TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

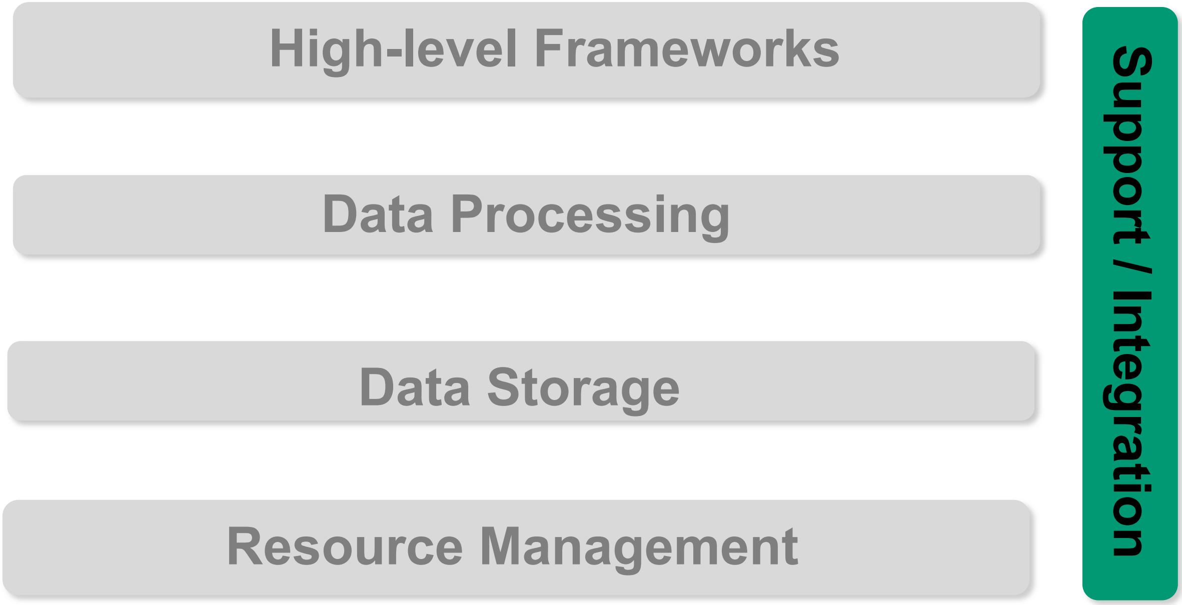
Macroarea di Ingegneria
Dipartimento di Ingegneria Civile e Ingegneria Informatica

Data Acquisition and Ingestion

Corso di Sistemi e Architetture per Big Data
A.A. 2022/23
Valeria Cardellini

Laurea Magistrale in Ingegneria Informatica

The reference Big Data stack



Data acquisition and ingestion

- How to collect data from external (and multiple) data sources and ingest it into a system where it can be stored and later analyzed?
  - Using distributed file systems, NoSQL data stores, batch processing frameworks
- How to connect external data sources to stream or in-memory processing systems for immediate use?
- How to perform some preprocessing (e.g., data transformation or conversion)?

Driving factors

- Source type and location
  - Batch data sources: files, logs, RDBMS, …
  - Real-time data sources: IoT sensors, social media feeds, stock market feeds, …
  - Source location
- Velocity
  - How fast data is generated?
  - How frequently data varies?
  - Real-time or streaming data require low latency and low overhead
- Ingestion mechanism
  - Depends on data consumer
  - Pull vs. push based approach

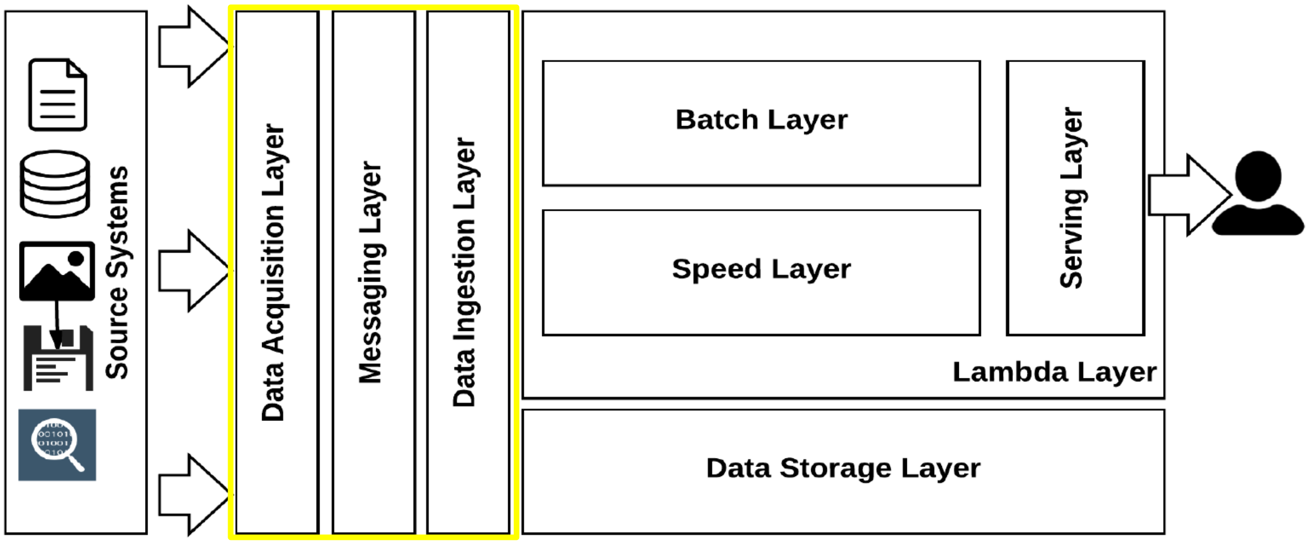
Requirements for data acquisition and ingestion

- Ingestion
  - Batch data, streaming data
  - Easy writing to storage (e.g., HDFS)
- Decoupling
  - Data sources should not directly be coupled to processing framework
- High availability and fault tolerance
  - Data ingestion available 24x7
  - For streaming data: buffering (persistence) in case processing framework is not available
- Scalability and high throughput
  - Number of sources and consumers will increase, amount of data will increase

