inceptez

Web: Inceptez.com Mail: info@inceptez.com Call: 7871299810, 7871299817

Databricks – Structure Streaming

Spark Structure Streaming

- Apache Spark Structured Streaming is a near real-time stream processing engine
- Provides end-to-end fault tolerance with exactly-once processing guarantees
- Uses familiar Spark APIs (DataFrame / SQL) for both batch and streaming
- Treats streaming data as an unbounded table and processes data incrementally
- Continuously updates results as new data arrives

Read from a Data Stream

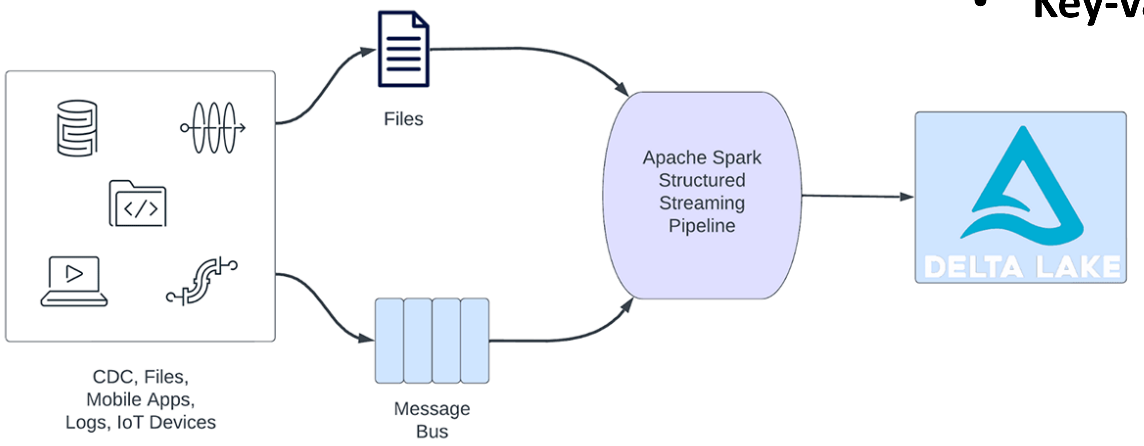
Structured Streaming supports incremental ingestion from:

- **Cloud object storage** (via Auto Loader)
- **Message queues / buses** (Kafka, Event Hubs)
- **Delta Lake tables**

Write to a Data Sink

Common streaming sinks in Databricks:

- **Delta Lake**
- **Message queues**
- **Key-value databases**



File Based Source - Autoloader

- Databricks Auto Loader enables incremental and efficient ingestion of new files as they arrive in cloud object storage
- Built on Spark Structured Streaming
- Requires minimal setup and supports exactly-once processing

How Auto Loader Works

- Uses the **cloudFiles** streaming source
- Automatically detects and processes **new (and optionally existing) files**
- Scales to **billions of files** and **millions of files per hour**

Key Benefits

- Tracks ingestion using **file metadata stored in checkpoint (RocksDB)**
- Automatically resumes from failures
- No manual state management required
- Guarantees **exactly-once writes** to Delta Lake
- Schema inference & evolution (add/modify columns)

Supported Sources & Formats

Cloud Storage

- Amazon S3 (`s3://`)
- Azure Data Lake Storage (`abfss://`)
- Google Cloud Storage (`gs://`)
- Azure Blob Storage (`wasbs://`)
- Databricks File System (`dbfs:/`)

File Formats

- JSON, CSV, XML
- Parquet, Avro, ORC
- Text, Binary

Spark Structure Streaming - Sources

Delta Lake Source - Delta Table as a Streaming Source

- Read **new commits** from a Delta table incrementally.

Syntax: spark.readStream.format("delta").table("bronze_db.events")

Kafka Source

- Native Spark connector for real-time messaging systems.

Syntax:
```
spark.readStream \
  .format("kafka") \
  .option("kafka.bootstrap.servers", "host:9092") \
  .option("subscribe", "events") \
  .load()
```

Event Hub

- Native event hub connector to read from azure event-hub

Syntax:
```
spark.readStream \
  .format(“eventhubs") \
  .option(**ehconf)
  .load()
```

Rate Source (Testing / Demo Only)

- Synthetic data generator - Generates rows at a fixed rate.

