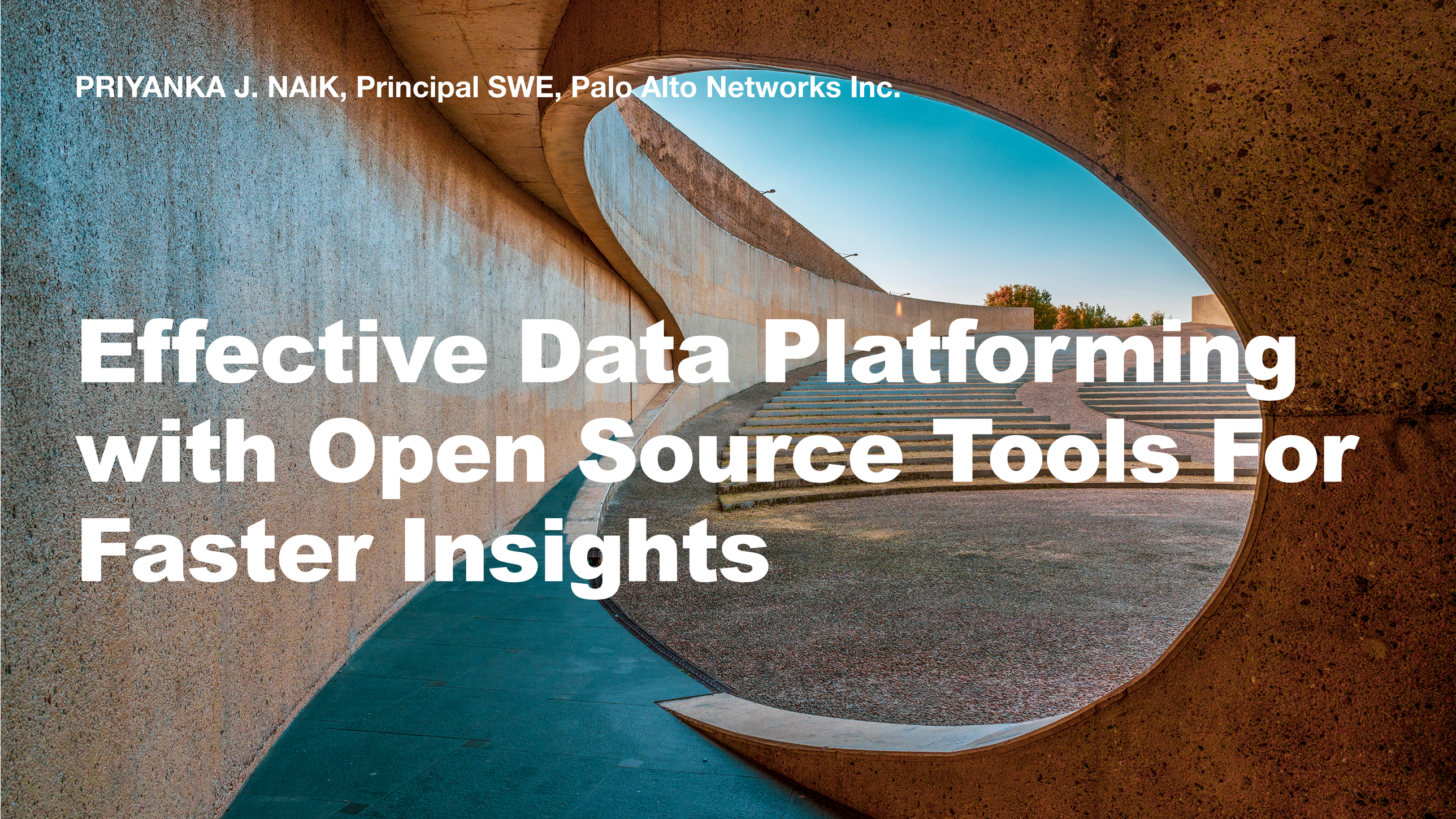
PRIYANKA J. NAIK, Principal SWE, Palo Alto Networks Inc.



Effective Data Platforming with Open Source Tools For Faster Insights

Speaker Bio

Priyanka J. Naik

- **Engineer for about 17 years**
  - Full-stack + Backend + Data
- **Current:** Palo Alto Networks Inc.
- **Past**:
  - AppFolio Inc
  - Citrix Systems Inc/LogMeIn Inc
  - Tower Technologies
  - Accenture
- **Education**:
  - Master’s (Comp Sci) - Univ. of Southern California, Los Angeles, USA
  - Bachelor’s (Info Sci) - Siddaganga Institute of Technology (SIT, Visvesvaraya Technological Univ,), India

linkedin.com/in/priyankajnaik/

Data Democratization

	Before	After
Stakeholders	• Senior Leaders • Marketing teams • Sales teams	• Software Engineers • AI/ML engineers • Data Scientists • Research teams • Senior Leaders • Marketing teams
Tools	Mostly Batch processing tools and Data warehouses	More Stream processing and transactional databases + data warehouses
Automation	Minimal	On par with software engineering

Need for Efficient & Robust Data Platform Architectures

- A **shift** from original approaches is needed
- **3rd party services** (Confluent etc) can be very **expensive** to start off with
- Based on usage and cost, data engineering teams have to consider
  - 3rd party vs open source
  - Delivering value **early** and **continuously**
- Sound architecture with clear **aims**, is key…

Data Platform Architectural Aims

Architectural Aims

Software + Data Engineering

- Modularity
- Separation of Concerns
- Loose Coupling
- Scalable
- Low bar for entry and to maintain
- Low Cost

Architectural Aims

Software + Data Engineering

- Modularity
- Separation of Concerns
- Loose Coupling
- Scalable
- Low bar for entry and to maintain
- Low Cost

What about Stakeholder Requirements?