Requirements for data acquisition and ingestion

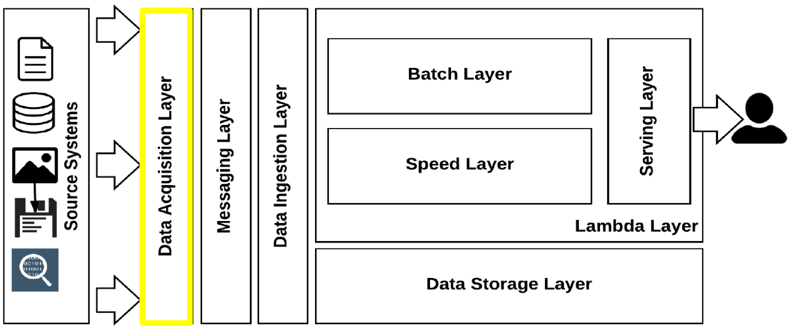
- Data provenance
- Security
  - Data authentication and encryption
- Data conversion
  - From multiple sources: transform data into common format
  - Also to speed up processing
- Data integration
  - From multiple flows to single flow
- Data compression
- Data preprocessing (e.g., filtering)
- Data routing
- Backpressure
  - Data buffering in case of temporary spikes in workload, so that data can be replayed later without loss

A unifying view



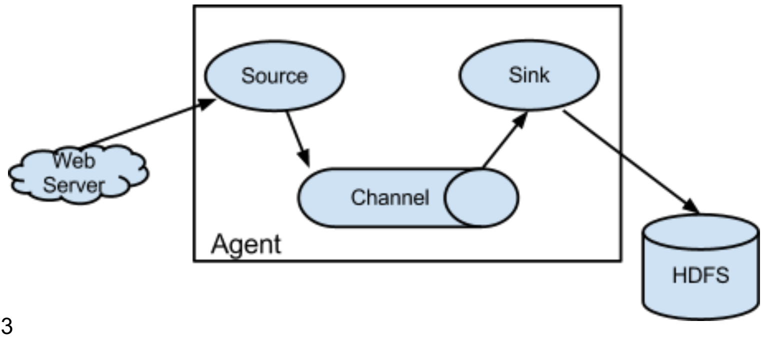
Data acquisition layer

- Allows collecting, aggregating and moving data
- From various sources (server logs, social media, IoT sensors, …)
- To a data store (messaging system, distributed file system, NoSQL data store)
- We analyze
  - **Apache Flume**
  - **Apache NiFi**

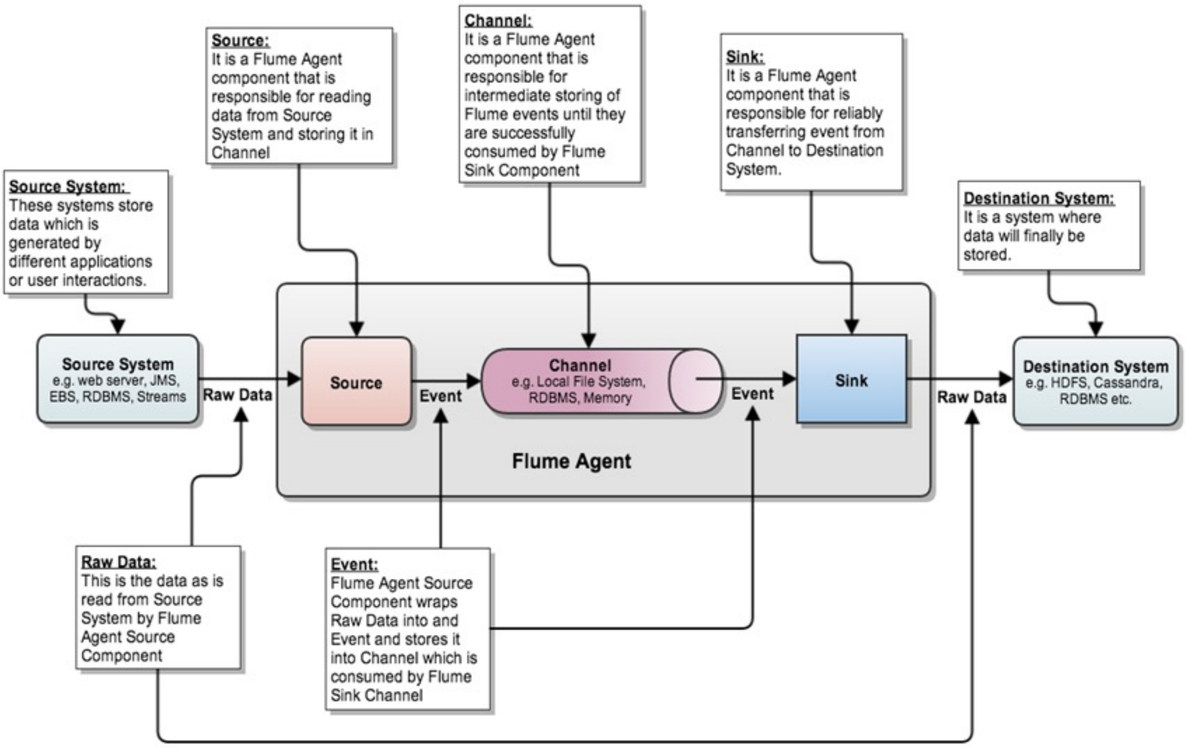


Apache Flume

- Distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of stream data (e.g., log data)
- Robust and fault tolerant with tunable reliability mechanisms and failover and recovery mechanisms
  - Tunable reliability levels
    - Best effort: “Fast and loose”
    - Guaranteed delivery: “Deliver no matter what”
- Suitable for streaming analytics



Flume: architecture


Source System
e.g. web server, JMS, EBS, RDBMS, Streams
Raw Data
Source
Event
Channel
e.g. Local File System, RDBMS, Memory
Event
Sink
Raw Data
Destination System
e.g. HDFS, Cassandra, RDBMS etc.
Flume Agent

Source: It is a Flume Agent component that is responsible for reading data from Source System and storing it in Channel

Channel: It is a Flume Agent component that is responsible for intermediate storing of Flume events until they are successfully consumed by Flume Sink Component

Sink: It is a Flume Agent component that is responsible for reliably transferring event from Channel to Destination System.

Source System: These systems store data which is generated by different applications or user interactions.

Destination System: It is a system where data will finally be stored.