Syntax:
```
spark.readStream \
  .format("rate") \
  .option("rowsPerSecond", 10) \
  .load()
```

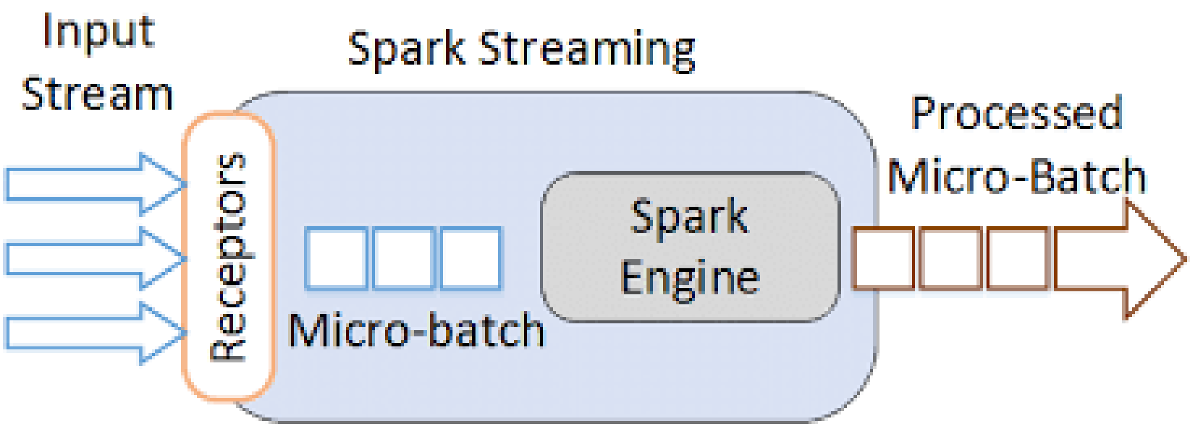
Micro-Batch

Structured Streaming processes data in **small batches** at regular intervals.

Each micro-batch does:

- Discover new data
- Read a bounded chunk
- Apply transformations
- Commit results
- Update checkpoint

Micro-batch size = how much data is read per trigger



checkpointLocation — Streaming Progress & Exactly-Once

- Tracks which files are already processed
- Stores micro-batch progress (offsets, commit logs)
- Enables fault tolerance and exactly-once ingestion

schemaLocation — Schema Tracking & Evolution

- Stores inferred schema
- Tracks schema evolution (new columns, type changes when allowed)
- Avoids re-inferring schema on every restart

Spark Structure Streaming - Output Modes

Append Mode

- Writes **only the newly added rows** in the result table since the last trigger.
- Once data is written, it is **never modified or removed**.
- **Most efficient and commonly used** mode in production pipelines.

Update Mode

- Writes **only the rows that have changed** since the previous trigger.
- Primarily used for **aggregation queries**, where intermediate results are updated as new data arrives.
- If no aggregations are involved, Update mode behaves similarly to Append mode.
- Suitable for **real-time dashboards and incremental metrics**.

Complete Mode

- Writes the **entire result table** to the external sink on every trigger.
- Used when the full result set must be **recomputed and rewritten** each time.
- The way data is written (overwrite, replace, etc.) depends on the **capabilities of the sink connector**.
- Best suited for **small datasets, debugging, or learning scenarios** due to higher computational cost.

