Building Secure, Reliable and Efficient Data ingestion pipelines

Suman Karumuri

Data Council, Austin, March 28, 2024

He/Him

Introduction

Big Data vs Observability

Observability pipelines for ETL

Conclusion

For simpler data pipelines (ETL) Observability pipeline tools are Simpler, Efficient and More Reliable Than Spark/Flink Data pipelines.

Take away

About me




- Principal Engineer @ Airbnb*.
- Work at intersection of Big data and Observability.
- LogSearch: ELK, Loglens, Astra (prev. KalDB)
- Tracing: Zipkin, PinTrace, SlackTrace, OpenTracing author.
- Large scale distributed systems.

*All opinions are my own and not my employers.

Introduction

Big Data vs Observability

Observability pipelines for ETL

Conclusion

Data about Data

120ZB	22%	280B	30%
Data generated In 2023	YoY data volume growth	Data processing market size.	Data real time processing.

Data Analytics

from 30000 feet

- We live in a world of ever increasing data volumes.
- Data is the new oil.
- Processing the data efficiently in real time unlocks new insights that help us build new customer experiences.

Data Analytics

Big Data

Observability

Introduction

Intro to Big data

Big Data vs Observability

Observability pipelines for ETL

Conclusion

Big Data

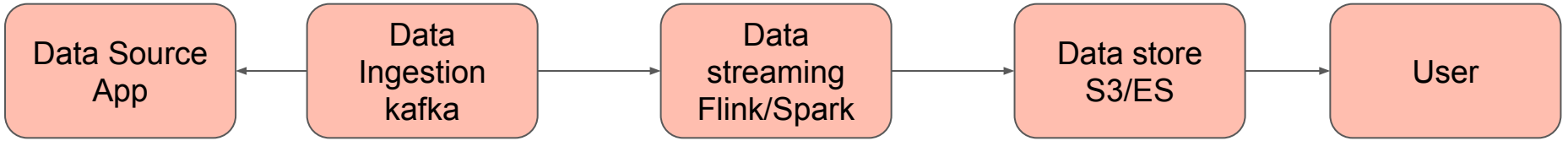
Data streaming and Batch analytics

- We live in a world of ever increasing data volumes.
- Real time applications, mobile and cloud computing, AI, increasing need for privacy and security are all driving the increasing need for advanced data analytics systems.
- Data streaming systems like Flink, Spark etc.. have become very common tools for data processing and analysis.
- Wide range of applications across every aspect of modern life.

ETL

- ETL(Extract-Transform-Load) is a very popular application of data streaming jobs.
- Data is typically loaded from a source, applied light transformation and then ingested into a sink.
- A good chunk of data streaming jobs are ETL jobs.
    - *Per ChatGPT, 50-70% of data streaming jobs are ETL.*

