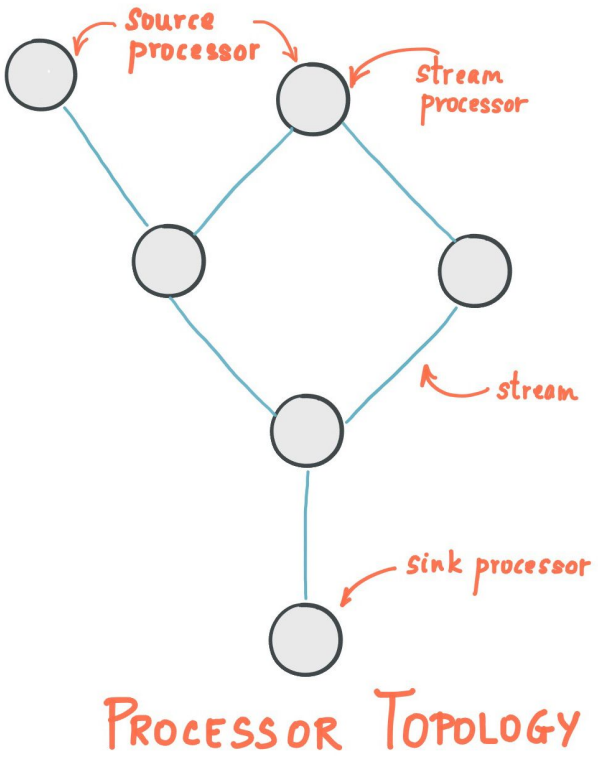
Kafka Streams: Stream-Table duality

A table

key1	value1
key2	value2
key3	value3



Kafka Streams: Processor Topology



- Read from Source processor
- Write into Sink processor
- Transform in Stream processor
  - Streams DSL is available in Java: `map`, `filter`, `join`
  - Aggregations are also possible
  - Windowing with a grace period
  - State stores: persistent key-value or in-memory hashmap or else

Aggregating Wikipedia events: demo app

https://github.com/ilia-khaustov/sytac-devjam2021-wikiapp