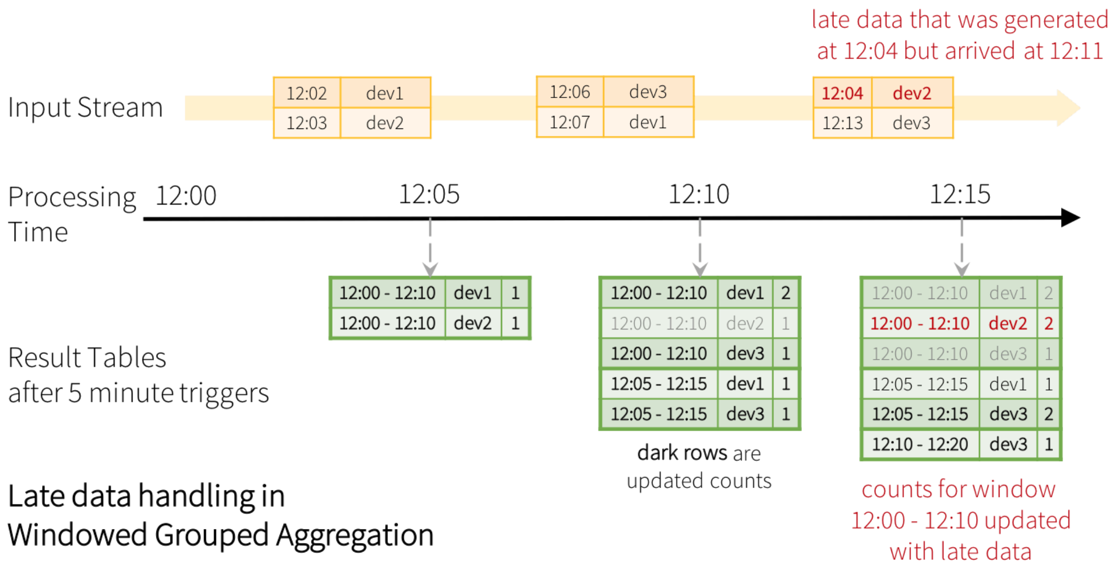
Time-Based Aggregation and Watermarking

It groups streaming records into **time windows** based on a **timestamp column** and applies aggregations such as:

- count
- sum
- avg
- min / max

Example:

- Orders **per hour**
- Errors **per 5 minutes**
- Sessions **per day**



Window Types

- Tumbling Window (Fixed, Non-Overlapping)
  groupBy(window("event_time", "10 minutes"))
- Sliding Window (Overlapping)
  groupBy(window("event_time", "10 minutes", "5 minutes"))

Watermarking

- Defines how late event-time data is allowed to arrive
- Used with time-based aggregations (window / session window
- Based on event time, not processing time
- Late data within watermark updates aggregates
- Data older than watermark is dropped

Spark Structure Streaming - Triggers

Default Trigger (Micro-batch – As Fast As Possible)

- Spark processes data as soon as possible
- Next batch starts immediately after the previous one finishes
- No fixed interval

syntax: df.writeStream.format(“delta”).start()

AvailableNow Trigger (Databricks-Optimized Once Trigger)

- Processes **all available data** and exits
- Optimized for **Auto Loader and Delta sources**

Syntax: df.writeStream.trigger(availableNow=True).start()

Processing Time Trigger (Fixed Interval Micro-batch)

- Spark runs a micro-batch at a fixed time interval
- If processing takes longer than the interval, Spark skips waiting and runs continuously

syntax: **df.writeStream.trigger(processingTime="30 seconds").start()**

Continuous Trigger (Low-latency Streaming)

- Processes data continuously, not in micro-batches
- Achieves sub-second latency

Syntax: df.writeStream.trigger(continous=“1 second”).start()

Apache Kafka

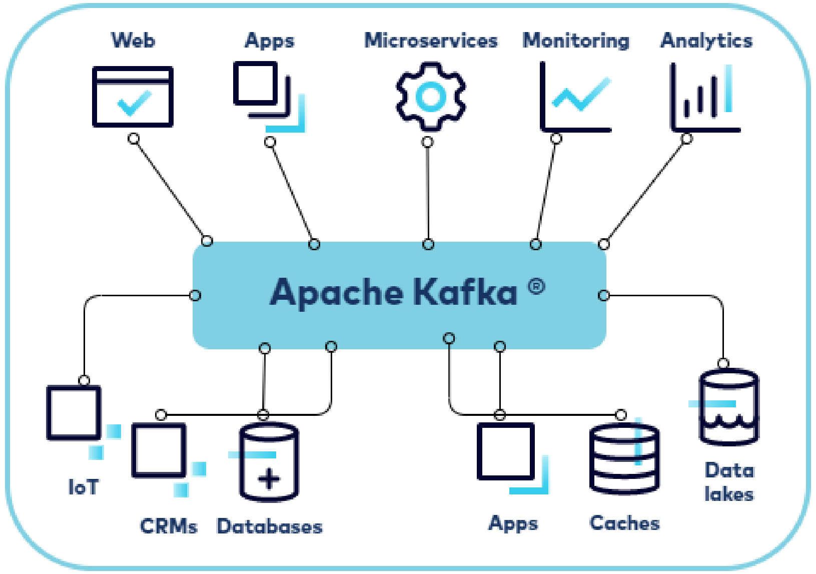
Kafka is a leading general purpose publish-subscribe distributed messaging system, which offers strong durability, scalability and fault-tolerance support.

