A comparison between Streaming Apps

Seshagiri Sriram

The core

Feature	Kafka Streams	Apache Flink	Apache Spark Structured Streaming	Apache Samza	AWS Kinesis
Latency	Low (ms-level)	Very Low (sub-ms)	Moderate (seconds, micro-batch)	Low	Low to Moderate
Processing Model	Event-by-event	True stream (event-by-event)	Micro-batch	Event-by-event	Micro-batch or event-based
Stateful Processing	Yes (local state stores)	Yes (robust state mgmt)	Yes (via checkpointing)	Yes	Limited
Fault Tolerance	Kafka-based recovery	Checkpointing + state backend	Checkpointing	Checkpointing	Built-in
Scalability	Moderate (single JVM)	High (clustered)	High (clustered)	Moderate	High
Ease of Use	Simple for Kafka users	Steep learning curve	Easy with DataFrames/Datasets	Moderate	Easy via AWS Console/API
Integration Ecosystem	Kafka-native	Rich connectors	Spark ecosystem	Kafka-native	AWS-native
Deployment Model	Embedded in app	Standalone cluster	Clustered (Spark jobs)	Clustered	Fully managed (serverless)

Kafka Streams

Use When:

You already use Kafka and need lightweight, embedded stream processing.

Best For:

- Real-time transformations and aggregations
- Enriching Kafka topics with minimal overhead
- Stateless or simple stateful apps (e.g., counters, filters)

Apache Flink

Use When:

You need ultra-low latency, complex event processing, or dynamic windowing.

Best For:

- Fraud detection, anomaly detection
- Stateful stream joins and CEP (Complex Event Processing)
- Real-time dashboards with fine-grained control

Apache Spark

Use When:

You already use Spark and want unified batch + streaming with SQL-like APIs.

Best For:

- ETL pipelines with streaming ingestion
- Streaming analytics with high throughput
- Data lake ingestion and enrichment

Apache Samza

Use When:

You need tight integration with Kafka and YARN, and prefer JVM-based apps.

Best For:

- Stateful stream processing with durable state
- Applications embedded in existing Kafka/YARN ecosystems
- Legacy systems with Samza infrastructure

Thank you!