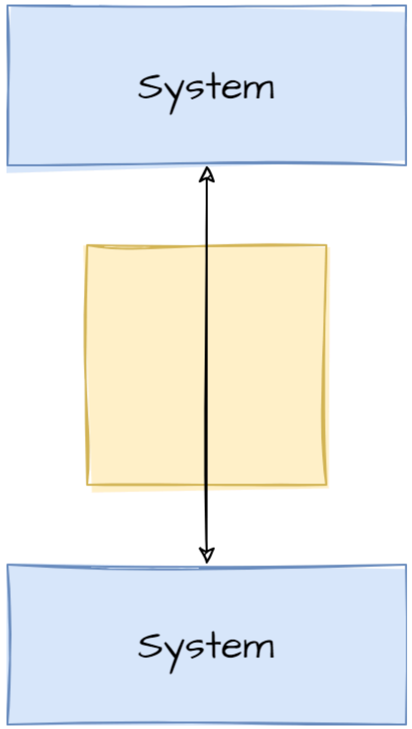
Apache Kafka for Java Developers

Introduction to Apache Kafka

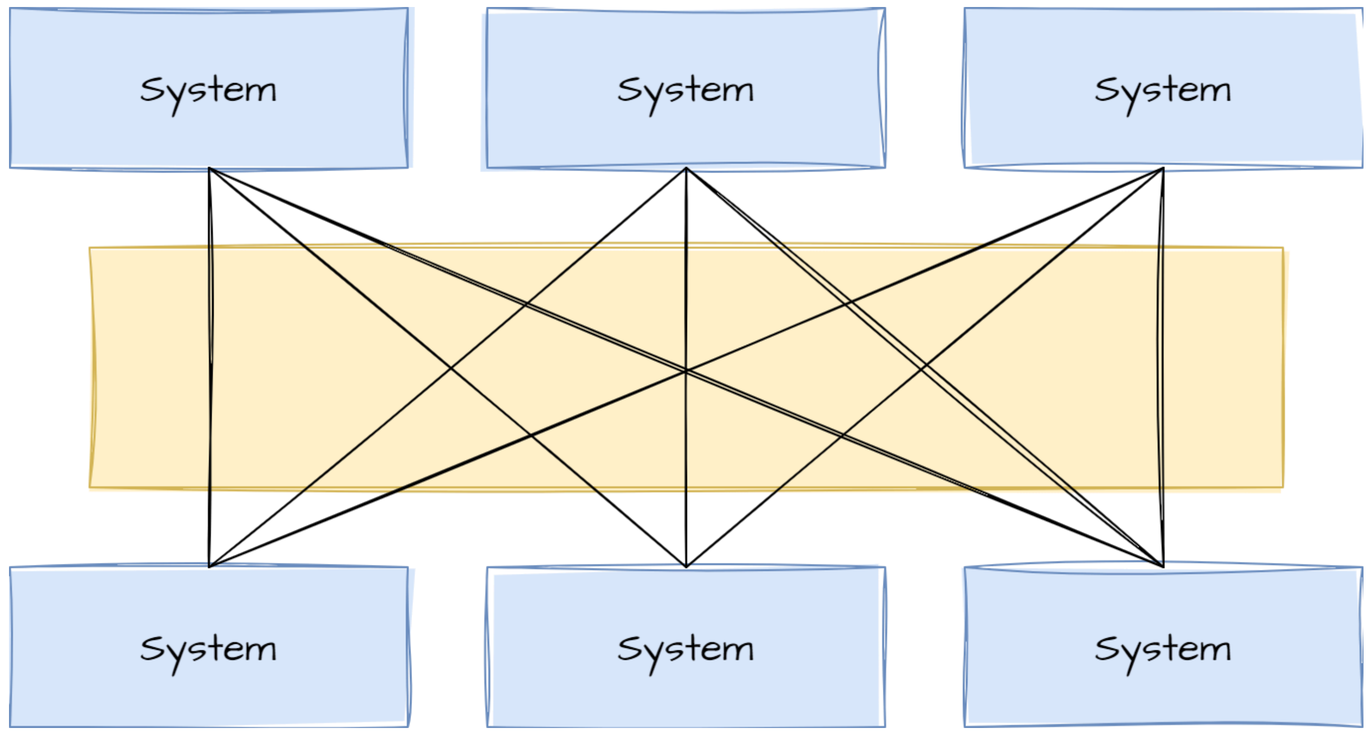
Boris Fresow, Markus Günther

JavaLand 2024, Nürburgring

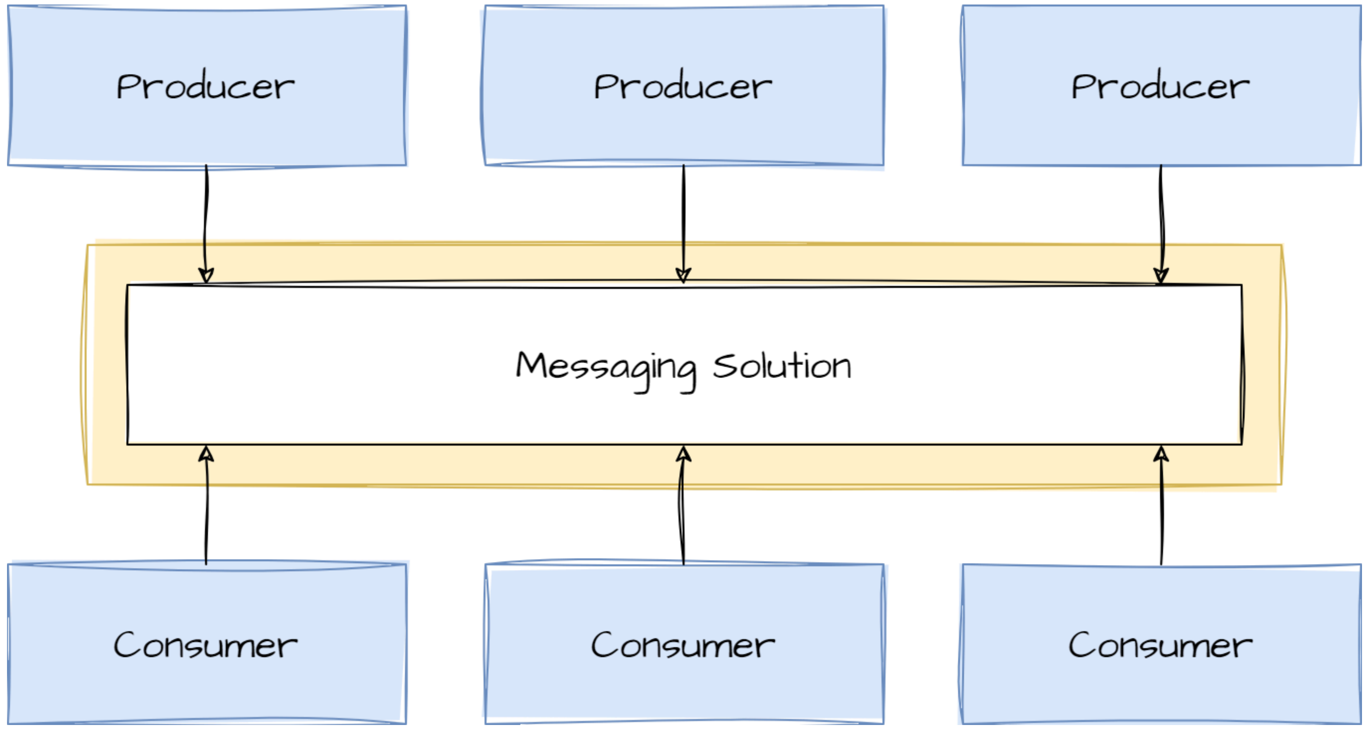
Point-to-point communication is simple to maintain - especially with few systems involved.