- Distributed **event streaming platform**
- Designed for **high-throughput, low-latency** data pipelines
- Stores data as **immutable events (logs)**
- Originally developed by **LinkedIn**, now open-source
- Commonly used for **real-time data streaming**
- Horizontally scalable and fault-tolerant

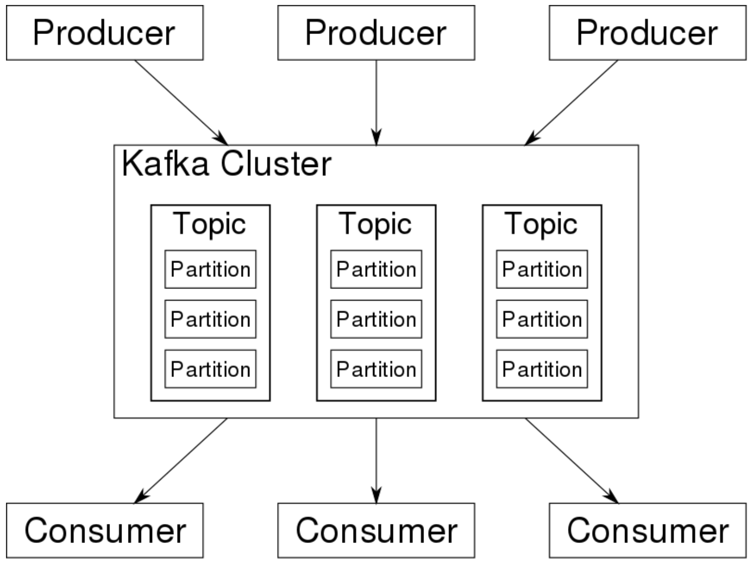
Why Kafka

- Handles millions of events per second
- Scales horizontally by adding partitions
- Designed for real-time workloads
- Producers and consumers are loosely coupled
- Kafka retains data for a configured time

Feature	Kafka	Traditional MQ
Throughput	Very High	Moderate
Data Retention	Yes	No (usually)
Replay	Yes	No
Scalability	Horizontal	Limited
Real-time analytics	Excellent	Limited



Kafka Components



Messages :

- The unit of data within Kafka with optional byte header, eg. Record in a table.

Producer:

- Producers push data to brokers.

Consumer:

- Consumer reads or consumes messages from the Kafka cluster.
- Keep track of offset of last consumed messages

Broker:

- A single Kafka server
- Broker receives messages from producers, assigns offsets to them, and commits the messages to storage on disk.
- The broker serves consumer responding with the messages based on the partitions.

Topic :

- Category of messages eg. Table/Folder
- Spread across all Brokers/nodes.

