Chronicle Queue vs Kafka - Detailed Comparison

This document provides a detailed comparison between Chronicle Queue and Apache Kafka — two distinct technologies designed for different messaging and event streaming use cases. Chronicle Queue focuses on ultra-low-latency, file-based communication, whereas Kafka provides distributed, fault-tolerant, scalable event streaming.

Feature	Chronicle Queue	Apache Kafka
Type	Embedded, append-only message queue	Distributed, pub-sub streaming platform
Primary Use Case	Low-latency inter-thread/process communication	Durable, scalable data streaming
Architecture	File-based queue (memory-mapped files)	Broker-based cluster (distributed commit log)
Persistence	Writes to disk (memory-mapped)	Replicated across brokers
Throughput	Millions of msgs/sec, <10µs latency	Hundreds of thousands msgs/sec, ms latency
Scalability	Limited to single machine or shared file system	Horizontally scalable
Reliability	File durability, manual replication	Built-in replication and fault tolerance
Language Integration	Java-native (JVM only)	Cross-language (Java, Python, Go, etc.)
Message Ordering	Strictly ordered	Ordered per partition
Best for	Trading, real-time local systems	Distributed microservices, analytics

Architecture Comparison

Chronicle Queue: An append-only log structure built on memory-mapped files for near-zero latency. Writers and readers directly access the same file without any broker or serialization overhead.

Apache Kafka: A distributed commit log where producers send messages to brokers, which store them across partitions and replicas. Consumers subscribe to topics and read data over the network with configurable delivery semantics.

Performance Snapshot

Metric	Chronicle Queue	Kafka
Latency (99th percentile)	1–10 µs	2–10 ms
Throughput (single node)	~5M msgs/sec	~500K msgs/sec
Jitter	Very low (deterministic)	Moderate (network & GC overhead)
CPU Overhead	Very low	Moderate

When to Use Which

Use Chronicle Queue if:
- You need ultra-low-latency (<10µs) communication.
- The system runs on a single host or shared disk.
- Deterministic performance is critical (e.g., trading engines).

Use Kafka if:
- You need distributed scalability and reliability.
- Producers/consumers are across different nodes or services.
- You need long-term event retention, replication, and recovery.