Raw Data: This is the data as is read from Source System by Flume Agent Source Component

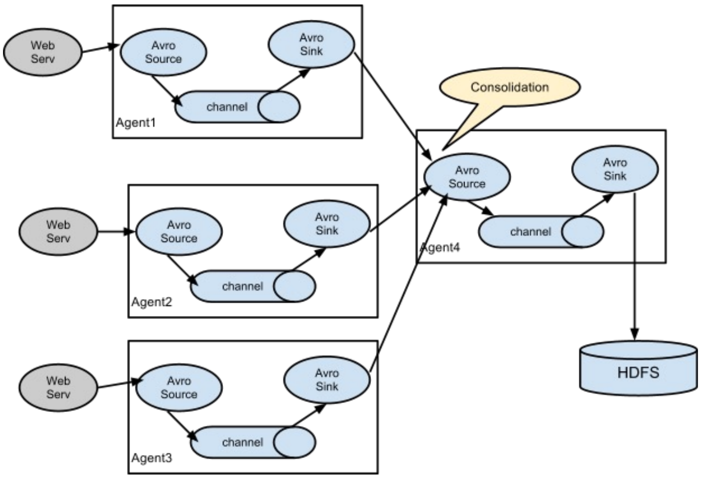
Event: Flume Agent Source Component wraps Raw Data into and Event and stores it into Channel which is consumed by Flume Sink Channel

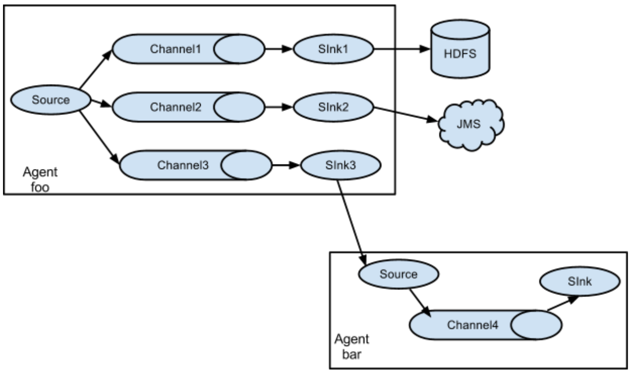
Flume: architecture

- Agent: JVM running Flume
  - One per machine
  - Can run many sources, sinks and channels
- Event
  - Basic unit of data that is moved using Flume (e.g., Avro event)
  - Normally ~4KB
- Source
  - Produces data in the form of events
- Channel
  - Connects source to sink (like a queue)
  - Implements the reliability semantics
- Sink
  - Removes an event from a channel and forwards it to either to a destination (e.g., HDFS) or to another agent

Flume: data flows

- Flume allows a user to build multi-hop flows where events travel through multiple agents before reaching the final destination
- Supports multiplexing the event flow to one or more destinations
- Multiple built-in sources and sinks (e.g., Avro, Kafka)





Flume: reliability

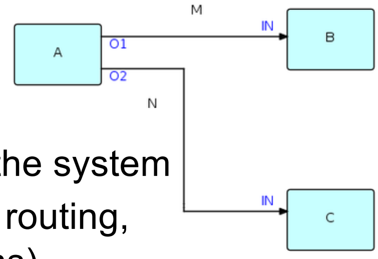
- Events are staged in a channel on each agent
  - Channel can be either durable (FILE, will persist data to disk) or non durable (MEMORY, will lose data if machine fails)
- Events are then delivered to next agent or final destination (e.g., HDFS) in the flow
- Events are removed from a channel *only after* they are stored in the channel of next agent or in the final destination
- Transactional approach to guarantee the reliable delivery of events
  - Sources and sinks encapsulate in a transaction the storage/retrieval of events

Apache NiFi

- Easy to use, powerful and reliable system to automate the flow of data between systems, mainly used for data routing and transformation
- Highly configurable
  - Flow specific QoS: loss-tolerant vs guaranteed delivery, low latency vs high throughput
  - Dynamic prioritization of queues
  - Flow can be modified at runtime: useful for preprocessing
  - Backpressure control
- Ease of use: drag-and-drop web-based UI to create, manage and monitor the dataflow
  - Allows to define sources from where to collect data, processors for data transformation, destinations to store data
- Data provenance and security (SSL, data encryption)

NiFi: core concepts

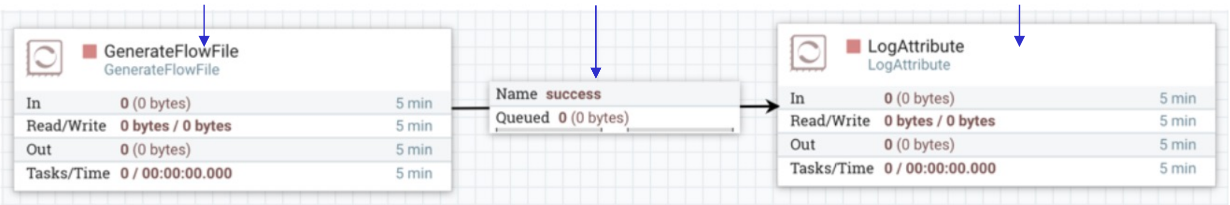
- Based on flow-based programming
- Main NiFi concepts:
  - FlowFile: each piece of user data moving in the system
  - FlowFile Processor: performs the work (data routing, transformation, or mediation between systems)
  - Connection: linkage between processors; acts as queue
  - Flow Controller: maintains knowledge of how processes connect and manages threads and allocations
  - Process Group: specific set of processes and their connections



processor

connection

processor



GeneratedFlowFile
GenerateFlowFile

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name success
Queued 0 (0 bytes)