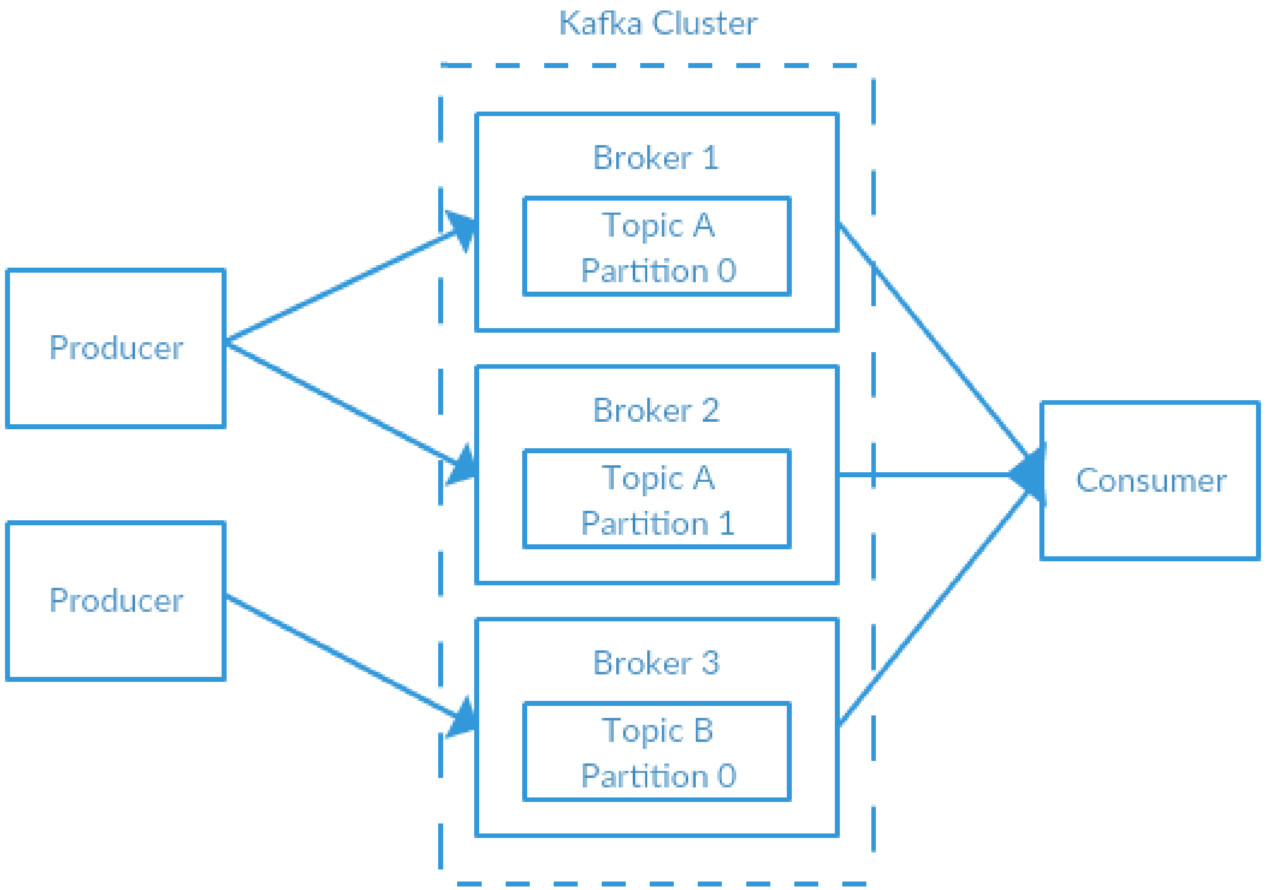
Partition:

- Horizontal division of topics.
- Scaled across multiple broker nodes.