Typical ETL Data Pipeline

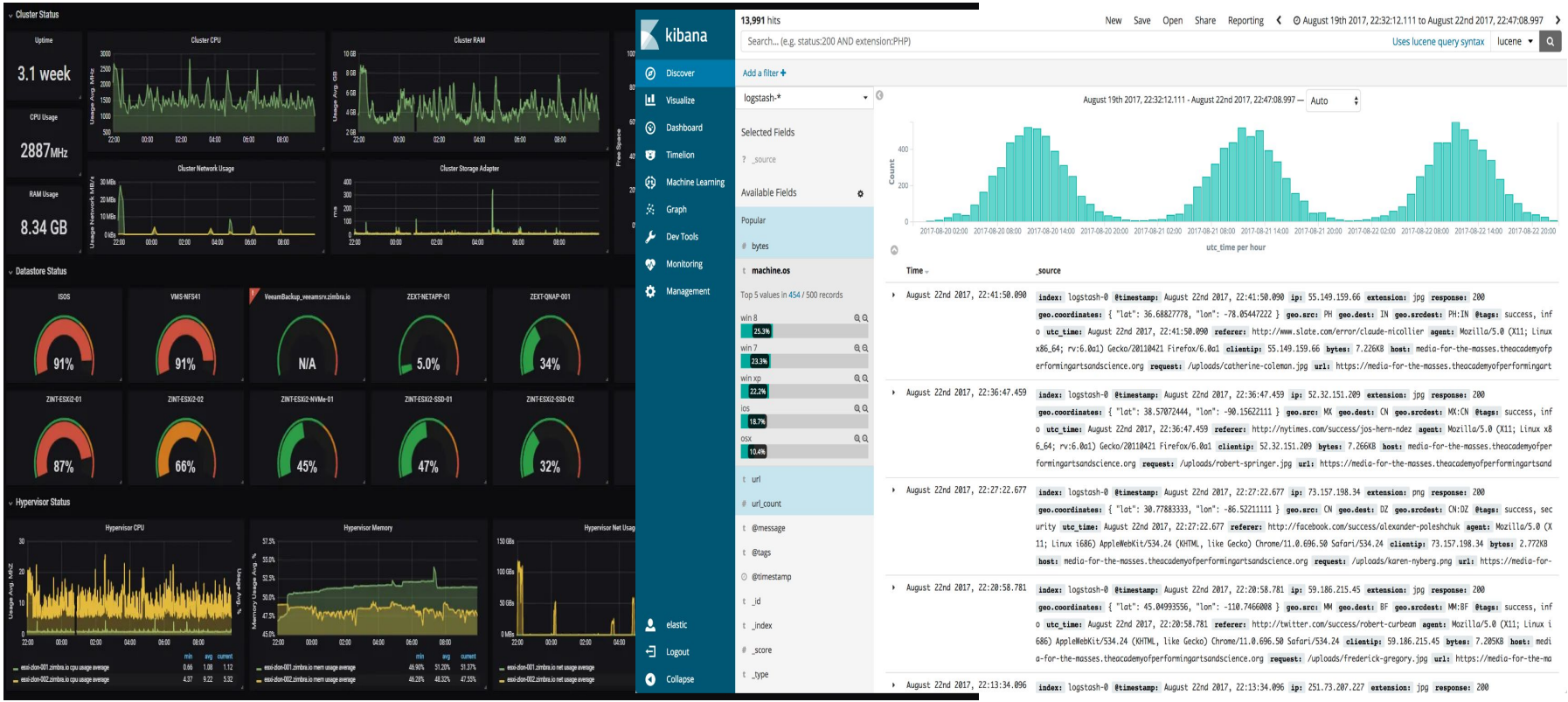


Observability

- Observability is a term originated in control theory.
- It refers to measure the internal state of systems by examining its outputs.
- Typically used to understand the behavior and performance of systems and to diagnose issues in applications and infrastructure.
- Often achieved through collection and analysis of data in the form of logs, metrics and traces.
- Observability is foundational for achieving reliability and performance of modern applications.
- Typical Observability tools are:
    - Datadog, Prometheus, ELK (Elasticsearch, Logstash, Kibana), Grafana etc..

Typical observability data pipeline

Screenshot of Observability tools



Introduction

Big Data vs Observability

Observability pipelines for ETL

Conclusion

Observability vs Big data: Market.

	Big data	Observability
Instrumentation and emit	Custom	Open Telemetry
Process	Flink/Spark/Other	Vector, Logstash, Open Telemetry Collector
Store	Columnar stores	Metrics (pre-aggregated) Logs (semi-structured data) Traces (structured, hierarchical) Events (Structured)
Monitor/Analyze/Decide	UI, Notebooks	Grafana, Alerting
Users	Data engineers/Data scientists	Software Engineers/SRE
Vendors	Snowflake, Databricks, DBT, Airflow, Looker, BI tools	Datadog, Splunk, Grafana, Prometheus, Elasticsearch

Observability vs Big data: Tech stack.

