Design doc for real time analytics for CK

Summary

This design doc lays out the different use-cases for the need of real-time analytics solution for CK followed by a listing of different requirements for the said solution. In the end, an architecture is proposed along with a listing of features for the said solution.

Use-cases

Below is a outlining of possible use-cases for real-time analytics -

- As a **operation manager** maintaining the kitchen(s), I would like a near real-time view of operational KPIs like
  > number of orders pending
  > current lag time (defined as the time difference between order received to order delivered to delivery partner)
  > orders received per minute in the trailing hour

- As a **operations manager**, I want to be alerted when
  > the orders received per minute exceeds a certain threshold or falls below a certain threshold
  > number of orders pending exceeds a certain threshold or falls below a certain threshold
  > Lag time exceeds a certain threshold
  > When the productivity of chefs falls below a certain threshold

- As a **program manager** monitoring a certain geographical area, I want to monitor operational KPIs as indicated above but at a higher aggregation (i.e. at a area level vs. at a restaurant level)

- As a **program manager** incharge of pricing, I want to monitor pricing levels for possibility of charging surge premium

Requirements

Data Freshness
The use-cases listed above are all time sensitive hence the data needs to be near real-time with a latency of less than 5 minutes.

Near real-time interactive query
The stakeholders would require the ability to write interactive queries to extract data near real-time.

Rich dimensionality
In addition to providing interactive querability, users would require rich dimensionality so they can slice and dice the data per their use-case(s).

Batch ingestion
While this solution is focused primarily on ingesting and showing near real-time data, it should support mechanism so data can be ingested for batch processing which allows for richer/enhanced datasets which can afford a time lag of 1+ day

Scalability
The solution should be able to scale appropriately so it can handle peak order times.

Fraud and anomaly detections
The solution should provide a mechanism to detect and flag possible fraudulent orders. Example, a previously flagged customer whose payment method failed previously is ordering again in higher volumes.

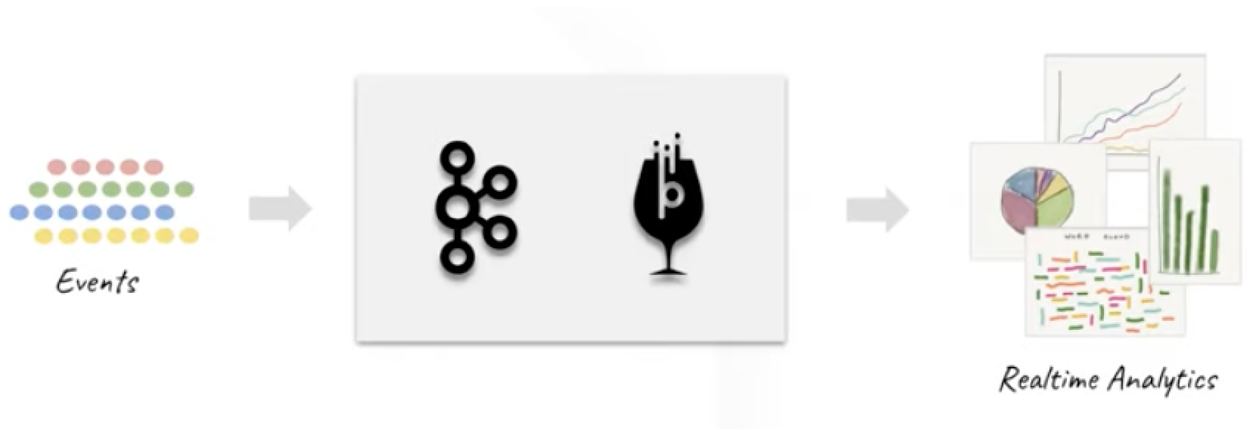
Data Uniqueness
The solution should provide the ability to perform data deduping at the lowest grain i.e. customer order level.

