Welcome & Introduction
- Greeting and welcome to Community Over Code.
- Speaker: Amrit Sarkar, Senior Software Engineer, Apple (AI/ML domain).
- Experience: 8+ years in Kubernetes & Search use-cases, 18 months in streaming tech (Kafka, Spark, Flink).

Session Overview
- Focused on Apache Kafka cluster monitoring from an ops lens.
- Aiming to provide a holistic view for both new and experienced administrators/users.
- Initial discussion: Brief overview of Kafka's internal workings.
- Importance: Understanding core fundamentals for effective issue troubleshooting.

Apache Kafka: A Primer
- Definition: A distributed publish-subscribe messaging system.
- Key Features: High data volume handling, persistent messaging, suitable for offline and online consumption, low-latency delivery, high availability, fault tolerance.
- Use Cases: Monitoring data, aggregating stats, centralized log collection across an organization.

Key Components & Concepts in Kafka
- **Producers**: Write data/messages to Kafka clusters.
- **Kafka Cluster**: Consists of Kafka servers (brokers); messages are persisted in a distributed and replicated manner.
- **Kafka Topics**: Logical queues where messages are written, which can be partitioned for concurrency.
- **Consumers**: Subscribe to topics to read messages.
- **Zookeeper**: Used for cluster state management and coordination in versions until Kafka 3.x (deprecated in newer versions).

In-Depth Kafka Mechanics
- **Kafka Topics and Partitions**:
  - Topics can be partitioned to facilitate concurrent data handling.
  - Each partition can be consumed by one or more consumer groups.
  - Each message entity in a partition is referred to as a Record.
- **Consumer Groups and Partitions**:
  - Comprised of multiple consumers, each tasked with reading records from one or more partitions.
  - Goal: Read records from all partitions of a topic.
- **Record Offsets**:
  - Each record is associated with a unique integer offset within its partition.
  - Consumers track offsets to read the next record sequentially.

Rebalancing in Kafka
- **Rebalancing**:
  - Activity of redistributing partitions among consumers.
  - Triggered when consumers are added or removed.
- **Consumer Group Leader**:
  - Elected by the Group Coordinator.
  - Responsible for rebalancing.
- **Heartbeats**: Sent by a Group Coordinator to check consumer status within a group.

Broker and Topic Configuration
- **Brokers**: Run on servers, offering durability and high availability to topics.

- **Replication**:
  - Topics are divided and replicated across multiple brokers.
  - Each partition has one Leader Replica responsible for read/write operations.
  - Messages written to a leader are propagated to follower replicas.

Discussion Points
- Exploring Kafka performance areas and metrics classification.
- Insight into consumer lag as a critical performance metric.
- Derivation of discussed factors based on Kafka observation for streaming services over recent years.
- Open invitation for insights from experienced Kafka practitioners post-presentation.

Session Expectation
- Although the audience may be familiar with certain aspects, a foundational coverage ensures comprehensive understanding for subsequent discussions.
- Engaging in exploring monitoring options and adopting an observability mindset for Kafka.
- Delving into performance metrics and deriving actionable insights from them for effective Kafka cluster management.

Conclusion
- Interactive Q&A and knowledge sharing invited at the end of the session.

Talking Bullet Points for Concise Presentation on Kafka

Controller Broker in Kafka
- Assumes administrative role, handling create, update, and delete topic requests.
- Processes reassignments and tracks replicas' states for in-sync partition follower replicas.
- Critical for managing Partition Leadership.

KRaft Protocol and Zookeeper Phasing Out
- KRaft Protocol has been implemented, diminishing Zookeeper's role in cluster management.
- Promotes Kafka's evolution and maintenance in open-source technology.

Performance Optimization Areas for Kafka Pipelines
- Addressing Throughput and Latency is crucial for distributed systems like Kafka.
- Focus on optimizing production and consumption rates.

Producer-Consumer Synchronization
- Importance of aligning producer throughput and consumer read/process capacity.
- "Customer’s Lag" denotes misalignment between producer and consumer, should be minimized.

Ensuring Data Integrity in Kafka
- Kafka allows tunability in write/read durability, influencing throughput and latencies.
- Acknowledgment levels influence write latencies and data durability.

Kafka Cluster and Broker Distribution
- Adequate broker distribution across servers, racks, or regions ensures stability.
- Considerations needed in formulating cluster state configurations for different environments.

Resource Utilization Insights

- Balancing utilization of CPUs, Memory, and other resources is essential.
- Ensure that storage and network components are optimal to avoid potential issues.

Going Beyond Basic Monitoring to Observe Systems
- Traditional monitoring might be insufficient for understanding complex application failures.
- Building observability frameworks helps comprehend system’s internal state, identifying faults and causes.

Kafka’s Metrics Reporting
- Kafka reports metrics, not problems; the Active Controller Count should be 1.
- Good observability helps identify root causes of issues, like server restarts or network issues.

Kafka Cluster Components as Performance Gates
- All components (producers, brokers, etc.) affect the system's performance.
- Each component requires meticulous monitoring and management.