More systems increase the complexity of communication channels in this architecture.



A messaging solution decouples producing from consuming systems.



- Systems become decoupled and easier to maintain because they can produce and consume data without knowledge of other systems.

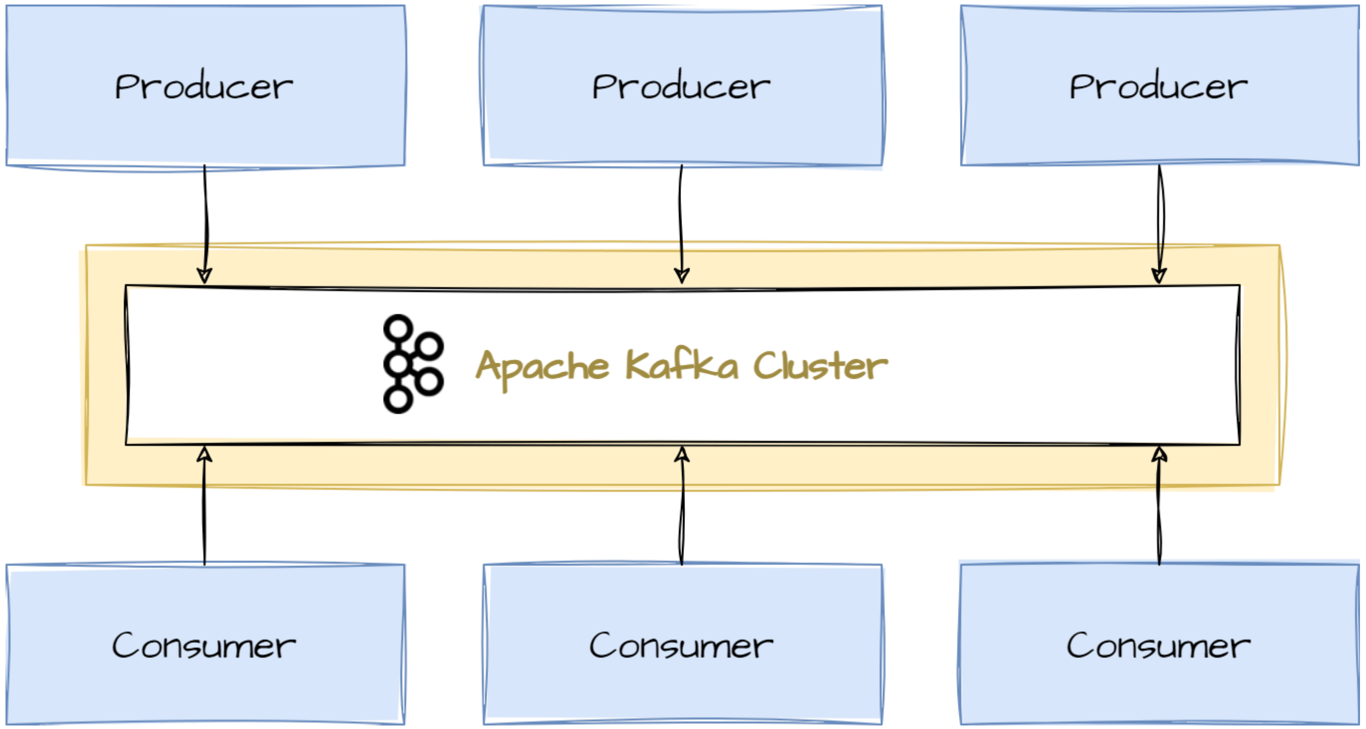
- Asynchronous communication comes with stronger delivery guarantees. If a consumer goes down, it will simply pick up from where it left off when it comes back online again (or shift load to other consumers of a group of consumers).

- Systems can standardize on the communication protocol, as well as scaling strategies and fault-tolerance mechanisms.