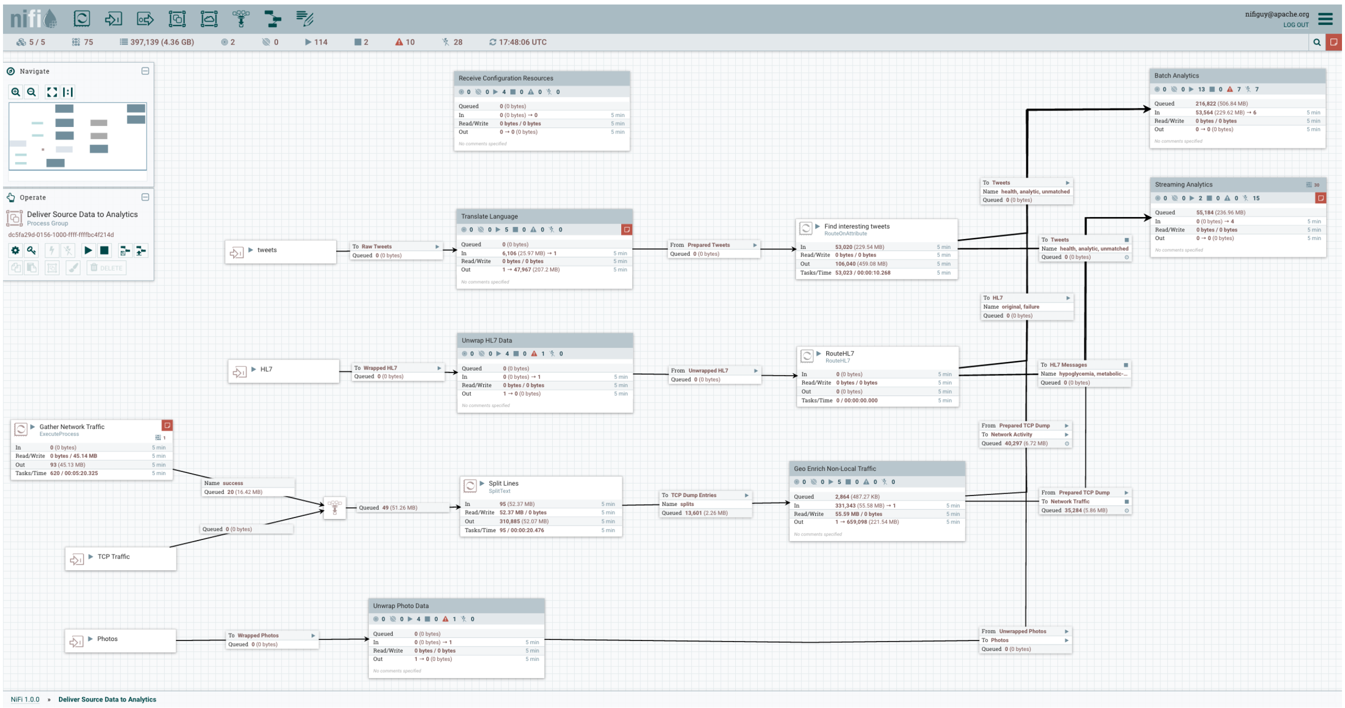
LogAttribute
LogAttribute

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

NiFi: visual command & control

- Drag and drop Processors to build a flow
  nifi.apache.org/docs/nifi-docs/html/getting-started.html
- Start, stop and configure components in real time
- View errors and corresponding messages
- View statistics and health of data flow
- Create templates (i.e., reusable sub-flows) for common Processors and Connections

NiFi: visual command & control



NiFi: processors

- Main steps to create and run the dataflow
  - Add Processors
  - Configure Processors
  - Connect Processors among them
  - Start and stop Processors
  - Get info on Processors

NiFi: processors

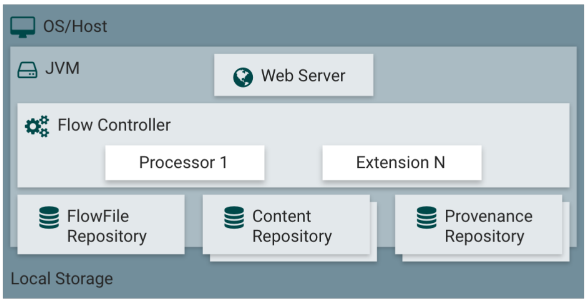
- NiFi provides many different Processors out of the box
  - Capabilities to ingest data from many different systems, route, transform, process, split, and aggregate data, and distribute data to many systems
  - Classified by category
- Data transformation
  - E.g., `CompressContent`, `EncryptContent`, `ReplaceText`
- Routing and mediation
  - E.g., `ControlRate`, `DistributeLoad`, `RouteOnContent`
- Database access
  - E.g., `ExecuteSQL`, `PutSQL`
- Attribute extraction
  - E.g., `ExtractText`, `HashContent`, `IdentifyMimeType`

NiFi: processors

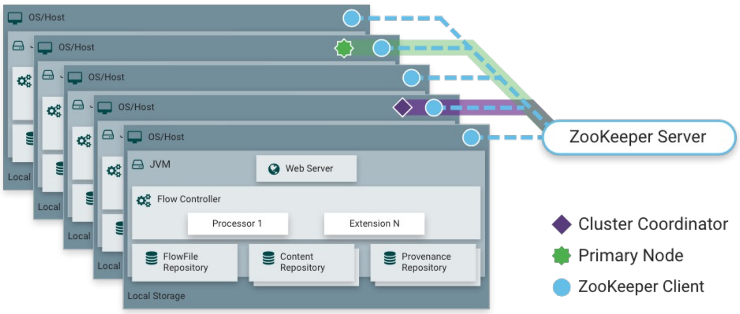
- System interaction
  - E.g., `ExecuteProcess`
- Data ingestion
  - E.g., `GetFile`, `GetFTP`, `GetHTTP`, `ListenUDP`, `GetHDFS`, `FetchS3Object`, `ConsumeKafka`, `GetMongo`, `GetTwitter`
- Data egress / Sending data
  - E.g., `PutEmail`, `PutFile`, `PutFTP`, `PutHDFS`, `PutSQL`, `PublishKafka`, `PutMongo`
- Splitting and aggregation
  - E.g., `SplitText`, `UnpackContent`, `MergeContent`, `SplitContent`
- HTTP
  - E.g., `GetHTTP`, `PostHTTP`, `InvokeHTTP`, `ListenHTTP`
- Amazon Web Services
  - E.g., `FetchS3Object`, `PutS3Object`, `GetSQS`, `PutSQS`

NiFi: architecture

- NiFi executes within a JVM

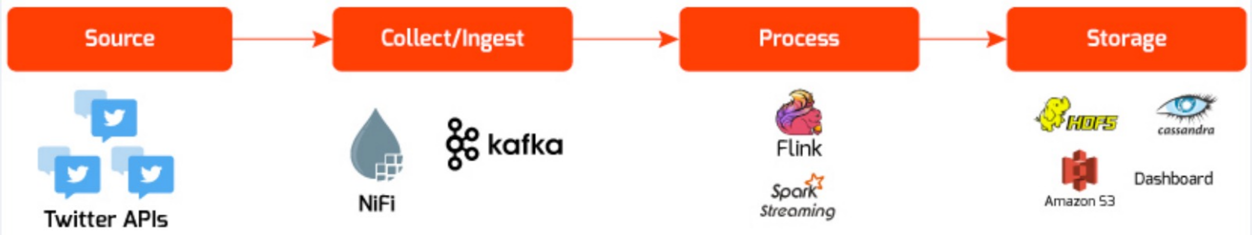


- Multiple NiFi servers can be clustered for scalability



NiFi: use case

- Use NiFi to fetch tweets by means of NiFi's processor 'GetTwitter'
  - Use Twitter Streaming API to retrieve tweets
- Move data stream to Apache Kafka using NiFi's processor 'PublishKafka'