	Big data	Observability
Data freshness	Minutes	Real time
Scale	High latency, Low QPS, 10s of PBs	Sub-second latency, high QPS, PBs
Data model	Relational OLAP	Metrics (pre-aggregated) logs (semi-structured data) Traces (structured, hierarchical)
Schema	Fixed	Dynamic
Query language	SQL	PromQL, Text search, TraceQL
Users	Data engineers/Data scientists	Software Engineers/SRE
Use case	Analyze large volumes of data.	Real time monitoring and alerting
Retention	Long.	Short.

Introduction

Big Data vs Observability

Observability pipelines for ETL ingestion

Conclusion

Observability pipelines

for ETL pipelines

- Data pipelines are implemented using data streaming systems like Spark/Flink etc..
- Observability pipelines systems are implemented using systems like Open Telemetry collector, Vector etc..
- Applies to simple ETL pipelines:
  - Kafka -> Process -> Elasticsearch.
  - Network -> Process -> Kafka.
- Simple ETL pipeline.
  - No Joins.
  - No to very low state.
  - No complex computations.
- Data streaming pipelines are ideal for large scale complex computations.
  - Trade off efficiency for scale and durability.

Simple

Built-in Data Ingestion layer.

- Data pipelines written in Flink*/Spark can’t be run as a service listening for requests.
- Data pipelines have a separate data ingestion component.
    - Often bespoke and custom.
- Observability pipelines software can be run as a service.
    - No need to run a separate ingestion layer.
    - Same component for ingestion and transformation.

Simple

Config driven

- Data pipelines are often implemented using code.
- Observability pipelines are mostly config driven.
- Observability pipelines can perform simple filtering and transformation operations.
- Faster development cycles.

Simple

Sample Flink job for ingesting Events into elasticsearch.

// Flink job - ~80 lines

```
public class KafkaToElasticsearch {
    public static void main(String[] args) throws Exception {
        final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

        Properties kafkaProps = new Properties();
        kafkaProps.setProperty("bootstrap.servers", "localhost:9092");
        kafkaProps.setProperty("group.id", "test-group");

        // Adjust the Kafka consumer
        FlinkKafkaConsumer<String> consumer = new FlinkKafkaConsumer<>(
                "your-topic-name",
                new SimpleStringSchema(),
                kafkaProps);

        Map<String, String> esConfig = new HashMap<>();
        esConfig.put("cluster.name", "elasticsearch");
        esConfig.put("bulk.flush.max.actions", "1");
        esConfig.put("hosts", "localhost:9200");

        ElasticsearchSink.Builder<String> esSinkBuilder = new ElasticsearchSink.Builder<>(
                esConfig,
                new ElasticsearchSinkFunction<String>() {
                    public IndexRequest createIndexRequest(String element) {
                        Map<String, String> json = new HashMap<>();
                        json.put("data", element);
                        return Requests.indexRequest()
                                .index("your-index")
                                .type("your-type")
                                .source(json);
                    }

                    @Override
                    public void process(String element, RuntimeContext ctx, RequestIndexer indexer) {
                        indexer.add(createIndexRequest(element));
                    }
                });

        // Kafka source
        DataStream<String> sourceStream = env.addSource(consumer);

        // Filter out messages based on a regular expression
        DataStream<String> filteredStream = sourceStream.filter(new FilterFunction<String>() {
            @Override
            public boolean filter(String value) throws Exception {
                // Adjust the field and regex according to your needs
                return !value.matches("your-regex-here");
            }
        });

        // Sink to Elasticsearch
        filteredStream.addSink(esSinkBuilder.build());

        env.execute("Kafka to Elasticsearch");
    }
}
```

Simple

Config driven development using vector sample.

```
[sources.kafka_source]

  type = "kafka" # type of the source

  bootstrap_servers = "<KAFKA_BROKERS>"

  group_id = "kafka-consumer-group"

  topics = ["<KAFKA_TOPIC>"]


[transforms.filter_transform] # Drop messages based on a
regular expression

  type = "filter"

  inputs = ["kafka_source"]

  condition = '''!matches(.message, "<REGEX_PATTERN>")'''


[sinks.elasticsearch_sink]

  type = "elasticsearch"

  inputs = ["filter_transform"]

  endpoint = "<ELASTICSEARCH_ENDPOINT>"

  index = "<INDEX_NAME>-%F"

  bulk_action = "index"
```

Simple

Deploy

- Flink/Spark jobs have complex state and are complex to run.. They are different than running a micro service.
  - Needs special deployment process.
  - Custom observability to monitor.
- Observability pipelines can be deployed on K8s easily.
  - Works like a microservice.
  - Observability built in.

