CLOUDERA

Cloudera Data Flow
CDF Workshop

Data in Motion Field Team

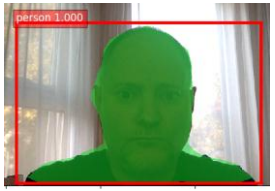
Welcome to our Cloudera Data Flow Workshop

Who are we?

Cloudera Data in Motion Field Team







@PaasDev



Welcome to Your Local Cloudera Team

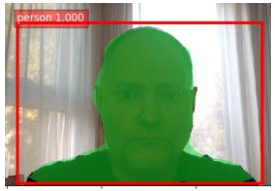
Who are we?

Camila Hiskey
Ifi Derekli, Senior Solutions Engineer - ifi@cloudera.com
Marty Lurie
Eran Orgad
William Brooks, Solutions Engineer - wbrooks@cloudera.com @wcbdata

Today's Lead

Who am I?

Cloudera Data in Motion Field Engineer



@PaasDev

DZone Zone Leader and Big Data MVB;
Princeton NJ Future of Data Meetup;
ex-Pivotal Field Engineer;
Author of Apache Kafka RefCard
https://github.com/tspannhw
https://www.datainmotion.dev/

AGENDA

- Cloudera DataFlow Overview
- Cloudera Flow Management
- Cloudera Edge Management
- Cloudera Stream Processing
- Cloudera Stream Analytics
- Labs Architecture
- Summary

AGENDA

8:30am-9:00am – Check In & Complimentary Breakfast

9:00am-9:15am – Welcome & Kickoff

9:15am-9:30am – Cloudera DataFlow (CDF)

9:30am-10:30am – Cloudera Flow Management - NiFi (Core concepts, Designing Data Flows)

10:30am-10:45am – Break

10:45am-11:15am – NiFi (Extending NiFi, Data Distribution, CI/CD)

11:15am-12:00am – Lab

12:00am-12:30pm – MiNiFi and Cloudera Edge Management (CEM)

12:30pm-1:15 pm – Complimentary Lunch/Lab

1:15pm-1:45pm – Cloudera Stream Processing (CSP) - Kafka, Schema Registry and Streams Messaging Manager (SMM)

1:45pm-3:30pm – Lab

3:30pm-4:00pm – Kafka Streams

4:00pm-4:30pm - SRM - Mirror Maker 2, Streams and Flink

4:45pm-5:00pm – Summary And Wrap-up

Labs are available at

https://github.com/asdaraujo/edge2ai-workshop

Your Environment:
http://35.175.115.227

Labs Summary:

- Lab 1 - On the Gateway host, run a simulator to send IoT sensors data to the MQTT broker.
- Lab 2 - On the Gateway host, configure and start MiNiFi, which will read from the MQTT broker, filter and forward to the NiFi cluster.

- Lab 3 - Create the MiNiFi flow on the Edge Flow Manager and publish it for the MiNiFi agent to start sending data to the NiFi cluster.
- Lab 4 - On Schema Registry, register the schema describing the data generated by the IoT sensors.

- Lab 5 - On the NiFi cluster, prepare the data and send it to the Kafka cluster.

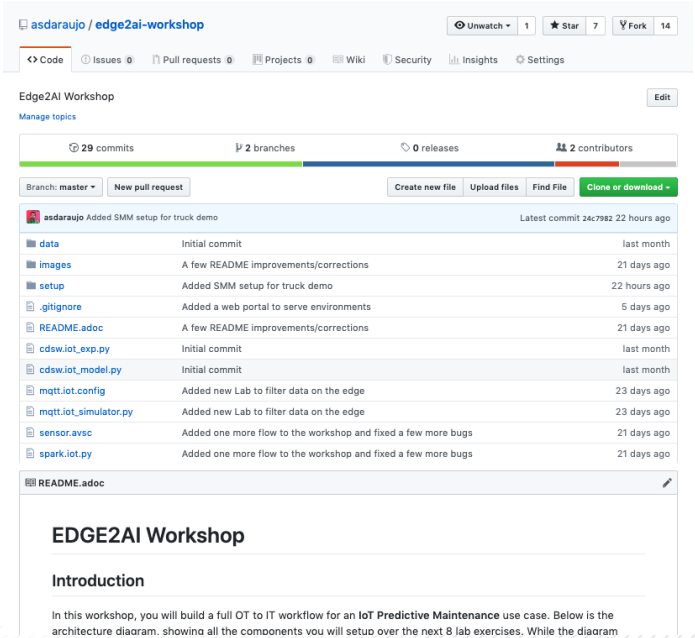
- Lab 6 - On the Streams Messaging Manager (SMM) Web UI, monitor the Kafka cluster and confirm data is being ingested correctly.
- Lab 7 - Use the Edge Flow Manager to update existing edge flows and perform additional processing on the edge
- Lab 8 - Use NiFi to process each record, calling the Model endpoint and save results to Kudu.

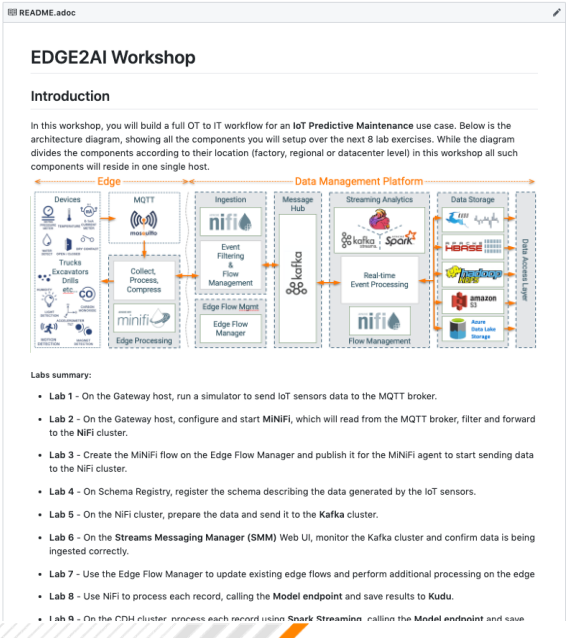
https://github.com/asdaraujo/edge2ai-workshop#labs-summary

LABS

Step by step instructions

https://github.com/asdaraujo/edge2ai-workshop



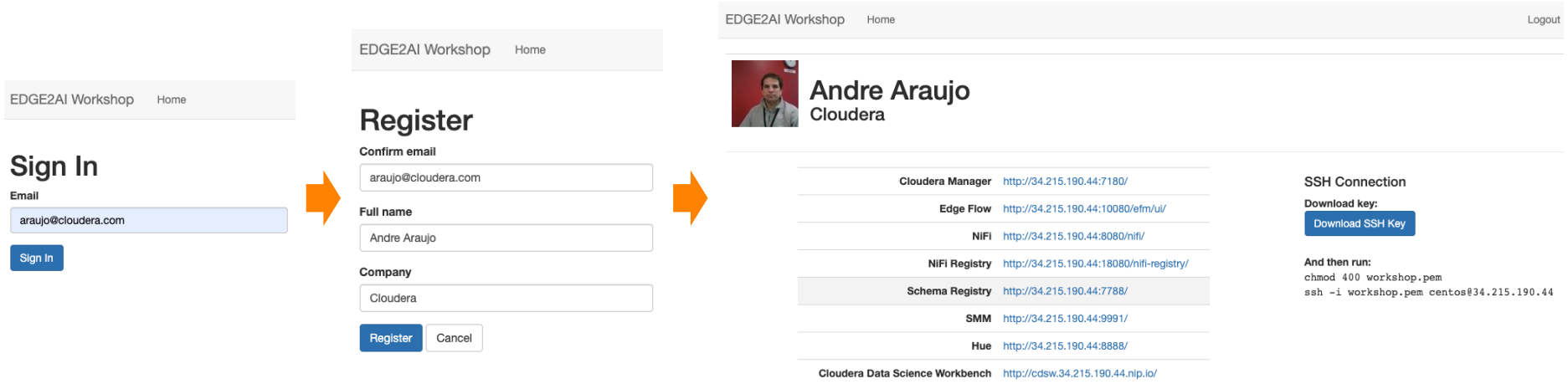


HOW DO I GET ACCESS TO A CLUSTER?

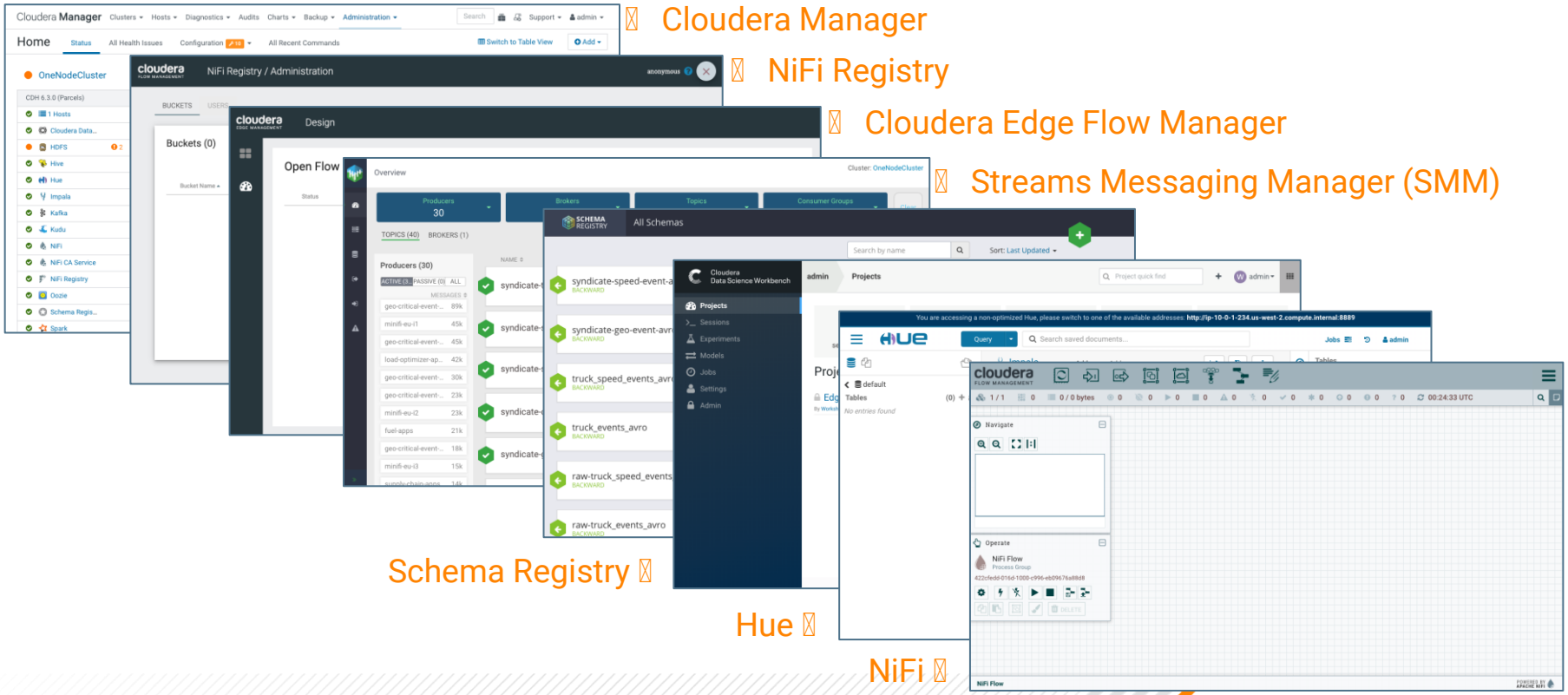
Register for a Single-Node Cluster in AWS

admin/admin

http://35.175.115.227

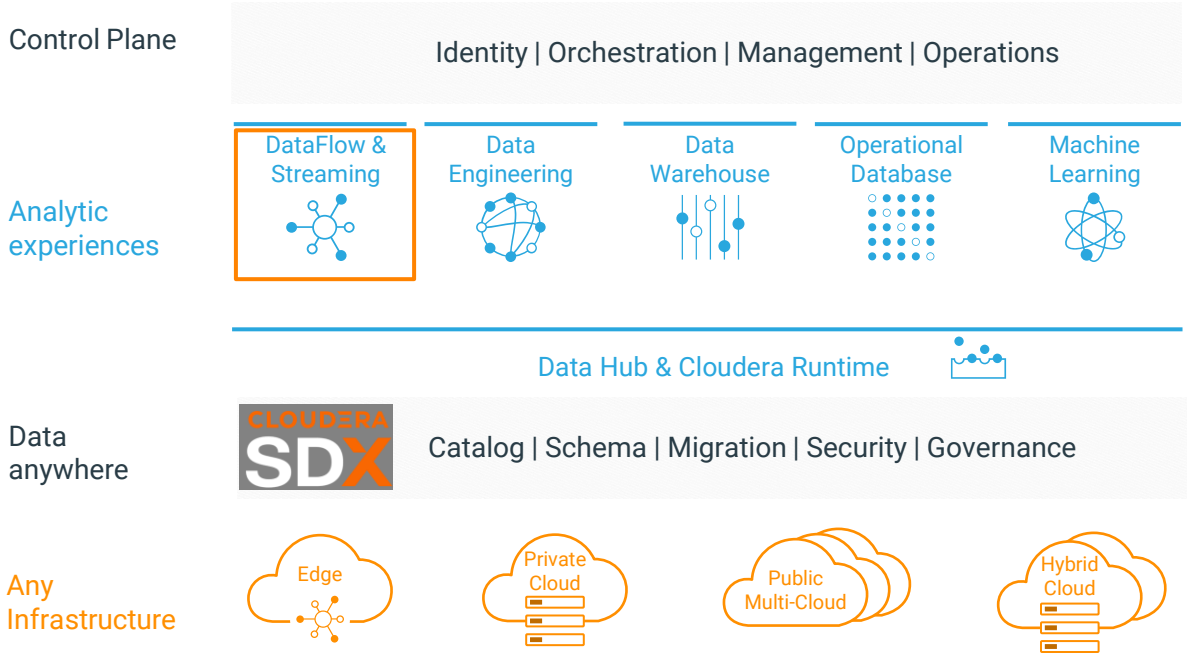


HOW DO I GET ACCESS TO A CLUSTER? (continued)



Cloudera Data Platform

- Public, private & hybrid cloud
- Shared data experience
- Powered by open source
- Analytics from the **Edge to AI**
- Unified data control plane



KEY DIFFERENTIATORS

100% open source technology – Only vendor with this strategy; prevents vendor lock-in

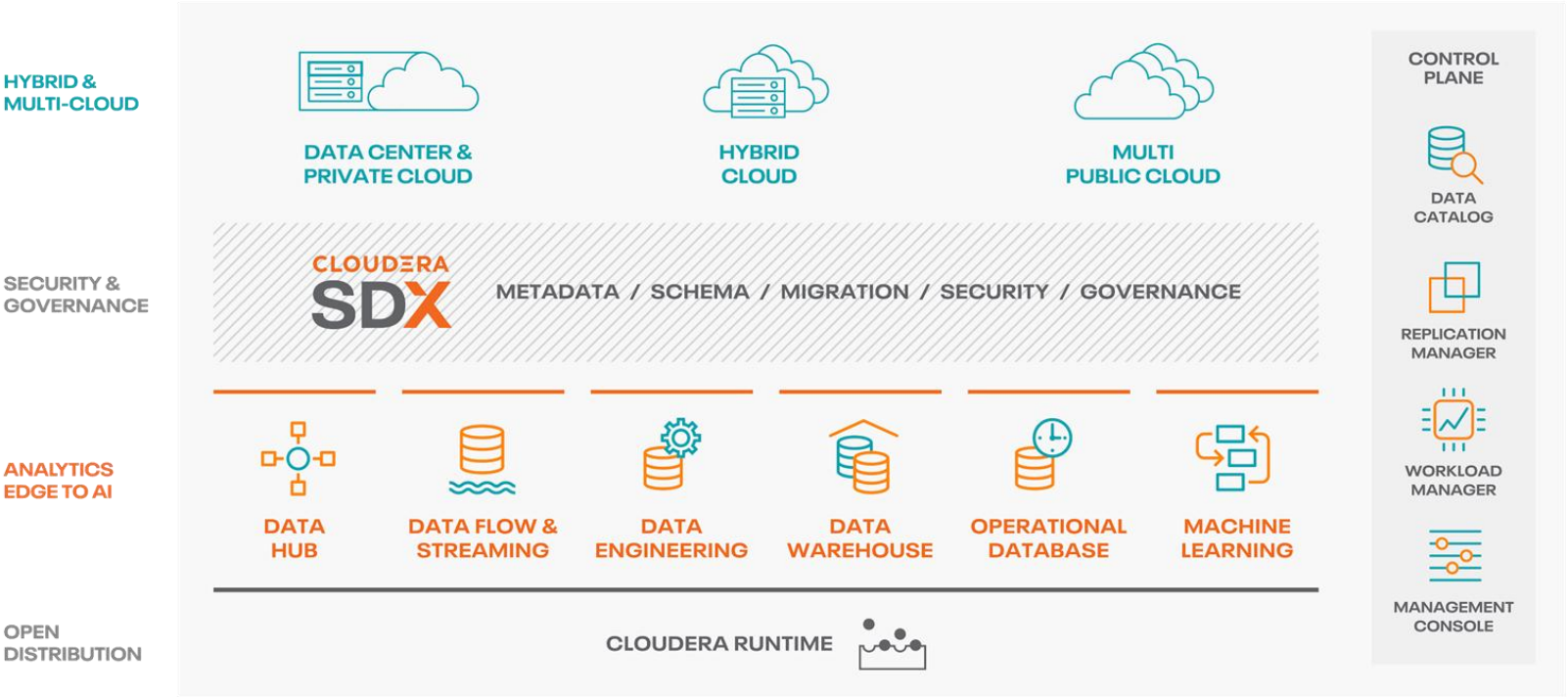
300+ pre-built processors – Only product to offer such comprehensive connectivity from edge to enterprise

3 Streaming analytics engines – Only vendor to offer a choice of three streaming analytics engines to customers for all their streaming architecture needs

Built-in data provenance – Only product in the market to offer out-of-the-box data provenance on data-in-motion

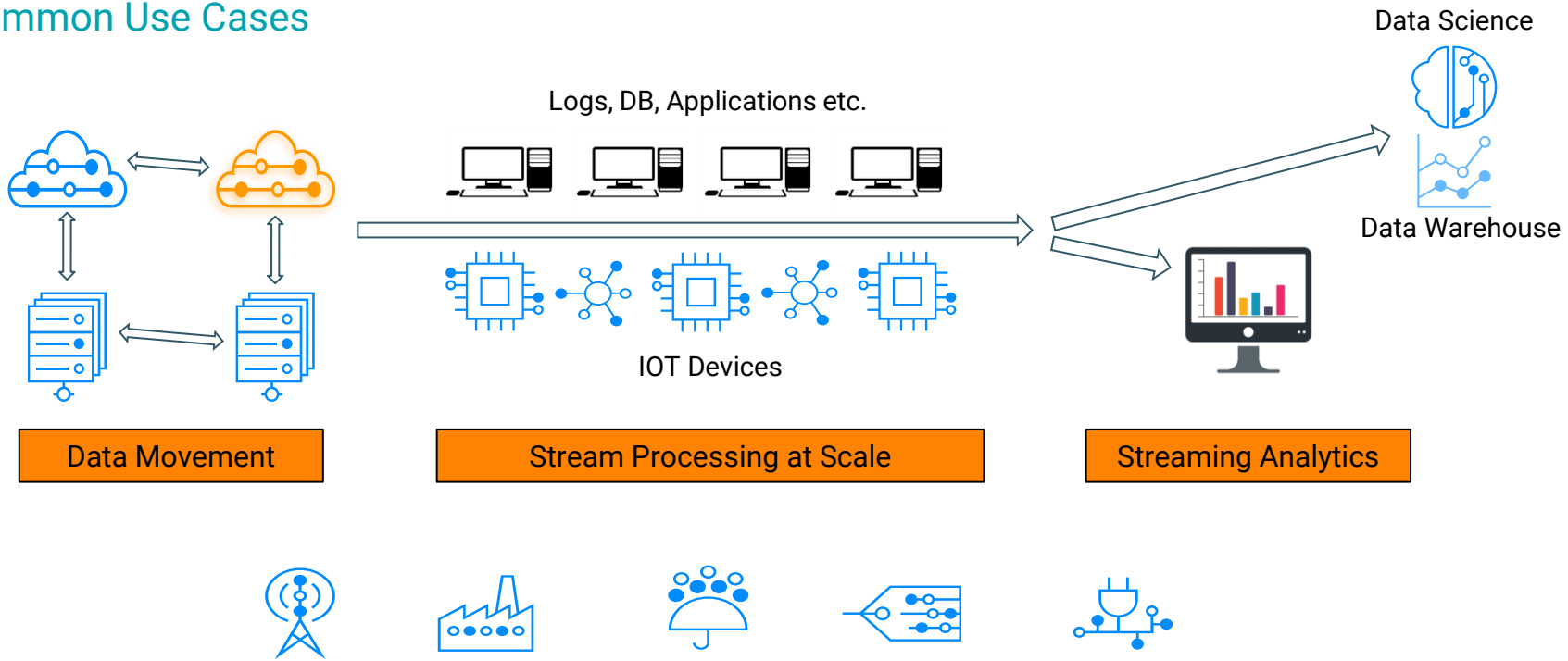
Comprehensive streaming platform – Only big data vendor to offer a comprehensive streaming platform from real-time data ingestion, transformation, routing to descriptive, prescriptive and predictive analytics.

CLOUDERA DATA PLATFORM



Cloudera Data Flow

Common Use Cases



KEY TAKEAWAYS

Cloudera Data Platform

CDF is the only comprehensive streaming data platform for your end-to-end data journey

SMM and SRM

SMM is the cure to “Kafka Blindness” with a single pane of glass across all Kafka clusters

SRM is the answer to Kafka Replicator

Enterprise Capabilities

Only CDF offers out-of-the-box unified security & governance from Edge to AI

KEY CUSTOMER CHALLENGES

DATA INGESTION

High-volume streaming sources

Multiple message formats

Diverse protocols

Multi-vendor devices

REAL-TIME INSIGHTS

Analyzing continuous and rapid streaming data

Handling high streaming data volumes

Running predictive models on streaming data

VISIBILITY

Monitor end-to-end streaming data flows

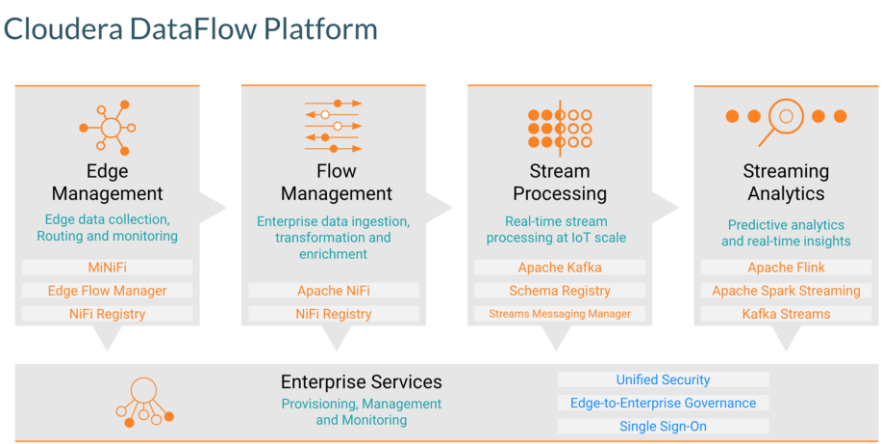
Troubleshoot bottlenecks

Understand consumption patterns

HISTORY OF CDF

Data-in-Motion:

- Comprehensive real-time streaming data platform
- Manage data-in-motion from edge-to-enterprise
- Power IoT-scale streaming architectures



Bring this to the edge with connected platforms

Enable next generation Modern Data Architecture

Mid-2000's
NiFi was developed and used at NSA

2015
Onyara is acquired
HDF is born

2018
Strong Streaming Platform
- Support for Kafka 2.0
- SMM is introduced

2019
Cloudera merger
Enable Edge Intelligence

Tomorrow:
Edge-to-AI

CDF Debuts as Strong Performer in 2019 Forrester Wave for Streaming Analytics

Key Highlights

- Cloudera has been named as a **Strong Performer in the Forrester Wave for Streaming Analytics, Q3 2019**.
- Our **current offering is considered to be on par if not superior to the key cloud players like AWS, Google and Microsoft.**
- **Our key competitors, Confluent and Streamsets, are missing from this wave**. This is a significant competitive advantage for us to press upon in the field.

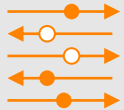
Data Flow in the Enterprise

Cloudera DataFlow Platform

Edge Management

Edge data collection, Routing and monitoring

- MiNiFi
- Edge Flow Manager
- NiFi Registry



Flow Management

Enterprise data ingestion, transformation and enrichment

- Apache NiFi
- NiFi Registry

Stream Processing

Real-time stream processing at IoT scale

- Apache Kafka
- Schema Registry

Streams Management

- Streams Messaging Manager
- Streams Replication Manager

Streaming Analytics

Predictive analytics and real-time insights

- Apache Flink
- Kafka Streams
- Spark Streaming

Enterprise Services

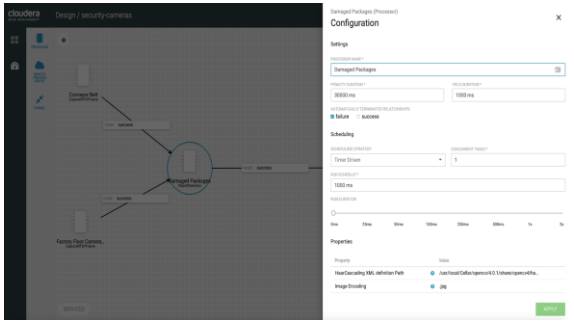
Provisioning, Management and Monitoring

- Unified Security
- Edge-to-Enterprise Governance
- Single Sign-On

Cloudera EDGE Management

Edge device data collection and processing with easy to use central command and control

Edge Flow Manager



APACHE NIFI minifi

A lightweight edge agent that implements the core features of Apache NiFi, focusing on data collection and processing at the edge

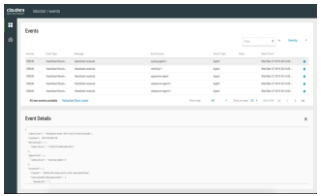
Flow Authorship

- Small footprint agent with MiNiFi
- Java and C++ agents
- Rich edge processors (edge collection & processing)
- End to end lineage and security

Flow Deployment

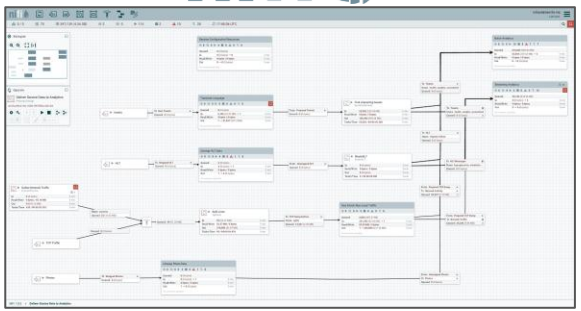
- Central Command and Control (C2)
- Design and deploy to thousands of agents
- Edge Applications lifecycle management
- Multitenancy with Agent classes
- Native integration with other CDF services

Flow Monitoring



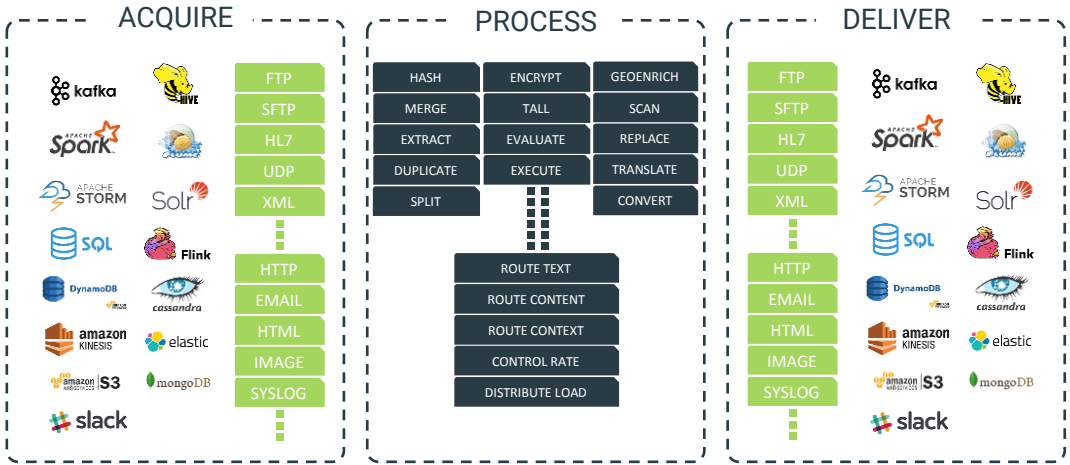
Cloudera Flow Management

Enable easy ingestion, routing, management and delivery of any data anywhere (Edge, cloud, data center) to any downstream system with built in end-to-end security and provenance



APACHE NIFI registry

Advanced tooling to industrialize flow development (Flow Development Life Cycle)



- Over 300 Prebuilt Processors
- Easy to build your own
- Parse, Enrich & Apply Schema
- Filter, Split, Merger & Route
- Throttle & Backpressure

- Guaranteed Delivery
- Full data provenance from acquisition to delivery
- Diverse, Non-Traditional Sources
- Eco-system integration

Stream Processing and Analytics

Providing simpler ways of complex event processing at scale for the enterprise

kafka + KStreams

"Pub & Sub"

Massively Scalable Publish -Subscribe Message Queue, API for building real-time microservices with Kafka Streams

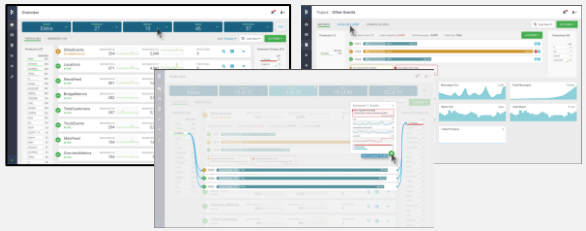
Flink

"Process/Analyze"

Real-Time, Distributed Data and Task Processing Engine

Streams Messaging Manager

"Manage & Monitor"



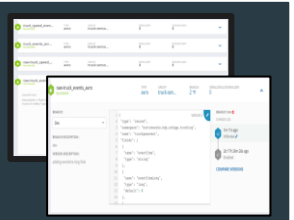
- Cure Kafka blindness and help the different streaming personas be more productive
- End-to-end integration with Ambari, Grafana, Ranger & Atlas
- Comprehensive REST service for open integration

Streams Replication Manager (SRM)

"Mirroring & DR"

Replicate data & configuration in Realtime, Smart clients for easy fail-over & fail-back, Support active active scenarios, Monitoring

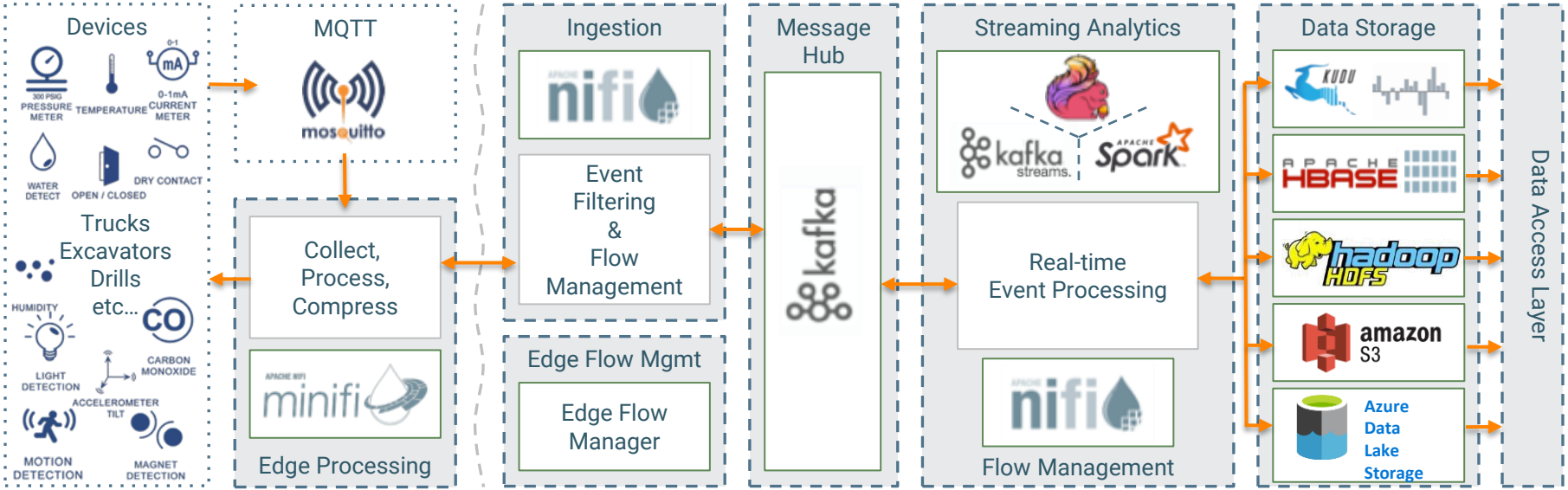
SCHEMA REGISTRY



"Manage"

A shared schemas repository that allows applications to save, retrieve and reuse schemas and interact with each other

And by the way we will be building a complete IoT use case



Cloudera Flow Management Apache NiFi

DATAFLOW MANAGEMENT:
The systematic process by which data is acquired from all producers and delivered to all consumers

NIFI OVERVIEW

WHY NIFI?