Data serialization formats for Big Data

- Serialization: process of converting structured data into a compact (binary) form
- Data serialization formats you already know
  - JSON
  - Protocol buffers
- Other serialization formats
  - Apache Avro (row-oriented)
  - Apache Parquet (column-oriented)
  - Apache Thrift

Apache Avro

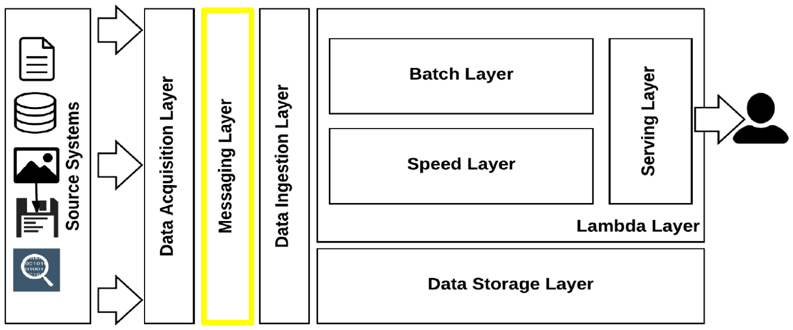
- Key features avro.apache.org
  - Compact, fast, binary data format
  - Supports a number of data structures for serialization
  - Neutral to programming language
  - Simple integration with dynamic languages
  - Relies on *schema:* data+schema is fully self-describing
    - JSON-based schema segregated from data
  - Can be used in RPC
  - Spark (and Hadoop) can access Avro as data source

  spark.apache.org/docs/latest/sql-data-sources-avro.html

- Comparing performance of serialization formats
  - Avro should not be used from small objects (high serialization and deserialization times)
  - Interesting for large objects

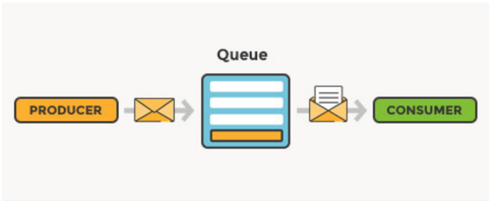
Messaging layer: use cases

- Mainly used in data processing pipelines for data ingestion or aggregation
- Typically used at the beginning or end of a data processing pipeline
  – E.g., at beginning of data processing pipeline:
    - Incoming data from various sensors: ingest data into a streaming system for real-time analytics or a distributed file system for batch analytics

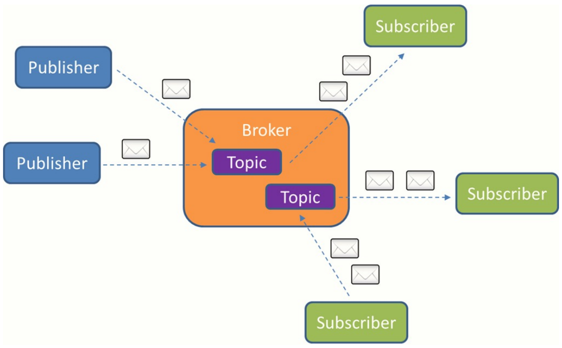


Messaging layer: architectural choices

- Message queue
  – ActiveMQ
  – RabbitMQ
  – ZeroMQ
  – Amazon SQS



- Publish/subscribe
  ☞ Kafka
  ☞ Apache Pulsar
  – NATS
  – Redis

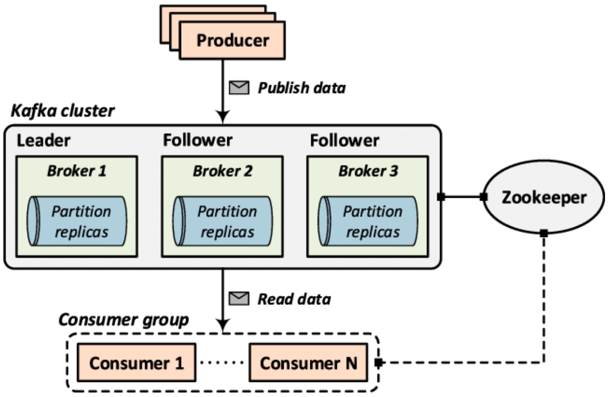


Apache Kafka

- Analyzed in SDCC course
- In a nutshell
  - Open-source, distributed pub/sub and event streaming platform
  - Designed as a replicated, distributed, persistent commit log
  - Clients produce or consume events directly to/from a cluster of brokers, which read/write events durably to the underlying local file system and also automatically replicate the events synchronously or asynchronously within the cluster for fault tolerance and high availability
- Let's recall the main points

Kafka: architecture

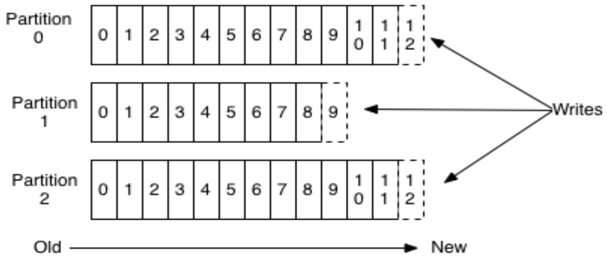
- Kafka maintains feeds of messages in categories called topics
- Producers publish messages to a topic, while consumers subscribe to topics and process published messages



- Kafka cluster: distributed and replicated commit log of data over servers known as *brokers*