Stakeholder Requirements

Stakeholder Requirements

A General Laundry list

- Ability to handle and join/enrich data from varied sources
  - Documents
  - Existing data in warehouses
  - Uncollected tool data
- Transmission of varied data sources to points of usage
  - Reporting
  - Third party tools
- Quick ingestion, processing and availability of lots of data sources

Stages of Development

The How

1. Understanding Your Data/Pre-processing
2. Gather your tools
3. Identify Platform Build tools
4. Assemble
    A. Ingestion
    B. Streaming & Stream Processing
    C. Storage & Dissemination
    D. Monitoring & Security
5. Other Features
    - Data Catalog

The How:
1. Understanding Your Data/ Pre-Processing

Understanding Your Data/Pre-Processing

- Understand your data and data sources before making the system understand it
  - Structured/Semi-structured/Unstructured or a mix
  - Streamed data or batch to stream conversion?
  - Frequency of data changing
  - Size of data growth
  - Purge policy
- Map your most ubiquitous synonymous fields in master data
  - “Is Product ID the same as Team ID?”
- Clean your data
  - Prune unused data fields
- Standardize data/event representation -> For instance, CNCF CloudEvents
- Needs of stakeholders
  - Reports or insights?
  - Hands-on or hands-off?

The How:
2. Gather Your Tools

Gather Your Tools

Considerations

- Schema. Schema. Schema.
  - Confluent Schema Registry (OS, Confluent Community license) or
  - RedHat Artifact Registry (OS FSD)
  - Confluent Avro Converter - schema detection
- Open Source and Cloud Native
  - K8s on cloud of choice
  - Kafka
  - Camel Connectors
  - KSqlDB, Flink
  - Strimzi
  - CloudEvents

Gather Your Tools

Considerations (Contd.)

- In-flight processing, optimized transport of large data feeds with in-built scalability
  - Kafka
- Ability to handle both streaming and data at rest & integrations for common data sources
  - Kafka Connect
- Observability
  - Data catalog
  - Platform Monitoring
    - Grafana + Prometheus

The How:
3. Identify Platform Build Tools

Build & Deploy

- Helm
  - Kafka, Kafka Connect (connectors too) (CRDs)
  - Strimzi on K8s (CRDs)
  - Schema Registry (CRDs)
- Terraform
  - KSqlDB or Flink Applications
  - Cloud-specific resources like K8s clusters, KMS etc
- Deployment tools of choice - Jenkins or Git pipelines

The How:

4.A Ingestion

Ingestion

- Push events to Kafka via https
- Kafka Connect for pull-based streams
  - **Source Connectors** for **different source types (via Apache Camel project or Confluent Hub)**
    - Files, REST APIs, PubSub, Lambda triggers
    - Single Message Transforms (**SMTs**)
      - Cast(), Filter() etc.
- Confluent Schema Registry
  - **Automatic Schema deduction** by Confluent Avro Converter (Apache License)
  - **Versioning**
  - Different **levels of compatibility** (backward, full, forward, transitive, none, etc)
- Also consider - RedHat Service Registry as a drop-in replacement for Confluent Schema Registry

The How:

4.B Stream Processing

Stream Processing

- **Input** : Kafka topic data
- **Output**: Into another Kafka topic
- Amortized (**stream processing**) vs all at once (batch processing)
  - **Optimal** resource usage
  - **Quicker insights** on incoming data
- KSqlDB/Flink **Lookup tables** (for referential data or master data)
- Streaming data + Lookup tables = **Enrichment**
- Enriched data can then be transformed and aggregations done over for **insights**

The How:

4.C. Storage & Dissemination

Storage or Dissemination

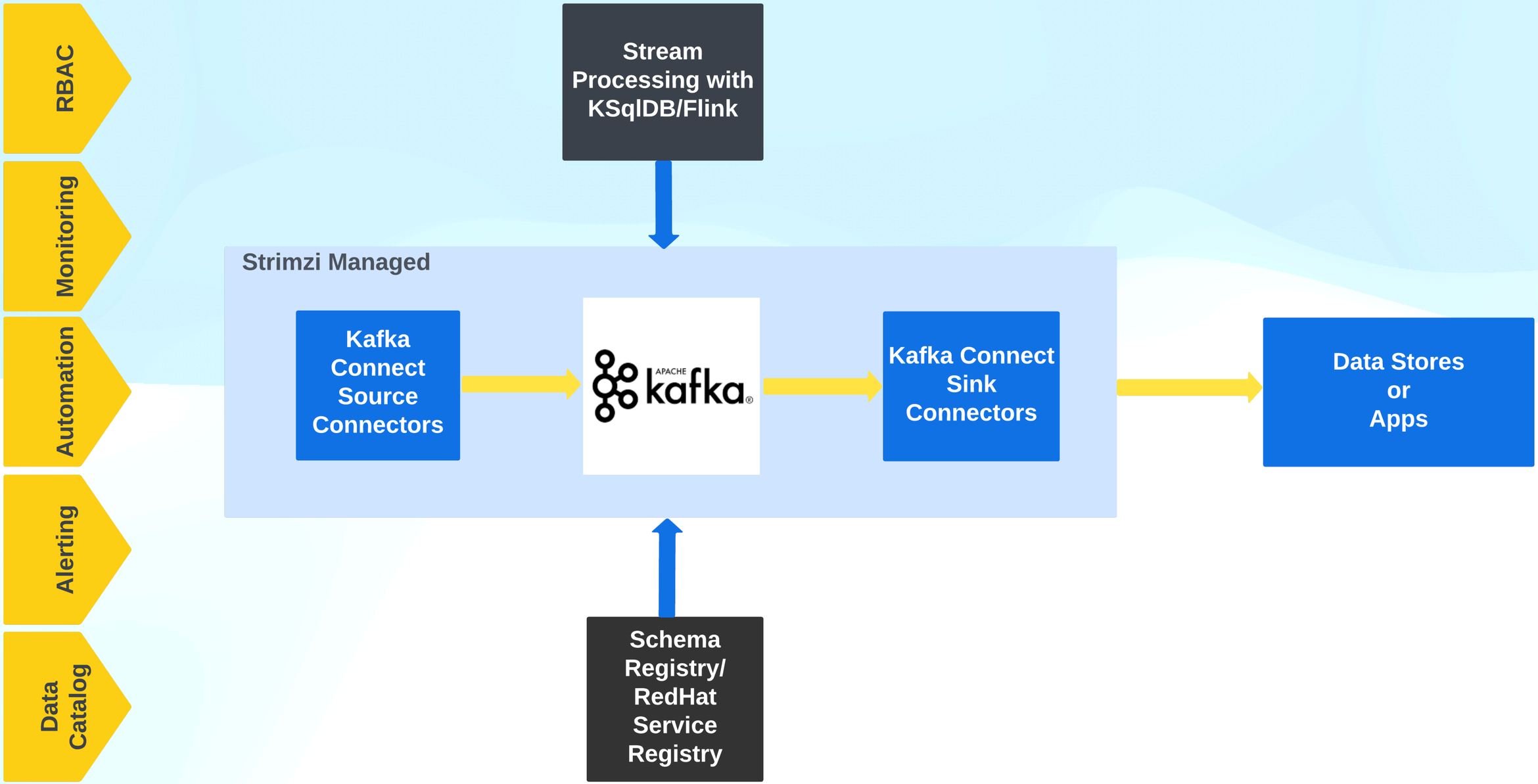
- Kafka Connect
  - **Sink Connectors** for **different target locations (via Confluent hub or Apache Camel project)**
    - BigQuery sink connector
      - Creates, updates tables based on schema & upserts into tables or new partitions
    - Files, REST APIs and many more
    - Build Your Own Connectors (BYOC)
- Data Warehouse Storage used for
  - Reporting
  - Training Machine Learning + AI algorithms
- Send data to other systems via APIs

The How:
4.D Monitoring & Security

Monitoring & Security

- Use monitoring and alerts extensively using Grafana and Prometheus integrated with your platform and data streams
- Secure everything with RBAC
  - Onboard users based on roles and defined access policies
  - For adhoc accesses, use time-based revocation

Architecture



Learnings

Learnings

- Kafka/KConnect advantage of good integrations and ubiquitousness due to usage of standards and useful tools built around it
- KSqlDB downsides vs Flink
- Resolve your Schemas automatically
  - Use Confluent’s Avro converter with Confluent Schema/Redhat Service registry (don’t burn fingers writing and maintaining schema versions)
- Data Catalog is useful for easy lookup for stakeholders who are hands off
- Optimize your transformations with stream processing

More Learnings

- Always talk to stakeholders to ensure they are benefitting from the data
- If no connector exists, you can build them yourself, learn from existing ones
- Don’t hold back on looking under the hood and fixing things
- Open source communities:
  - CNCF for Strimzi or CloudEvents (Slack)
  - Apache Camel project (Github)
  - Various connector library communities (Github)

“Ancora Imparo”
“Yet, I am learning”

Michaelangelo at 87

Thanks!

Resources

- Apache Camel connectors for Kafka connect (GitHub)
- Strimzi (GitHub)
- Confluent Hub
- Confluent Schema Registry
- RedHat blogpost on Service Registry as a drop-in for Confluent Schema Registry
- Apache Flink
- Confluent ksqlDB
- CNCF CloudEvents