**Stream Processing Examples Recap**

Stream Processing is a critical component in a number of familiar technology applications:

* Finding patterns and meaningful data in disparate log messages in a micro services architecture
* Tracking user-engagement in real time with streaming website analytics
* Real-time pricing in ride-sharing applications based on demand and environmental conditions
* Stock buying/selling based on price, news, and social media sentiment

**Batch Processing**

* Runs on a scheduled basis
* May run for a longer period of time and write results to a SQL-like store
* May analyze all historical data at once
* Typically works with mutable data and data stores

**Stream Processing**

* Runs at whatever frequency events are generated
* Typically runs quickly, updating in-memory aggregates
* Stream Processing applications may simply emit events themselves, rather than write to an event store
* Typically analyzes trends over a limited period of time due to data volume
* Typically analyzes immutable data and data stores
* *Batch and Stream processing are not mutually exclusive*. Batch systems can create events to feed into stream processing applications, and similarly, stream processing applications can be part of batch processing analyses.

**Components of a Stream Processing Solution**

Play Video

**Streaming Data Store**

* May look like a message queue, as is the case with Apache Kafka
* May look like a SQL store, as is the case with Apache Cassandra
* Responsible for holding all of the immutable event data in the system
* Provides guarantee that data is stored ordered according to the time it was produced
* Provides guarantee that data is produced to consumers in the order it was received
* Provides guarantee that the events it stores are immutable and unchangeable

**Stream Processing Application and Framework**

1. Stream Processing applications sit downstream of the data store
2. Stream Processing applications ingest real-time event data from one or more data streams
3. Stream Processing applications aggregate, join, and find differences in data from these streams
4. Common Stream Processing Application Frameworks in use today include:
   1. Confluent KSQL
   2. Kafka Streams
   3. Apache Flink
   4. Apache Samza
   5. Apache Spark Structure Streaming
   6. Faust Python Library

**Further Optional Reading on Message Queues**

* [**RabbitMQ**](https://www.rabbitmq.com/)
* [**ActiveMQ**](https://activemq.apache.org/)

**Benefits of Stream Processing**

1. Faster for scenarios where a limited set of recent data is needed
2. More scalable due to distributed nature of storage
3. Provides a useful abstraction that decouples applications from each other
4. Allows one set of data to satisfy many use-cases which may not have been predictable when the dataset was originally created
5. Built-in ability to replay events and observe exactly what occurred, and in what order, provides more opportunities to recover from error states or dig into how a particular result was arrived at

**Key concepts to remember about stream processing**

* Stream processing applications consist of a stream data store and a stream processing application framework
* Stream processing solutions do not operate on a scheduled basis
* Stream processing solutions provide real-time insights based on event data
* Stream processing solutions are built around generic data events, allowing for flexibility in data processing and highly scalable applications
* Batch and stream processing solutions can coexist and feed into each other

**Append-only logs**

* Append-only logs are text files in which incoming events are written to the end of the log as they are received.
* This simple concept -- of only ever appending, or adding, data to the end of a log file -- is what allows stream processing applications to ensure that events are ordered correctly even at high throughput and scale.
* We can take this idea a step farther, and say that in fact, streams *are* append-only logs.

**Log-structured streaming**

* Log-structured streams build upon the concept of append-only logs. One of the hallmarks of log-structured storage systems is that at their core they utilize append-only logs.
* Common characteristics of all log-structured storage systems are that they simply append data to log files on disk.
* These log files may store data indefinitely, for a specific time period, or until a specific size is reached.
* There are typically many log files on disk, and these log files are *merged* and *compacted* occasionally.
* When a log file is *merged* it means that two or more log files are joined together into one log file.
* When a log file is *compacted* it means that data from one or more files is deleted. Deletion is typically determined by the age of a record. The oldest records are removed, while the newest stay.
* Examples of real world log-structured data stores: Apache HBase, Apache Cassandra, Apache Kafka

**Further Research on Append-Only Logs**

* [**Apache Cassandra**](http://cassandra.apache.org/)
* [**Apache HBase**](https://hbase.apache.org/)
* [**Apache Kafka**](https://kafka.apache.org/)

**Log-Structured Storage**

One of the key innovations over the past decade in computing has been the emergence of log-structured storage as a primary means of storing data.

**Apache Kafka as a Stream Processing Tool**

* Kafka is one of the most popular streaming data platforms in the industry today.
* Provides an easy-to-use message queue interface on top of its append-only log-structured storage medium
* Kafka is a log of *events*
* In Kafka, an event describes something that has occurred, as opposed to a request for an action to be performed
* Kafka is distributed by default
* Fault tolerant by design, meaning it is hard to lose data if a node is suddenly lost
* Kafka scales from 1 to thousands of nodes
* Kafka provides ordering guarantees for data stored within it, meaning that the order in which data is received is the order in which data will be produced to consumers
* Commonly used data store for popular streaming tools like Apache Spark, Flink, and Samza

**Kafka History**

* Created at Linkedin to service internal stream processing needs
* Kafka is one of the Apache Foundation’s most popular projects
* Used widely in production. Some famous users include Uber, Apple, and Airbnb
* Creators of Kafka left LinkedIn to found Confluent, which now acts as the owner and leader of the Kafka project
* Jay Kreps, one of the core authors of Apache Kafka, named the system after Czech author Franz Kafka. Kreps, who enjoys Kafka’s work, thought the name was a good fit because Kafka was built to be a “system optimized for writing.”