- Moving data is multifaceted in its challenges and these are present in different contexts at varying scopes
- Provide common tooling and extensions that are commonly needed but be flexible for extension
  - Leverage existing libraries and expansive Java ecosystem for functionality
  - Allow organizations to integrate with their existing infrastructure
- Empower folks managing your infrastructure to make changes and reason about issues that are occurring
  - Data Provenance to show context and data's journey
  - User Interface/Experience a key component

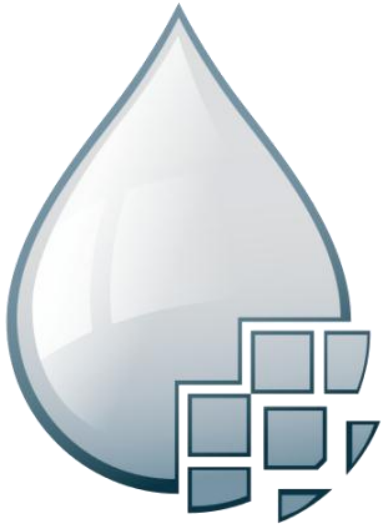
https://dzone.com/articles/apache-nifi-10-cheatsheet

THE NSA YEARS

- Created in **2006**
- Improved over eight years
  - Simple initial vision – Visio for real-time dataflow management
- Key Lessons Learned
  - What scale means – down, up, and out
  - The fearsome force known as Compliance Requirements
  - The power of provenance!
  - Operational best-practices and anti-patterns
- NSA donated the codebase to the ASF in late 2014

APACHE NIFI

Key features



- Guaranteed delivery
- Data buffering
  - Backpressure
  - Pressure release
- Prioritized queuing
- Flow specific QoS
  - Latency vs. throughput
  - Loss tolerance
- Data provenance
- Supports push and pull models

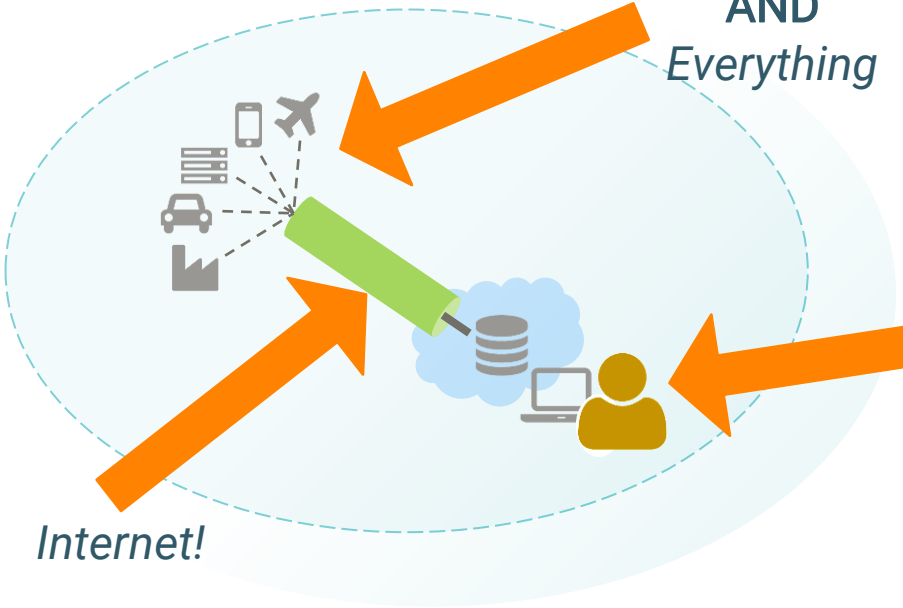
- Recovery/recording a rolling log of fine-grained history
- Visual command and control
- Flow templates
- Pluggable/multi-role security
- Designed for extension
- Clustering

What is Dataflow?

Moving some content from A to B

Content could be any bytes

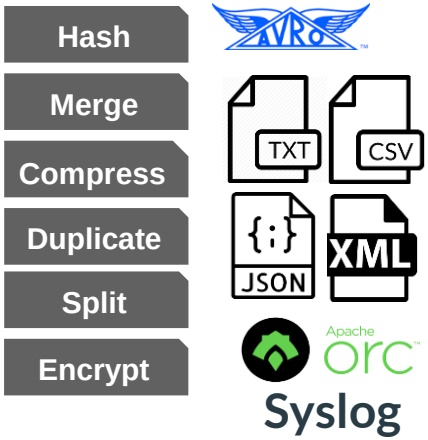
- Logs
- HTTP
- XML
- CSV
- Images
- Video
- Telemetry



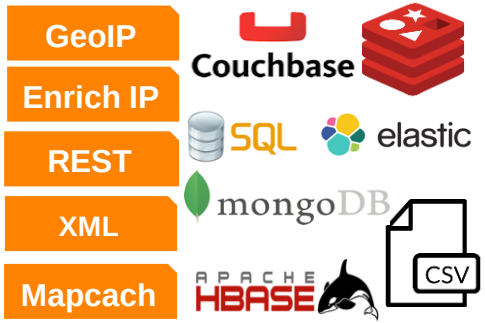
An Overview of Apache NiFi Capabilities

Data Ingest

Data Transformation

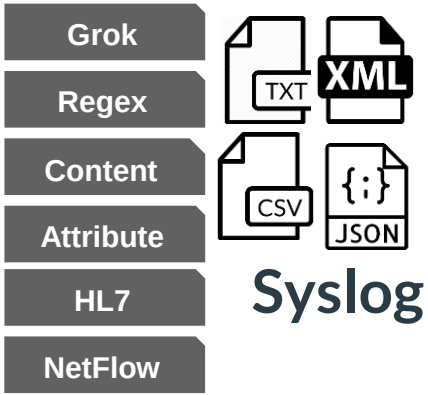


Data Enrichment

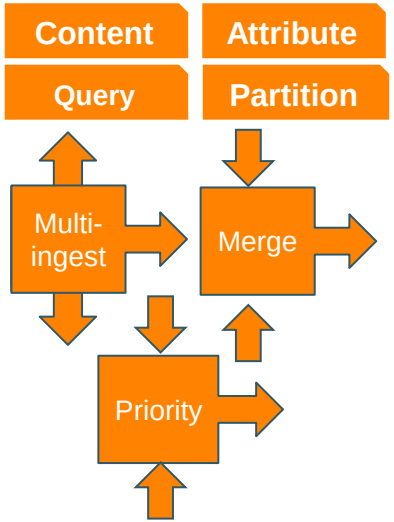


An Overview of Apache NiFi Capabilities

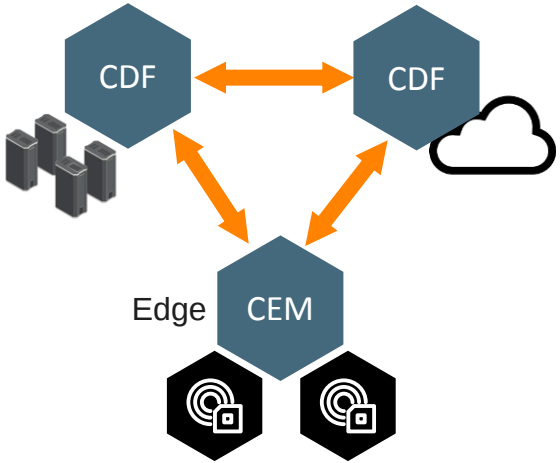
Parsing



Routing



Data Movement



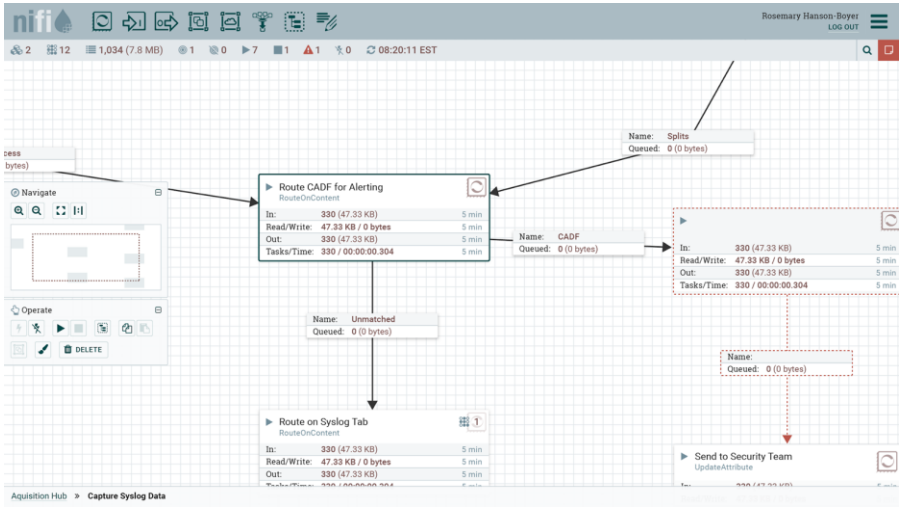
APACHE NIFI 1.9.2

Key New Features

Feature	Description	Apache JIRA
Nar Autoloading	Ability for Apache NiFi to load new nar bundles without restarting.	NIFI-5673
Remote Input/Output Ports	Remote input Input/Output Ports no longer require being placed in the root Process Group	NIFI-2933
PutKudu	Ability for Apache NiFi to write data into Apache Kudu on Cloudera Managed environments.	NIFI-5989
Kerberized Database Connection Pools	Ability for Apache NiFi to utilize JDBC connection pools that are secured via Kerberos (IE: Kudu & Impala)	NIFI-5985
Full Support for Cloud Processors	List too large for here. See release notes.	Multiple

High Level Capabilities

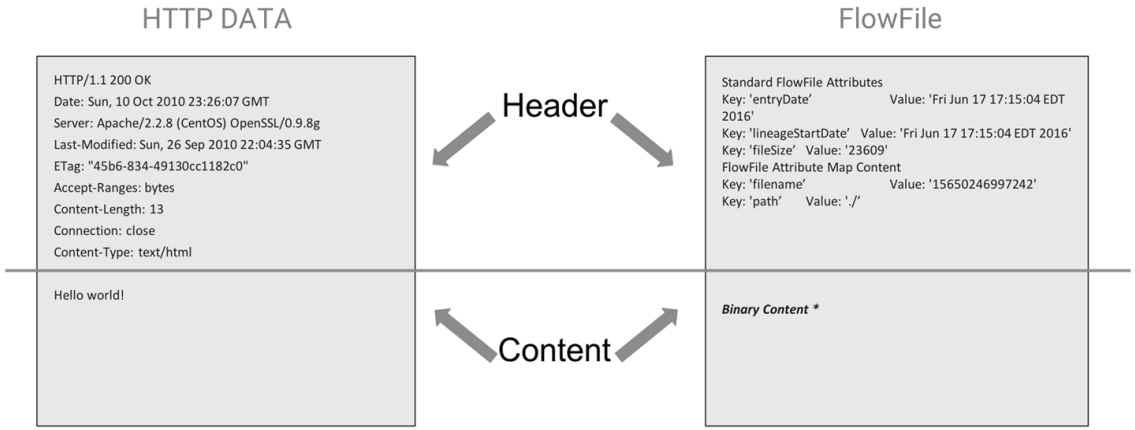
- Scale horizontal and vertically
  - Scale your data flow to millions event/s
  - Ingest TB to PB of data per day
- Adapt to your flow requirements
  - Back pressure & Dynamic prioritization
  - Loss tolerant vs guaranteed delivery
  - Low latency vs high throughput
- Secure
  - SSL, HTTPS, SFTP, etc.
  - Governance and data provenance
- Extensible
  - Build your own processors and Controller services (providers)
  - Integrate with external systems (Security, Monitoring, Governance, etc)



NIFI CORE CONCEPTS

NIFI—TERMINOLOGY

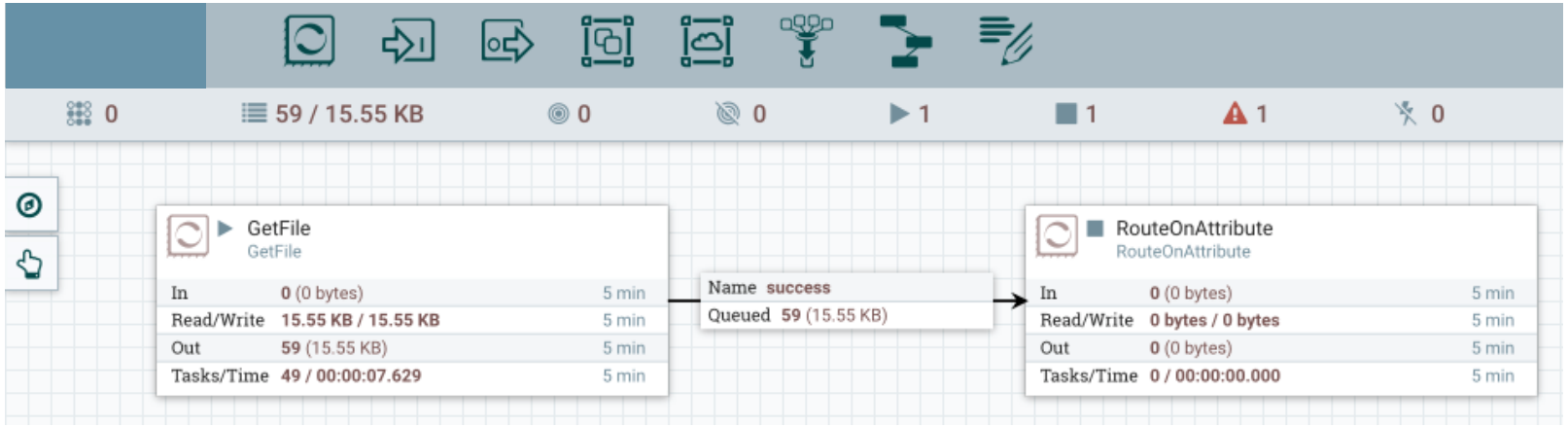
- FlowFile
  - Unit of data moving through the system
  - Content + Attributes (key/value pairs)
- Processor
  - Performs the work, can access FlowFiles
- Connection
  - Links between processors
  - Queues that can be dynamically prioritized
- Process Group
  - Set of processors and their connections
  - Receive data via input ports, send data via output ports



NIFI IS BASED ON FLOW BASED PROGRAMMING (FBP)

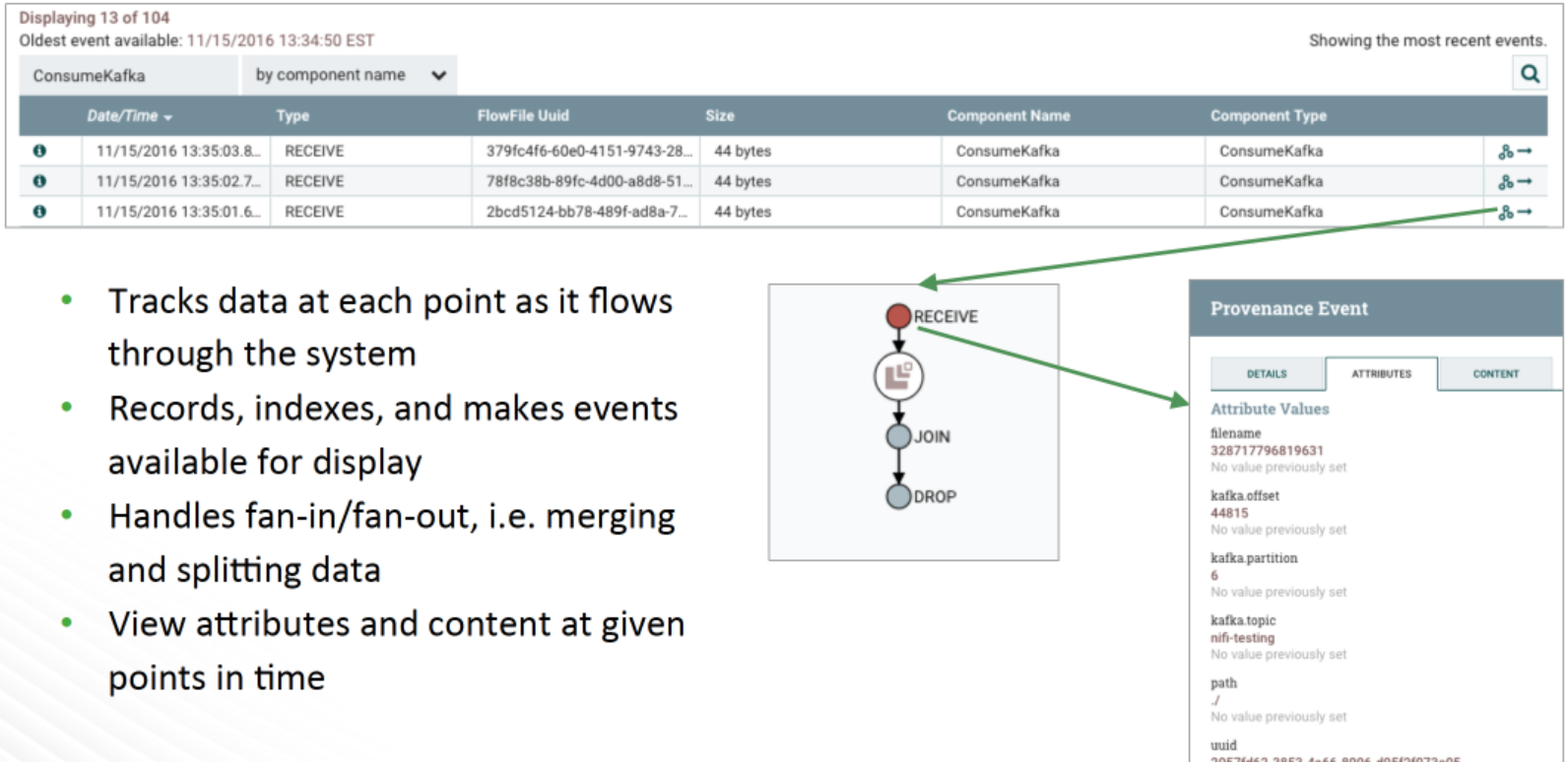
FBP Term	NiFi Term	Description
Information Packet	FlowFile	Each object moving through the system.
Black Box	FlowFile Processor	Performs the work, doing some combination of data routing, transformation, or mediation between systems.
Bounded Buffer	Connection	The linkage between processors, acting as queues and allowing various processes to interact at differing rates.
Scheduler	Flow Controller	Maintains the knowledge of how processes are connected, and manages the threads and allocations thereof which all processes use.
Subnet	Process Group	A set of processes and their connections, which can receive and send data via ports. A process group allows creation of entirely new component simply by composition of its components.

VISUAL COMMAND AND CONTROL



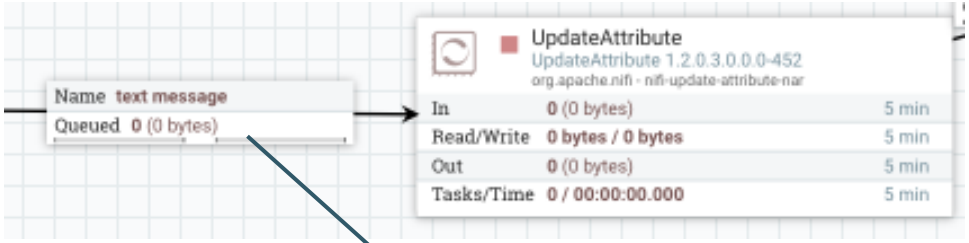
- Drag and drop processors to build a flow
- Start, stop, and configure components in real time
- View errors and corresponding error messages
- View statistics and health of data flow
- Create templates of common processor & connections

PROVENANCE/LINEAGE

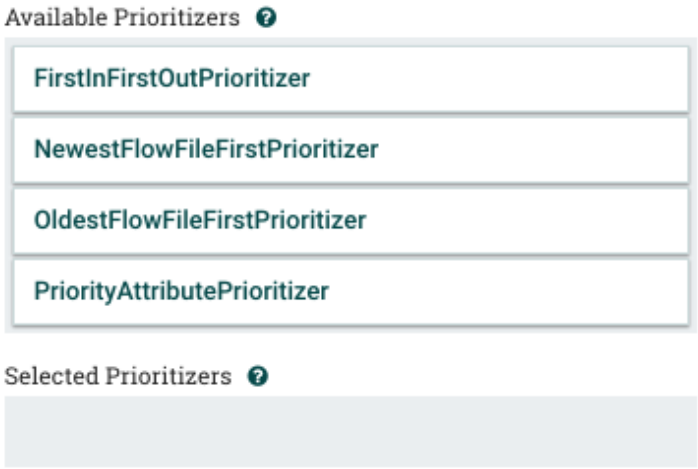


- Tracks data at each point as it flows through the system
- Records, indexes, and makes events available for display
- Handles fan-in/fan-out, i.e. merging and splitting data
- View attributes and content at given points in time

PRIORITIZATION

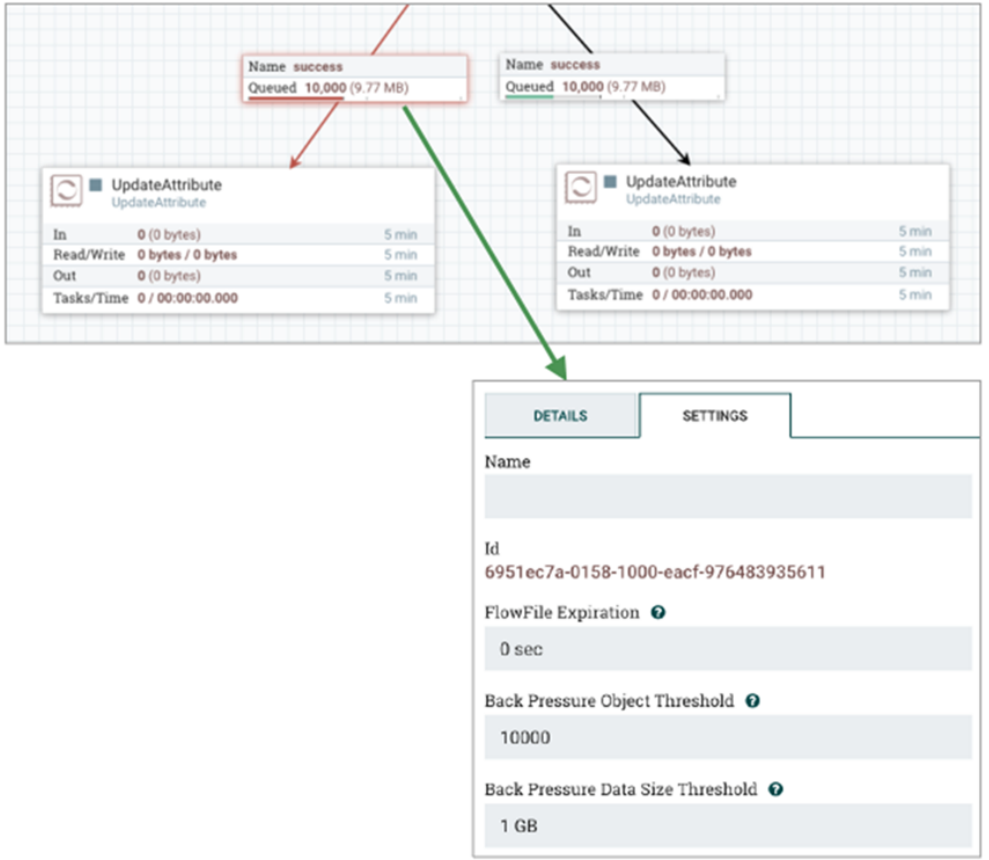


- Configure a prioritizer per connection
- Determine what is important for your data – time based, arrival order, importance of a data set
- Funnel many connections down to a single connection to prioritize across data sets
- Develop your own prioritizer if needed



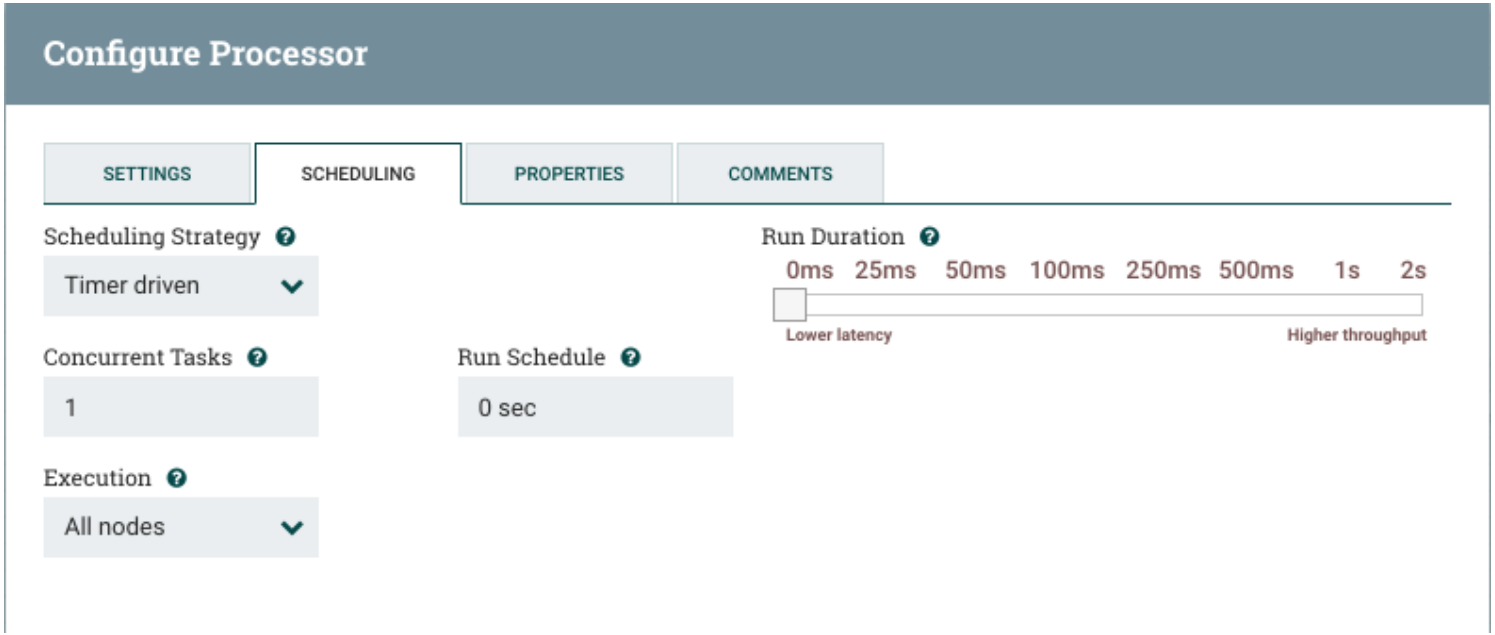
BACK PRESSURE

- Configure back-pressure per connection
- Based on number of FlowFiles or total size of FlowFiles
- Upstream processor no longer scheduled to run until below threshold



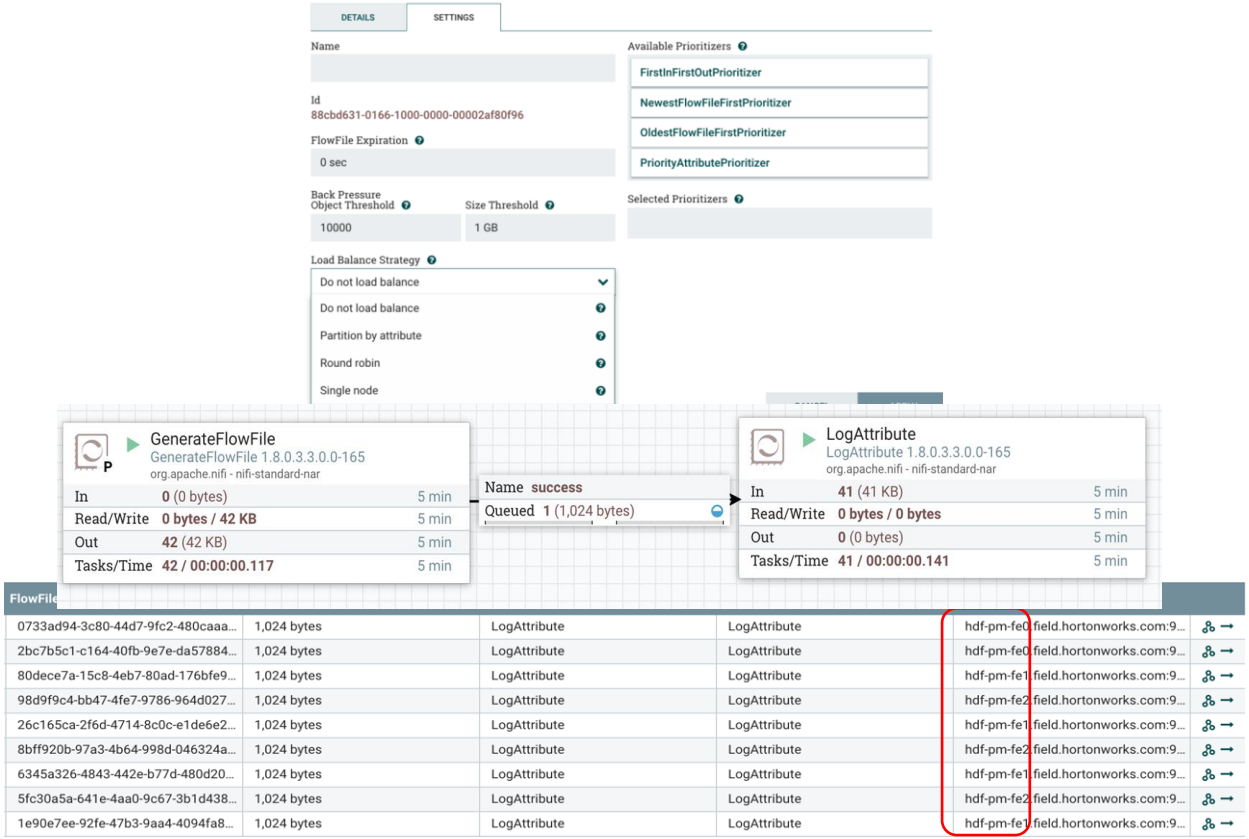
LATENCY VS. THROUGHPUT

Choose between lower latency, or higher throughput on each processor



NiFi Load Balancing

- Improve NiFi cluster throughput
- Defined at connection level
- Configurable balancing strategies
- Critical for scale up paradigm in Kubernetes
- Alleviates S2S balancing “hack” customers use



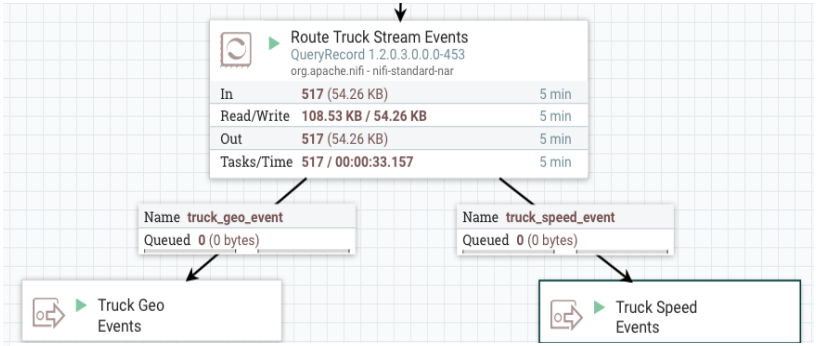
SQL BASED ROUTING WITH NiFi's QueryRecord Processor

QueryRecord Processor- Executes a SQL statement against records and writes the results to the flow file content.