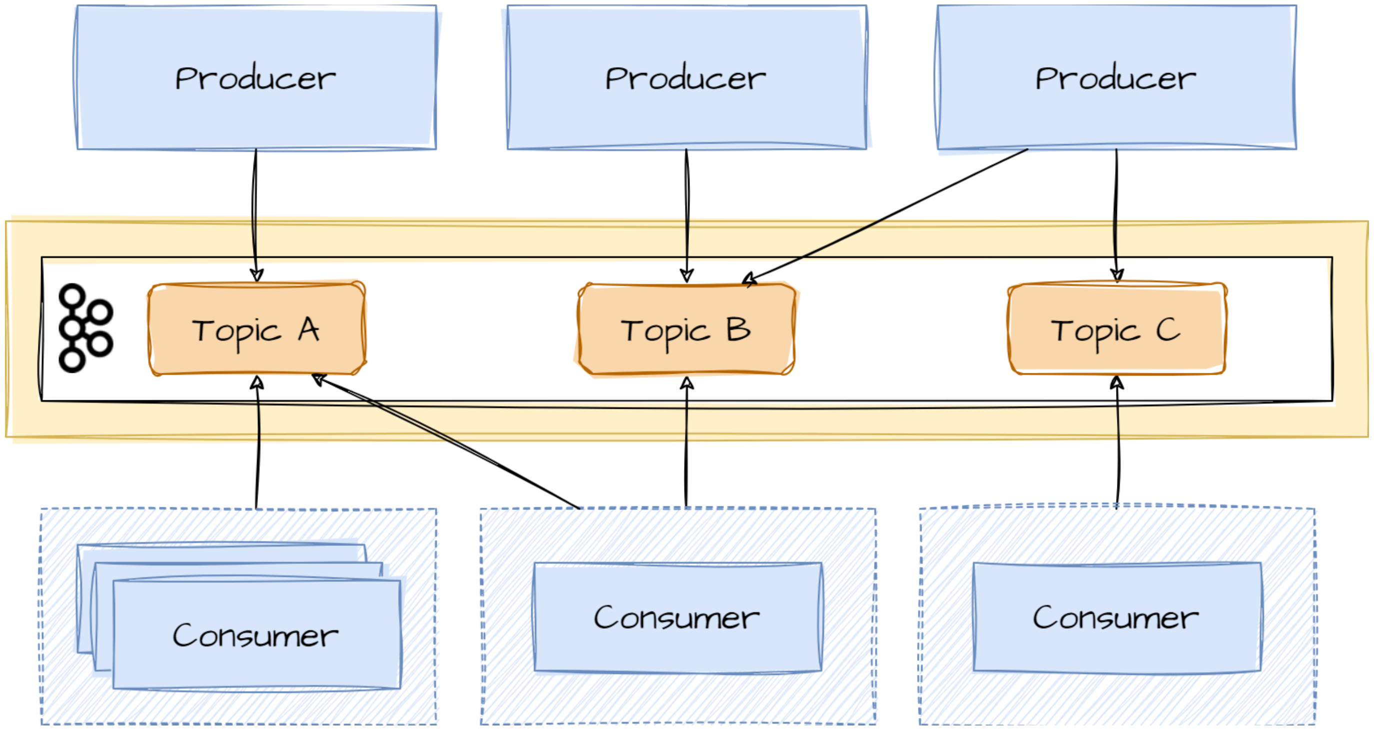
- Consumers consume data at a rate they can handle.

- Systems can rebuild their state anytime by replaying events in a topic.

Apache Kafka supports this communication model.



Producers publish data to topics, consumers subscribe to them and read at their own pace.



- Instead of having multiple systems communicate directly with each other, producers simply publish their data to one or more topics, without caring who comes along to read the data.

- Topics are named streams (or channels) of related data that are stored in a Kafka cluster.

- Topics serve a similar purpose as tables in a database. In fact, there is an underlying duality between a stream and a table that is important when we talk about stream processing for analytics.

- The data in a topic imposes no particular schema as far as Kafka is concerned. Kafka stores raw binary data. You can of course enforce schemas at the application level.

Apache Kafka is a distributed pub-sub messaging system with topic access semantics.

History

- Apache Kafka originated at LinkedIn
- Maintained by the Apache Foundation
- Confluent drives further development
- Confluent provides various system components that enrich the Kafka ecosystem

Apache Kafka is a distributed pub-sub messaging system with topic access semantics. (cont.)

Intentions

- Designed for near-real-time processing of events
- Supports multiple delivery semantics
  - At-least-once
  - Exactly-once (well, not quite)
- Optimized binary protocol for client-to-broker communication
  - No integration with JMS ...

Apache Kafka is a distributed pub-sub messaging system with topic access semantics. (cont.)