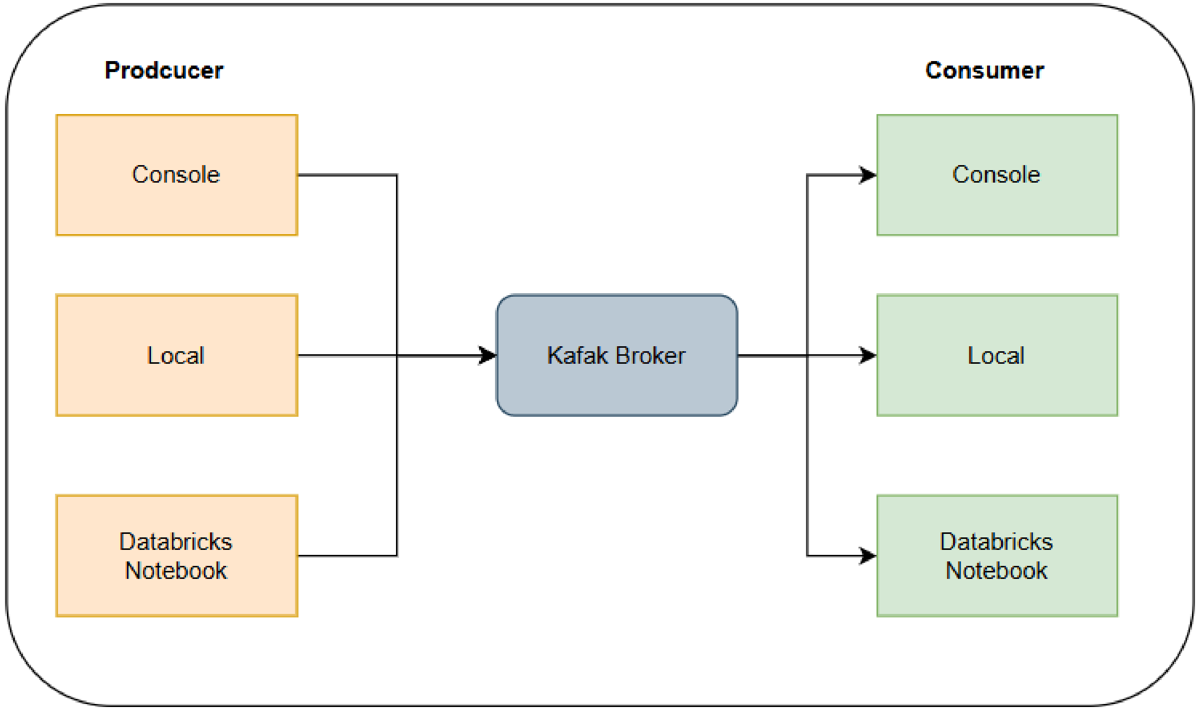
Kafka Cluster:

- Group of Brokers
- One Broker will act as a cluster controller and assigned leader for the partition.