CSVReader: Looking up schema from SR, it will converts CSV Records into ProcessRecords

SQL execution via Apache Calcite: execute configured SQL against the ProcessRecords for routing

CSVRecordSetWriter: Converts the result of the query from Process records into CSV for the for the flow file content



Property	Value
Record Reader	Routing-CSVReader
Record Writer	Routing-CSVWriter
Include Zero Record FlowFiles	false
Cache Schema	false
truck_geo_event	SELECT * FROM FLOWFILE where eventSource = 'truck_geo_event'
truck_speed_event	SELECT * FROM FLOWFILE where eventSource = 'truck_speed_event'

Do routing(routing geo and speed streams) using standard SQL as opposed to complex regular expressions.

Data Protection

- *Example*
  - Encrypt data
    - Can use asymmetric encryption for destination-only decrypt
    - Can use symmetric encryption for speed
  - Handles standard AES and GPG ciphers out of the box

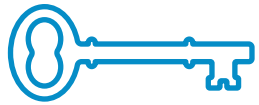
Decrypting AES data inline

View as: original

```
1  驸옙ω袴□擘[?][?]槔梗嘛嗷□럳□惹ㅜ萊鼬邝銈蜓旣□□□卟핔굄垮ㄥ講擘瓔쨍黔□贈瓏斃僑
```

View as: hex

```
0x00000000 CB 54 29 E4 BA 64 3B 1B 55 85 84 70 22 9B 77 03 .T)..d;.U..p".w.
0x00000010 4E 69 46 69 49 56 2C 19 3C B1 DA AF 45 65 F9 A9 NiFiIV,.<...Ee..
0x00000020 E8 D6 F7 B1 53 79 A1 11 81 83 96 FA E6 48 20 B5 ....Sy.......H .
0x00000030 CD AE 71 66 03 C7 20 11 C0 DD 65 9E 4A 54 59 4B ..qf.. ...e.JTYK
0x00000040 4E 6A CD D7 5D 85 7D 68 C6 AF EF 70 F8 23 E0 D6 Nj..].}h...p.#..
0x00000050 3D 95 8E                                       =..
```

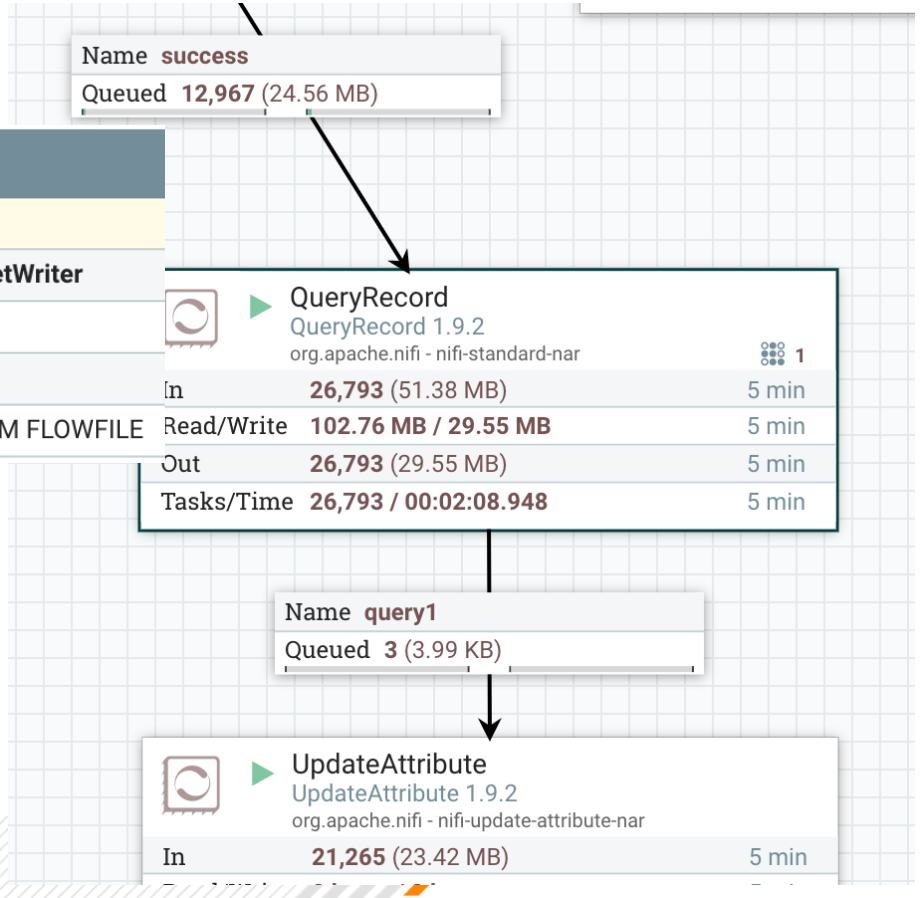


View as: original

```
1  This is a message at 2017/02/21 22:16:27.986Z
```

XML -> JSON or AVRO

Property	Value
Record Reader	**XMLReader**
Record Writer	**JsonRecordSetWriter**
Include Zero Record FlowFiles	**false**
Cache Schema	**true**
query1	SELECT * FROM FLOWFILE



Advanced XML Processing

Property	Value
Schema Access Strategy	**Infer Schema**
Schema Registry	AvroSchemaRegistry
Schema Name	${schema.name}
Schema Version	No value set
Schema Branch	No value set
Schema Text	${avro.schema}
Schema Inference Cache	No value set
Expect Records as Array	**false**
Attribute Prefix	No value set
Field Name for Content	No value set
Date Format	No value set
Time Format	No value set
Timestamp Format	No value set

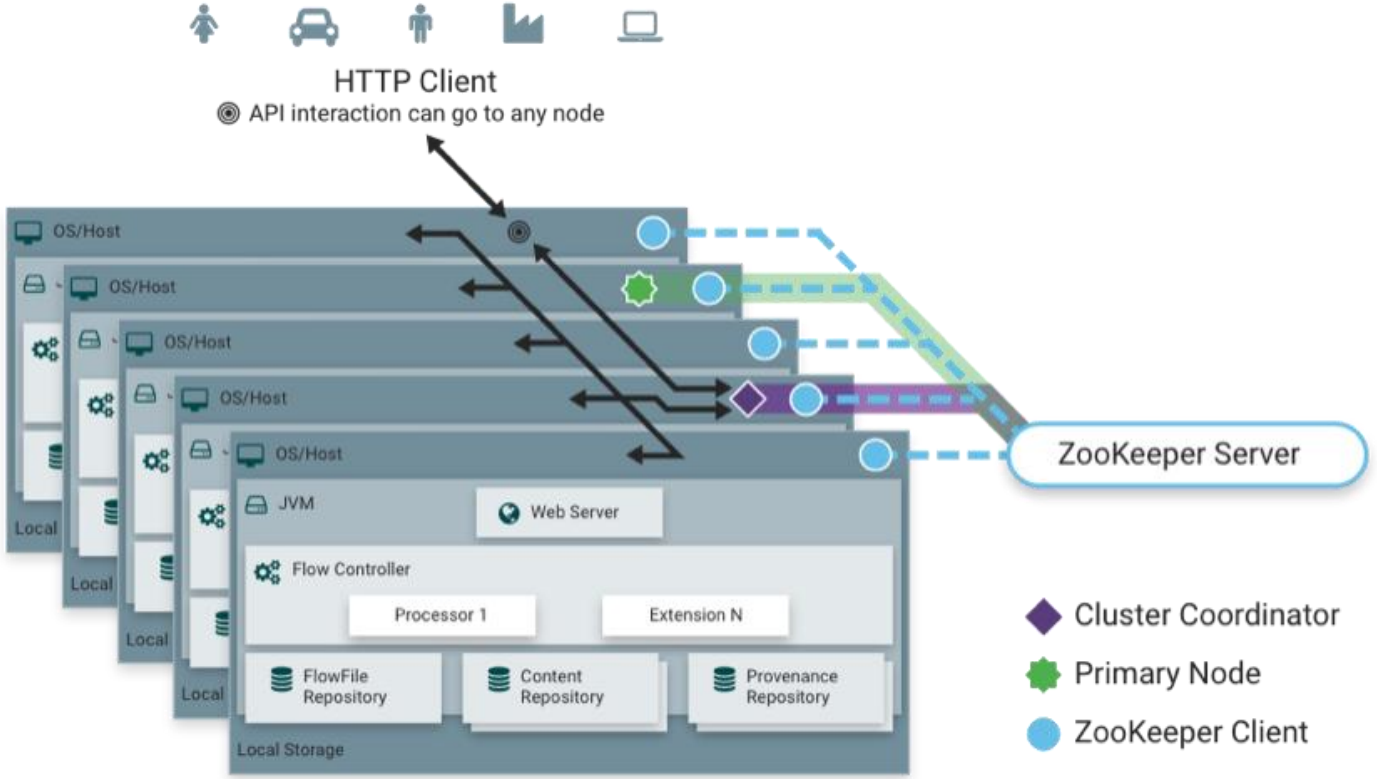
Property	Value
Record Reader	**XMLReader**
Record Writer	**JsonRecordSetWriter**
Include Zero Record FlowFiles	**false**
Cache Schema	**true**
query1	SELECT * FROM FLOWFILE

https://pierrevillard.com/2018/06/28/nifi-1-7-xml-reader-writer-and-forkrecord-processor/

https://www.datainmotion.dev/2019/03/advanced-xml-processing-with-apache.html

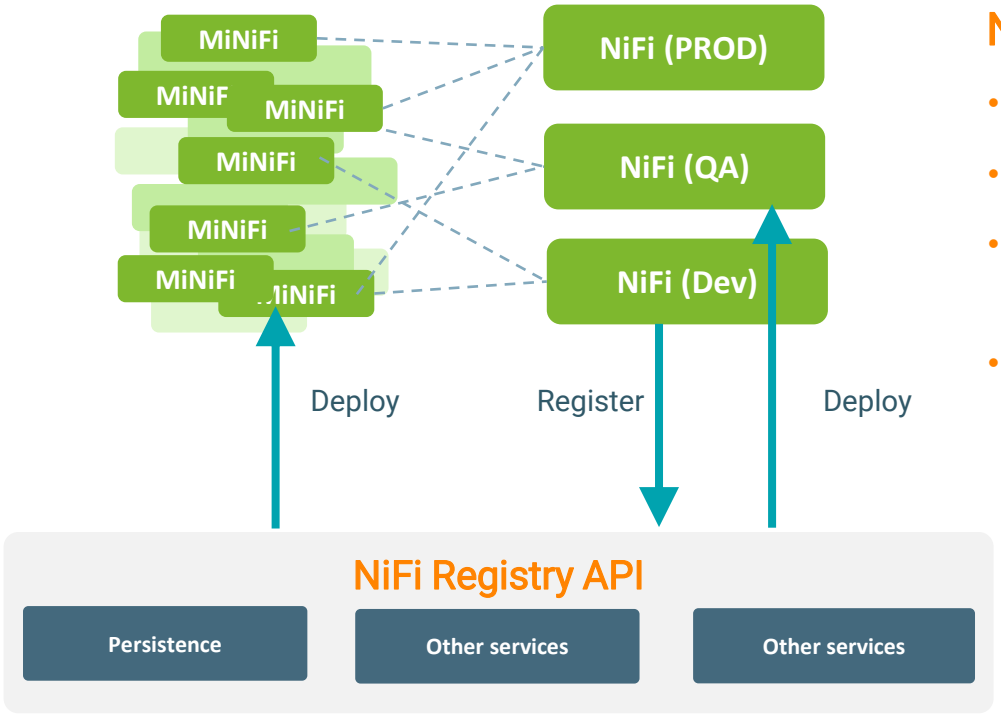
NIFI ARCHITECTURE

ARCHITECTURE



SOFTWARE DEVELOPMENT LIFECYCLE (SDLC)

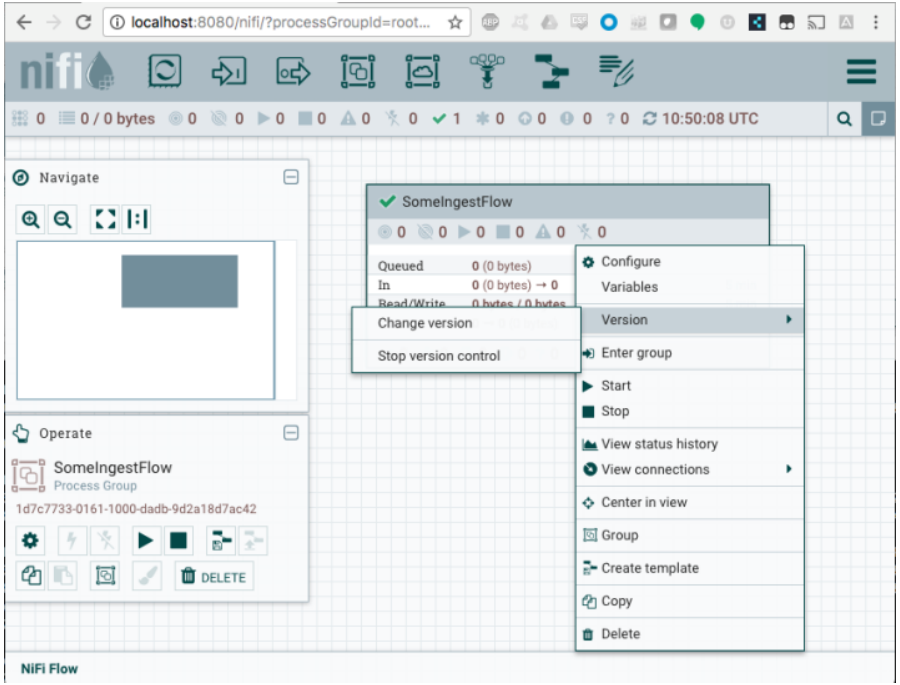
INCREASED DEVELOPER PRODUCTIVITY: APACHE NIFI REGISTRY



NiFi Registry

- Repository of versioned flows
- Portability
- Support multiple registries and interactions between them
- Design and deploy mechanism

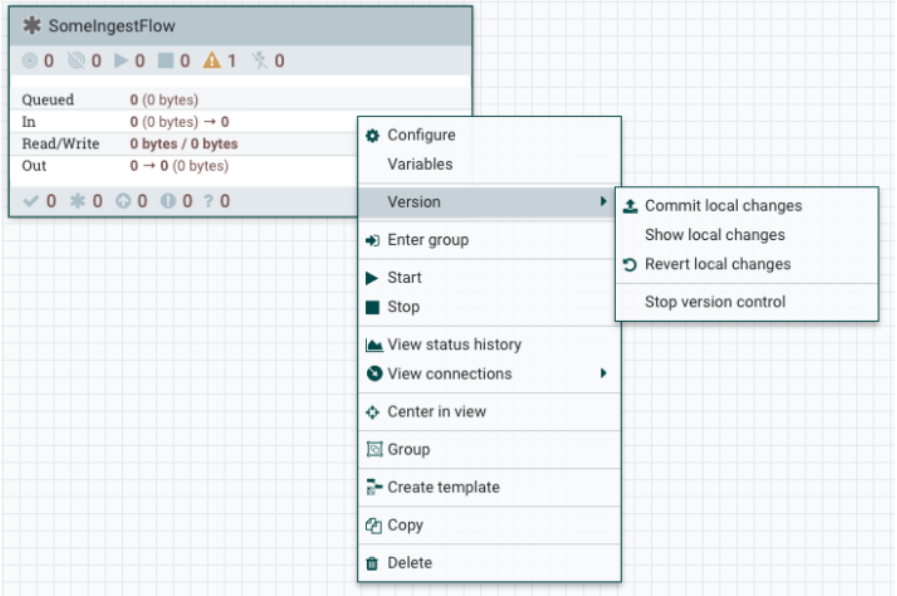
DESIGN & DEPLOY COMPLEMENTING COMMAND & CONTROL



- SDLC Dev: Place Process Groups under Version Control
- Make changes and commit to new version
- Roll versions back or forward

https://community.cloudera.com/t5/Community-Articles/Building-a-Custom-Apache-NiFi-Operations-Dashboard-Part-1/ta-p/249060

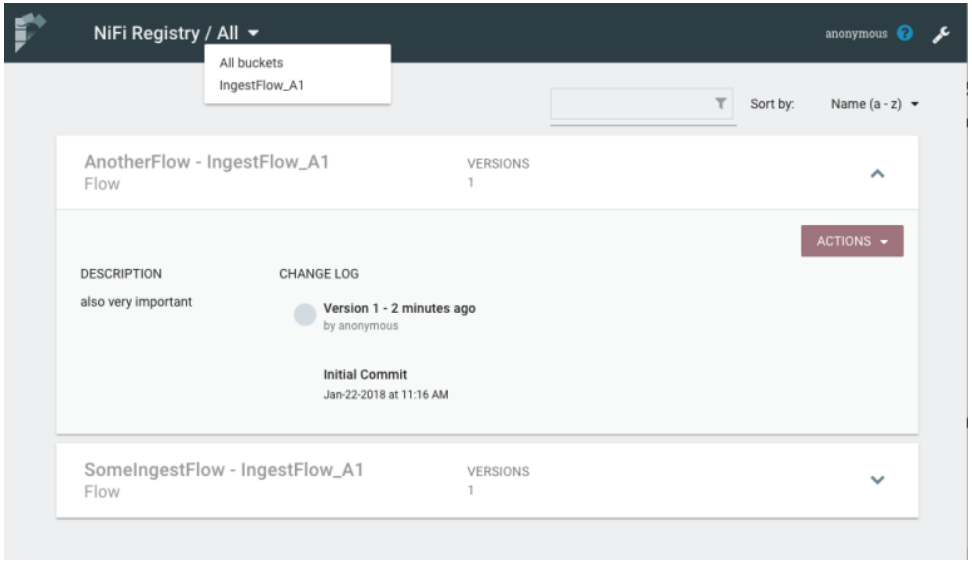
DESIGN & DEPLOY COMPLEMENTING COMMAND & CONTROL



- Get Notifications of local changes or new versions available in Repository
- Revert or Commit local changes via the GUI or Rest-API
- Use Rest-API to integrate with Jenkins, etc.

https://medium.com/@abdelkrim.hadjidj/fdlc-towards-flow-development-life-cycle-with-nifi-registries-82e1ee866fab

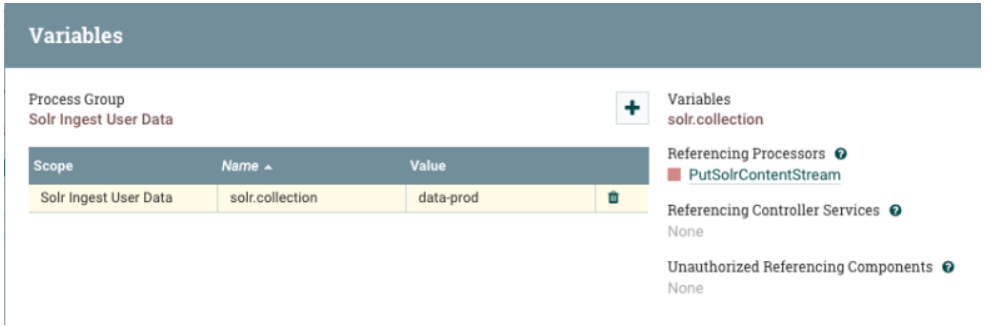
INTEGRATED FLOW REGISTRY SERVICE



- Integrated Flow Registry Service
- Sharable between NiFi environments for Dev/UAT/Prod promotion
- API or GUI driven
- Can be integrated with Enterprise Version Control e.g. GitLab
- 'Buckets' of Flows for security and access control

https://community.cloudera.com/t5/Community-Articles/More-DevOps-for-HDF-Apache-NiFi-Registry-and-Friends/ta-p/248668

INTEGRATED VARIABLE REGISTRY SERVICE



- Integrated Variable Registry
- Sets of key:value pairs available on every Process Group
- Referenced with NiFi Expression Language
- Dynamically changeable at runtime
- Use within Versioned Flows to set Environment Variables
- GUI or API driven

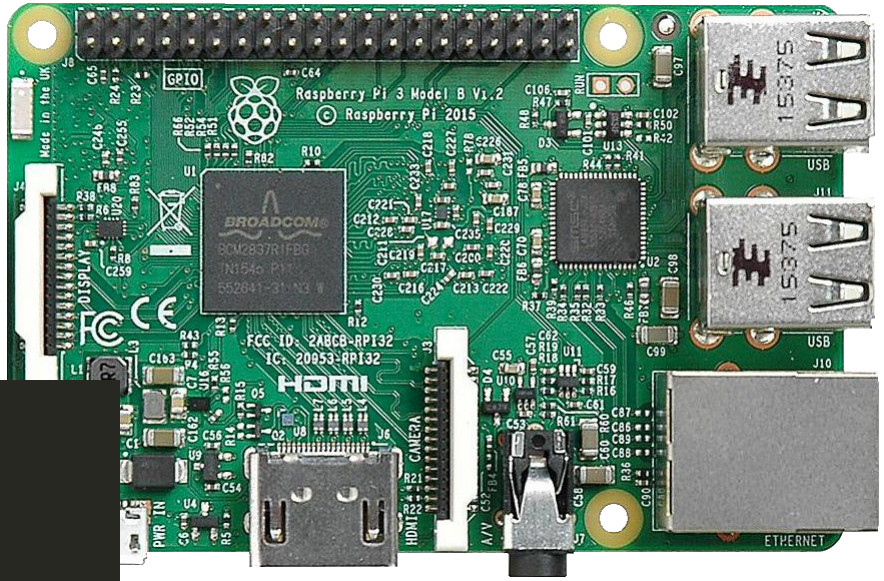
Mini NiFi (MiNiFi)

Apache NiFi Solves Everything*

- Runs on JVM
- Provides UI for flow design & monitoring
- Security built-in
  - TLS, authentication/authorization, encrypted data
- Handles practically any format/protocol

Apache NiFi for IoT

- NiFi supports AMQP, MQTT, UDP, TCP, HTTP(S), CEF, JMS, (S)FTP, AWS, Azure, GC, etc
- With a little pruning, NiFi can run on a Raspberry Pi



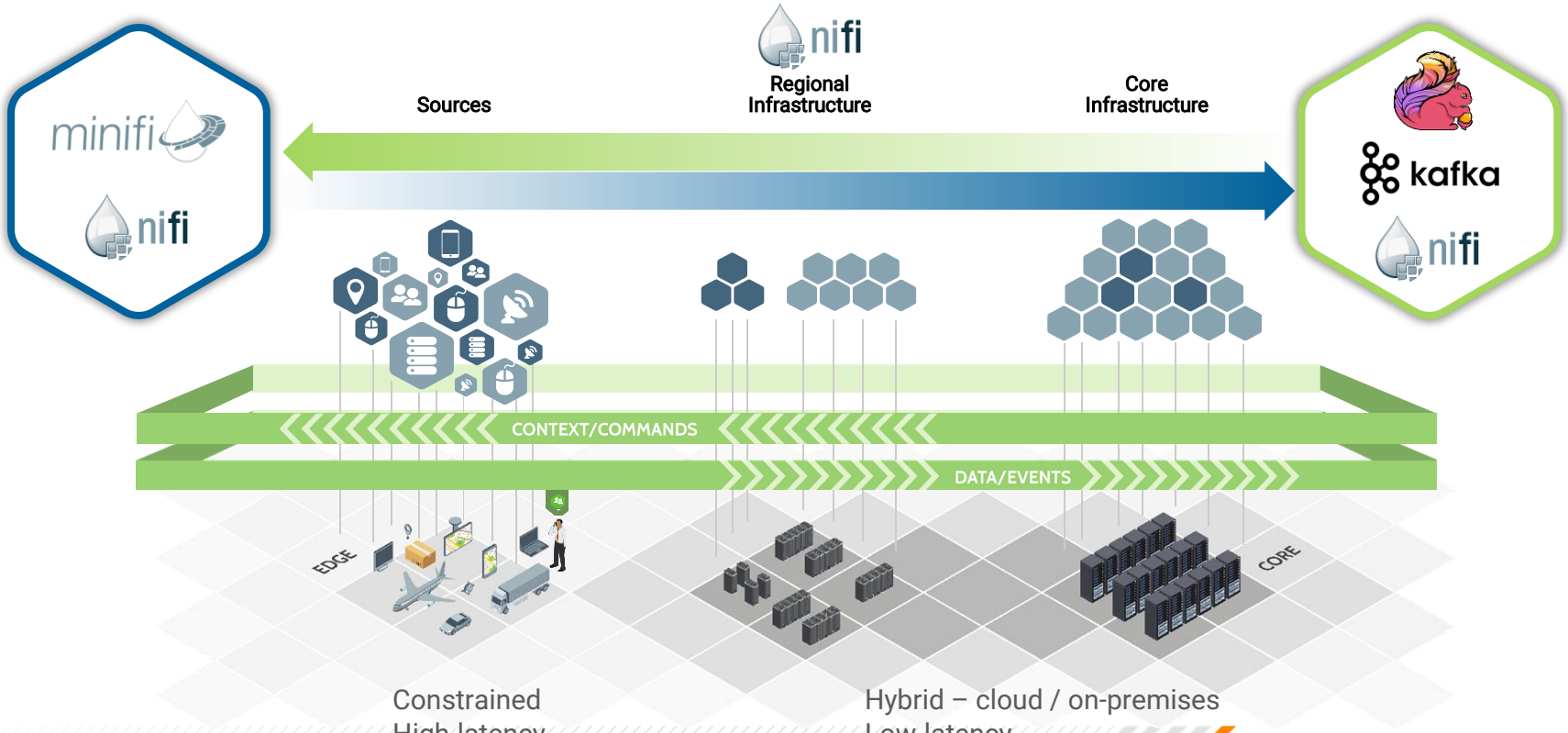

```
.
├── bootstrap
├── jcl-over-slf4j-1.7.12.jar
├── jul-to-slf4j-1.7.12.jar
├── log4j-over-slf4j-1.7.12.jar
├── logback-classic-1.1.3.jar
├── logback-core-1.1.3.jar
├── nifi-api-0.6.1.jar
├── nifi-documentation-0.6.1.jar
├── nifi-framework-nar-0.6.1.nar
├── nifi-html-nar-0.6.1.nar
├── nifi-http-context-map-nar-0.6.1.nar
├── nifi-jetty-bundle-0.6.1.nar
├── nifi-kerberos-iaa-providers-nar-0.6.1.nar
├── nifi-ldap-iaa-providers-nar-0.6.1.nar
├── nifi-nar-utils-0.6.1.jar
├── nifi-properties-0.6.1.jar
├── nifi-provenance-repository-nar-0.6.1.nar
├── nifi-runtime-0.6.1.jar
├── nifi-scripting-nar-0.6.1.nar
├── nifi-ssl-context-service-nar-0.6.1.nar
├── nifi-standard-nar-0.6.1.nar
├── nifi-standard-services-api-nar-0.6.1.nar
├── nifi-update-attribute-nar-0.6.1.nar
└── slf4j-api-1.7.12.jar
```

Edge Challenges

- Limited computing capability
- Limited power/network
- Restricted software library/platform availability
- No UI
- Physically inaccessible
- Not frequently updated
- Competing standards/protocols
- Scalability
- Privacy & Security

Apache NiFi is a an eco-system

End to End Data Management



OUTSIDE THE COMFORTS OF THE DATA CENTER

Realities of computing

- Limited computing capability
- Limited power/network
- Restricted software library/platform availability
- No UI
- Physically inaccessible
- Not frequently updated
- Competing standards/protocols
- Scalability
- Privacy & Security

Apache NiFi Subproject: MiNiFi

- Get the key parts of NiFi close to where data begins and provide bidirectional communication
- NiFi lives in the data center — give it an enterprise server or a cluster of them
- MiNiFi lives as close to where data is born and is a guest on that device or system
  - IoT
  - Connected car
  - Legacy hardware

Why build MiNiFi?