Efficient

>5x more efficient

- Data ingestion pipelines trade off performance for scalability. (COST Paper)
    - Spark jobs to take 10x more resources than an equivalent single threaded Java application.
- Observability pipelines are closer to single threaded Java programs so they perform a more efficient ETL.
    - Anecdotal benchmark is >5x.
- Observability pipelines can be scaled up and down using auto-scaling groups.

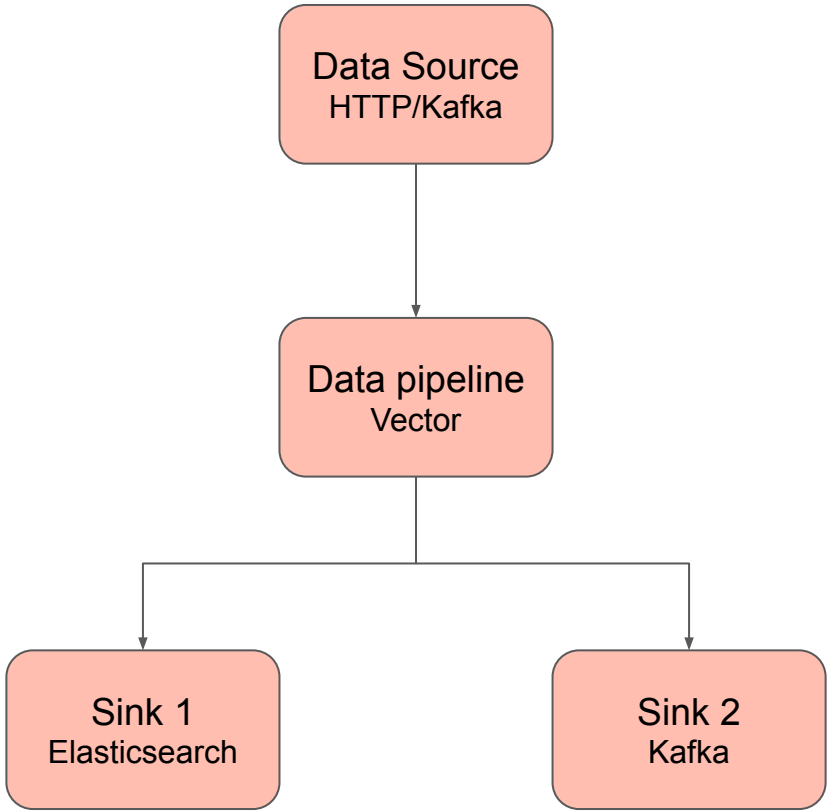
Efficient

Multi-sink ingestion

- Data pipelines used in Flink are ideal for single sink ingestion.
    - Infra cost is proportional to number of sinks in typical configuration.
- Observability pipelines enable multi-sink ingestion.
    - Infra cost is fixed.

