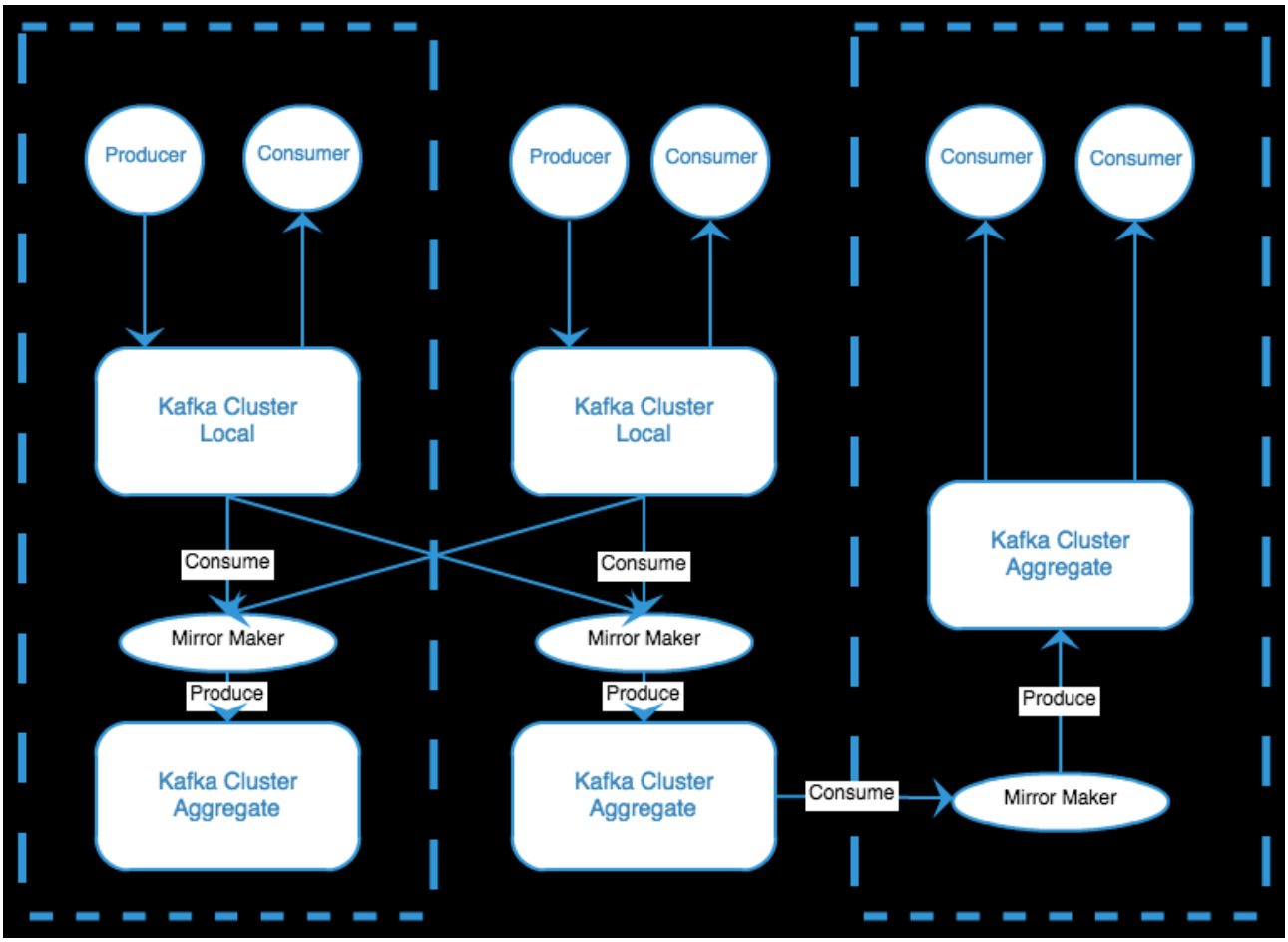
Kafka at LinkedIn

- kafka is not a DB but it does persist msgs for a configured amount of time (retention period)
- File-Based storage in brokers:
  - kafka msgs are stored on disk by kafka brokers in a distributed manner.
  - msg is written to a log file associated with the topic and partition it belongs to.
  - Each partition is an append-only log file.
  - kafka stores these logs in the filesystem using underlying disk storage.
- Retention policy:
  - time-based (retention.ms)
  - size-based (retention.bytes)
  - configurable settings at topic level.
- storage mechanics:
  - segmented logs: each partition is divided into smaller chunks called log segments to make deletion efficient.
  - index files: kafka maintains index files for quick lookup of offsets within the log segments.
  - Compaction (Optional): For topics with log compaction enabled, kafka retains only the latest message with a given key, deleting older versions.
- LinkedIn runs over 1100 brokers into more than 60 clusters.
- defines 4 categories of messages:
  - queuing
    - **Queuing** is the standard messaging type.
    - messages are produced by one part of an application and consumed by another part of that same application.
    - Other applications aren't interested in these messages,
    - because they're for coordinating the actions or state of a single system.
    - This type of message is used for sending out emails,
    - distributing data sets that are computed by another online application,
    - or coordinating with a backend component.
  - metrics
    - **Metrics** handles all measurements generated by applications in their operation.
    - It includes everything from OS and hardware statistics to application-specific measurements which are critical to ensuring the proper functioning of the system.

        - providing visibility into the status of all servers and applications, and driving our internal monitoring and alerting systems.
    - logs
        - **Logging** includes application, system, and public access logs.
        - Originally, metrics and logging coexisted on the same cluster for convenience.
        - We now keep logging data separate simply because of how much there is.
        - The logging data is produced into Kafka by applications,
        - and then read by other systems for log aggregation purposes.
    - tracking data
        - **Tracking** includes every action taken on the front lines of LinkedIn's infrastructure,
        - whether by users or applications.
        - These actions need to be communicated to other applications as well as to stream processing in Apache Samza and batch processing in Apache Hadoop.
        - This is the bread and butter of big data:
        - the information we need to keep
        - search indices up to date,
        - track usage of paid services,
        - and measure numerous growth vectors in real time.
    - All four types of messaging are critically important to the proper functioning of LinkedIn, however tracking data is the most visible as it is often seen at the executive levels and is what drives revenue.