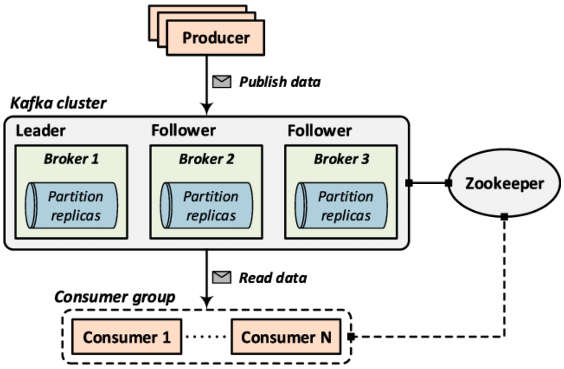
Kafka: topics and partitions

- For each topic, Kafka cluster maintains a partitioned log: topic is split into a fixed number of partitions
- Each partition is an ordered, numbered, immutable sequence of records that is continually appended to
- Each partition is replicated for fault tolerance across a configurable number of brokers
- Partitions are distributed across brokers for scalability



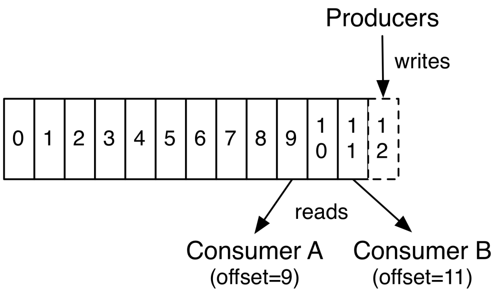
Kafka: partition replication

- Each partition has one leader broker and 0 or more followers
- Leader handles read and write requests
- A follower replicates leader and acts as backup
- Each broker is a leader for some of its partitions and a follower for others to distribute load



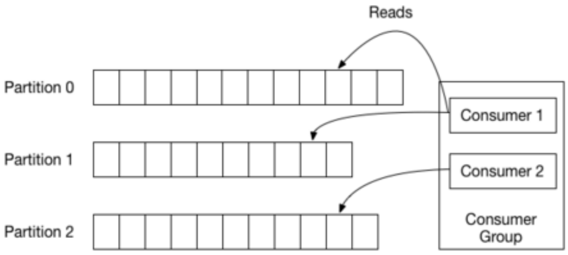
Kafka: partitions

- Producers publish their records to partitions of a topic (round-robin or partitioned by keys), and consumers consume published records of that topic
- Each record is associated with a monotonically increasing sequence number, called offset
  - Kafka provides the topic `__consumer offsets` to store offsets
- Consumers must manage their offset



Kafka: consumers

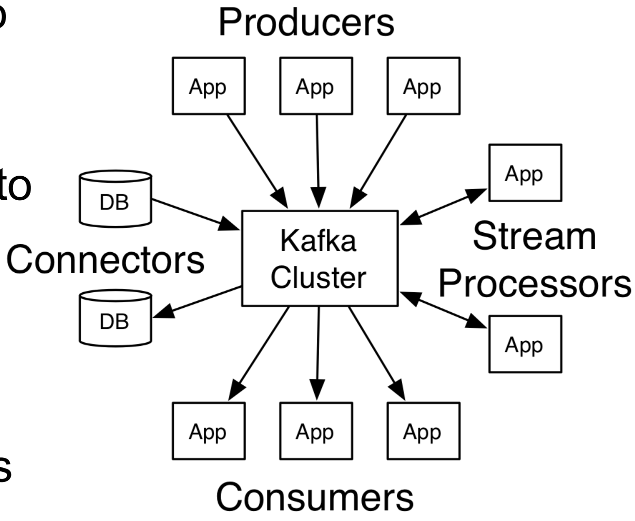
- In Kafka design, pull approach for consumers
  [kafka.apache.org/documentation.html - design_pull](kafka.apache.org/documentation.html#design_pull)
- Consumers use offset to track which messages have been consumed
  - Replay messages using offset
- Consumers can be grouped into a Consumer Group: set of consumers sharing a common group ID
  - A Consumer Group maps to a logical subscriber
  - Each group consists of multiple consumers for scalability and fault tolerance



Kafka: APIs

- Core APIs https://kafka.apache.org/documentation/#api

1. Producer API: allows apps to publish records of data (e.g., logs, IoT) to topics
2. Consumer API: allows apps to read records from topics
3. Connect API: reusable connectors (producers or consumers) that connect topics to existing applications or data systems so to move large collections of data into and out of Kafka
   - Connectors for AWS S3, HDFS, RabbitMQ, MySQL, Postgres, AWS Lambda, MongoDB, Twitter, …

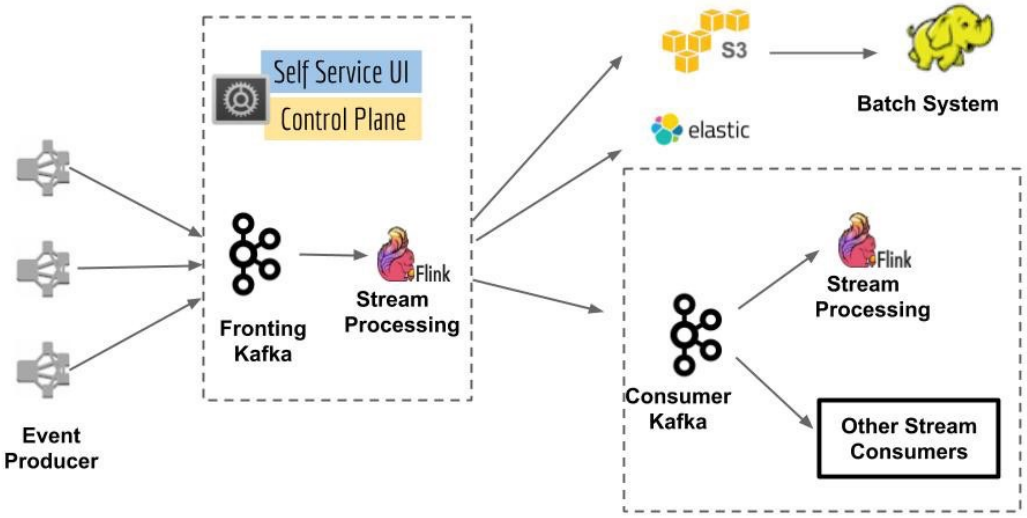


Kafka: APIs

- Streams API: allows transforming streams of data from input topics to output topics
  - Kafka as real-time streaming platform
- Hands-on: use Kafka Streams to process data in pipelines consisting of multiple stages