- NiFi is big
  - 1.7.0 release is 1.2 GB compressed
  - Can be modified to run in restricted environments, but requires manual surgery
  - Provides UI, provenance query, etc.
  - Runs on dedicated machines/clusters — “owns the box”
- MiNiFi lives at the edge
  - No UI
  - 0.5.0 Java binary is 67 MB, 0.5.0 C++ binary is 4.8 MB **(0.2.0 fits on a floppy disk)**
  - “Good guest”

APACHE MINIFI C++ 0.6.0

Key New Features

Feature	Description	Apache JIRA
NiFi JNI Bindings	Ability to run any native NiFi processor via JNI. Makes all NiFi processors available to MiNiFi C++	MINIFICPP-740
Native Python Processors	Ability to write native MiNiFi C++ processors in Python	MINIFICPP-750
PublishKafka	Ability to write to secured Kafka instance	MINIFICPP-731
CoAP Support	Ability to communicate with EFM server over CoAP. Drastically reduces the network impact of heartbeats	MINIFICPP-759
Windows Support	Ability to install MiNiFi via a MSI	MINIFICPP-700

MiNiFi Agents

- C++ and Java agents
- Security & Data provenance
- Guaranteed delivery
- Data buffering, Backpressure
- Prioritized queuing, Flow QoS
- Loss tolerance
- TensorFlow support of Edge AI
- Supports IoT ecosystem

C++ and Java MiNiFi capabilities

Choosing the right agent

MiNiFi C++

- Smaller footprint
- Can work without JVM
- JNI support for all NiFi processors
- Native Python processors
- Lowest network utilization
- More performant
- More processors
  - GPS, MQTT, Camera, etc

MiNiFi Java

- Fully supports all NiFi components
- Requires a JVM
- Benefits from JVM monitoring
- More easily deployed
- Easier to update
- Easier to develop custom features for clients

C++ and Java MiNiFi capabilities

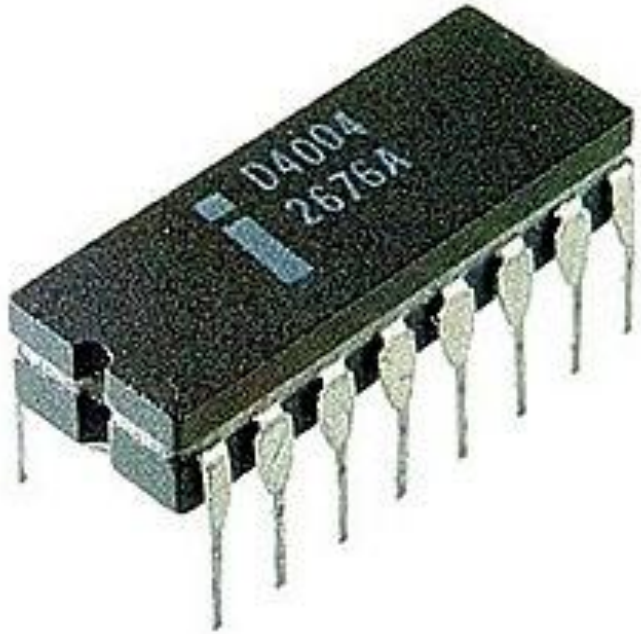
Choosing the right agent continued

- The two agents are 80% the same thing
- Java agent was short term solution while working on C++ agent
- C++ agent is now superior
- Java agent is still very valuable
  - Inherits all NiFi enhancements … thanks Java!
- Start with C++ agent and move to Java if needed
- Eliminate customer confusion by just saying “agent” and prescribing an agent

MiNiFi

Where not to position

- Never position MiNiFi to run on low level hardware (microprocessors)
  - * Caveat: we are working on a C implementation for this … not there yet.
- Requires local storage for data resiliency
- Requires Linux or Windows
- Requires networking interfaces



MiNiFi

Where to properly position

- IoT gateways with plethora of interfaces
- Mission critical servers where lightweight footprint for data collection is needed.
  - “Unclaimed environment”
- Ephemeral data collection endpoints
  - Containers, cloud instances, FaaS functions

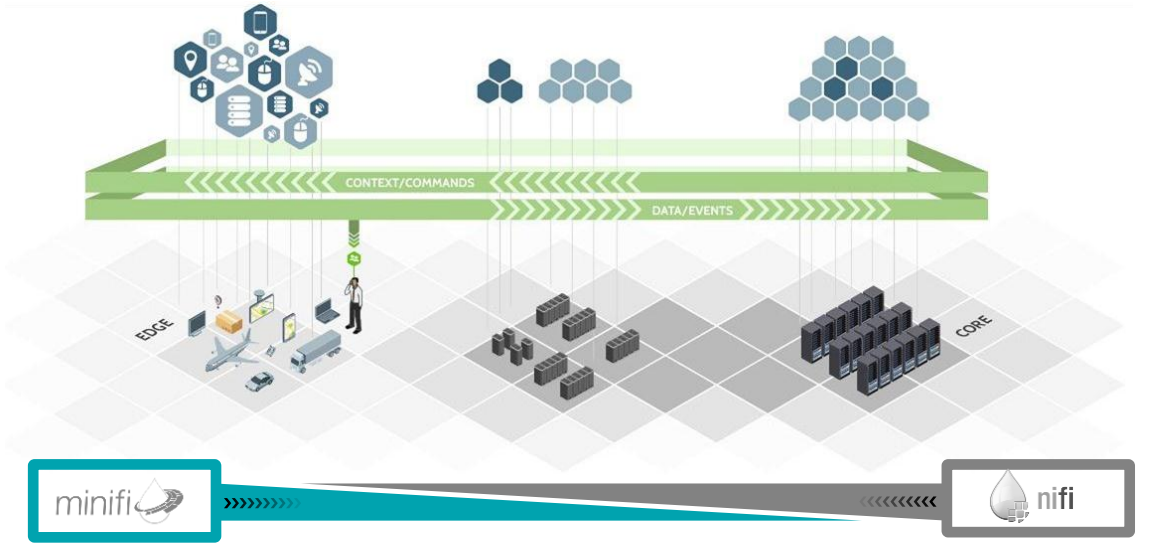
MiNiFi Exfil

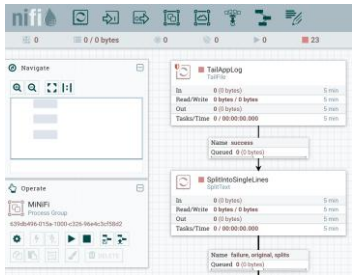
Getting Data from the Edge to the Core

- Site-to-Site
  - NiFi protocol
  - Two implementations
    - Raw socket
    - HTTP(S)
  - Secured with mutual authentication TLS
- *HTTP(S), (S)FTP, JMS, Syslog, File, Email, Process also available*

How Does MiNiFi Interact With NiFi?

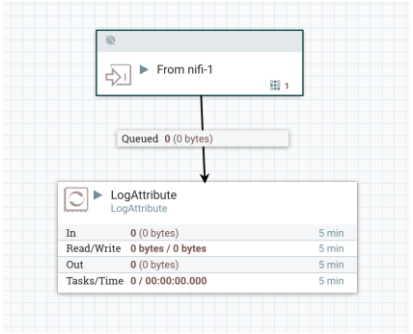
- MiNiFi
  - Receive flows
  - Collect data
  - Send for processing
- NiFi
  - Design flows
  - Aggregate data from many sources
  - Perform routing / analysis / SEP

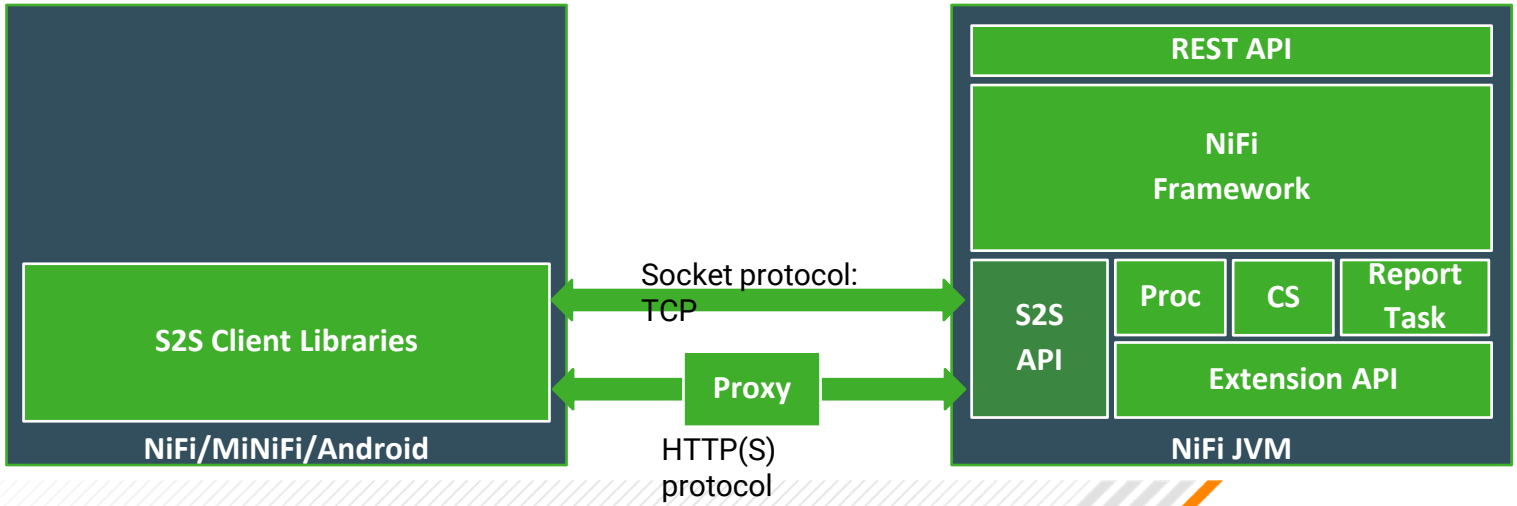




NiFi Site-to-Site (S2S)

- Guaranteed delivery
- 2-way SSL Encryption
- Built-in proxy capabilities





MINIFI: PRECEDENT FROM NIFI

A quick look at NiFi site to site

- Provides the semantics between two NiFi components across network boundaries
  - A custom protocol for inter-NiFi communication
  - Secure, Extensible, Load Balanced & Scalable Delivery to Cluster

- Extracted out to a client library which powers integration into popular frameworks like Apache Spark, Apache Storm, Apache Flink, and Apache Apex

- Attributes and the FlowFile format maintained

https://nifi.apache.org/docs/nifi-docs/html/user-guide.html#site-to-site

MINIFI: PRECEDENT FROM NIFI

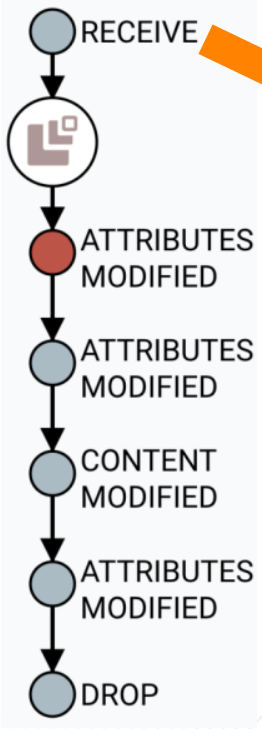
A deeper dive into provenance

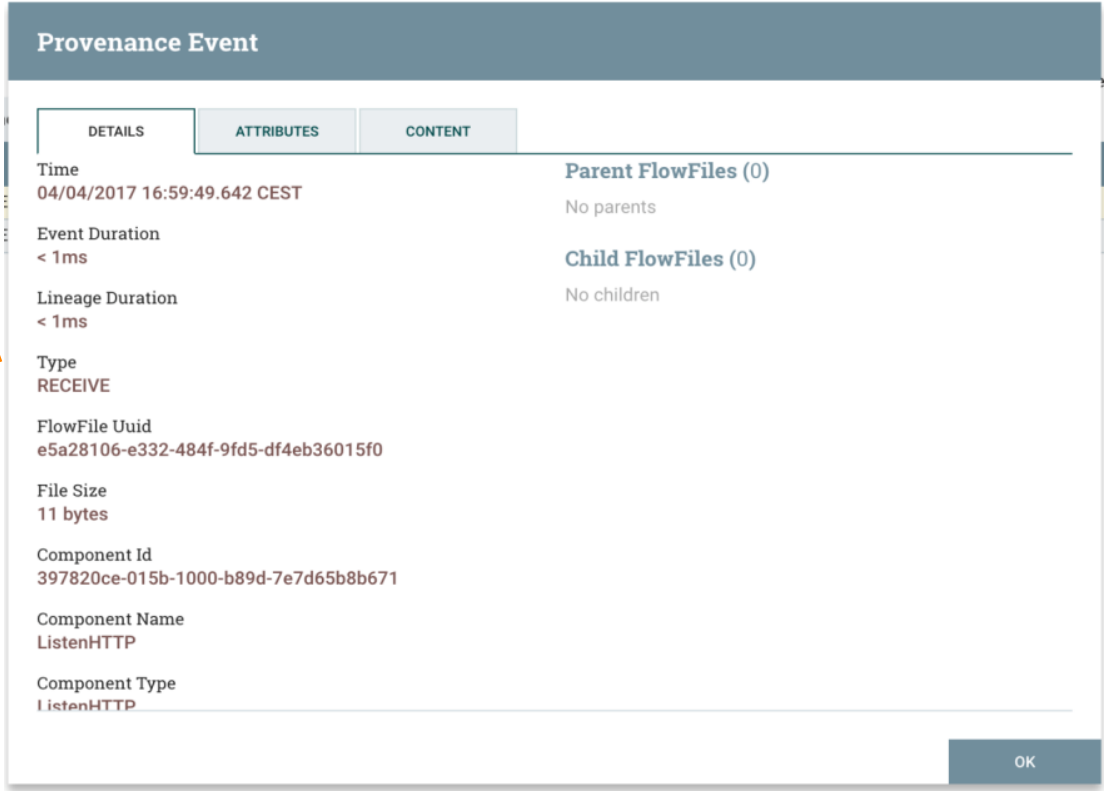
- Fine-grained, event level access of interactions with FlowFiles
  - CREATE, RECEIVE, FETCH, SEND, DOWNLOAD, DROP, EXPIRE, FORK, JOIN …
- Captures the associated attributes/metadata at the time of the event
- A map of a FlowFile's journey and how they relate to other FlowFiles in a system
  - MiNiFi enables us to get more and further illuminate the map of data processing

http://nifi.apache.org/docs/nifi-docs/html/user-guide.html#data-provenance

MINIFI: PRECEDENT FROM NIFI

RECEIVE event





APACHE NIFI - MINIFI

Departures from NiFi in getting the right fit

- The feedback loop is longer and not guaranteed
  - Removal of Web Server and UI

- Declarative configuration
  - Lends itself well to CM processes
  - Extensible interface to support varying formats
    - Currently provided in YAML

- Reduced set of bundled components

APACHE MINIFI: SCOPING

Provide all the key principles of NiFi in varying, smaller footprints

- ***Go small***: Java – *Write once, run anywhere**
  - Feature parity and reuse of core NiFi libraries
- ***Go smaller***: C++ – *Write once**, run anywhere*

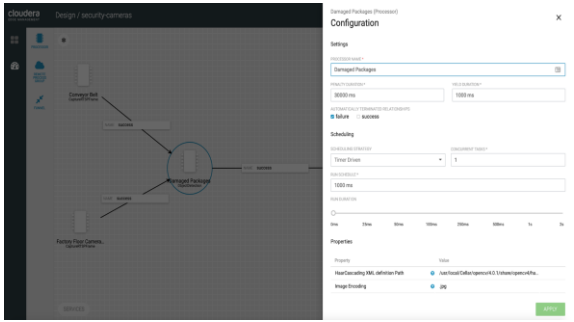
- ***Go smallest***: *Write n-many times, run anywhere*
  Language libraries to support tagging, FlowFile format, Site to Site protocol, and provenance generation without a processing framework
  - Mobile: Android & iOS
  - Language SDKs

Edge Flow Manager

Cloudera EDGE Management

Edge device data collection and processing with easy to use central command and control

Edge Flow Manager



APACHE NIFI minifi

A lightweight edge agent that implements the core features of Apache NiFi, focusing on data collection and processing at the edge

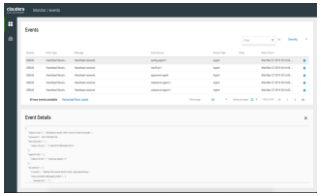
Flow Authorship

- Small footprint agent with MiNiFi
- Java and C++ agents
- Rich edge processors (edge collection & processing)
- End to end lineage and security

Flow Deployment

- Central Command and Control (C2)
- Design and deploy to thousands of agents
- Edge Applications lifecycle management
- Multitenancy with Agent classes
- Native integration with other CDF services

Flow Monitoring



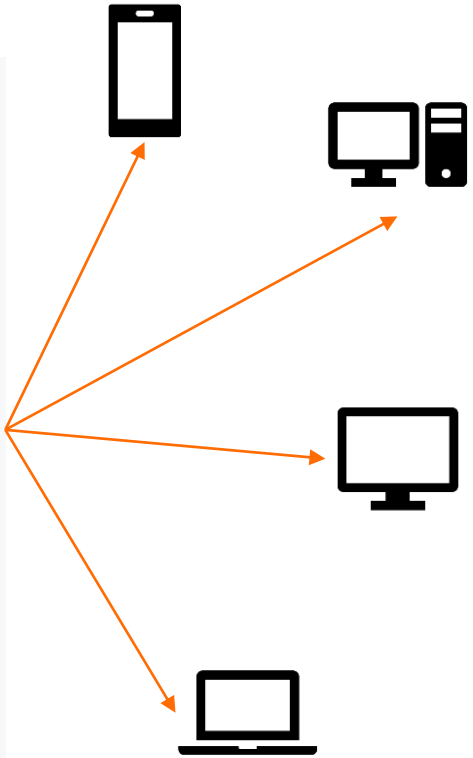
EFM

Flow Authorship - Previously

1. SSH to server
2. Enter credentials
3. SCP files to server
4. Stop MiNiFi Agent
5. Copy new config.yml file
6. Prep agent for restart
7. Restart MiNiFi agent
8. Tail logs to validate working
   a. If not start from scratch!

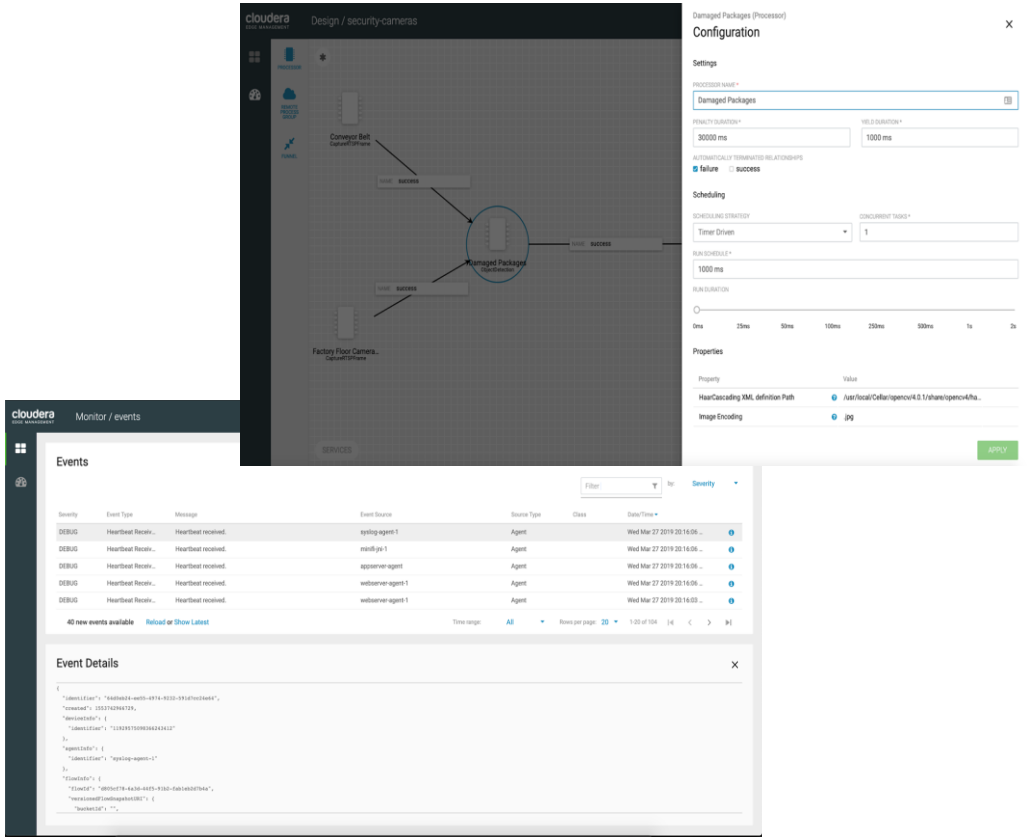
Author Flow Manually

```
Flow Controller:
    id: 471deef6-2a6e-4a7d-912a-81cc17e3a205
    name: MiNiFi Flow

Processors:
    - name: GetFile
      id: 471deef6-2a6e-4a7d-912a-81cc17e3a206
      class: org.apache.nifi.processors.standard.GetFile
      max concurrent tasks: 1
      scheduling strategy: TIMER_DRIVEN
      scheduling period: 1 sec
      penalization period: 30 sec
      yield period: 1 sec
      run duration nanos: 0
      auto-terminated relationships list:
      Properties:
          Input Directory: /tmp/getfile
          Keep Source File: true

Connections:
    - name: TransferFilesToRPG
      id: 471deef6-2a6e-4a7d-912a-81cc17e3a207
      source name: GetFile
      source id: 471deef6-2a6e-4a7d-912a-81cc17e3a206
      source relationship name: success
      destination id: 471deef6-2a6e-4a7d-912a-81cc17e3a204
      max work queue size: 0
      max work queue data size: 1 MB
      flowfile expiration: 60 sec

Remote Processing Groups:
    - name: NiFi Flow
      id: 471deef6-2a6e-4a7d-912a-81cc17e3a208
      url: http://localhost:8080/nifi
      timeout: 30 secs
      yield period: 10 sec
      Input Ports:
          - id: 471deef6-2a6e-4a7d-912a-81cc17e3a204
            name: From Node A
            max concurrent tasks: 1
            Properties:
```



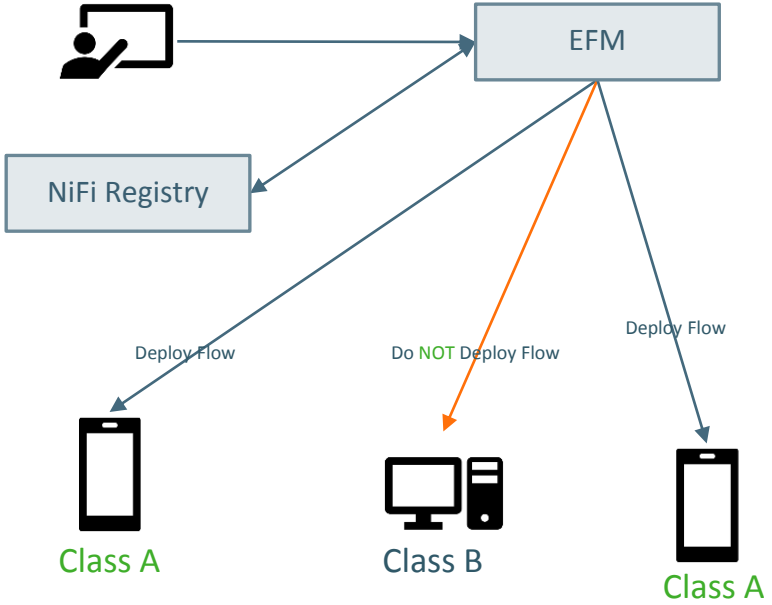
Edge Flow Manager (EFM)

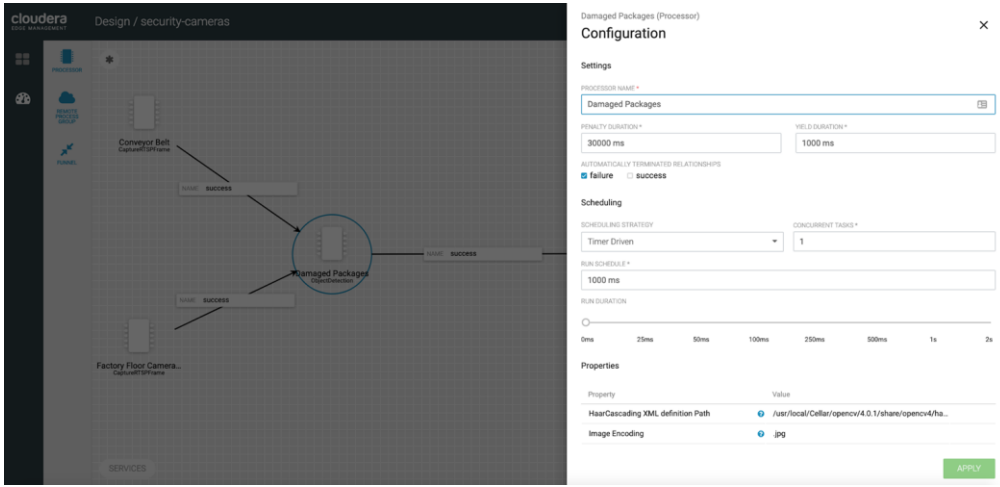
- Manage thousands of deployed agents
- NiFi-like user interface to develop and deploy flow files to the edge
- Application lifecycle management
- Update and deploy ML model files to the edge agents
- Monitor thousands of edge agents
- Integration with NiFi Registry



EFM Command & Control

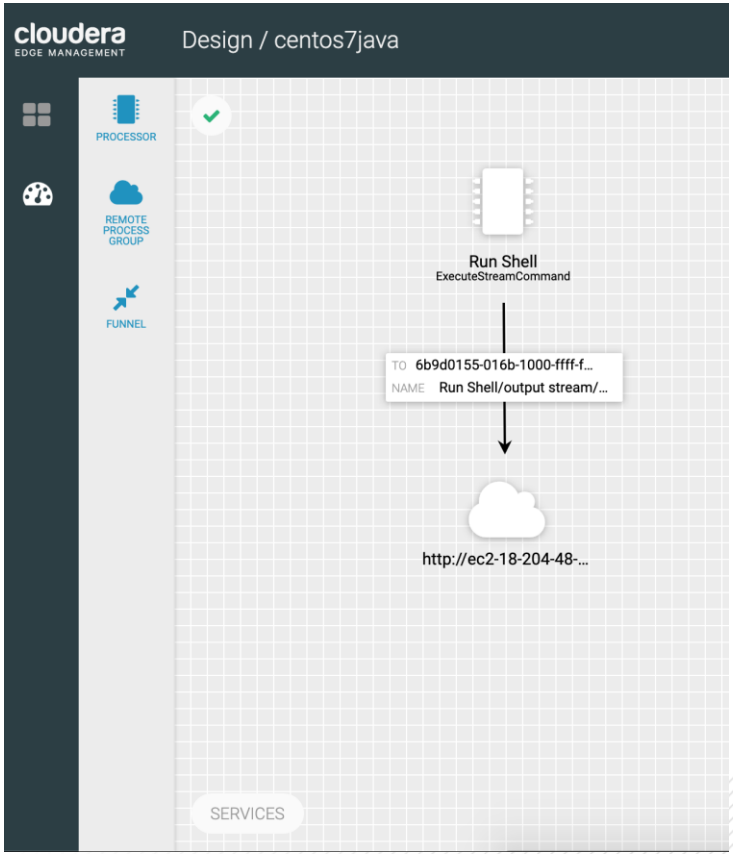
Flow Authorship & deployment

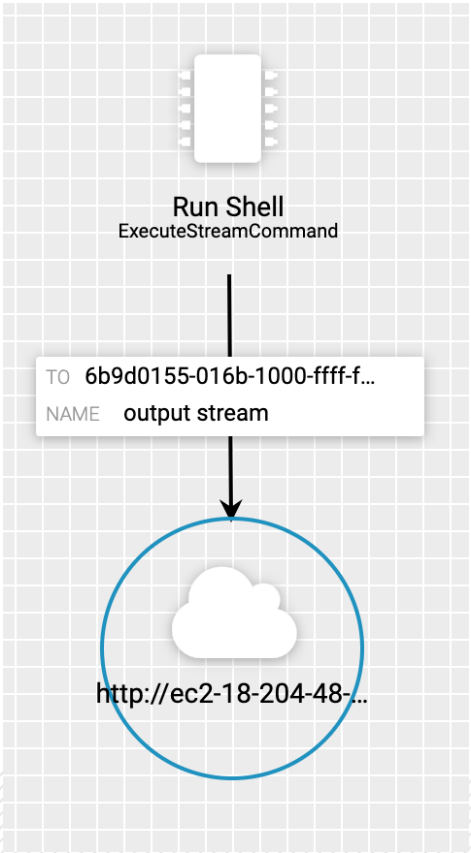




- User designs flow in EFM UI
- Completed flow is saved to NiFi Registry
- Class A flow is pushed to **only** class A devices