Internal Components and Partition Management in Kafka
- Keep a vigilant eye on topic health within the brokers.
- Ensure a stable distribution of partitions across brokers, considering server resources.

Kafka Message Distribution
- Messages are usually evenly distributed across partitions, but custom routing can create imbalances.
- Monitoring partition load distribution is necessary to avoid instability.

Tracking Consumer Rate and Lag
- Ensuring the consumer rate is tracked and managing lags is vital for system performance.
- Lag is a crucial metric in evaluating Kafka cluster performance.

Evaluating Elements Binding the System
- Monitor network latencies, CPU, memory, I/O spikes, etc., as they contribute to the whole system's performance.

Time-Series Data Visualization and Metric Actions
- Kafka and infrastructure metrics can be fed into a time-series format for visualization (e.g., using Prometheus and Grafana).
- Zookeeper can expose metrics using its exporter if still in use.

Kafka’s Open-Source Ecosystem
- Rich set of administrative, monitoring, alerting, and observability frameworks available.
- Includes popular tools like Confluent Control Center, KafDrop, and Yahoo Kafka Manager.

Producer Rate and Data Push to Kafka Cluster
- Different methods to push data, such as the `send()` function in Kafka Java client.
- Attention to batch size, compression formats, and ensuring balanced garbage collection activity.

Monitoring Key Producer Client Metrics
- Track batch-size-avg, compression rate average, and request-latency-avg.
- Ensure high throughput, low compression ratio, and optimal request latency.

Alert-Triggering Metrics

- ACC (ActiveControllerCount): Should always be 1.
- OfflinePartitionsCount: Should always be 0.
- UnderMinIsrPartitionCount: Should always be 0.

Consumer Lag and Observability
- Monitoring Consumer Lag provides insight into Kafka stream performance.
- Employ visualization tools like Grafana and Alertmanager to track and get notified about metric statuses.

Final Note
- Metrics in Kafka offer insights into the internal working of the clusters.
- Continuous monitoring and taking appropriate actions based on metrics are key to maintaining healthy Kafka clusters.

Bullet Point Summary:

Broker's Health Metrics and Load Distribution
- Distribute partitions of different topics fairly among brokers.
- Use the JMX metric `PartitionCount` to monitor partition distribution.
- Ensure balanced load to prevent broker overloads.

Network Behavior Monitoring
- Analyze network for successful and erroneous byte in/out.
- Track `Network Requests Per Sec` and `Network Error per Sec` in time-series format.

Broker Log File Management
- Ensure log files are flushed to disk efficiently to maintain throughput.
- Monitor log flush latency with a dedicated JMX metric.

Monitoring Follower Replica Lag
- Aggregate follower replica lag using `FetcherLag` metric.
- Ensure replicas are in sync within partitions.
- Visualize metrics to identify issues like load skewness among brokers.

Consumer Offsets Management
- Maintain two offsets: the current offset and the committed offset.
- Offsets are vital to resume consumption after consumer failures or rebalances.
- Carefully manage the frequency of offset commits to balance reliability and performance.

Consumer Groups and Offsets
- Track current and committed offsets for different consumer groups.
- Use a dedicated Kafka topic, `__consumer_offsets`, to store offset data.

Consumer Offsets Alerting and Best Practices
- Employ judicious offset commit strategies to avoid excessive load.
- Utilize auto-commit features and asynchronous commits judiciously.

Utilizing Monitoring Tools
- Utilize tools like Burrow for consumer lag checking without specifying thresholds.
- Metrics, such as Consumer Offset Lag, can be exposed to Prometheus.

Burrow's Capabilities
- Monitors all consumers using Kafka-committed offsets.
- Registers consumer groups, reads committed offsets, and makes metrics available.
- Provides HTTP endpoints and configurable notifiers for status updates.

Understanding Lag in Kafka
- Lag is the difference between the head offset and the consumer’s offset.
- Trends of lag are evaluated on a time-series format.

Evaluating Consumer Status and Potential Issues
- Monitor consumer offset and lag to identify issues and potential data loss risks.
- Identify stalled consumers and recognize potential reasons, such as server crashes or network issues.

Analyzing Consumer Lag Spikes and Offset Differences
- Spikes in consumer lag may result in false alarms.
- Calculating time-based lag can offer a clearer understanding of consumer performance.

Time-Based Lag Analysis
- Converting absolute offset lags to time units enables unified analysis across different Kafka pipelines.
- Aids in meeting SLAs and allows for load testing and performance comparison.

Analyzing and Anticipating Data Trends
- Evaluate topic growth rate and producer spikes to anticipate scaling needs.
- Analyzing trends helps predict future traffic and performance requirements.

Infrastructure Metrics and Monitoring
- Although not discussed in detail, infrastructure metrics (CPU, memory, storage, etc.) are crucial.
- Monitoring both Kafka functional and infrastructure metrics is vital to avoid performance issues and ensure optimal operation.