High volume of queries
The solution should be able to handle high volume of queries from end user(s) (like 1000 QPS)

Low latency on query results
The solution should be able to output results with low latency

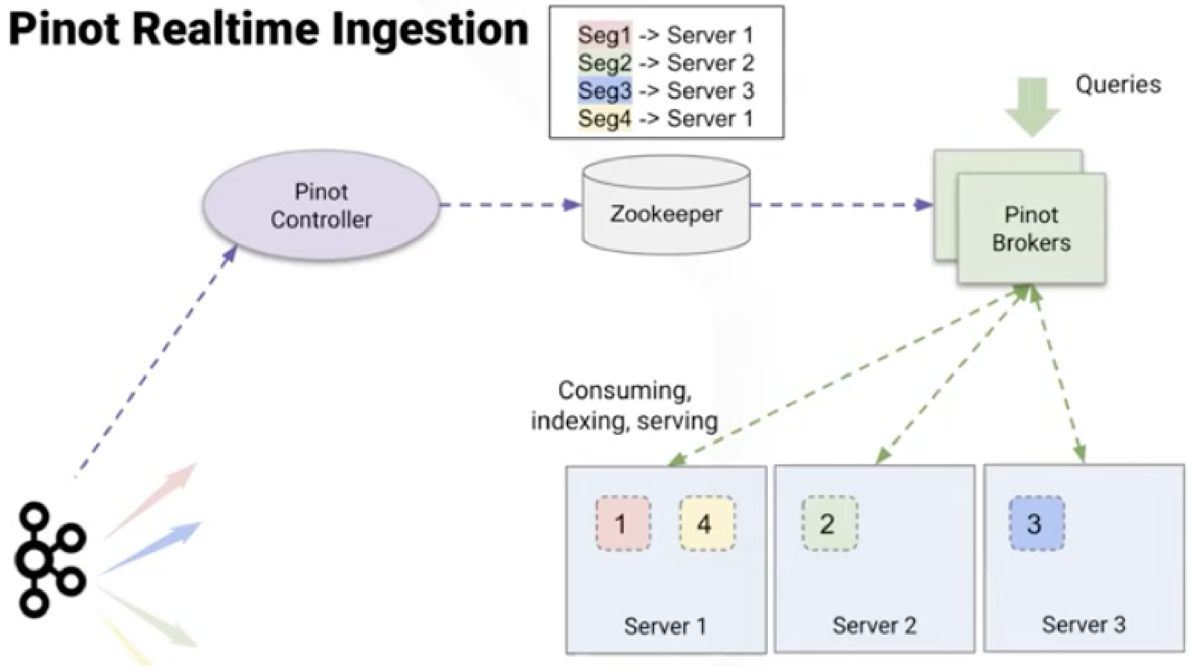
Architecture

The proposed solution includes two key components: **Apache Kafka** and **Apache Pinot**. At a high level this solution looks like below -



Kafka Producer will capture events and publish these events. Kafka is the key data infrastructure for event-streamed systems and can scale to consuming 1 million events per second. In the case of CK, incoming data is in the form of JSON format and both Kafka and Pinot can handle it efficiently (even deeply nested jsons). In fact, users can write UDF to extract the exact data element from a deeply nested json file in Pinot. With this set-up, stale data (anything over a week can be moved to S3 for archiving purposes. Hence Kafka becomes the system of record). Initiating this process is as simple as creating a topic in Kafka and starting emitting events. Within pinot, a schema and table can be defined which will consume these events (along with UDFs if any) and the user is now ready to perform real time analytics. With pinot, data in memory is instantly available for queries making for faster turnaround for key operational KPIs.

Data consumed in Pinot is handled by Pinot Controller which is a single coordinator in charge of segmenting data based on the chosen partitioning key. Pinot controller is nimble thus allowing for easy handling of changes like increase in server count for upcoming peak events.



In the case of CK, it would be prudent to choose customerID as a partitioning key since it would allow for all data about a relevant customer to stay on the same server thus allowing for complex joins and faster turnaround. In addition, Pinot has a presto connector which provides users the flexibility to join to other datasets in other applications. Pinot also supports powerful indexing methods (Sorted, inverted etc.) thus allowing for faster query performance.

This set-up is purely for real-time analytics. In addition to this set-up, I would recommend batch processing everyday during non-operational hours (from 11 pm till 6 am) to create richer, enhanced datasets for stakeholders to query the following day. Revenue metrics are a good example for an enhanced dataset.