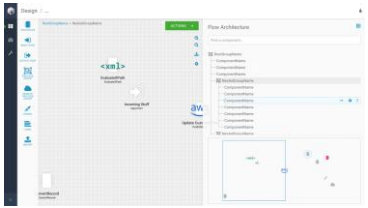
EFM Demo





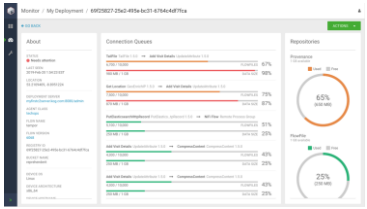
EFM ADDRESSES

Flow Authorship

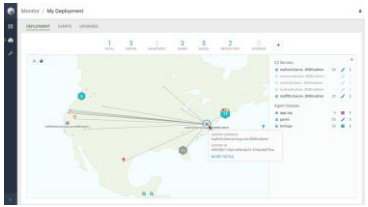


Flow Deployment

Flow Monitoring



The Disconnected Edge



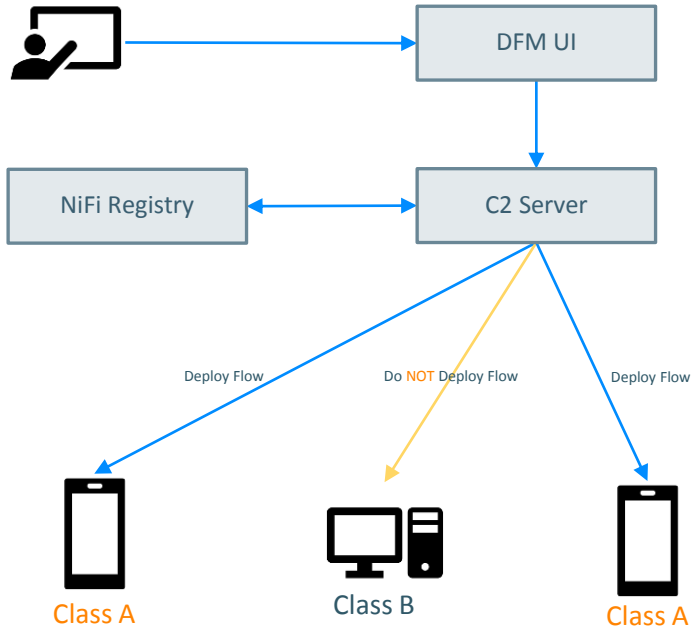
https://github.com/tspannhw/CDF-Workshop/blob/master/efm.md

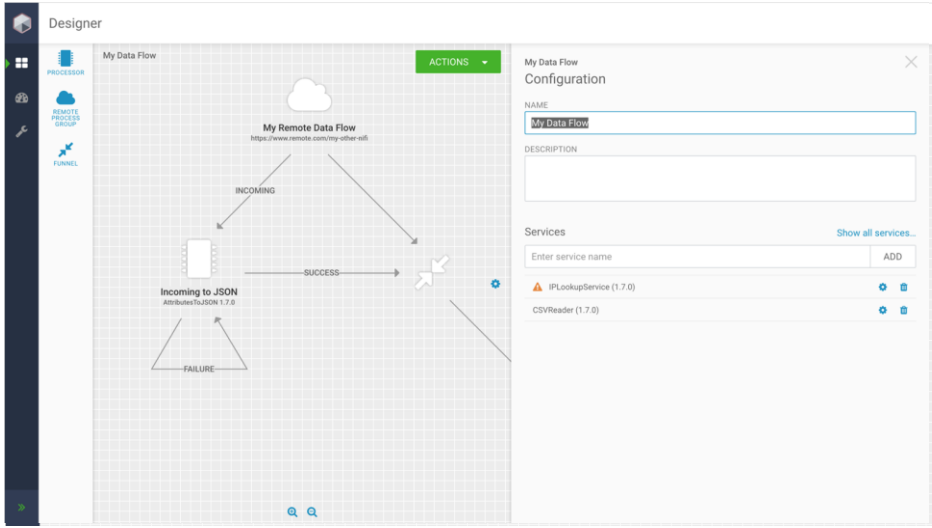
TERMINOLOGY

- Flow Authorship – Process of a flow operator designing/editing a flow in the DFM UI and then deploying those flows to targeted edge devices
- Flow Monitoring – Operational insights into flows deployed on Edge Devices.
- Device Manifest – Payload included in the edge device's heartbeat to their correlating C2 server that fully describes the processors supported by the edge device. This in turn drives the processors that are available to the flow author in the DFM UI
- Edge Device – Device running either MiNiFi Java or MiNiFi C++
- DFM – Dataflow Manager, DPS app that provides the actual UI components to users
- Class – A taxonomical grouping of Edge Devices

FLOW AUTHORSHIP

FLOW AUTHORSHIP





- Flow is sent to C2 server to be delivered to edge
- Flow is saved to NiFi Registry as versioned flow
- Class A flow is pushed to only class A devices

FLOW AUTHORSHIP – NEW DATA FLOW REQUIREMENTS

- A Data Flow developer has been tasked with modifying all gizmo Data Flows because all upstream environments now require JSON formatted data
- Developer must be able to load any existing flows into the DFM Designer
- Developer needs the ability to view all possible processors/services available to them for the specific class of devices
  - This is a very powerful and unique feature we offer
- Once Development is complete the developer needs the capability to commit their changes to the NiFi Registry and Git where it is staged for deployment

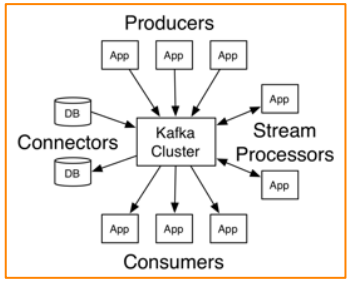
CSP : Cloudera Stream Processing

APACHE KAFKA CORE CONCEPTS

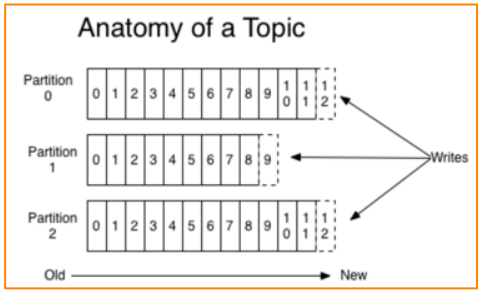
APACHE KAFKA BASICS

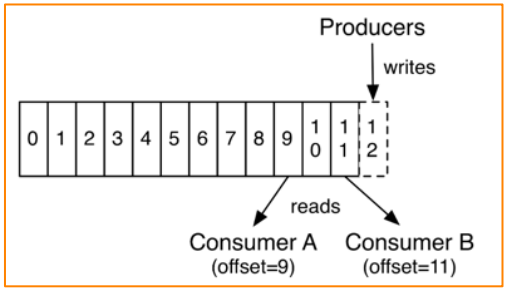
Kafka has 4 core APIs

1. Producer API
2. Consumer API
3. Streams API
4. Connector API

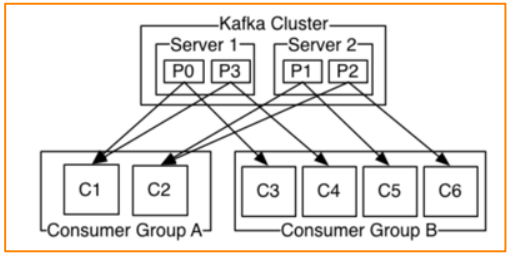


Anatomy of a Kafka Topic





Kafka Consumers



WHAT IS APACHE KAFKA?

- According to the Kafka website:

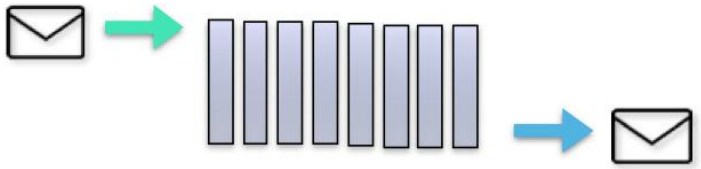
Kafka **is a distributed, partitioned, replicated commit log service. It provides the functionality of a messaging system, but with a unique design.**

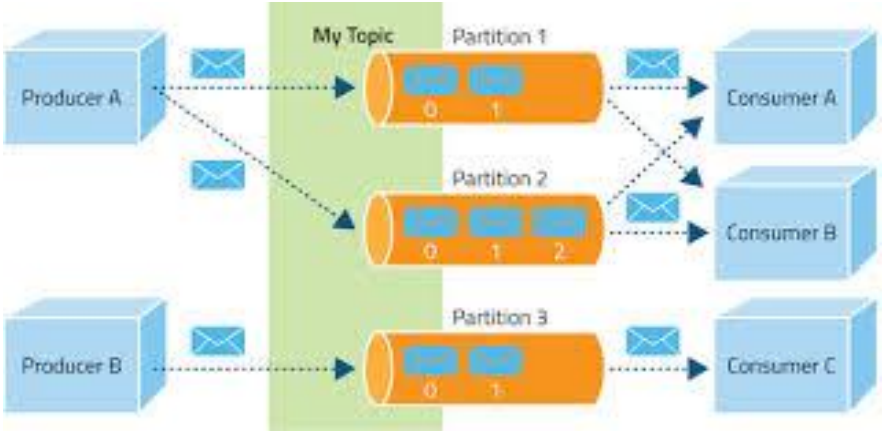
- **distributed:** horizontally scalable (just like Hadoop!)
- **partitioned:** the data is split-up and distributed across the brokers
- **replicated:** allows for automatic failover
- **unique:** Kafka does not track the consumption of messages (the consumers do)
- **fast:** designed from the ground up with a focus on performance and throughput

WHAT?

- A messaging system
- It can handle lots of messages
- Messages cannot get lost
- Message order is preserved
- Message will get delivered
- A producer sends the message and one or more
- consumers can read the message
- Kafka was built at Linkedin in 2011
- Open sourced as an Apache project

Inside Kafka

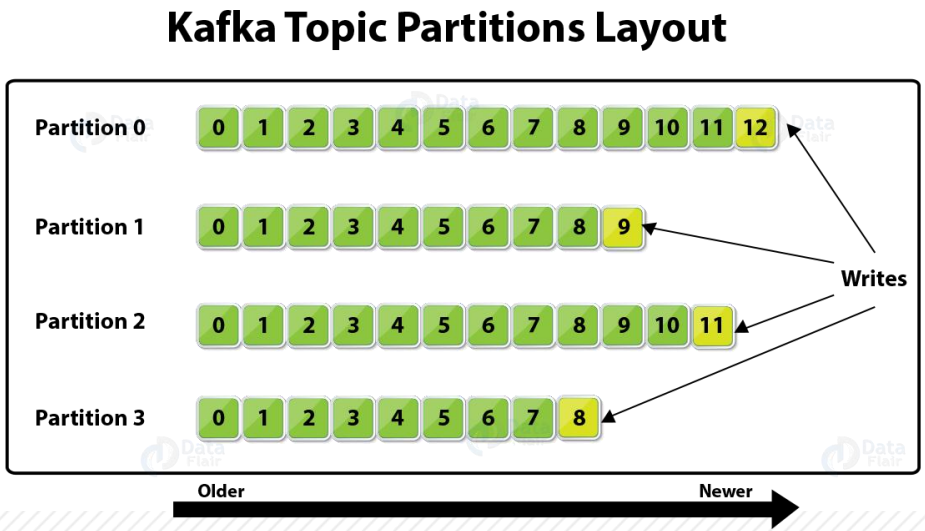




- Kafka topic and topic partition
- Kafka Broker
- Producers
- Consumers

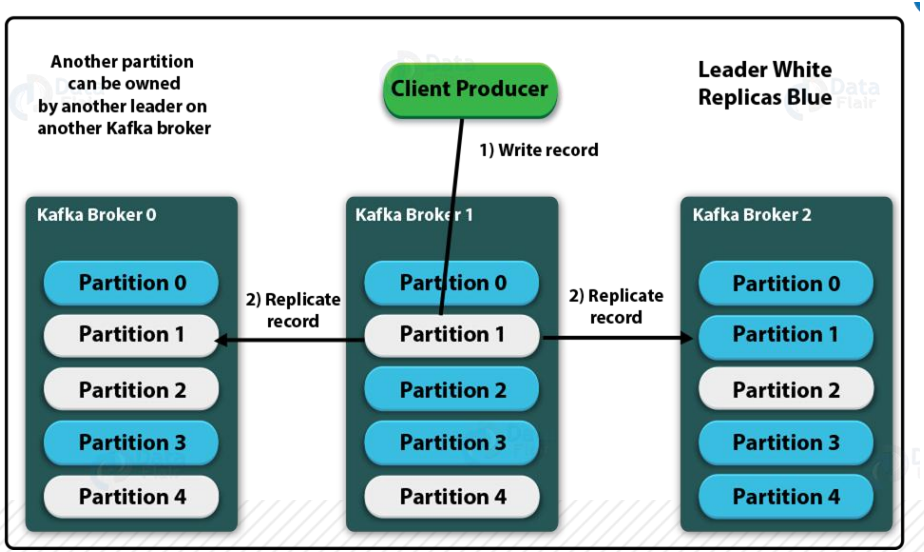
How does Kafka preserve message order

- Partition algorithm is fixed (hash on key)
- Stored as a log sequential write to a file
- Consume in order based on offset



How does Kafka prevent data loss

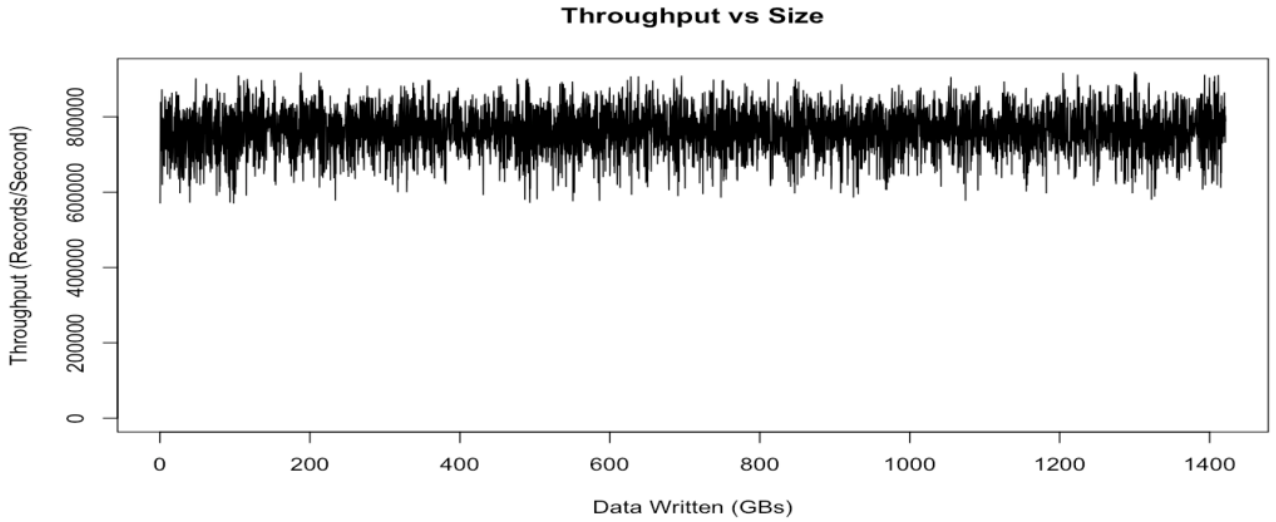
- Replicate, replicate, replicate
- Acknowledge you got the message
- Keep it even after it is consumed



HOW FAST IS KAFKA?

"Up to 2 million writes/sec on 3 cheap machines"

Using 3 producers on 3 different machines, 3x async replication



http://engineering.linkedin.com/kafka/benchmarking-apache-kafka-2-million-writes-second-three-cheap-machines

WHY IS KAFKA SO FAST?

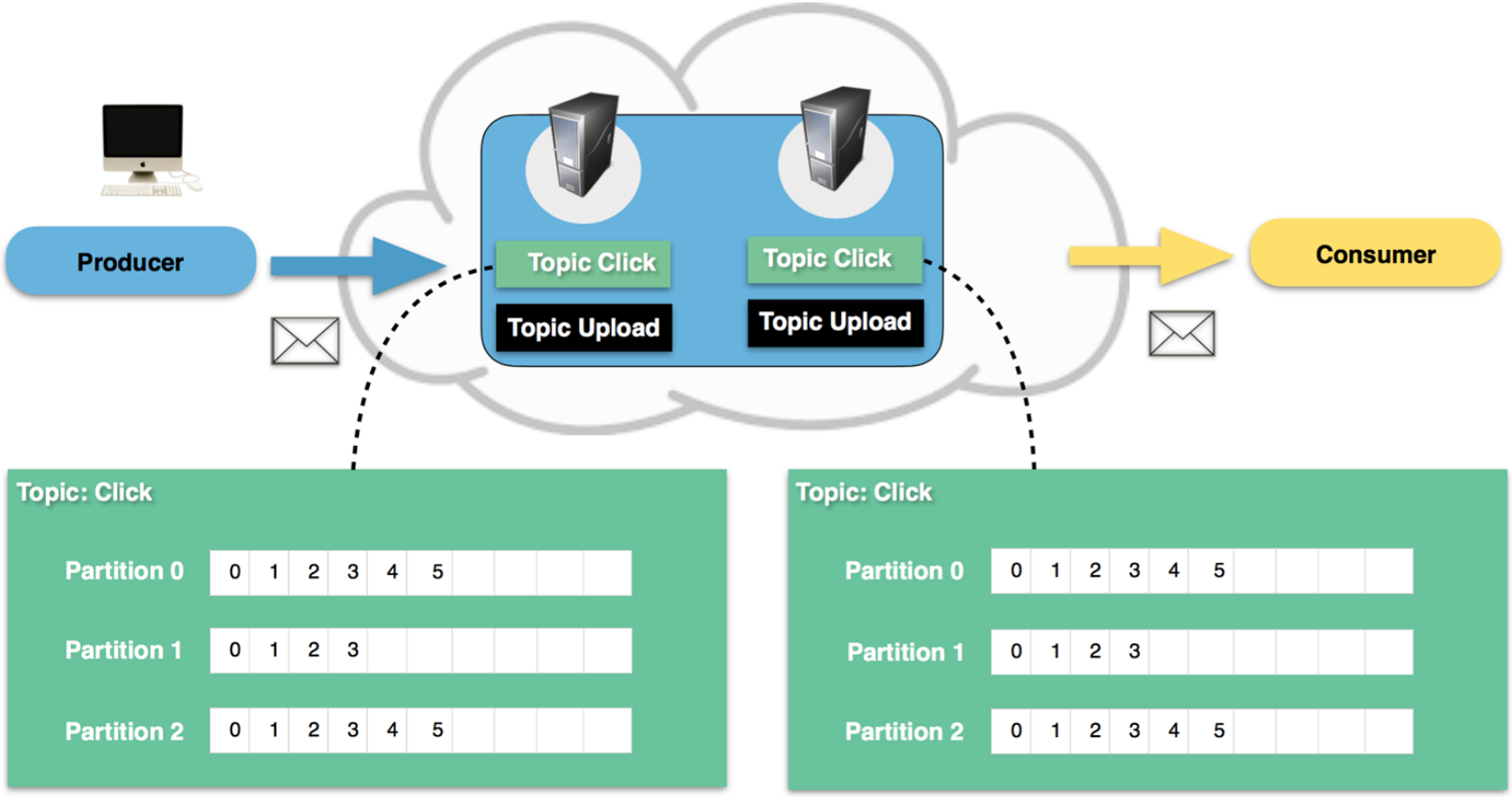
- **Fast writes:**
  - While Kafka persists all data to disk, essentially all writes go to the **page cache** of OS, i.e. RAM.
- **Fast reads:**
  - Very efficient to transfer data from page cache to a network **socket**
  - Linux: **sendfile()** system call
- **Fast writes + fast reads** = fast Kafka!
  - On a Kafka cluster where the consumers are mostly caught up, you will see no read activity on the disks as they will be serving data entirely from cache.

http://kafka.apache.org/documentation.html#persistence

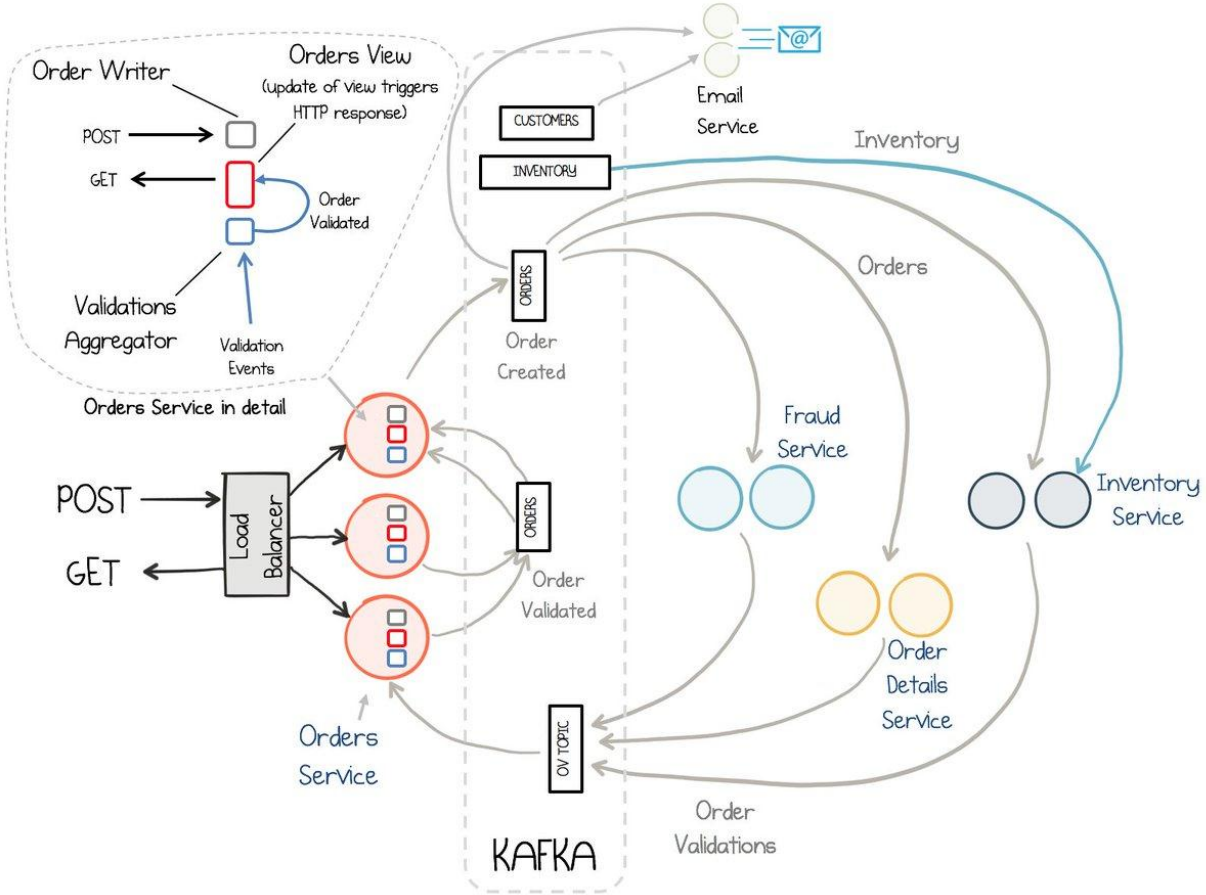
KAFKA USE CASES

- Web site activity: track page views, searches, etc. in real time
- Events & log aggregation: particularly in distributed systems where messages come from multiple sources
- Monitoring and metrics: aggregate statistics from distributed applications and build a dashboard application
- Stream processing: process raw data, clean it up, and forward it on to another topic or messaging system
- Real-time data ingestion: fast processing of a very large volume of messages

Kafka Use Case: Clickstream Data



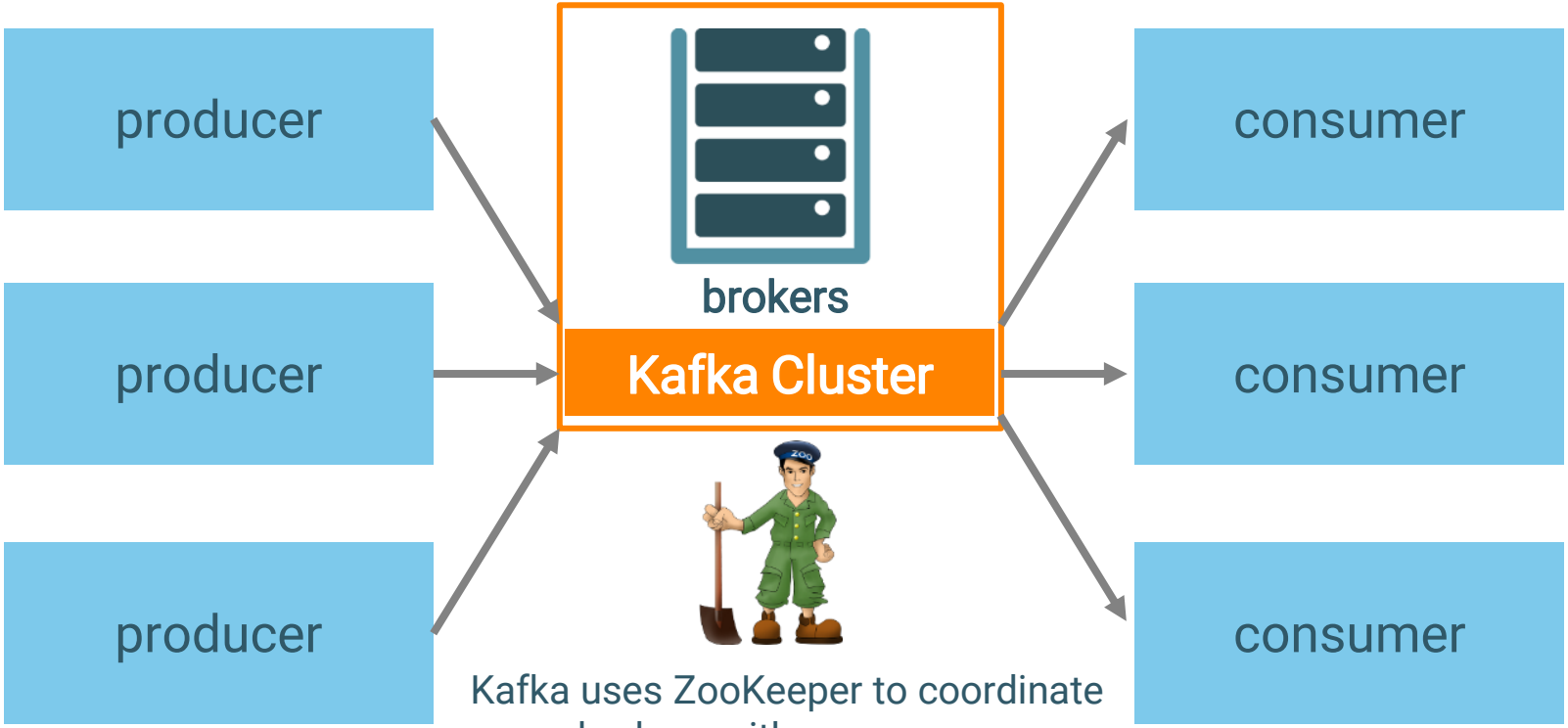
Kafka Use Case: Online ordering system



KAFKA TERMINOLOGY

- Kafka is a publish/subscribe messaging system comprised of the following components:
  - **Topic**: a message feed
  - **Producer**: a process that publishes messages to a topic
  - **Consumer**: a process that subscribes to a topic and processes its messages
  - **Broker**: a server in a Kafka cluster

KAFKA COMPONENTS

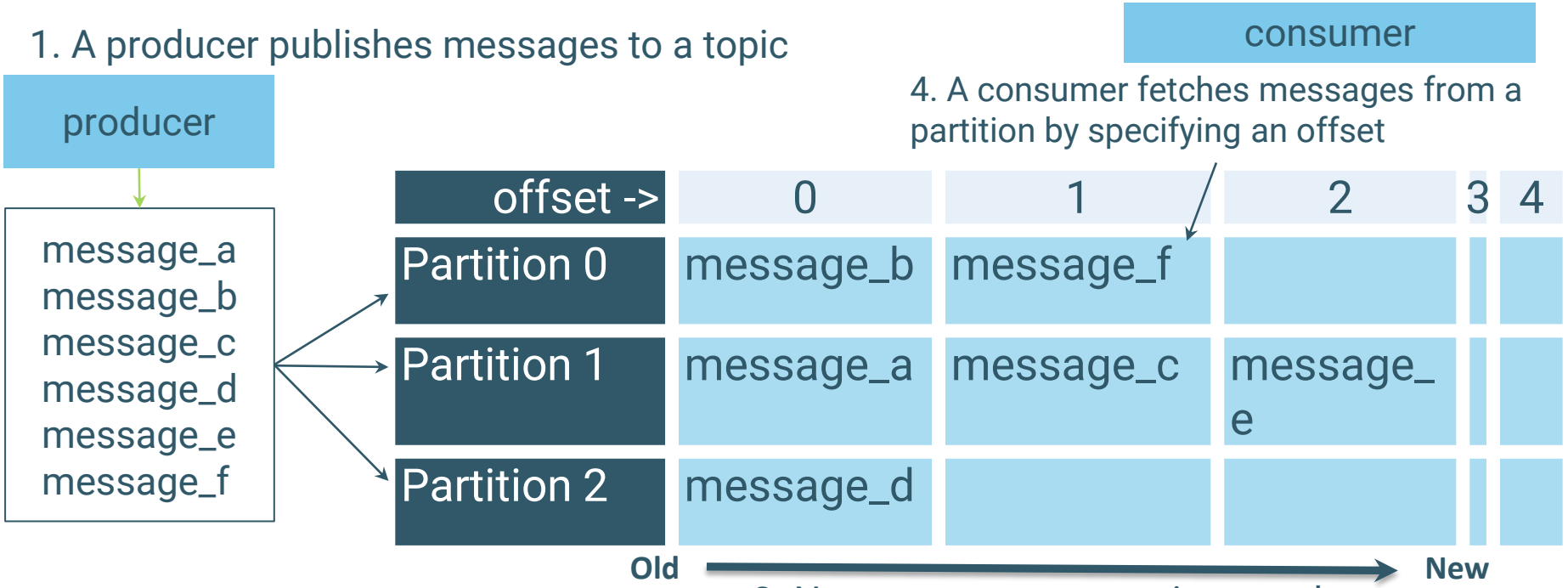


OVERVIEW OF TOPICS

- A topic is a name assigned to a feed to which messages are published
  - A topic in Kafka is partitioned
- Each partition is an ordered, immutable sequence of messages
  - it is continually appended to
  - each message is assigned a sequential id called an offset
- Messages are retained for a configurable amount of time (24 hours, 7 days, etc.)
- Each consumer retains its own offset in the partition
  - allows the consumer to go back and re-read messages without retaining the message
  - the offset is the only metadata that the consumer retains
  - different consumers maintain their own offset