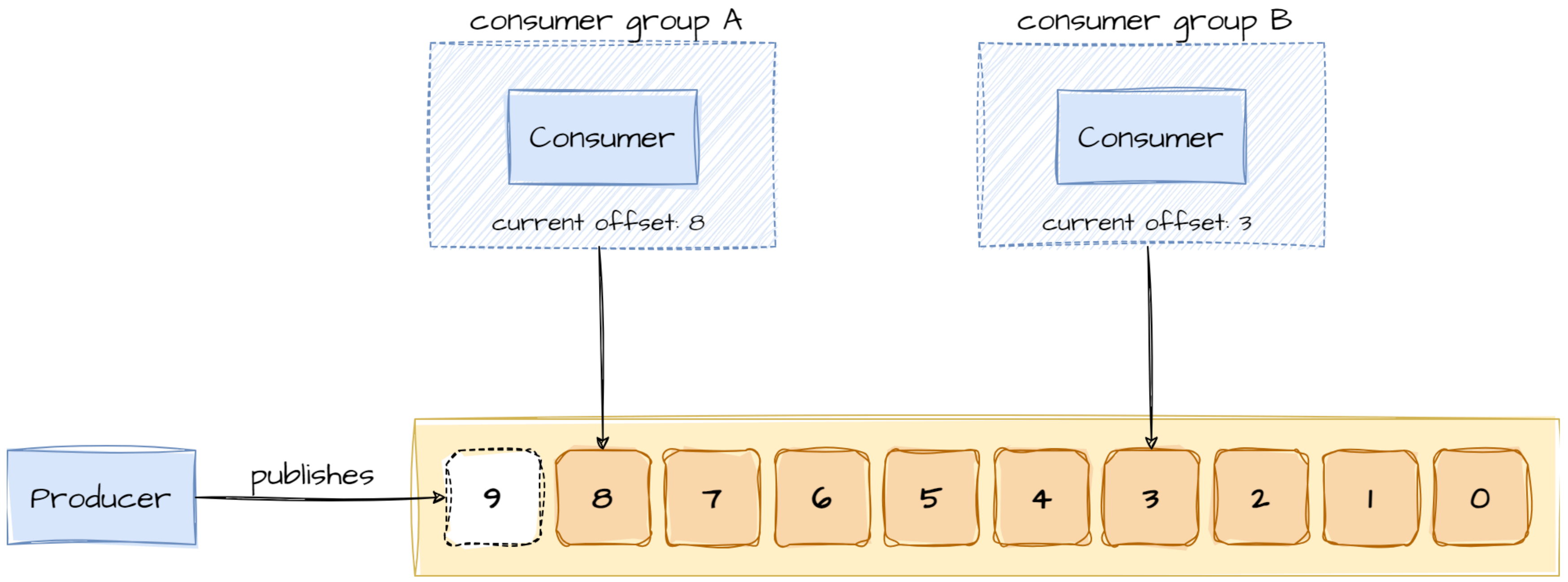
Innovations

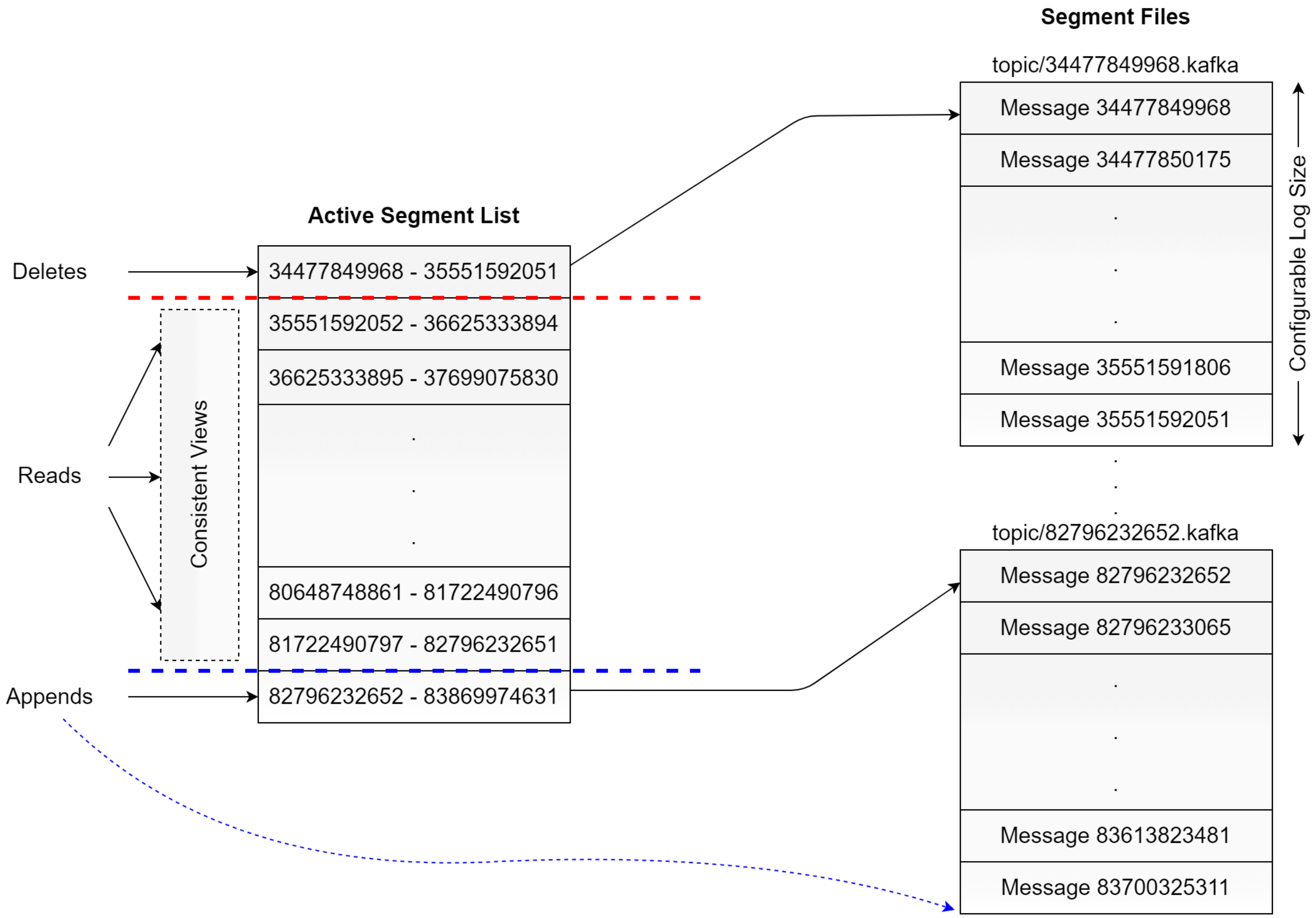
- Messages are acknowledged in order
- Messages are persisted for days / weeks / indefinite
- Consumers manage their offsets
- Very powerful configuration settings allow ...
  - different read/write strategies on the same topic
  - performance tuning on a very low level

- Messages are acknowledged in order: This eliminates the need to track acknowledgements on a per-message, per-listener basis and allows a reader's operation to be very similar to reading a file.
- Messages are persisted for days / weeks / indefinite: This eliminates the requirement to track when readers have finished with particular messages by allowing the retention time to be set so long that readers are almost certain to be finished with messages before they are added.
- Consumers manage their offsets: Consumers control from which offset they continue to read. Applications can even manage their offsets outside of Kafka entirely.

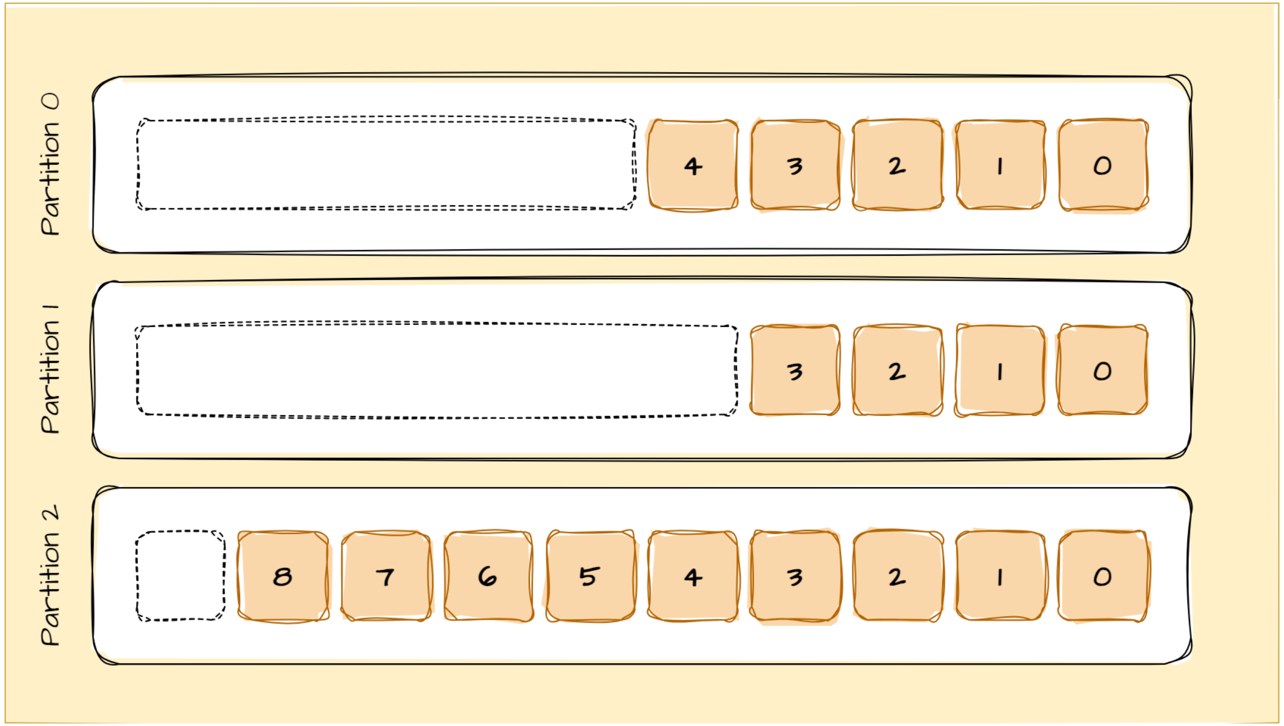
The technical impact of these innovations is that Kafka can write messages to a file system. The files are written sequentially as messages are produced, and they are read sequentially as messages are consumed. These design decisions mean that nonsequential reading or writing of files by a Kafka message broker is very, very rare, and that lets Kafka handle messages at very high speeds.