Efficient

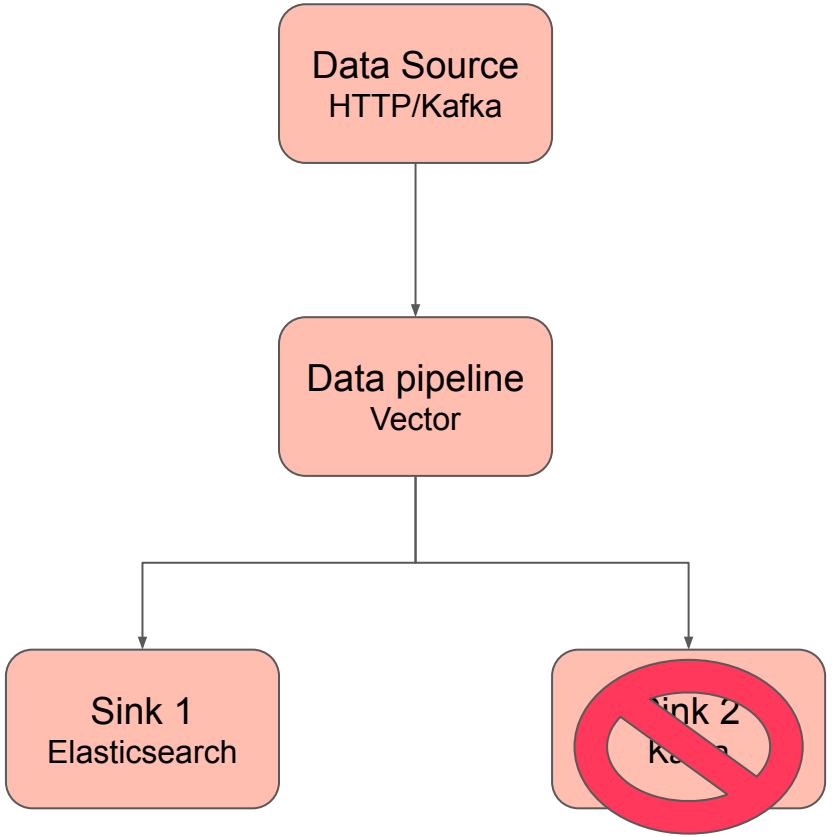
Multi-sink ingestion in Observability pipeline



Efficient

Multi-sink ingestion in Observability pipeline

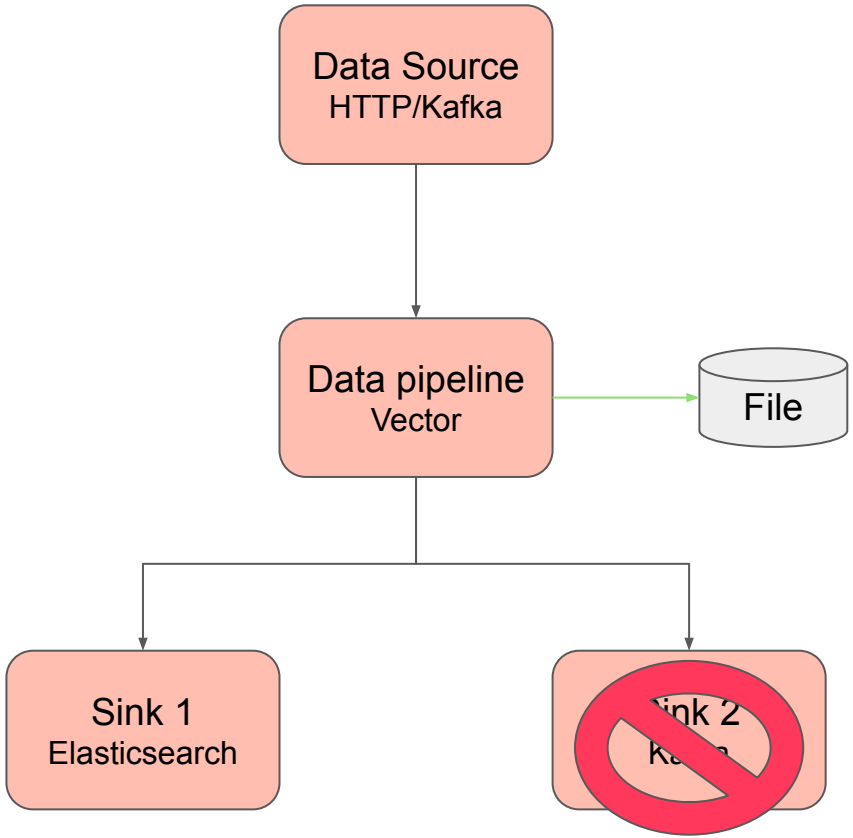
Sink 2 fails.