PUBLISHING MESSAGES

1. A producer publishes messages to a topic



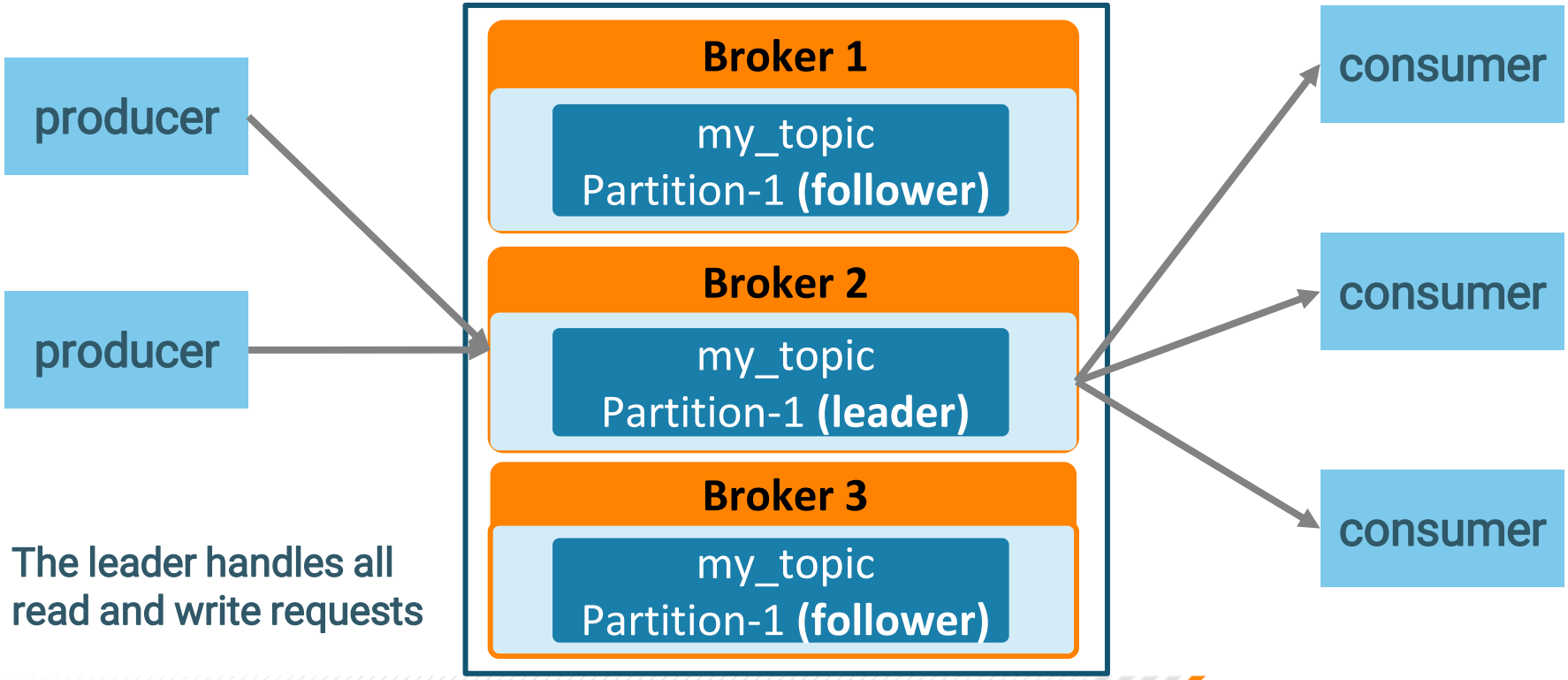
2. The producer decides which partition to send each message to

3. New messages are written to the end of the partition

UNDERSTANDING PARTITIONS

- Partitions are distributed across the cluster
- A partition is managed by a broker
- Each partition is replicated for fault tolerance
- A replicated partition has one broker that acts as the leader
- The other brokers of that partition act as followers
  - The followers passively replicate the leader
  - If the leader fails, one of the followers automatically becomes the new leader
  - Brokers distribute their roles as leaders and followers to maintain a well-balanced cluster

LEADER AND FOLLOWERS



The leader handles all read and write requests

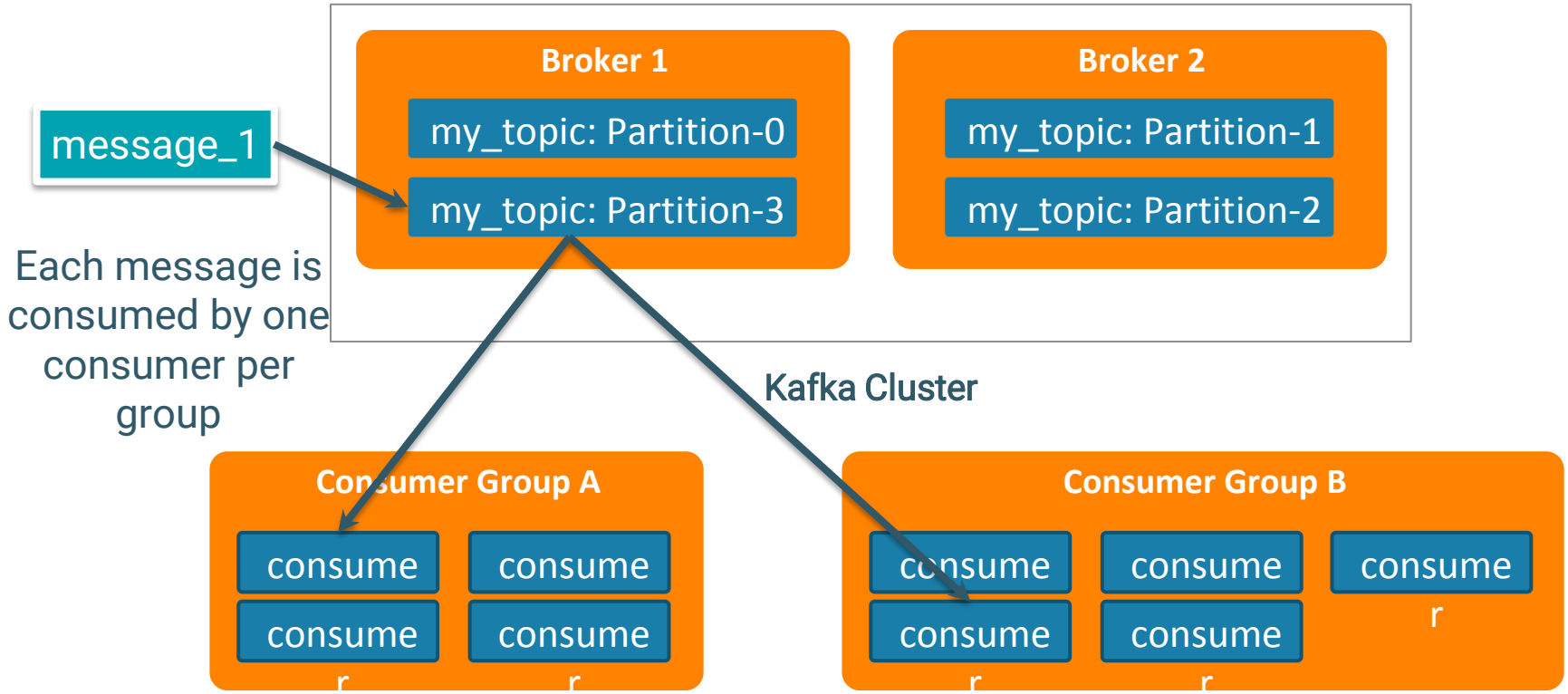
CONTROLLING PARTITIONING LOGIC

- The partitioning logic is performed by the producer
- This can happen various ways:
  - hash function (default behavior – the keys are hashed and divided by the # of partitioners)
  - random distribution (if the keys are null)
  - you can specify a partitioner using the partitioner.class config property (set to the name of a custom Java class that you write)

CONSUMING MESSAGES

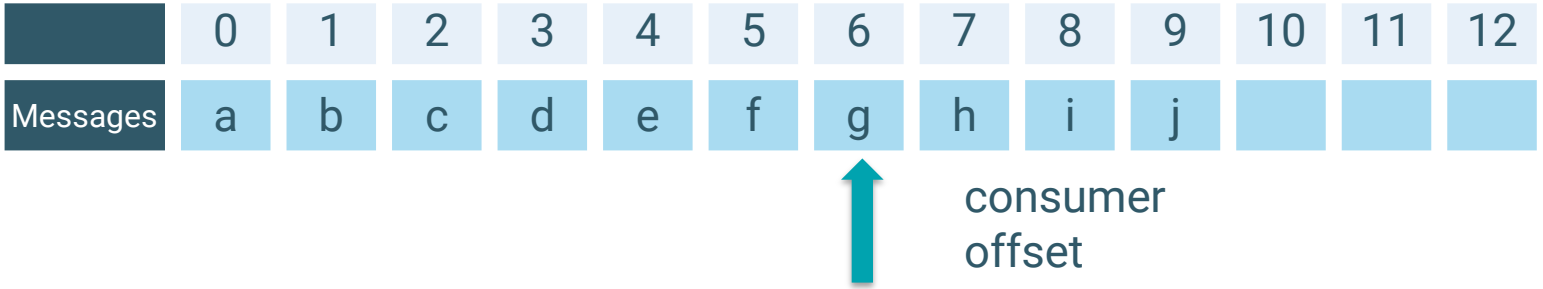
- Messages are consumed in Kafka by a consumer group
- Each individual consumer is labeled with a group name
- Each message in a topic is sent to one consumer in the group
- In other words, messages are consumed at the group level, not at the individual consumer level
    - This allows for fault tolerance and scalability of consumers
- This design allows for both queue and publish-subscribe models:
    - If you need a queue behavior, then simply place all consumers into the same group
    - If you need a publish-subscribe model, then create multiple consumer groups that subscribe to a topic

CONSUMER GROUPS



THE CONSUMER OFFSET

It is up to the consumer to maintain its offset in the partition (stored in a special topic named __consumer_offsets)



- This has several key benefits, including:
  - **performance**: there is no back-and-forth acknowledging of message consumption
  - **simplicity**: the consumer only has to maintain a single integer value for its state, which can be easily stored and shared between consumers (if a failure occurs)
  - **re-consume messages**: it becomes trivial for a consumer to re-consume messages

MESSAGE DELIVERY GUARANTEES

- Kafka guarantees at-least-once delivery by default
- At-most-once delivery is possibly by disabling retries on the producer (when a commit fails)
- Exactly-once delivery is possible (with clever coordination of your consumers and the consumer offset)
- Other guarantees:
    - Messages in a partition are stored in the order that they were sent by the publisher
    - Each partition is consumed by exactly one consumer in the group
    - That consumer is the only reader in the group of that partition in the group
    - Messages are consumed in order
    - Messages committed to the log are not lost for up to N-1 broker failures

IN-SYNC REPLICAS

- Kafka replicates the messages in each partition across multiple brokers
- New messages are always appended to the leader
- A follower that keeps up is called an ISR, or in-sync replica, which means:
- A message is considered committed when all ISRs have a copy of the message

Kafka's Omnipresence Has Led to the Onset of "Kafka Blindness"

- What is **"Kafka Blindness"**?
  - Customers who use Kafka today struggle with monitoring / "seeing"/troubleshooting what is happening in their clusters

- Who is **Affected**?
  - Platform Operation Teams
  - Developers / DevOps Teams
  - Security / Governance Teams

- What are the **Symptoms**?
  - Difficulty seeing who is producing and consuming data
  - Difficulty understanding the flow of data from producers -> topics consumers
  - Difficulty troubleshooting/monitoring.

Streams Messaging Manager (SMM)

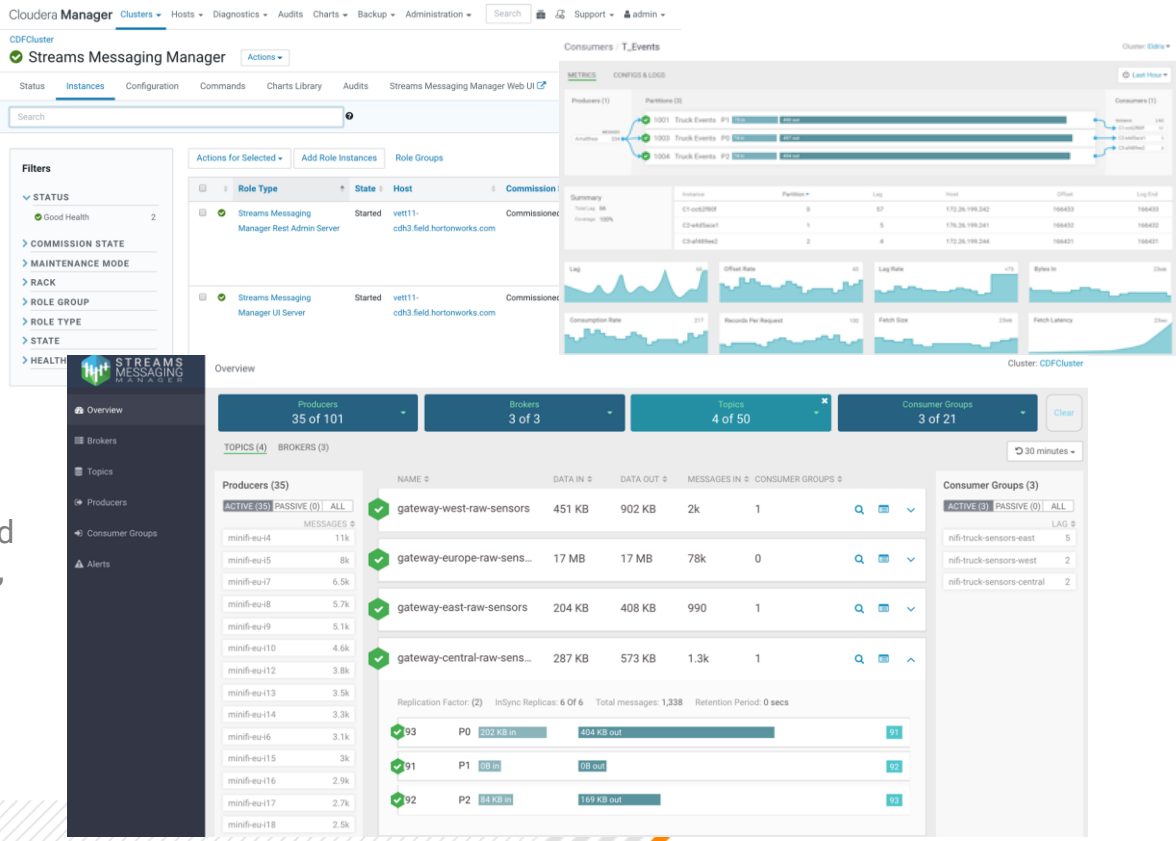
Curing Kafka Blindness

Problem Statement / Requirements

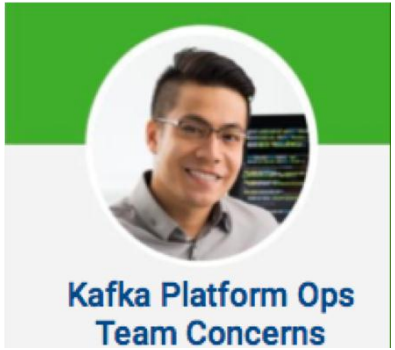
- Customers **struggle with monitoring / "seeing"/ troubleshooting what is happening in their Kafka clusters**
- They need an enterprise Kafka monitoring solution full integrated with platform services including CM and Sentry

Solution

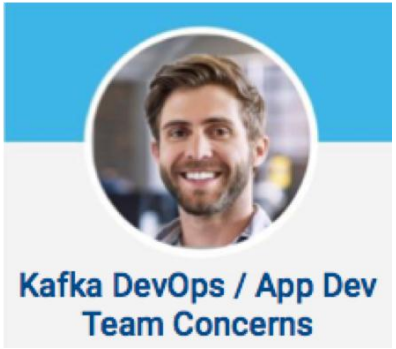
- SMM provides single monitoring Dashboard for Kafka Clusters across 4 entities: Broker, Producer, Topic, Consumer
- SMM integrated with CM Log Search
- Kerberized based authentication and rich access Control via Sentry
- DataPlane Platform NOT required



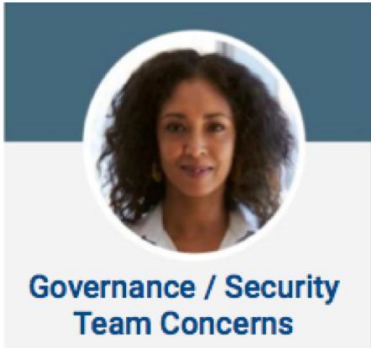
SMM Addresses the Distinct Needs of 3 Personas/Teams



Concerned with monitoring the overall health of the cluster and the infrastructure it runs on

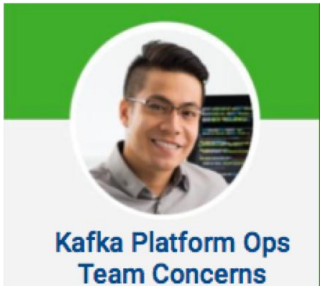


Concerned with monitoring the Kafka entities associated with their apps



Concerned with audit, compliance, access control & chain of custody requirements

SMM for the Platform Ops Team



Do I have any offline topic partitions?

Which consumer group us falling behind the most?

Are any of my brokers down?

Which producers are generating the most data right now?

What is the throughput in/out for a given partition on that broker?

What hosts are my brokers located on?

Are all my replicas in my topic in-synch?

How many total active producers/ consumers is there currently?

Which producers are generating the most data right now?

Are there any skewed partitions for a broker?

How many total active producers and consumers exist now?

Are any of my brokers running hot? Which broker has the highest throughput in and and rates?

How many total topics does my Kafka cluster have?

Which producers are generating the most data right now?

Are any of my brokers running out of disk space?

How of the cluster is being used, how much capacity do I have available per broker / cluster?

Which partitions are located on each broker?

Which of my topics has produced/consumers the most messages over the last N minutes/hours?

SMM for the DevOps/AppDev Teams

Find all entities (producer, consumers, topics) associated with my app.

What is the retention rate for my app topics?

What is the replication factor for my app topics?

What brokers holds the partitions for my app topics?

Who are all the producers and consumers connected to my app topics?

What type of events are in my application topics? What does the event look like?

What is the total number of messages into my topic over the last N minutes/hours?

Did a consumer rebalance occur for a given topic?

Are any of my consumers/ consumer-groups that are over-consuming?

Are there consumers in a consumer-group for a given topic slow/falling behind?

How many active consumers instances are in a given consumer group?

What topic(s) are the consumer group consuming messages from?

Are any of my consumers/consumer-groups that are under-consuming?

SMM for the Security and Governance Team

Governance / Security Team Concerns

When was a topic created?

What is the schema for a given topic?

What is the linage of a kafka topic?

How has the schema evolved for a given topic?

How does data flow across multiple kafka hops?

Who has edited the ACL policies of a given topic?

Which consumers have consumed from a topic?

What are the ACL policies for a given topic?

When were additional brokers added to the topic?

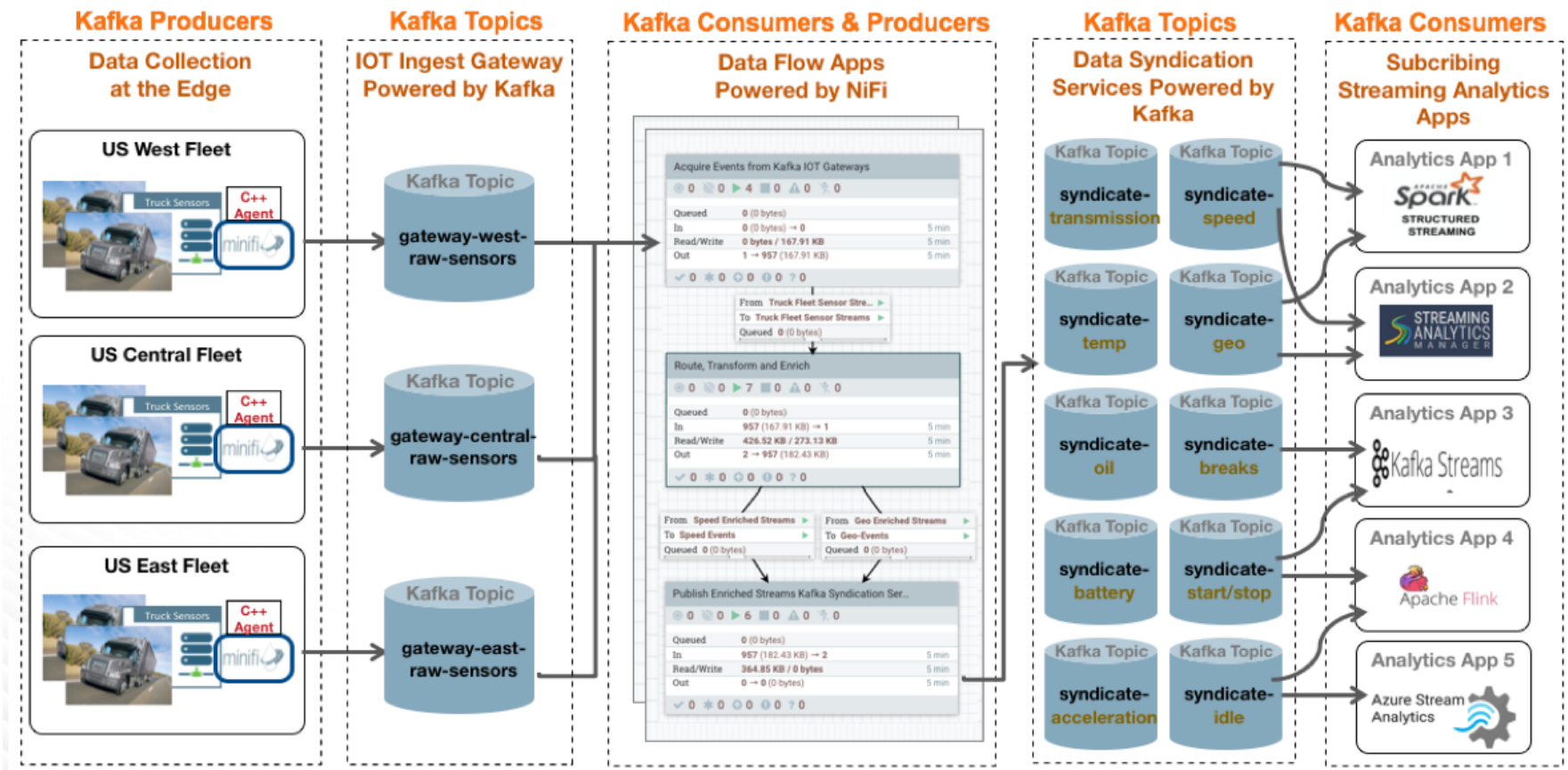
Which users/groups/service accounts have read from a given topic?

Which users/groups/service accounts have sent data to a topic?

When was the topic configuration last modified?

Which producers have sent data to the topic?

Dev Ops / App Dev Persona – Monitoring the Streaming Truck App



Kafka Challenges

Shared Schema Registry
How do I associate schemas for messages in Kafka topics

Agility and Self-Service
How do I develop, deploy & manage Kafka producers and consumers without code in self service manner

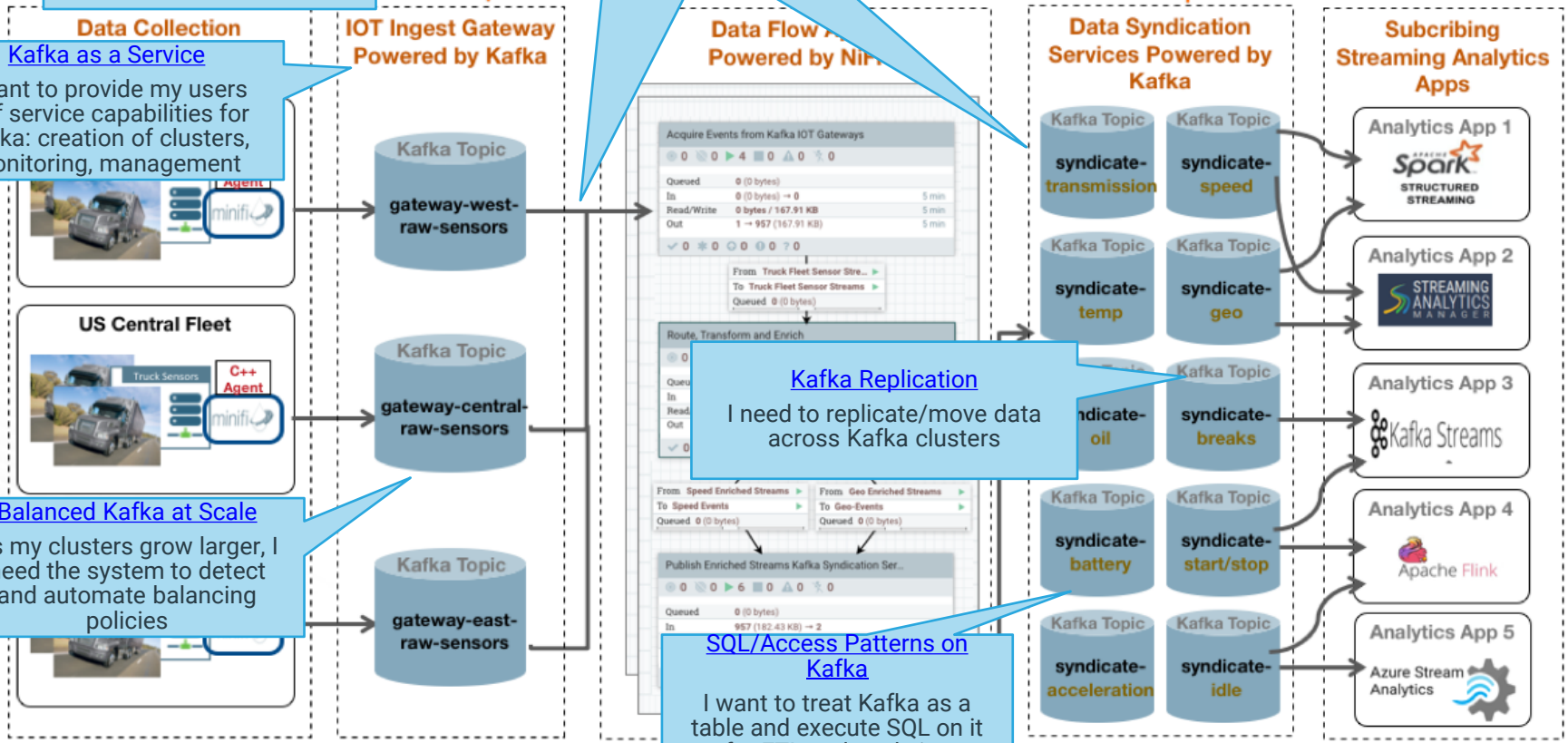
Kafka Blindness
How do I manage/monitor the different kafka clusters, topics, consumers, and producers

Kafka as a Service
I want to provide my users self service capabilities for Kafka: creation of clusters, monitoring, management

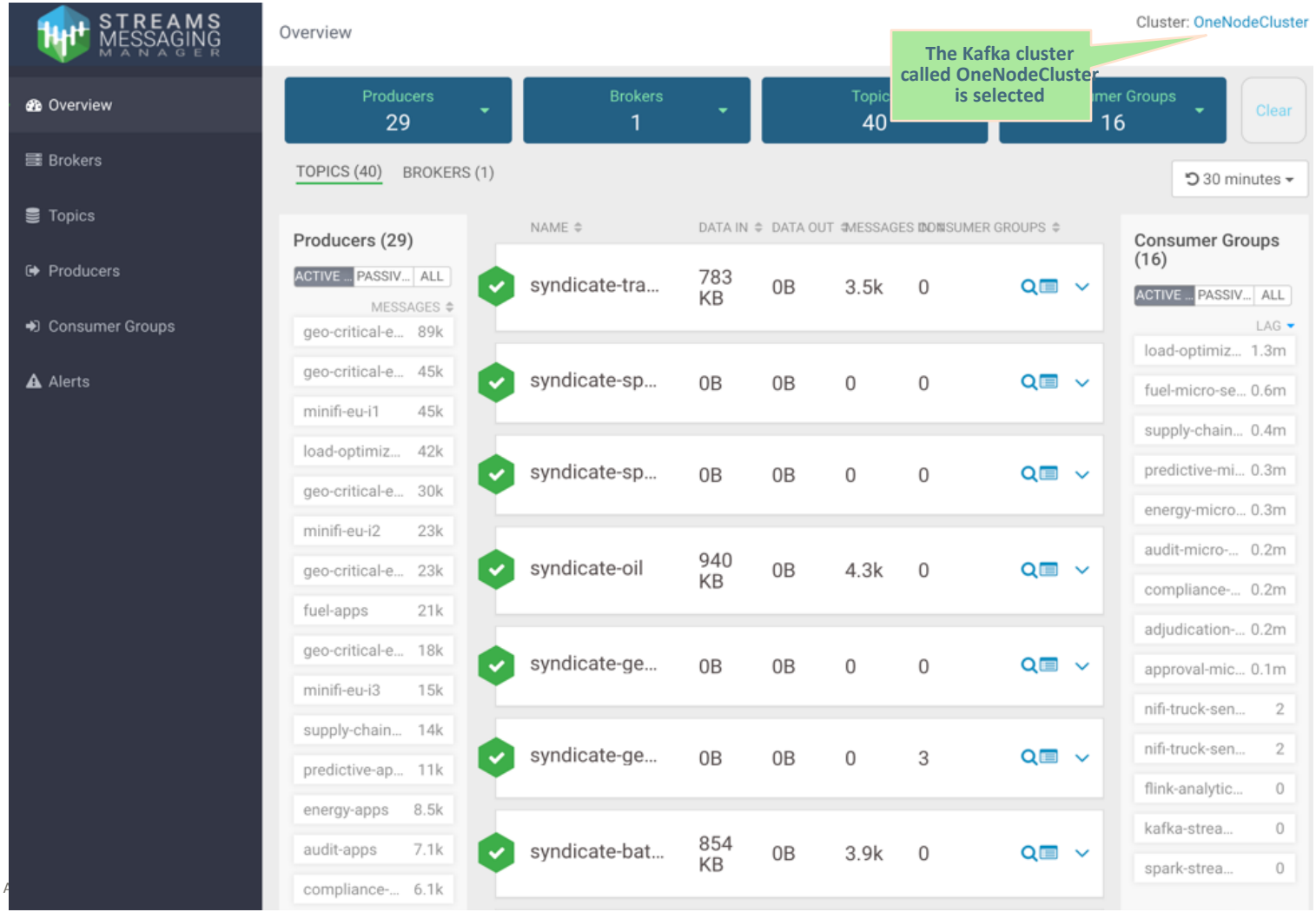
Kafka Replication
I need to replicate/move data across Kafka clusters

Balanced Kafka at Scale
As my clusters grow larger, I need the system to detect and automate balancing policies