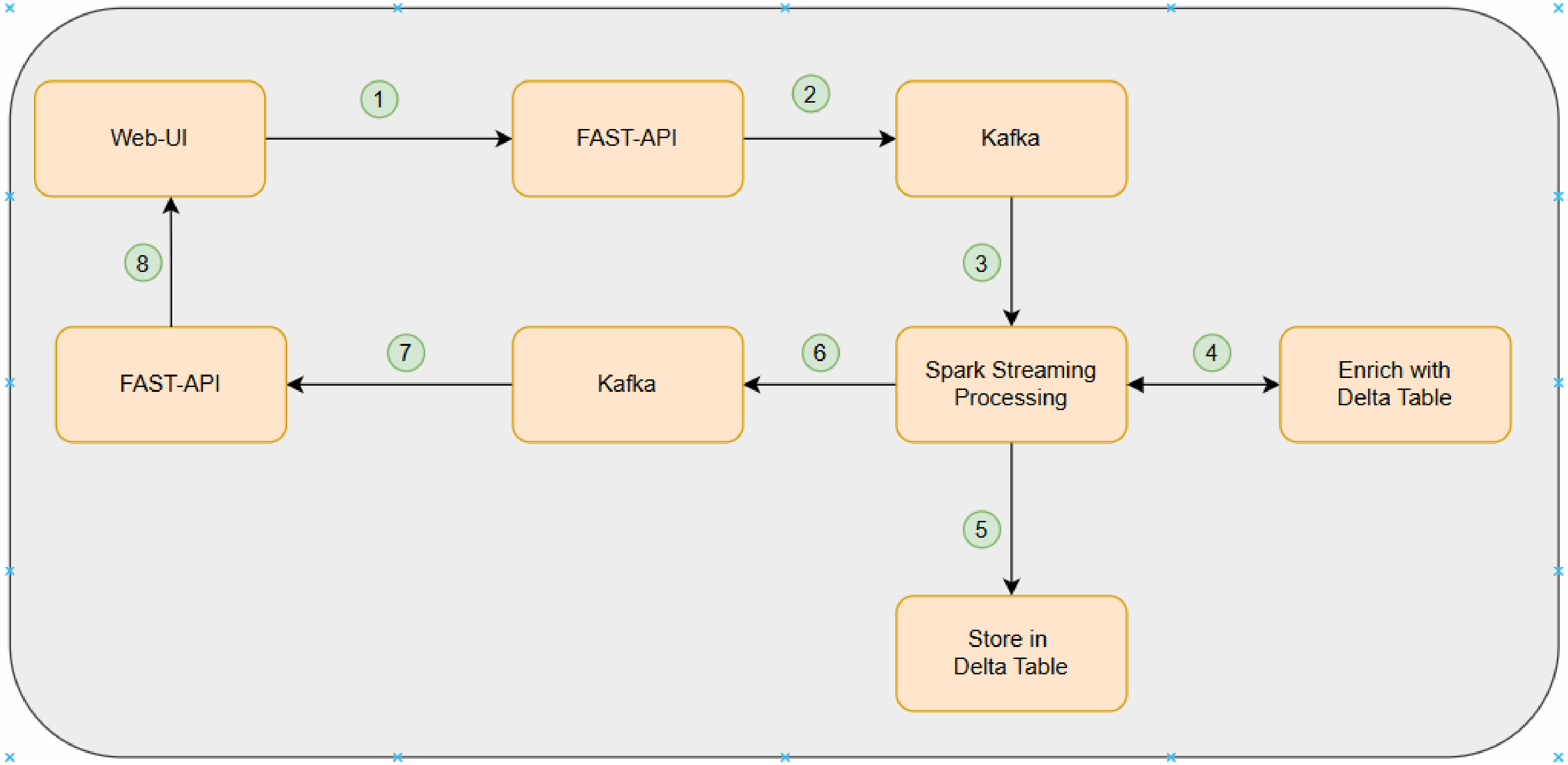
Kafka Architecture



Kafka Architecture



Use case – Realtime Claim Data Processing



Kafka Installation & Workouts

Azure EventHubs

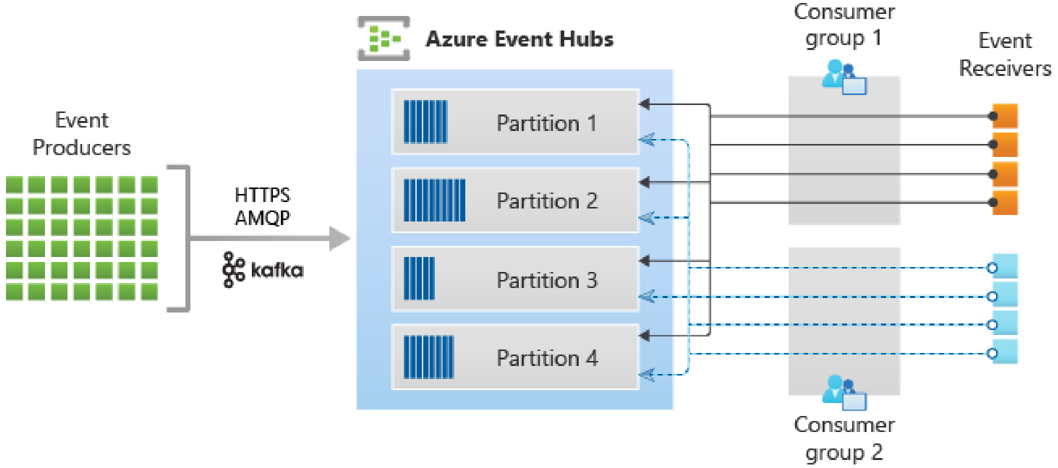
Azure Event Hubs is a **fully managed, cloud-native event streaming platform** provided by Microsoft Azure.

It is designed to **ingest, process, and stream massive volumes of real-time data** with very low latency.

In simple terms: **Azure Event Hubs is Azure's managed alternative to Apache Kafka for real-time data ingestion.**

Azure Event Hubs acts as a **central event ingestion service** where:

- Producers send millions of events per second
- Event Hubs stores them temporarily
- Consumers read and process events independently



Core Concepts of Azure Event Hubs

Concept	Description
Event Hub Namespace	Logical container for Event Hubs
Event Hub	Stream of events (similar to Kafka topic)
Partition	Ordered sequence of events
Producer	Application sending events
Consumer	Application reading events
Consumer Group	Logical view of the event stream
Offset	Position of an event within a partition