Tiers and Aggregation

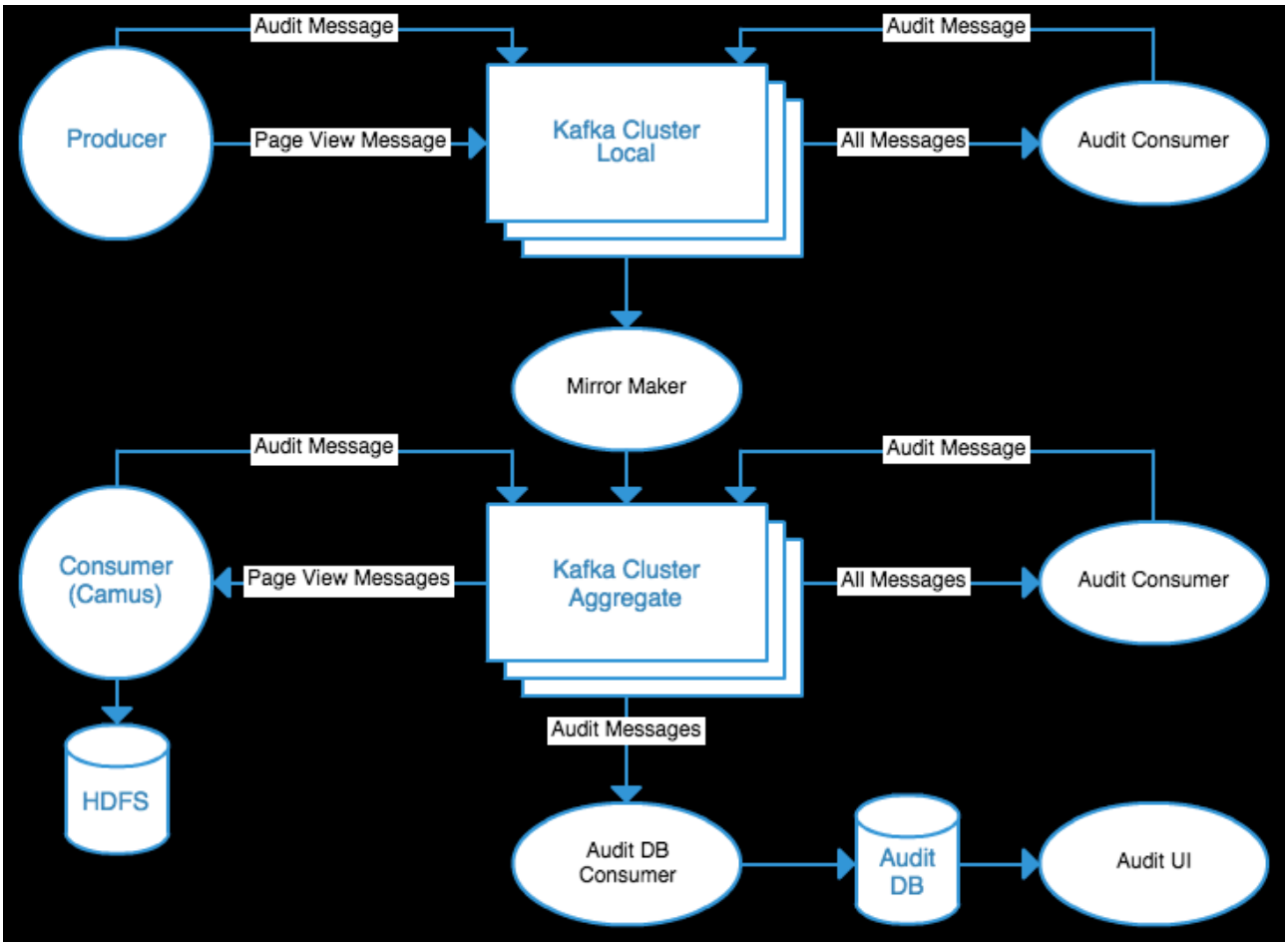
- multiple datacenters.
- Some applications, such as those serving a specific user's requests, are only concerned with what is going on in a single datacenter.
- Many other applications, such as those maintaining the indices that enable search, need a view of what is going on in all datacenters.
- for each msg category, we have cluster named "local", containing msgs created in datacenter.
- there is also an aggregate cluster, which combines msgs from all local clusters for a given category.
- use kafka mirrormaker application to copy msgs forward, from local into aggregate, this avoids any msg loops between local clusters.



- moving the data withing kafka infrastructure reduces network costs and latency by allowing to copy msgs a minimum number of times (once per datacenter).
- consumers access data locally, which simplifies their configuration and allows them to not worry about many types of cross-datacenter network problems.
- producer is the first tier.
- local cluster (Across all datacenters) is the second
- each of the aggregate clusters is an additional tier.
- consumer itself is the final tier.
- This tiered infrastructure solves many problems,
- but it greatly complicates monitoring Kafka and assuring its health.
- While a single Kafka cluster, when running normally, will not lose messages,
- the introduction of additional tiers, along with additional components such as mirror makers, creates myriad points of failure where messages can disappear.
- In addition to monitoring the Kafka clusters and their health,
- we needed to create a means to assure that all messages produced are present in each of the tiers, and make it to the critical consumers of that data.

Auditing Completeness:

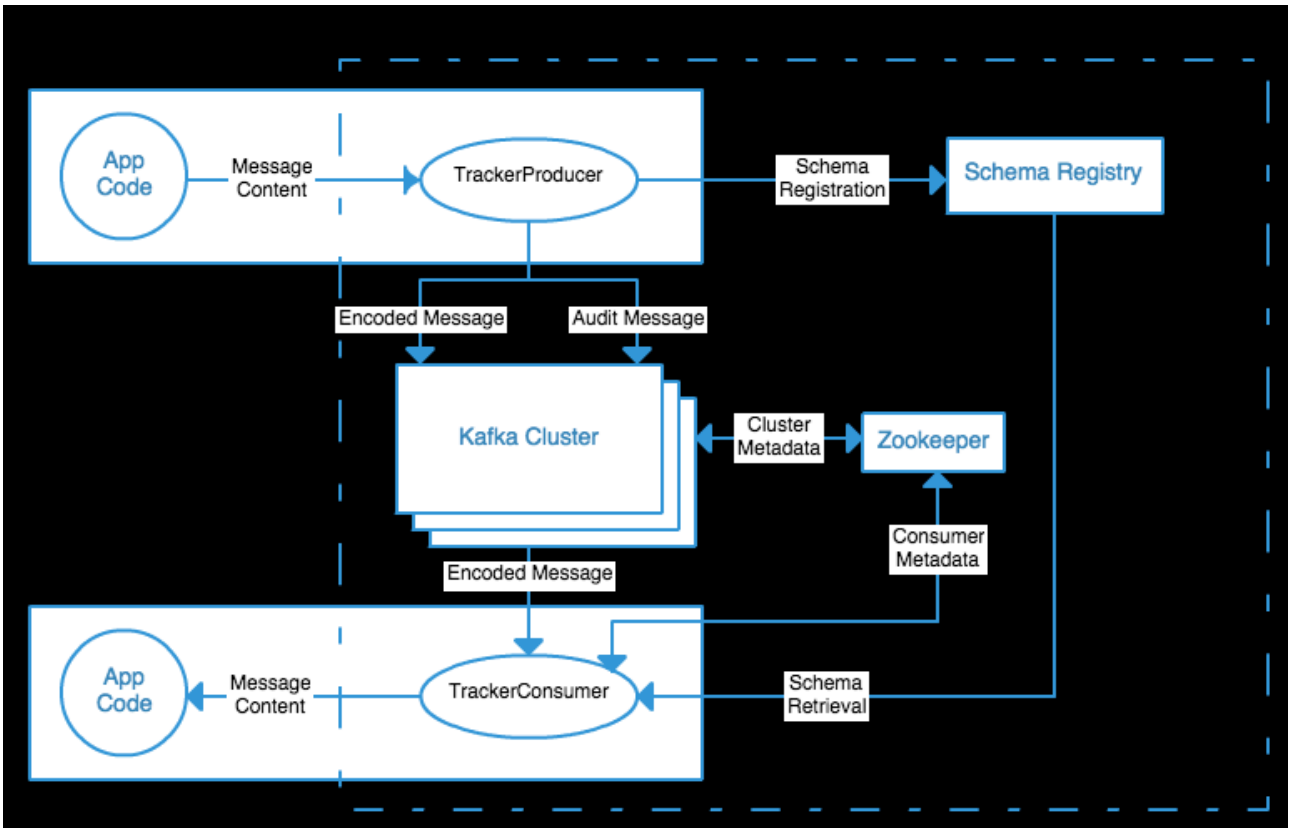
- internal tool that helps to make sure all msgs produced are copied to every tier without loss.
- msg schemas contains a header containing critical data common to every msg, such as timestamp, producing service, originating host.
- As an individual producer sends msgs into kafka, it keeps a count of how many msgs it has sent during current time interval.
- periodically, it transmits that count as a msg to a special auditing topic.
- this gives us info. about how many msgs each producer attempted to send into a specific topic.
- kafka console auditor, consumes all msgs from all topics in a single kafka cluster.
- like the producer, it sends msgs into auditing topic stating how many msgs it consumed from that cluster for each topic for the last time interval.
- by comparing counts to producer counts, we can determine that all msgs produced actually got to kafka.
- if number differs then we know producer is having problems and we can trace that back to specific service and host that is failing.
- each cluster has it own console auditor that verifies its msgs.
- by comparing each tier's counts against each other, we can assure every tier has same number of msgs present.
- this ensures no loss, no duplication.



- Certain critical consumers of messages, such as the Hadoop grids, also write back auditing information as a separate tier.
- This allows us to not only monitor to make sure the producers are all working and that Kafka is passing messages, but also validates that the consumer is receiving all of those messages.
- Should there be a problem with the application copying messages from Kafka to Hadoop, it will show up in the Kafka Audit tool as an error specific to the tier names that Hadoop uses.
- This final piece provides us with end-to-end assurance that every message produced was ultimately consumed.

Bringing all together:

- almost all producers of Kafka messages within LinkedIn use a library called `TrackerProducer`.
- When the application calls it to send a message, it takes care of inserting message header fields, schema registration, as well as tracking and sending the auditing messages.
- Likewise, the consumer library takes care of fetching schemas from the registry and deserializing Avro messages.

- The majority of the Kafka infrastructure applications, such as the console auditor, are also maintained by the development team.
- 

-