Kafka @ Netflix

- Netflix uses Kafka for data collection and buffering

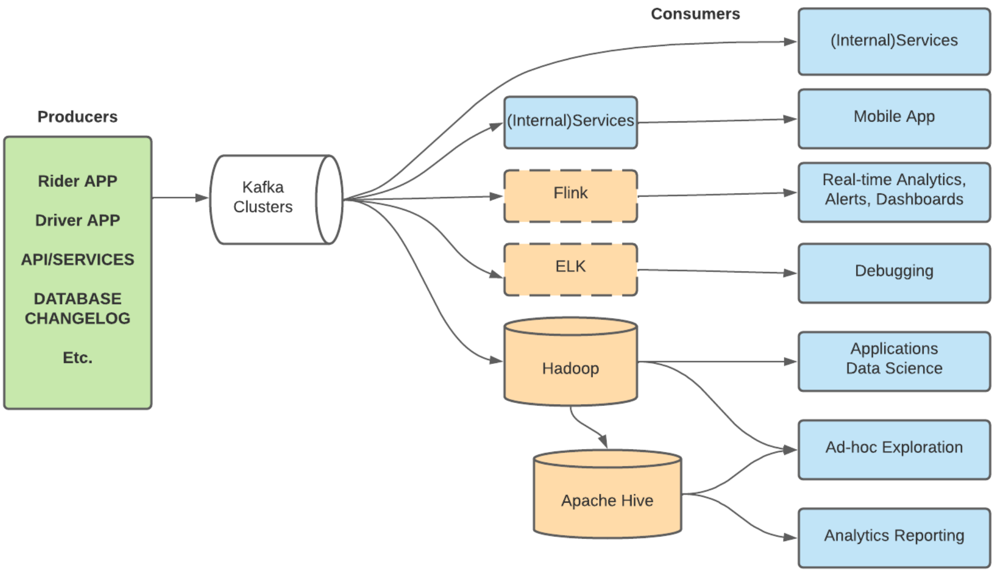


See netflixtechblog.com/kafka-inside-keystone-pipeline-dd5aeabaf6bb

- Another example: www.confluent.io/blog/how-kafka-is-used-by-netflix/

Kafka @ Uber

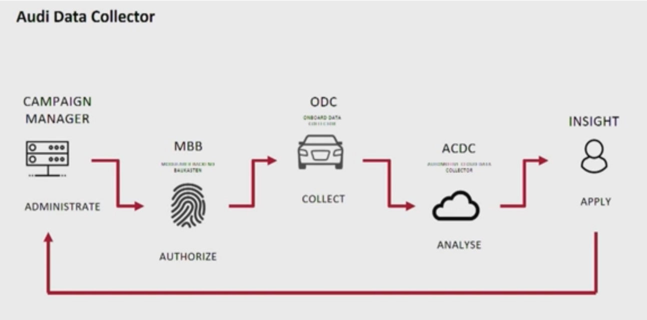
- Uber has one of the largest Kafka deployment in the industry

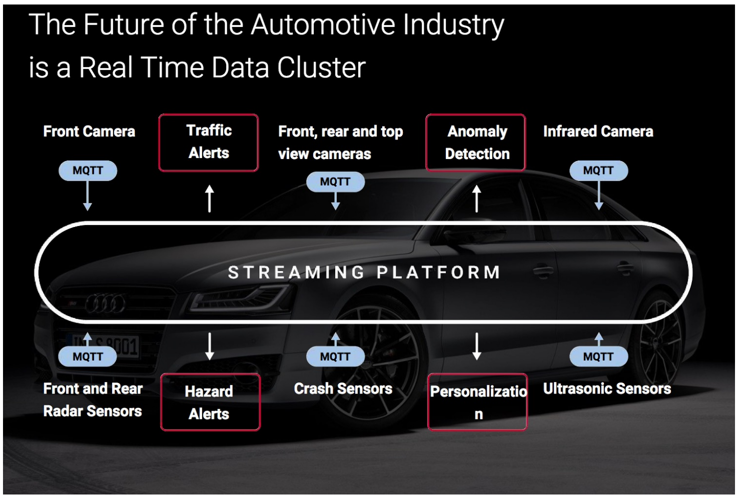


www.uber.com/en-IT/blog/presto-on-apache-kafka-at-uber-scale/

Kafka @ Audi

- Audi uses Kafka for real-time data processing
  - 850 sensors in each car



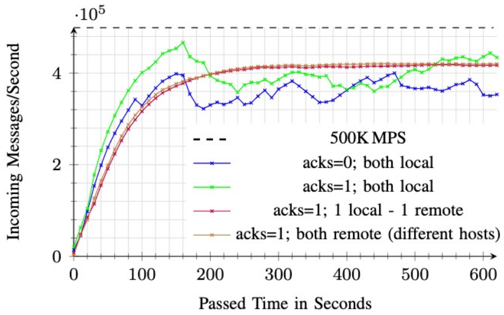


The Future of the Automotive Industry is a Real Time Data Cluster

https://www.youtube.com/watch?v=yGLKi3TMJv8

Kafka performance

- Performance evaluation study of Apache Kafka
  How Fast Can We Insert? An Empirical Performance Evaluation of Apache Kafka, ICPADS 2020
  - Achieves ingestion rate of about 421K messages/second or 92 MB/s (single topic with 1 partition and replication factor of 1) on commodity hardware and using 2 senders
  - Ack level choice influences performance: configurations with enabled acks showed better performance