Kafka uses a persistent log. Producers append to it, and consumers read from it sequentially.

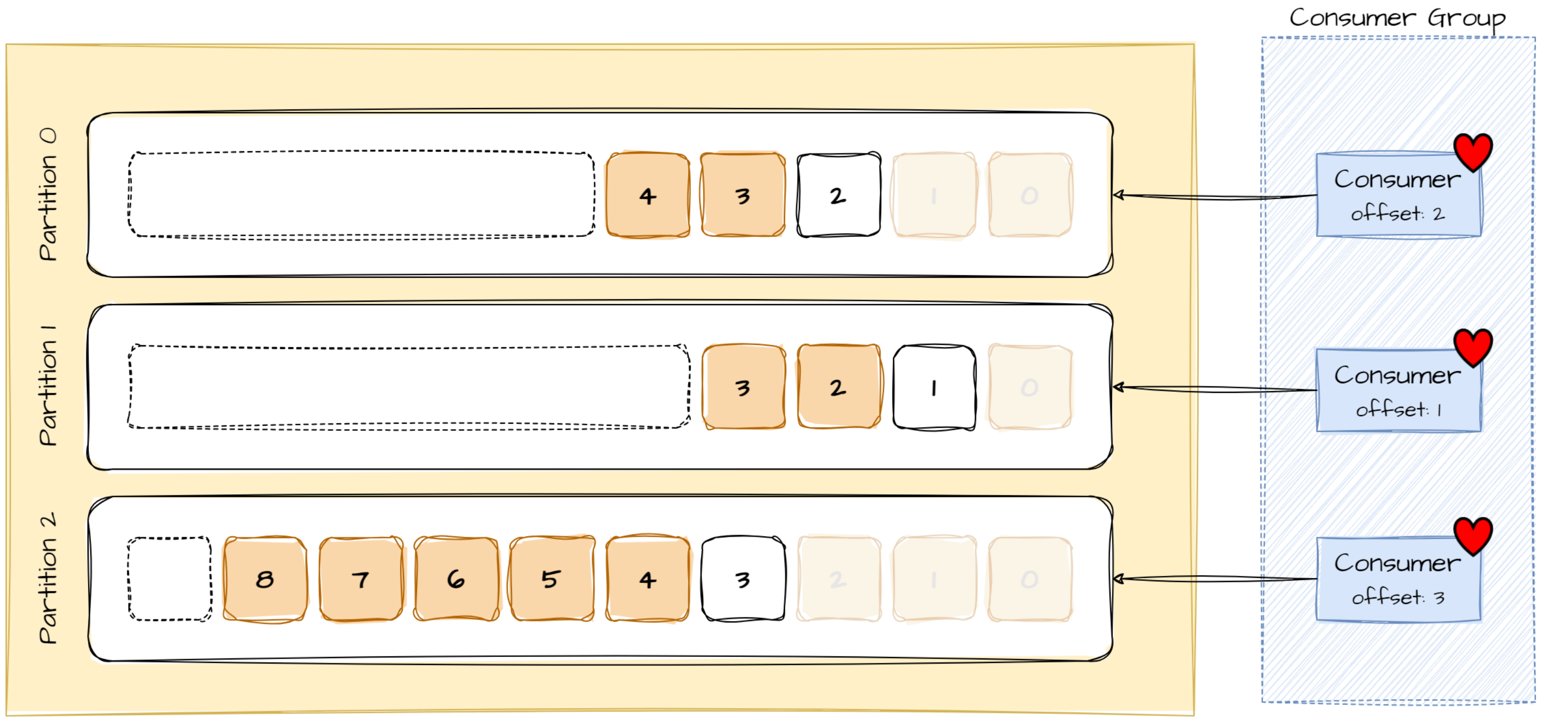



Segment Files
topic/34477849968.kafka
Message 34477849968
Message 34477850175
Message 35551591806
Message 35551592051
Configurable Log Size
topic/82796232652.kafka
Message 82796232652
Message 82796233065
Message 83613823481
Message 83700325311
Active Segment List
Deletes
34477849968 - 35551592051
35551592052 - 36625333894
36625333895 - 37699075830
80648748861 - 81722490796
81722490797 - 82796232651
82796232652 - 83869974631
Reads
Consistent Views
Appends

A Kafka topic is comprised of at least one partition.