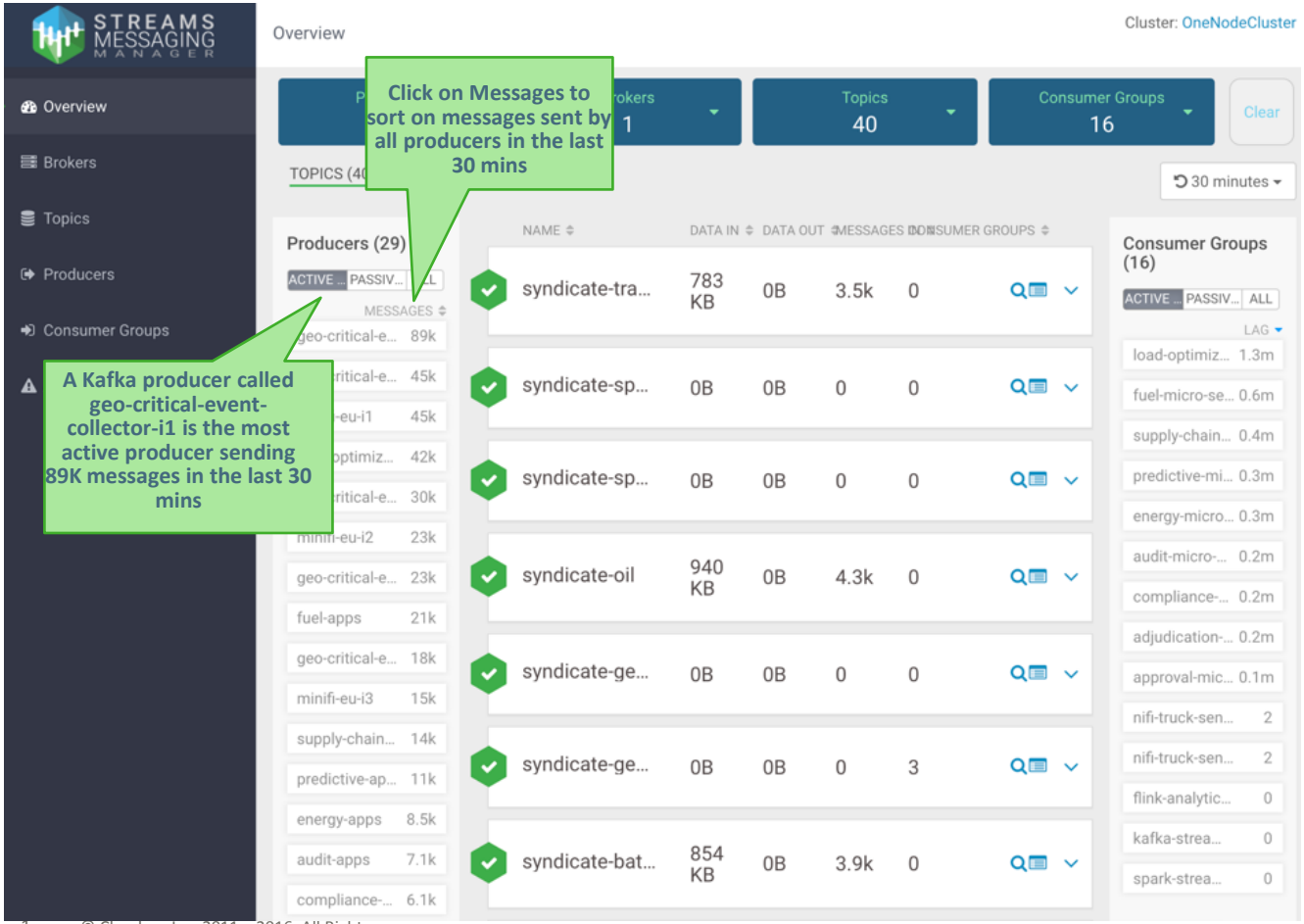
SQL/Access Patterns on Kafka
I want to treat Kafka as a table and execute SQL on it for ETL and analtyics



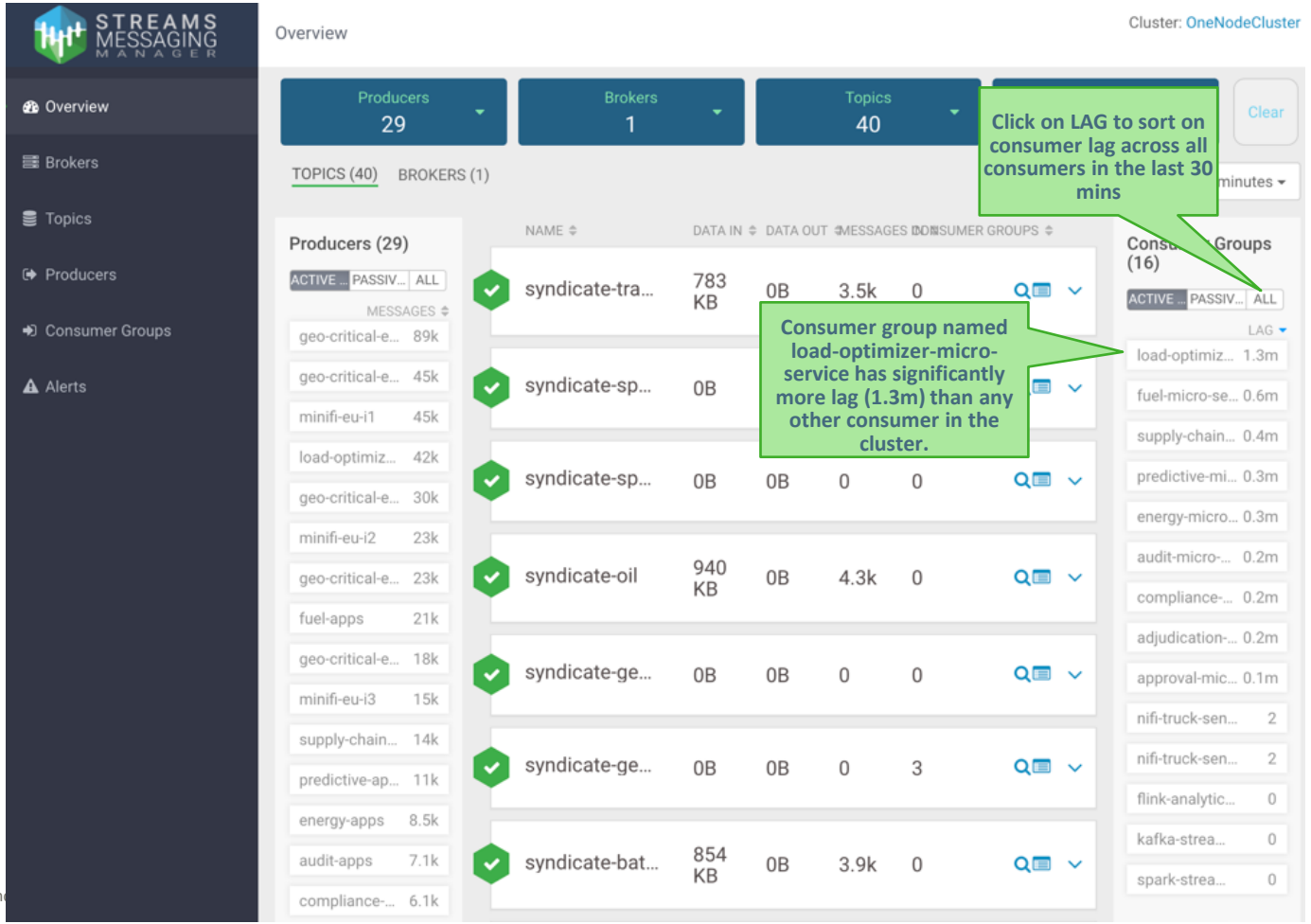
Main Dashboard View



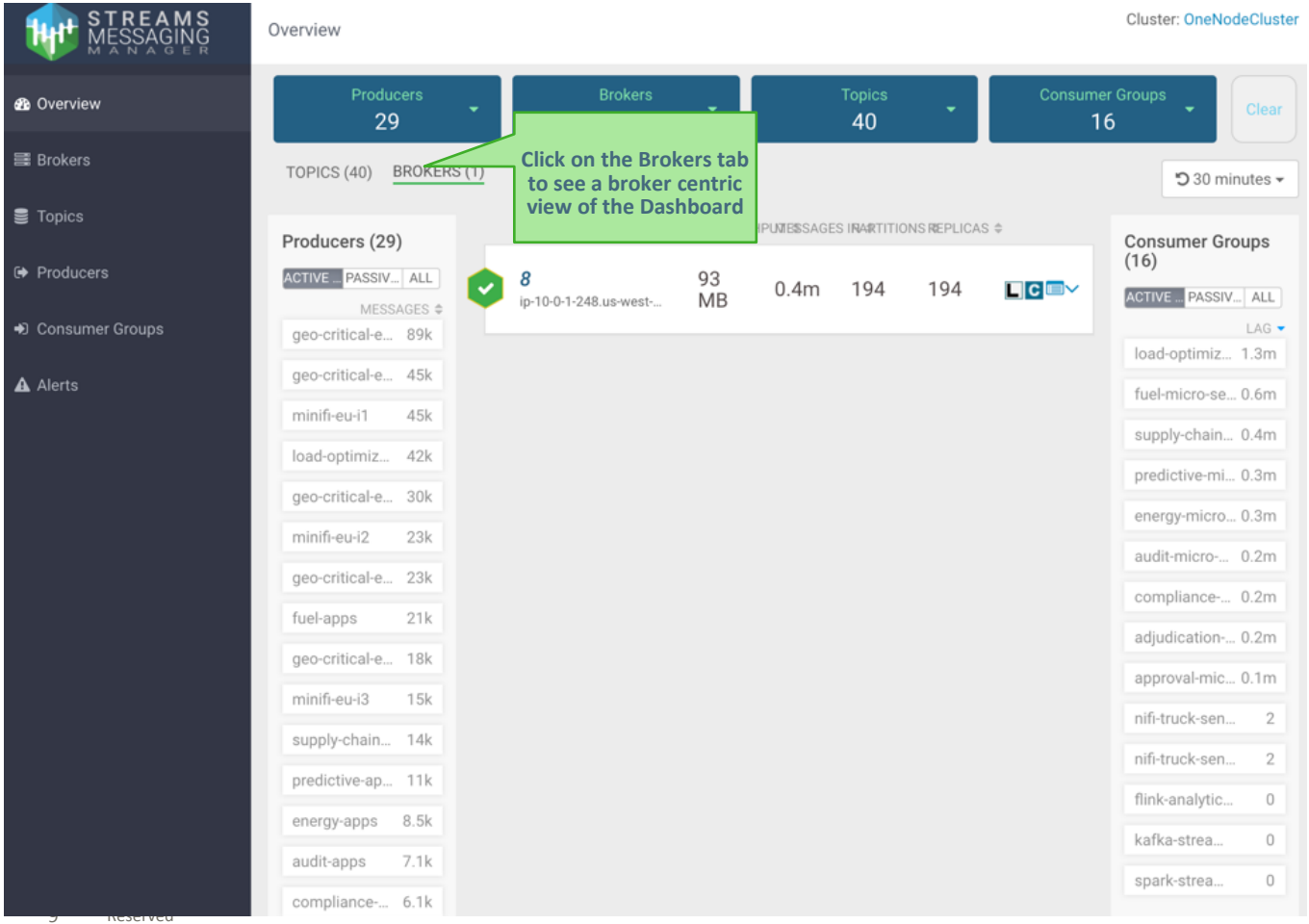
Find the Most Active Producer in my Cluster



Find the Consumer Who Has Fallen Behind the Most

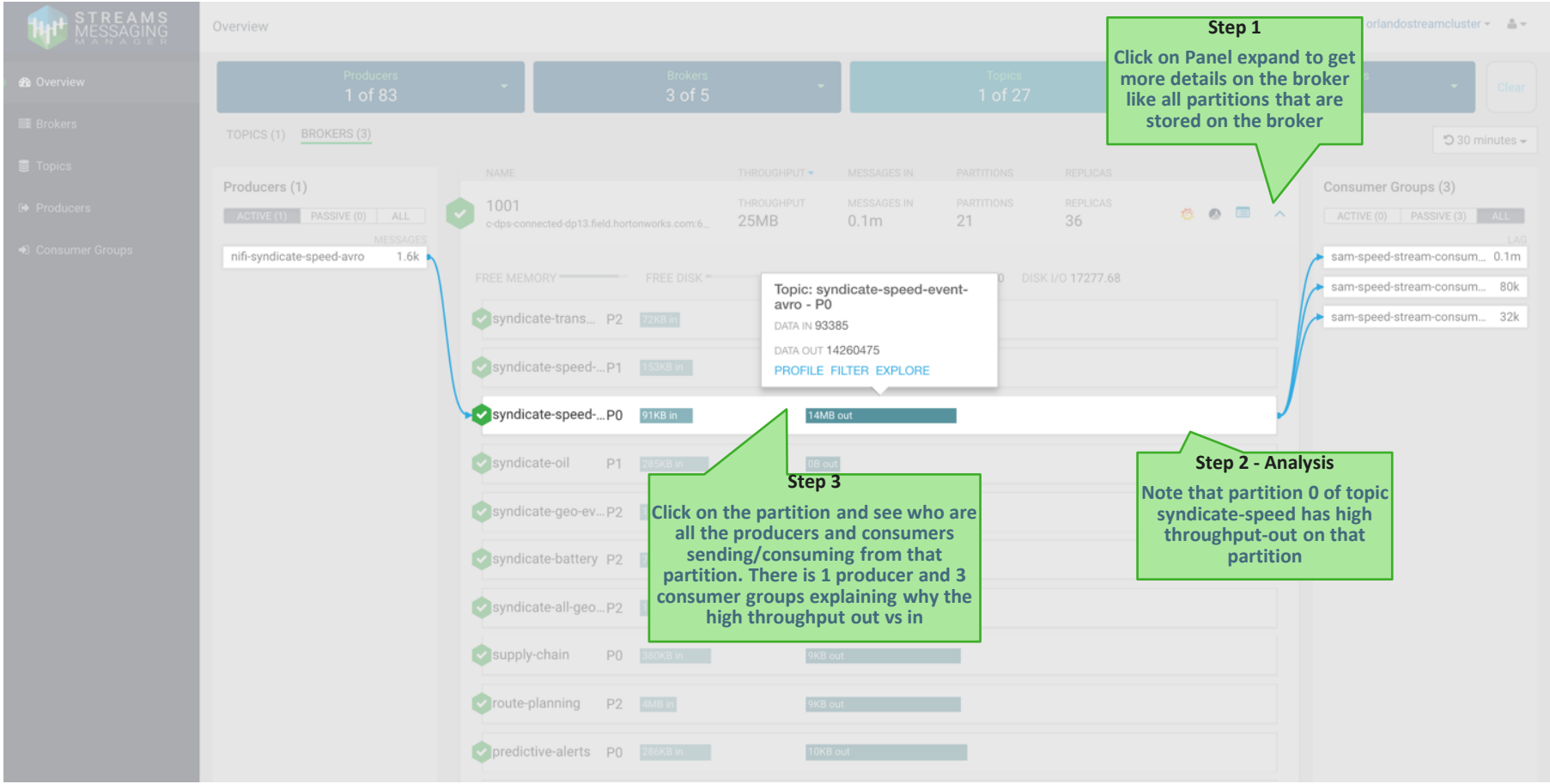


Broker Centric View – View Details of the Brokers in My Cluster

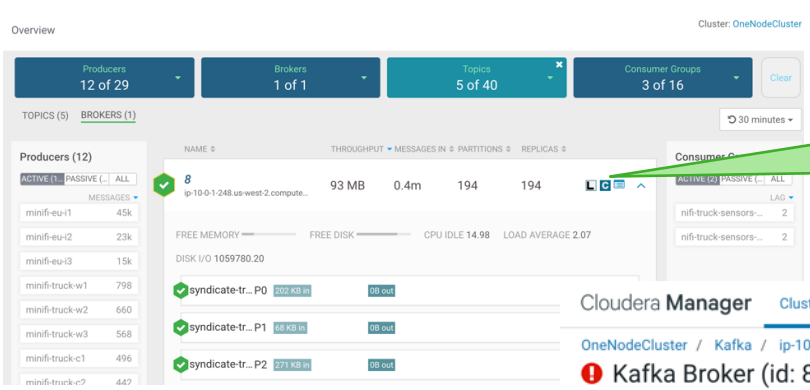


Broker Centric View: Find my Hottest Broker – Broker with Highest Throughput In

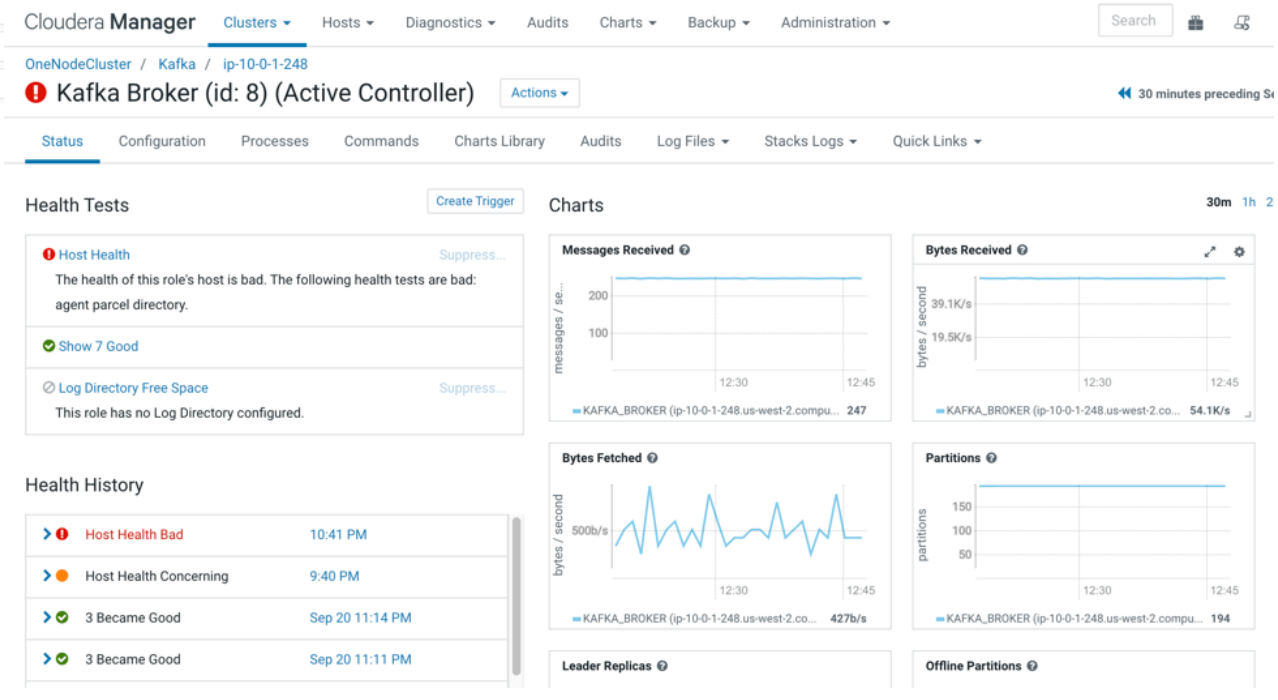
Find Partitions on a given Broker and Understand flow of data flow from Producer to selected Broker Partition to Consumer



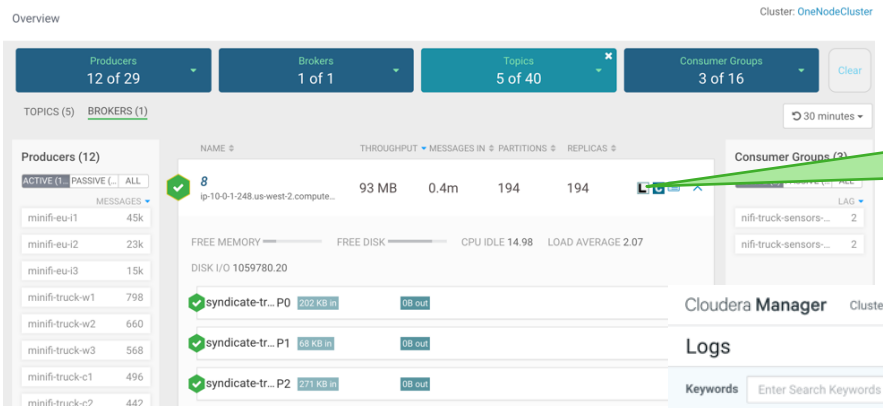
Analyze Detailed Broker Host Metrics – Cloudera Manager Integration



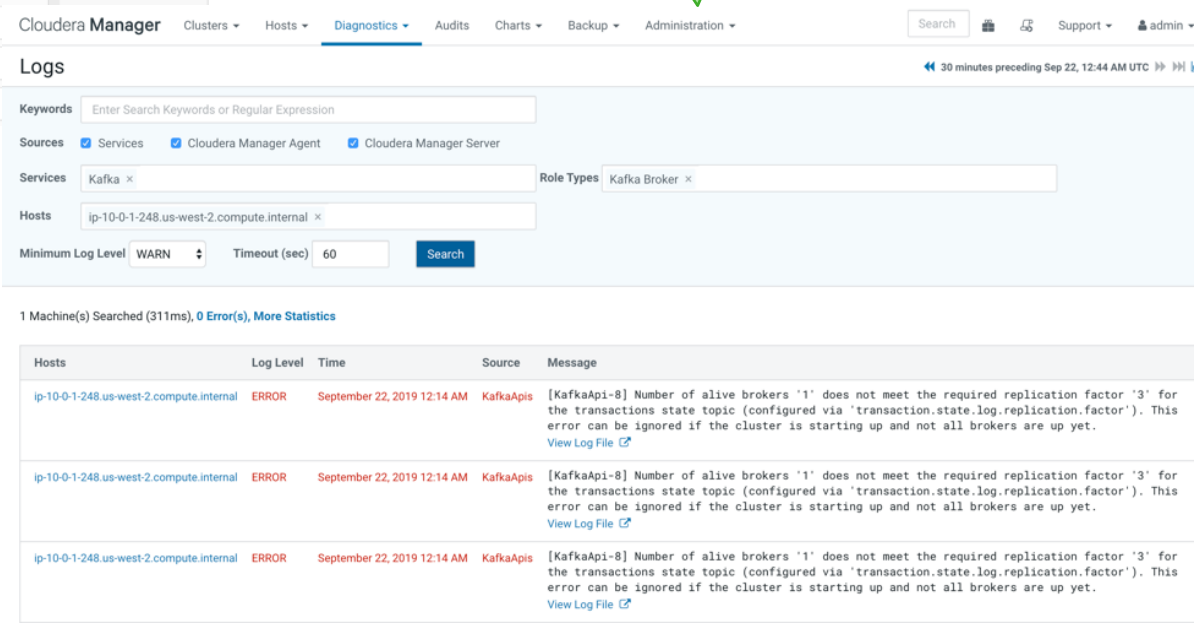
Click on the CM icon on the broker panel and the CM host detail view for that broker is displayed providing host level metrics and a view of other services running on that host



Keyword Search via Log Search

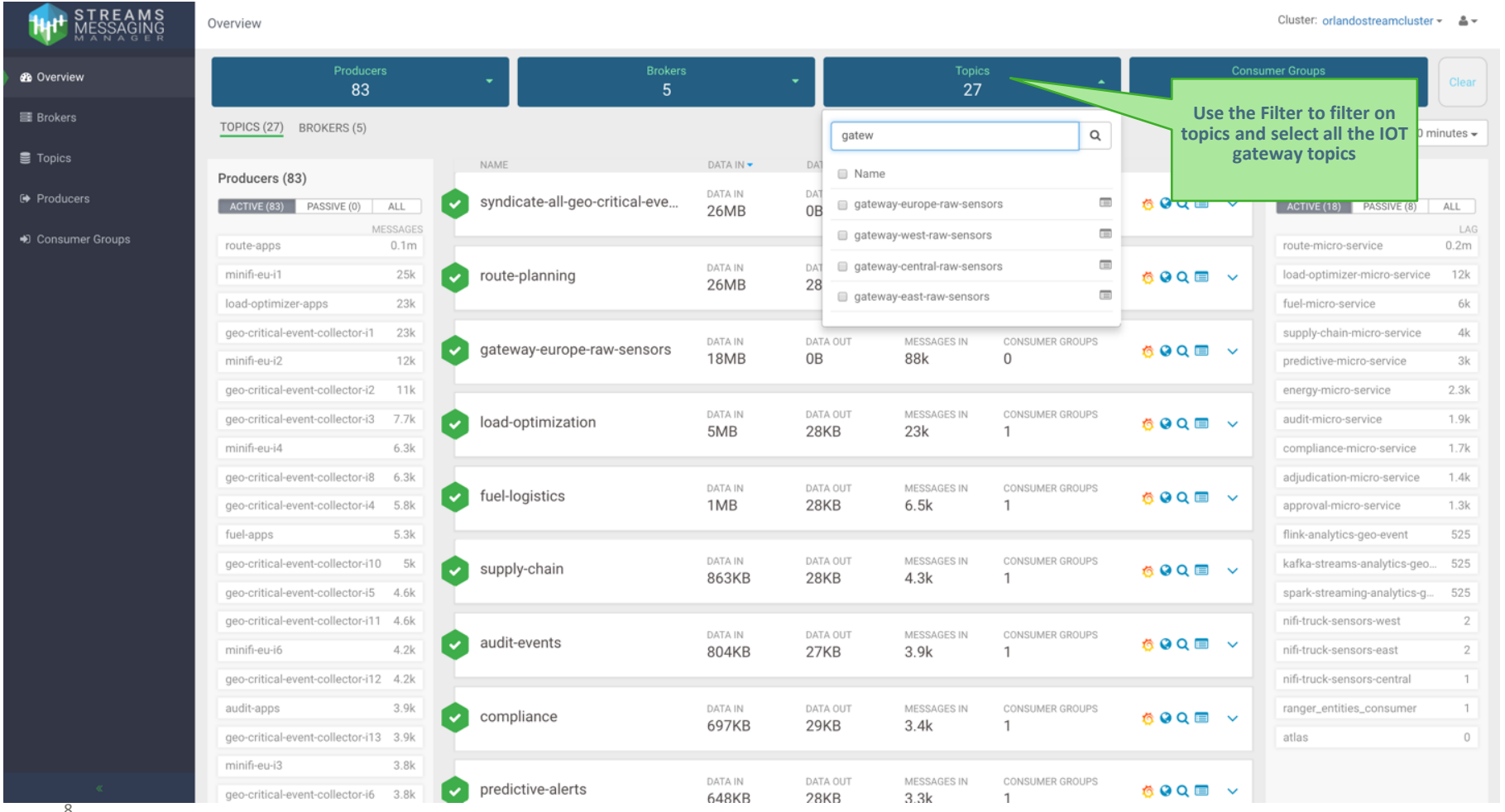


Click on the Log Search icon on the broker panel and the Log Search detail view is displayed. This enables you to search for specific keywords and to filter for specific log levels, components, and time ranges.