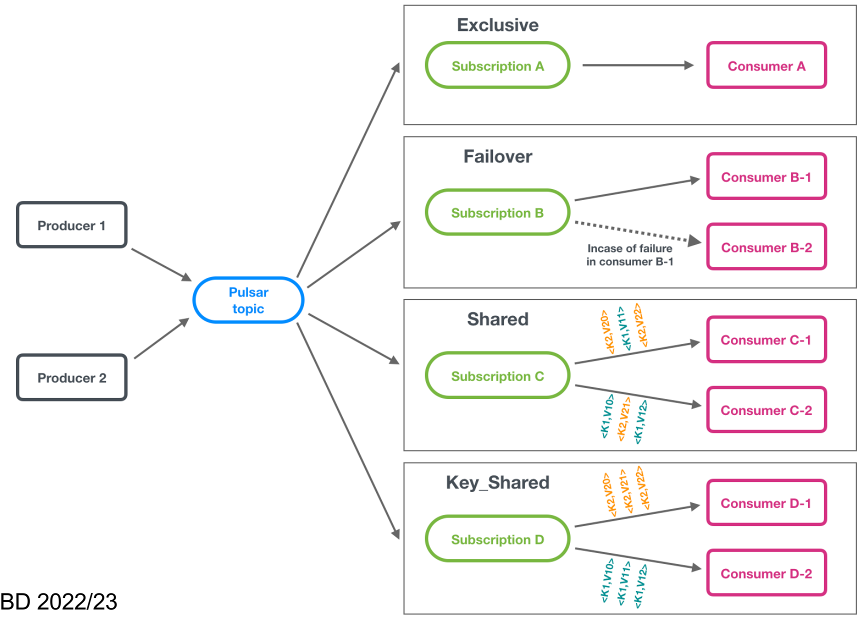


Apache Pulsar

- Cloud-native, distributed messaging and streaming platform, originally developed by Yahoo
- Scalable, low-latency and durable messaging based on pub-sub pattern, with support for geo-replication
- Multiple subscription types for topics
- Guaranteed message delivery with persistent message storage provided by Apache BookKeeper
- Enables also stream-native data processing through a serverless lightweight computing framework, named Pulsar Functions

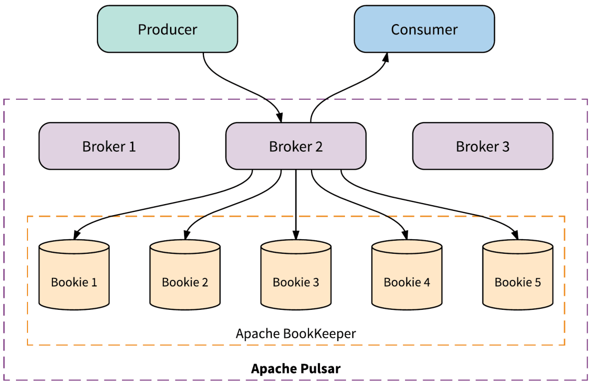
Pulsar: subscription types

- A subscription is a configuration rule that determines how messages are delivered to consumers
- Multiple subscription types: exclusive, shared (or round-robin), failover, and key-shared



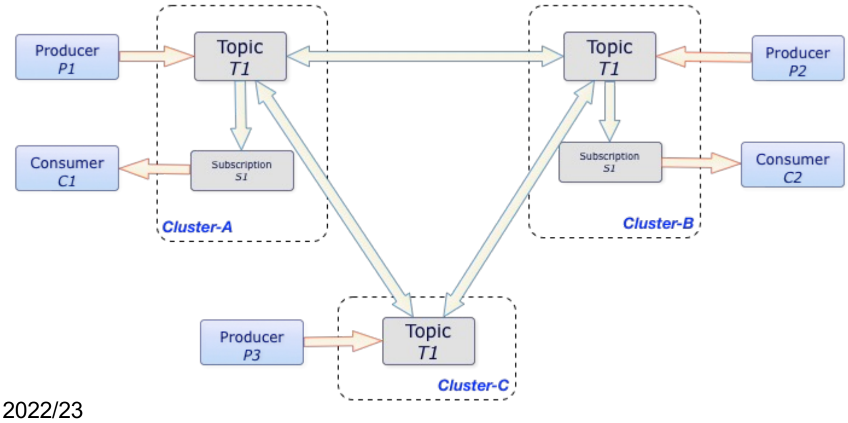
Pulsar: architecture

- Layered architecture designed to provide scalability and flexibility
  - Stateless serving layer and stateful persistence layer
  - Serving layer comprised of **brokers** that receive and deliver messages
  - Persistence layer comprised of Apache BookKeeper storage nodes called **bookies** that durably store messages
    - BookKeeper is a distributed write-ahead log



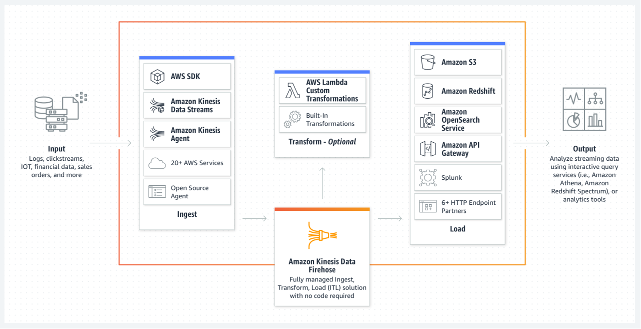
Pulsar: architecture

- Pulsar instance of Pulsar composed of one or more Pulsar clusters
  - Clusters may be geographically distributed and data can be geo-replicated among different clusters
  - Each cluster consiste of one or more brokers, an ensemble of bookies, and a ZooKeeper quorum
  - ZooKeeper is used for cluster-level configuration and coordination

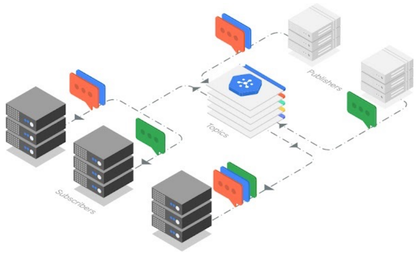


Cloud services for data ingestion

- Amazon Kinesis Data Firehose
  - Fully managed Ingest, Transform, Load, e.g., to S3 as data lake


Input
Logs, clickstreams, IOT, financial data, sales orders, and more
AWS SDK
Amazon Kinesis Data Streams
Amazon Kinesis Agent
20+ AWS Services
Open Source Agent
Ingest
AWS Lambda Custom Transformations
Built-In Transformations
Transform - Optional
Amazon Kinesis Data Firehose
Fully managed Ingest, Transform, Load (ITL) solution with no code required
Amazon S3
Amazon Redshift
Amazon OpenSearch Service
Amazon API Gateway
Splunk
6+ HTTP Endpoint Partners
Load
Output
Analyze streaming data using interactive query services (i.e., Amazon Athena, Amazon Redshift Spectrum), or analytics tools

  - Can transform and compress streaming data before storing it
  - Can invoke Lambda functions to transform source data
- Google Cloud Pub/Sub
  - Fully-managed real-time pub/sub messaging service



References

- Apache Flume documentation, flume.apache.org/FlumeUserGuide.html
- Apache NiFi documentation, nifi.apache.org/docs.html
- Apache Kafka documentation, kafka.apache.org/documentation/
- Apache Pulsar documentation, pulsar.apache.org/docs/3.0.x/concepts-overview/ pulsar.apache.org/docs/en/standalone/