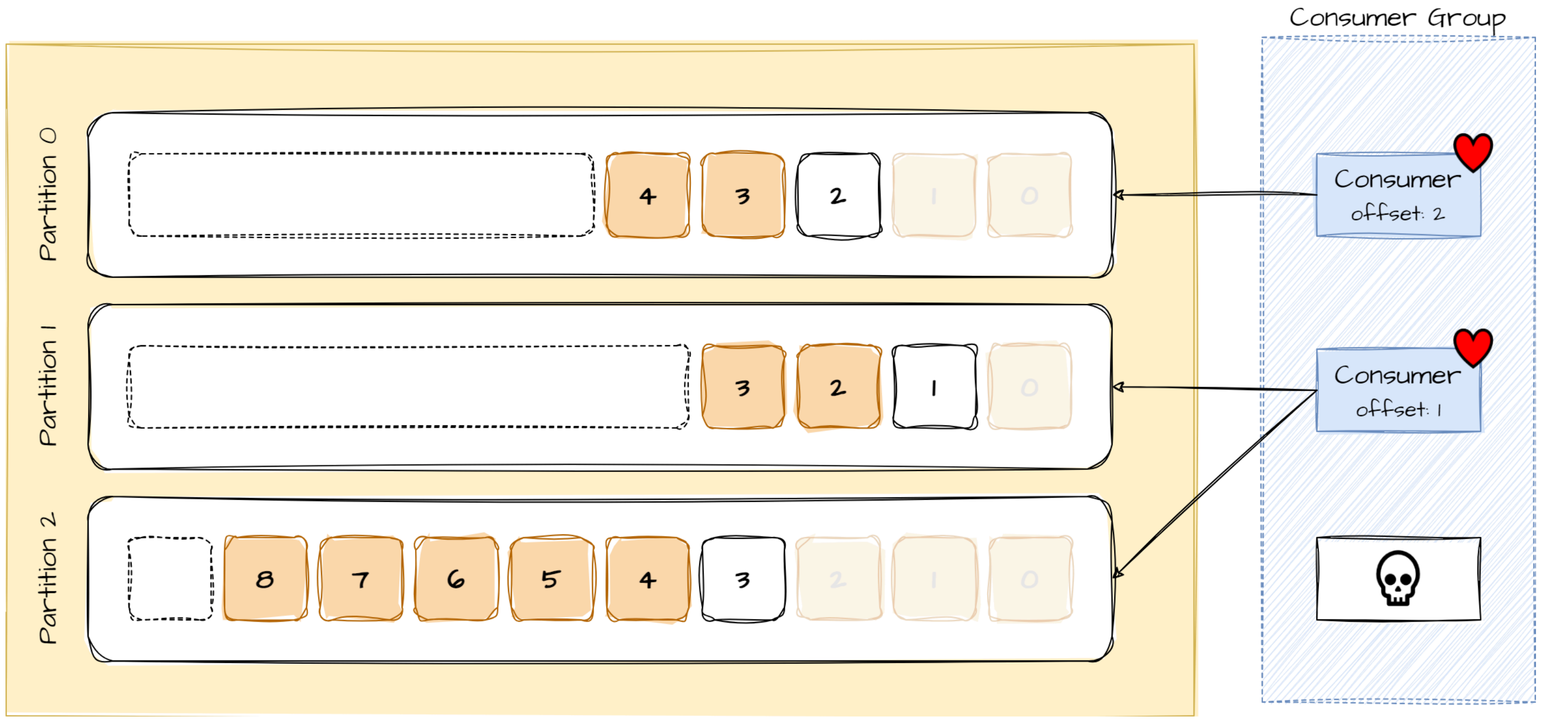


Consumers that participate in the same consumer group share the read workload on a topic.

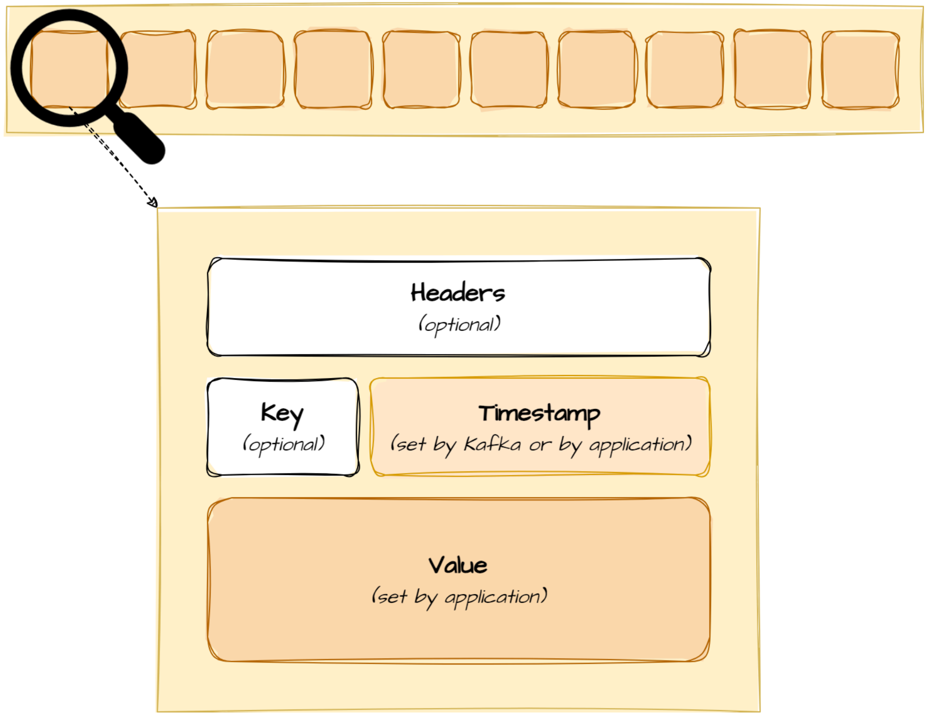


Recall: Messages are acknowledged in order. Since a producer only appends to the log and at most one consumer can consume messages from a topic-partition, this guarantees that we read all messages from a specific topic-partition in the same order in which they have been published.

Kafka redistributes work if a consumer fails and is no longer able to process messages.



A message (or record, or event) contains metadata alongside the message payload.