Efficient

Multi-sink ingestion in Observability pipeline

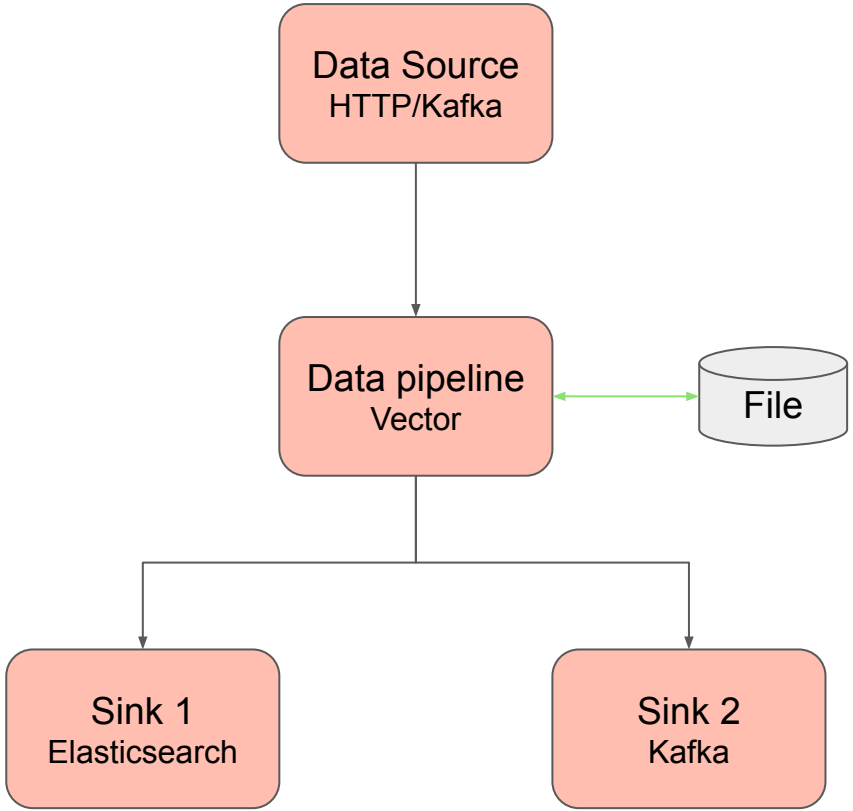
Sink 2 fails.
Persist data to local file.



Efficient

Multi-sink ingestion in Observability pipeline.

Data is backfilled when Sink 2 is back.



Available

Incremental deployments

- Data pipelines are complex stateful entities.
  - Designed for complex computations.
- Deploying data pipelines often takes several minutes. Proportional to job size and complexity.
  - Challenging to hit data freshness SLA.
- Observability pipelines can be deployed incrementally without downtime.

Available

Handle load spikes

- Data ingestion can spiky.
  - Data ingestion pipelines need techniques for load shedding, buffering, rate limiting etc..
- Data streaming pipelines need to built with these features in mind.
- Observability pipelines come with load shedding, buffering and rate limiting built in.

Secure

- Run multiple pipelines on the same host.
  - No code shared between apps.
- Practices from microservices world apply here.
  - No special security required.

Introduction

Big Data vs Observability

Observability pipelines for ETL

Conclusion

Conclusion

- Observability and Big data are two broad ways of analyzing data.
  - Lots of overlap between these domains.
  - Differ in scale of data, QPS and Latency.
- Observability pipelines can be more efficient, simpler, highly available and flexible for *simpler ETL* pipelines than data pipelines.
  - Data streaming pipelines are ideal for high fidelity complex computations over large data volumes.

Advantages of Observability vs data pipelines*

*For simple ETL jobs

Simple

Simpler to run and scale.
Config driven.
Exposed to internet.
Secure.
Same tool for ingestion and transformation.

Available

Highly available.
No availability loss during deployment.

Efficient

Resource efficient
Multiple-sinks.
Productive developers.
Fast iterations.

Q & A

Thank you!

Suman Karumuri
https://www.linkedin.com/in/mansu/