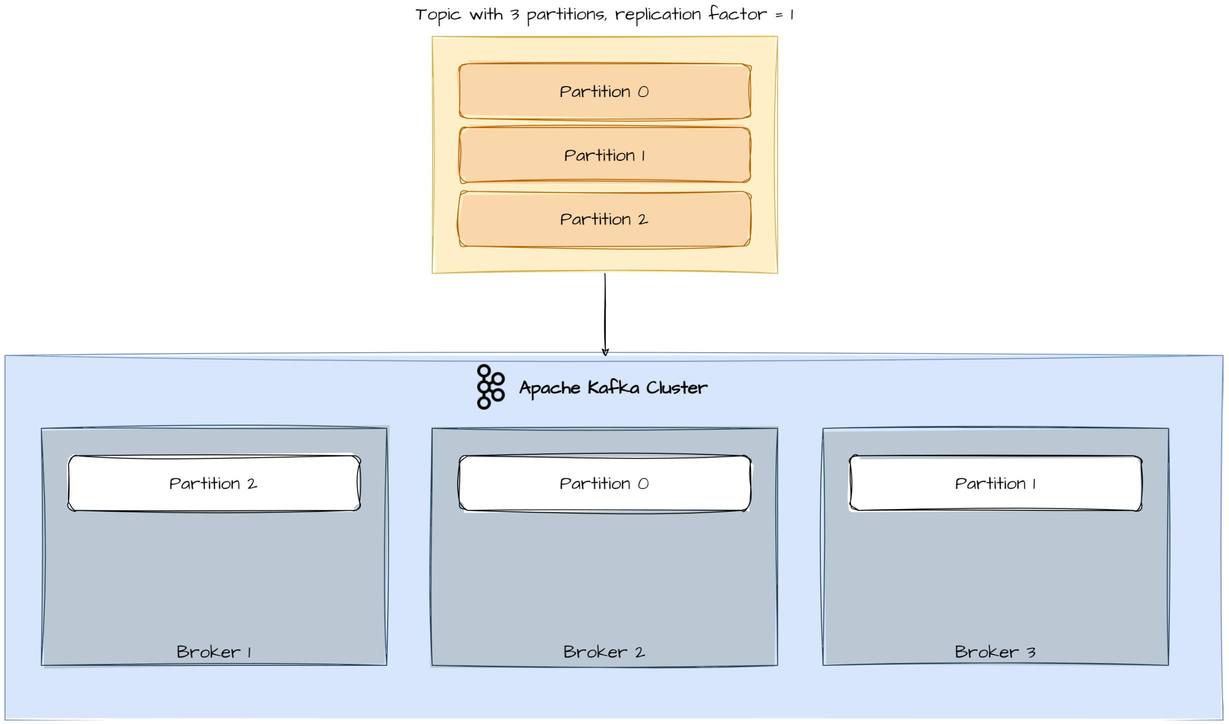
- Key-based partitioning can be used to ensure that all events that target a certain aggregate are written (and read) in-order into a topic-partition.

- Keys are hashed using a Murmur-based hashing algorithm (language agnostic).

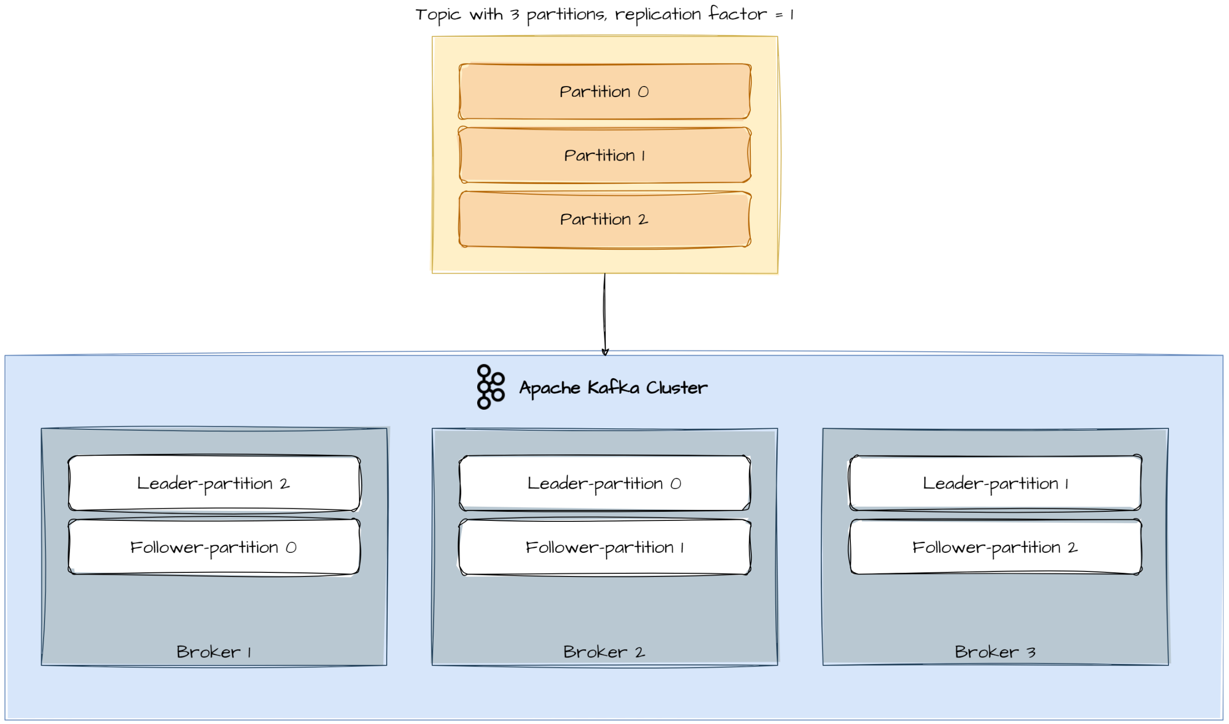
- It is possible to provide a custom partitioning strategy.

- It is possible to use the event time as timestamp. This requires the implementation of the `TimestampExtractor` interface. The event time is part of the message payload and thus must be extracted before it can be injected into the timestamp field.

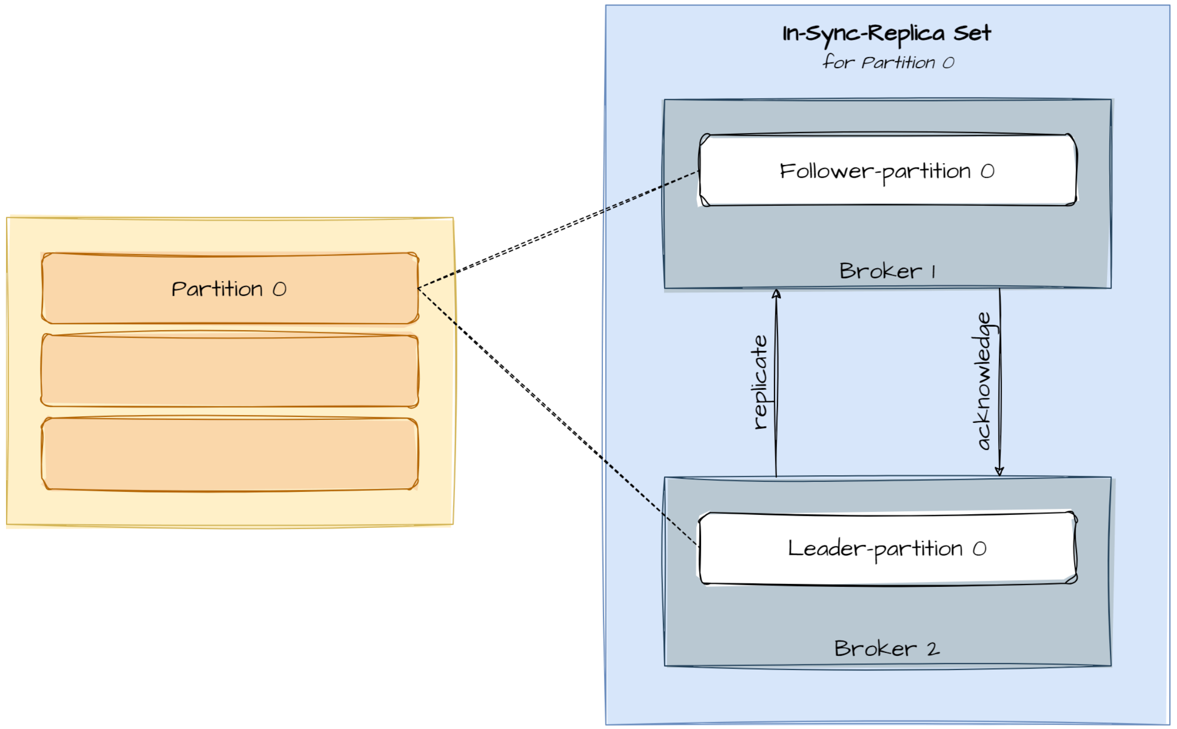
Partitions are spread across brokers and span multiple machines in a Kafka cluster.



Partitions are spread across brokers and span multiple machines in a Kafka cluster. (cont.)



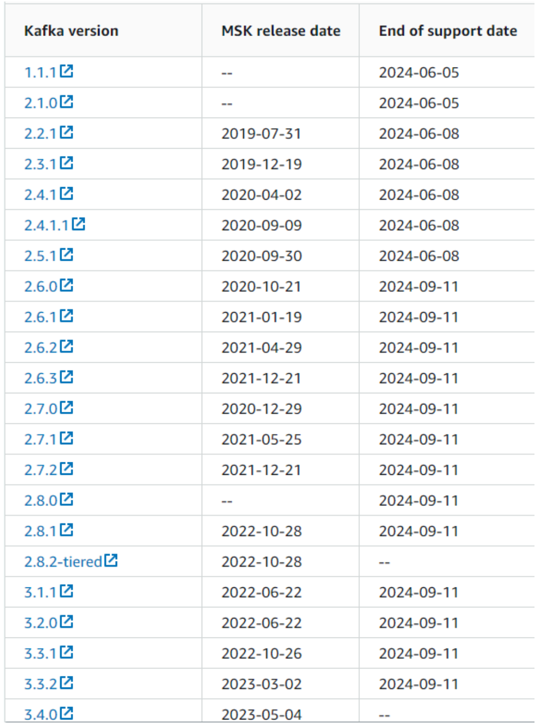
The In-Sync-Replica set contains brokers that are either a leader or follower of a partition.



Let's talk briefly about Kafka versions that you encounter in the wild.

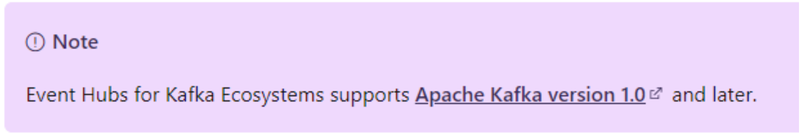
- In the wild you'll find a huge spread of Kafka versions
- Kafka Clients are usually (somewhat) compatible
- The configuration defaults are not
- Be **very** careful to use the same version in dev as in other stages

What's the situation on AWS?



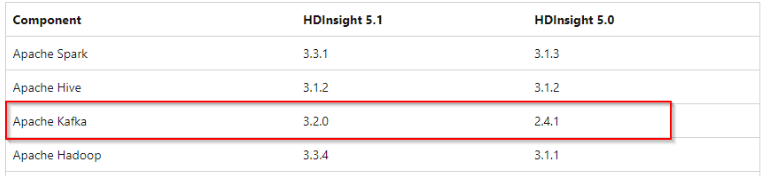
Kafka version	MSK release date	End of support date
1.1.1	--	2024-06-05
2.1.0	--	2024-06-05
2.2.1	2019-07-31	2024-06-08
2.3.1	2019-12-19	2024-06-08
2.4.1	2020-04-02	2024-06-08
2.4.1.1	2020-09-09	2024-06-08
2.5.1	2020-09-30	2024-06-08
2.6.0	2020-10-21	2024-09-11
2.6.1	2021-01-19	2024-09-11
2.6.2	2021-04-29	2024-09-11
2.6.3	2021-12-21	2024-09-11
2.7.0	2020-12-29	2024-09-11
2.7.1	2021-05-25	2024-09-11
2.7.2	2021-12-21	2024-09-11
2.8.0	--	2024-09-11
2.8.1	2022-10-28	2024-09-11
2.8.2-tiered	2022-10-28	--
3.1.1	2022-06-22	2024-09-11
3.2.0	2022-06-22	2024-09-11
3.3.1	2022-10-26	2024-09-11
3.3.2	2023-03-02	2024-09-11
3.4.0	2023-05-04	--

What's the situation on Azure?

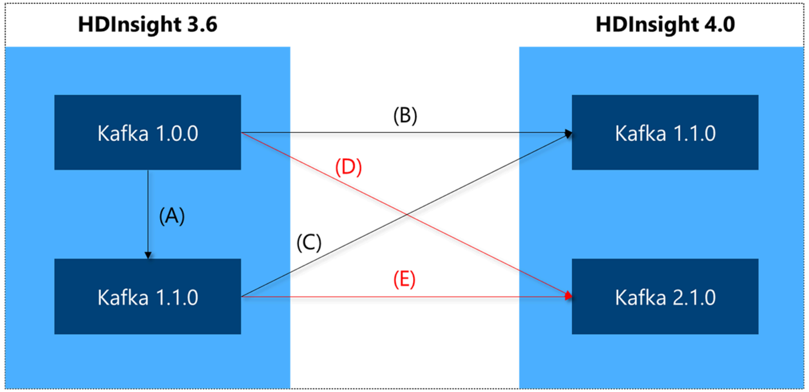


ⓘ Note

Event Hubs for Kafka Ecosystems supports Apache Kafka version 1.0 and later.



Component	HDInsight 5.1	HDInsight 5.0
Apache Spark	3.3.1	3.1.3
Apache Hive	3.1.2	3.1.2
Apache Kafka	3.2.0	2.4.1
Apache Hadoop	3.3.4	3.1.1



Questions?