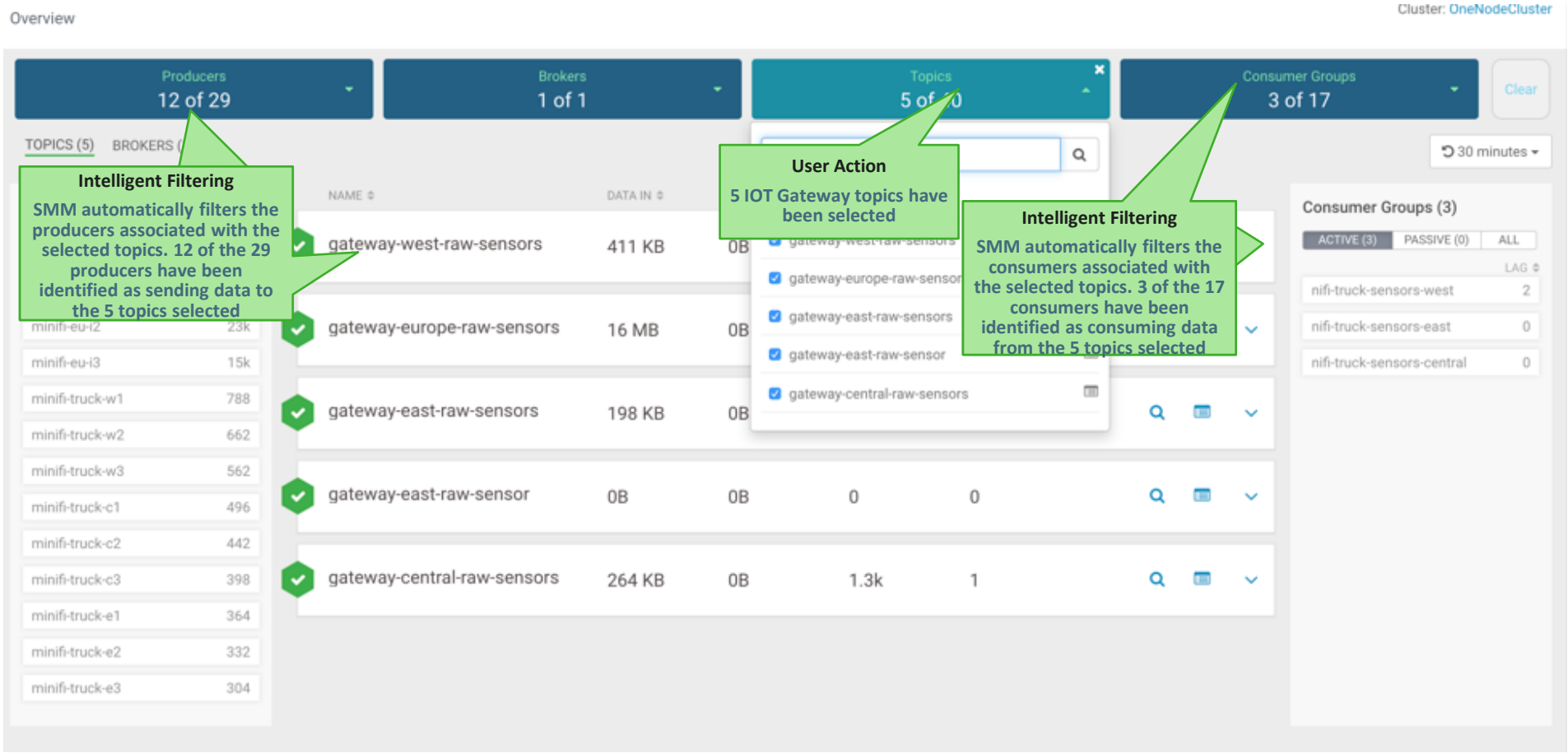


SMM DevOps/App Dev Use Cases

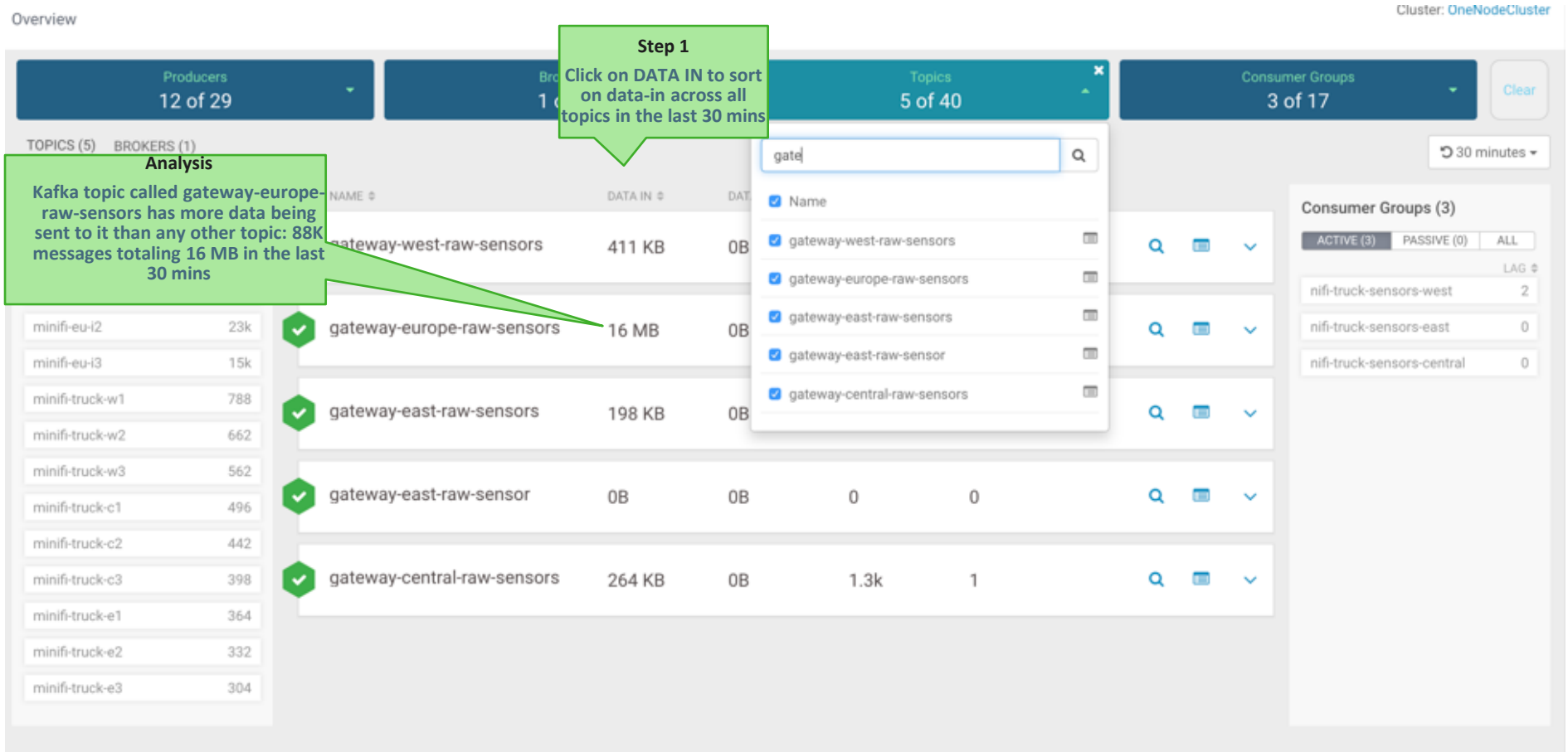
Topic Centric Dashboard View: Filter on Topics associated with my Topic



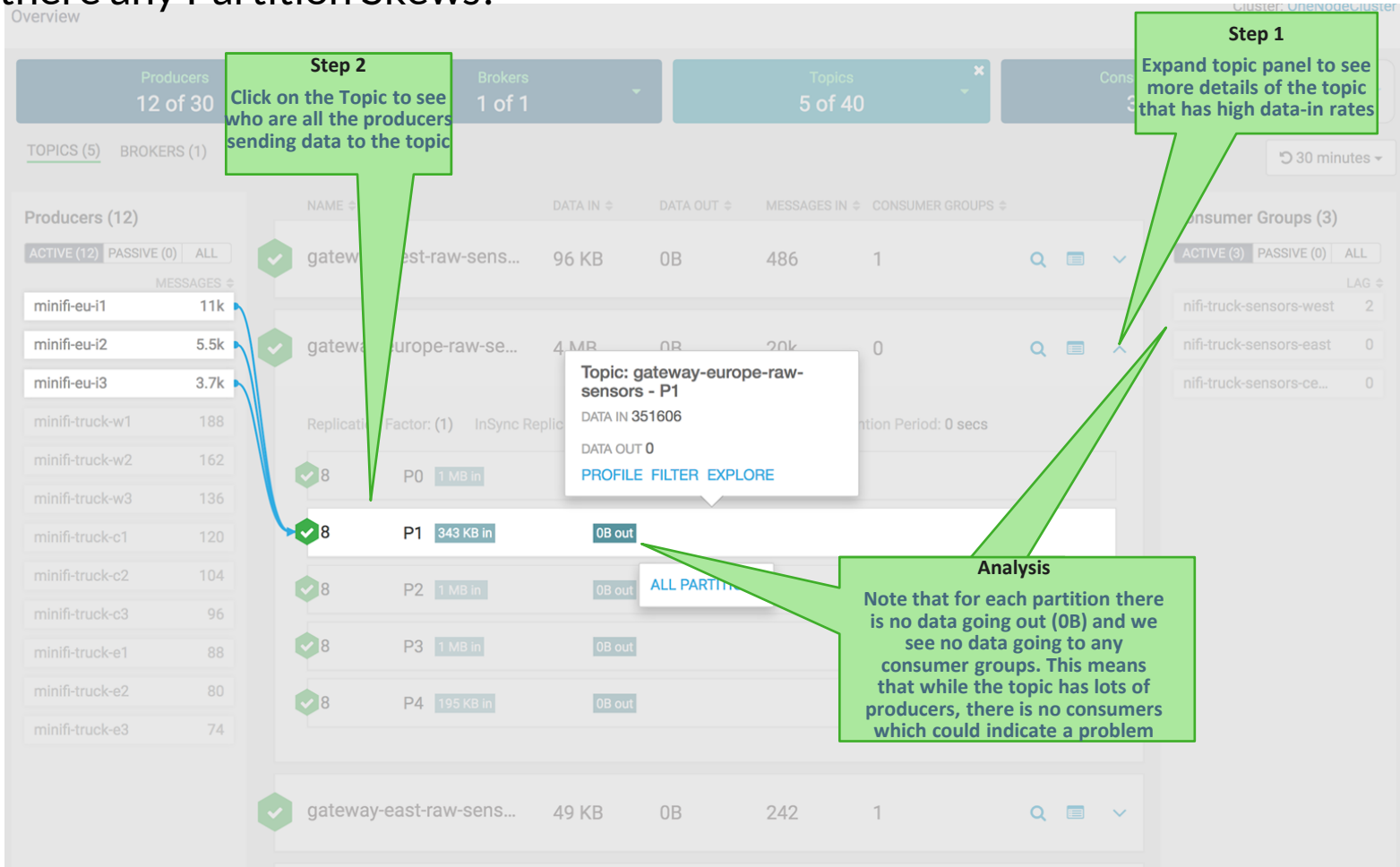
Intelligent Filtering – Selected Topics causes Producers / Consumer to be Intelligently Filtered



Find the Hottest Topic – Topic With Highest Throughput-In

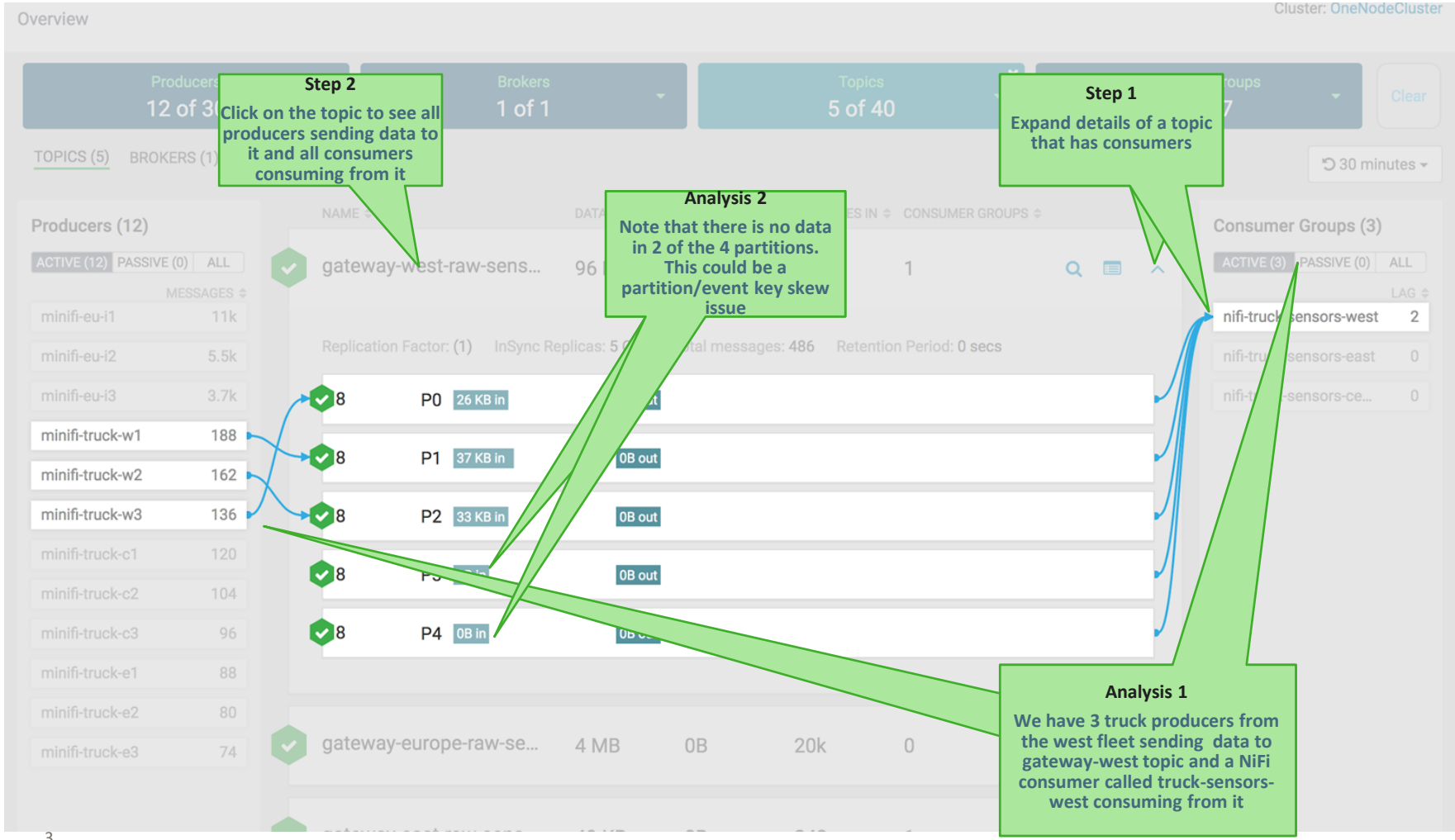


How are the Partitions Laid out for the Topic? Who are the Producers and Consumers? Are there any Partition Skews?

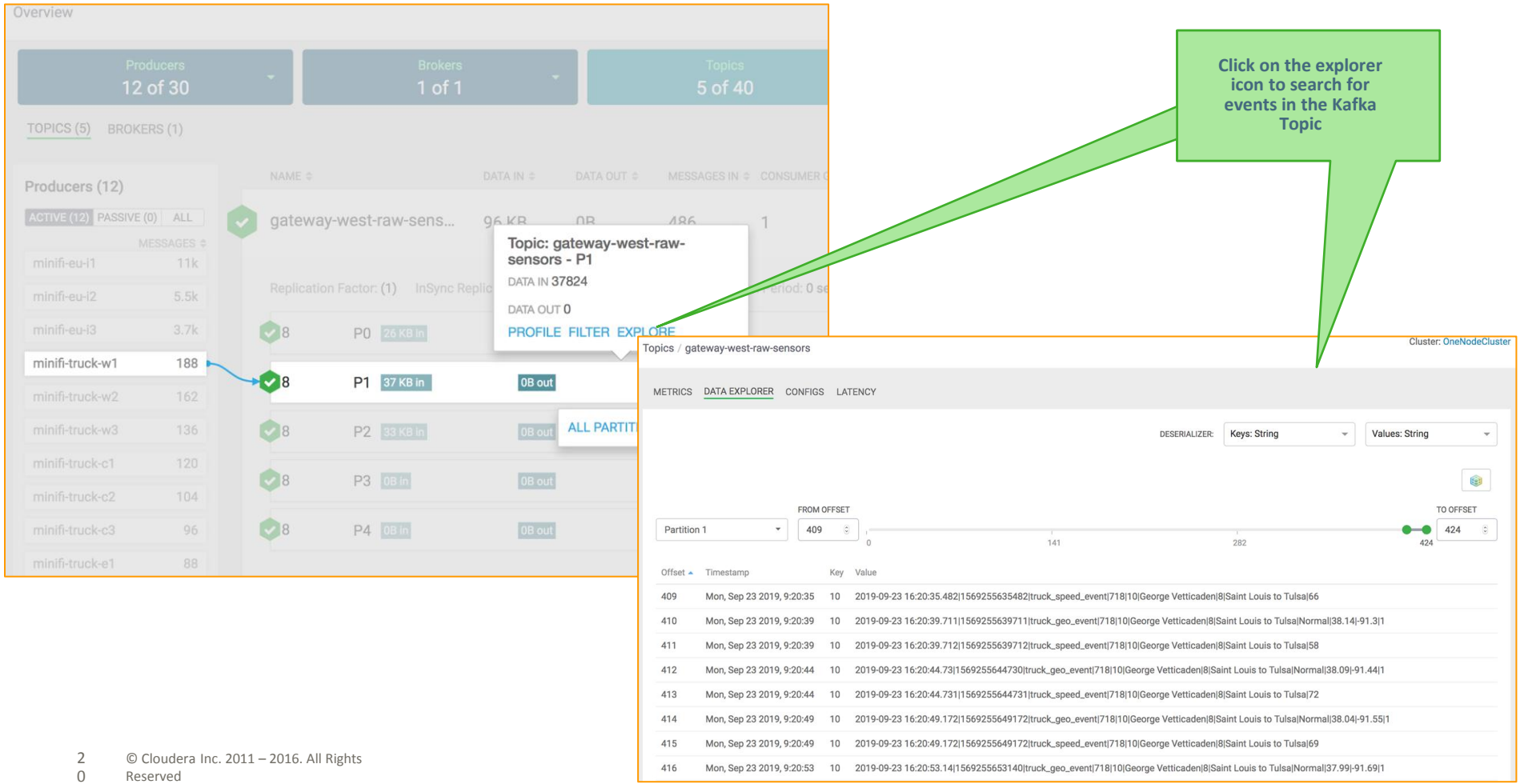


How does Data Flow between Producers to Topics to Consumers?

How does Data Flow between Producers to Topics to Consumers?

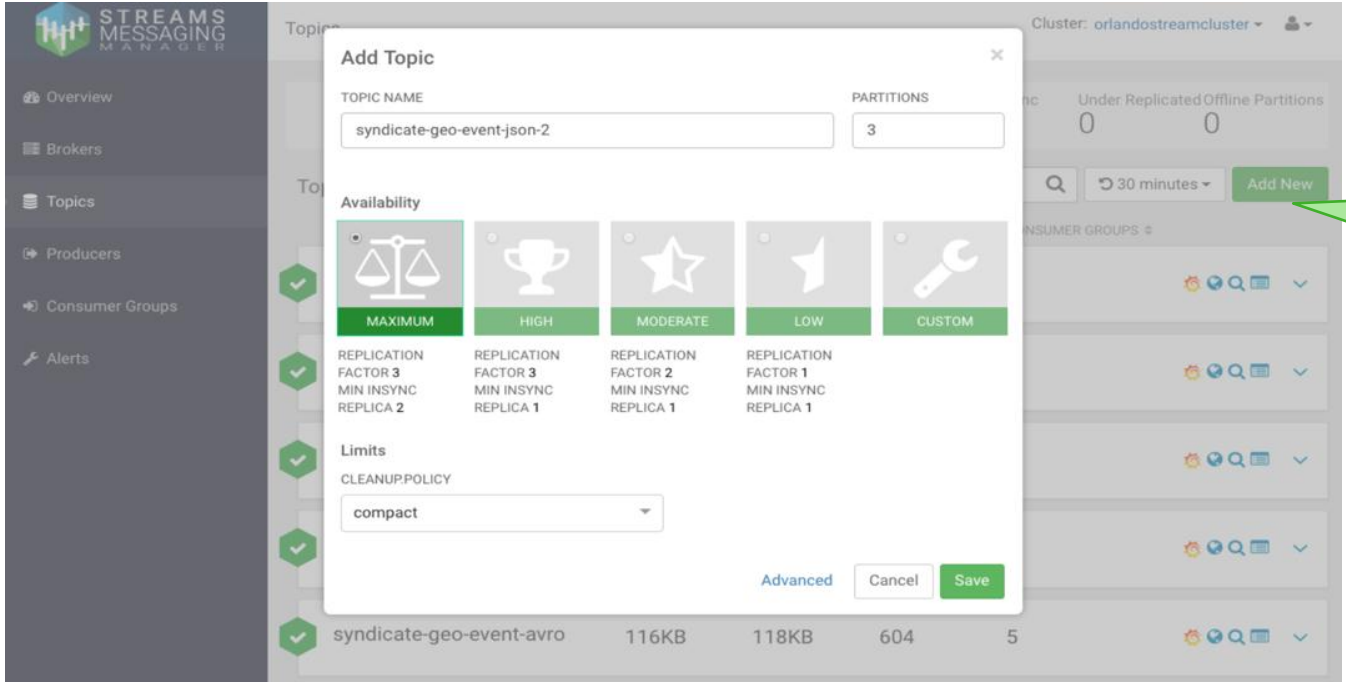


Explore/Search Messages in the Kafka Topic



SMM 1.2 New Features

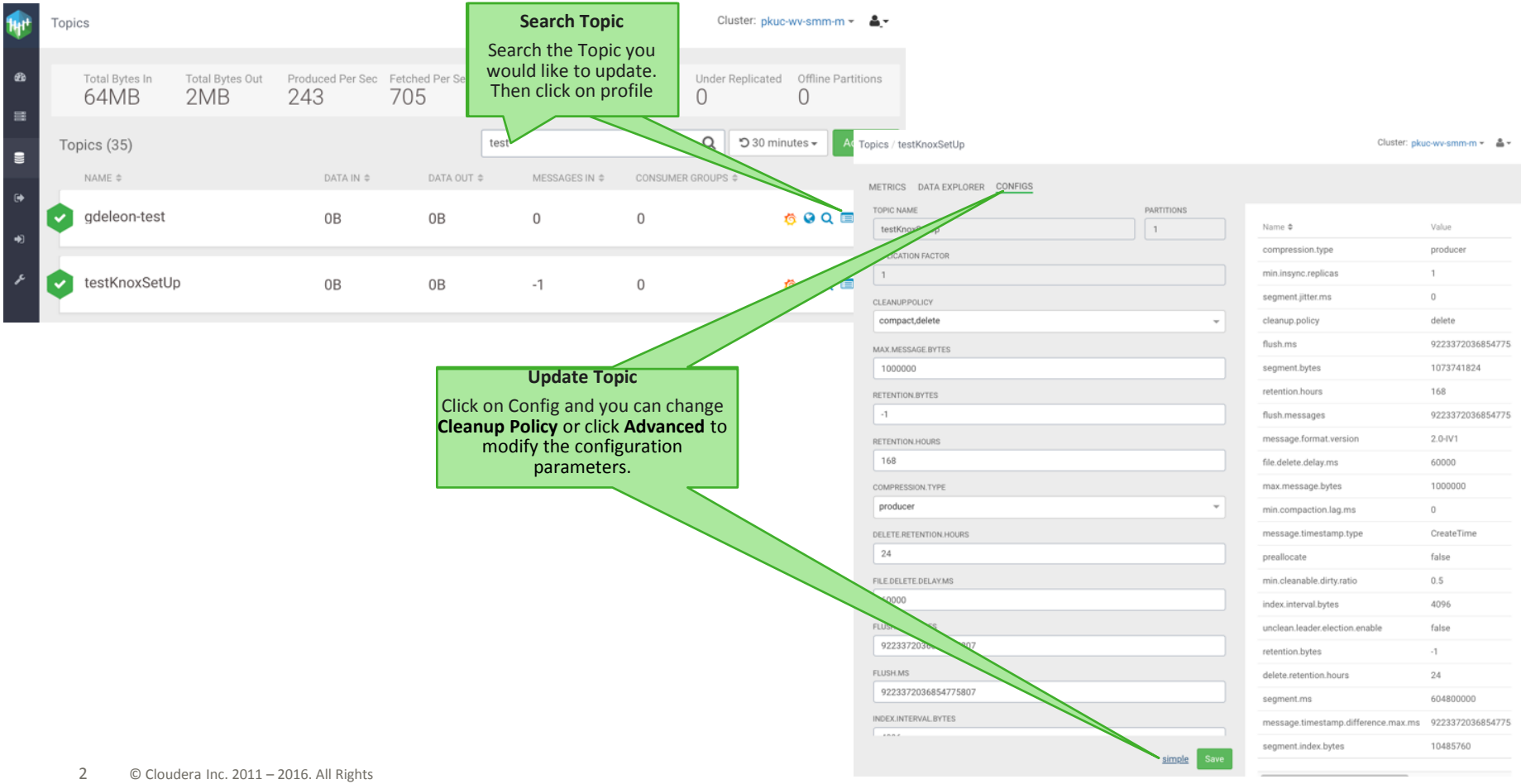
Topic Lifecycle Management



Add Topic

User friendly UI to create new topics. Simple and Advance features are available

Topic Update

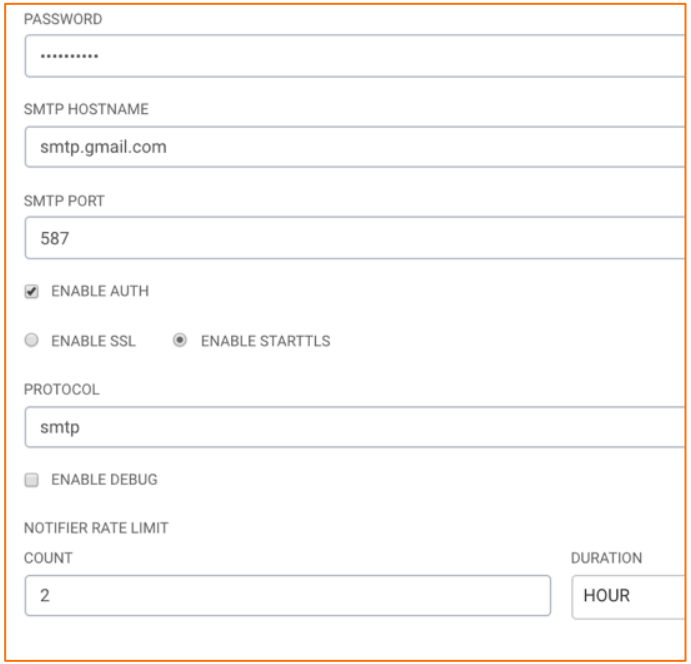


Alerting – Create Notifier

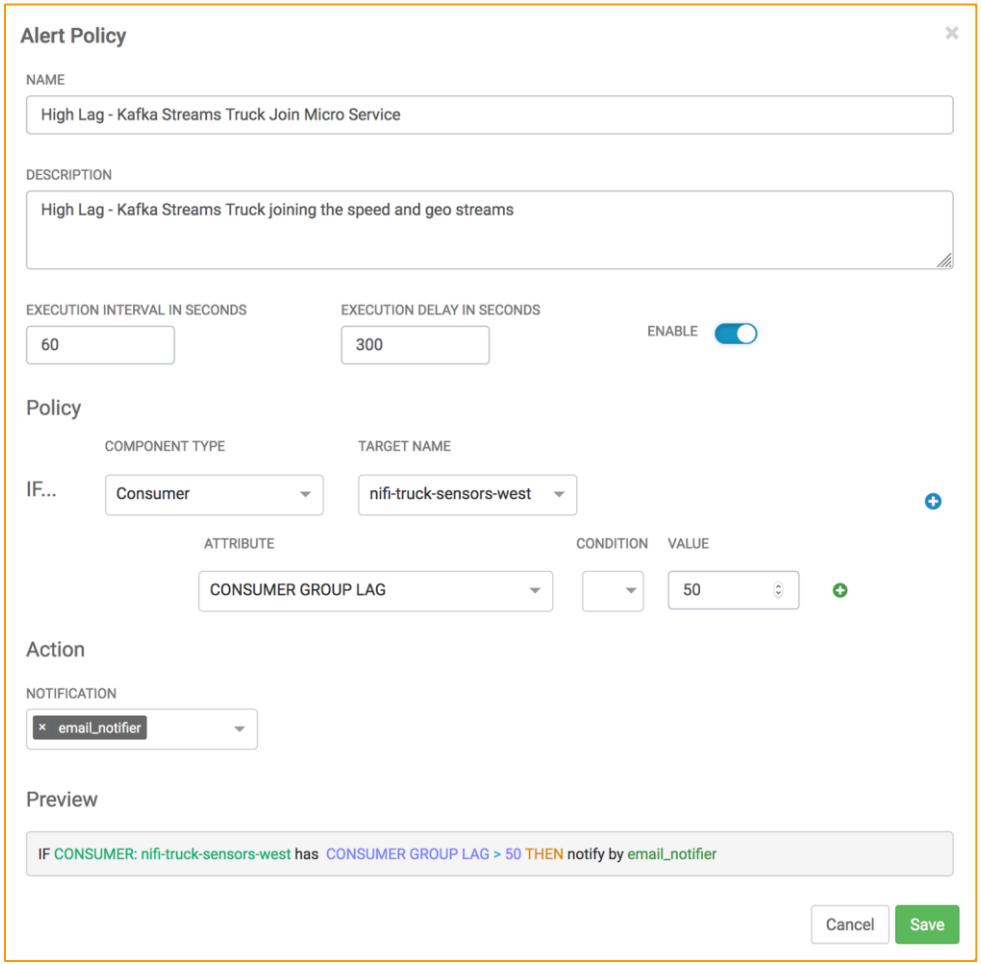
1st Key Construct of Alerts

Alert Notifier

1. Email
2. http end point
3. Kafka topic



Alerting – Create Alert

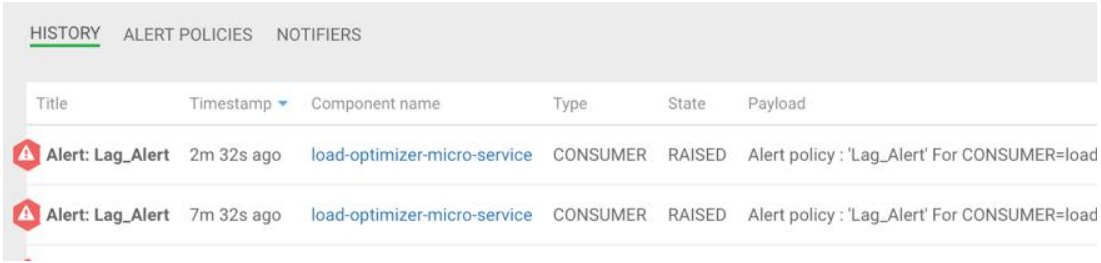


2nd Key Construct of Alerts

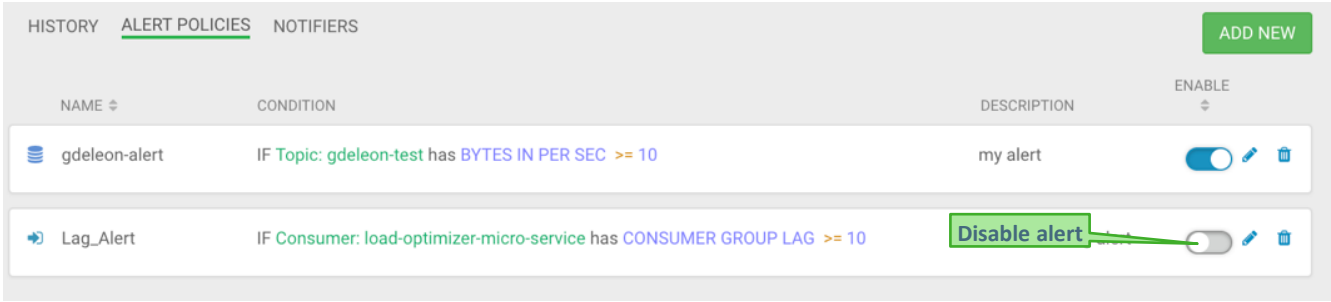
Alert Policy

1. Defined for 6 key entities
   (cluster, broker, topic, producer, consumer, latency, cluster replication)
2. Metrics defined on entities
3. Complex alerts
4. Includes notifier when triggered

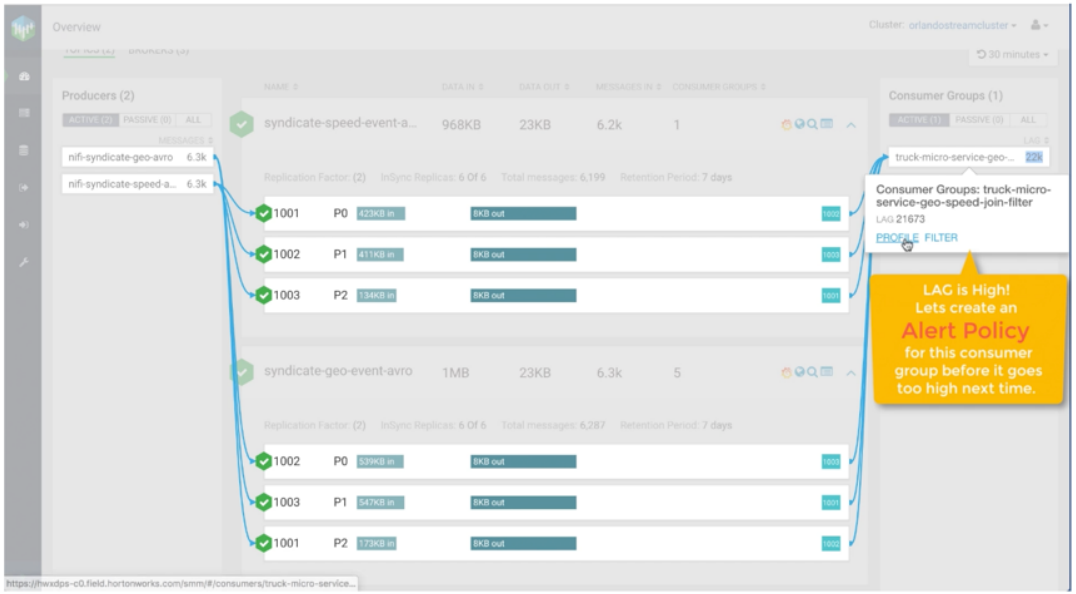
Alert History



Disable Alert



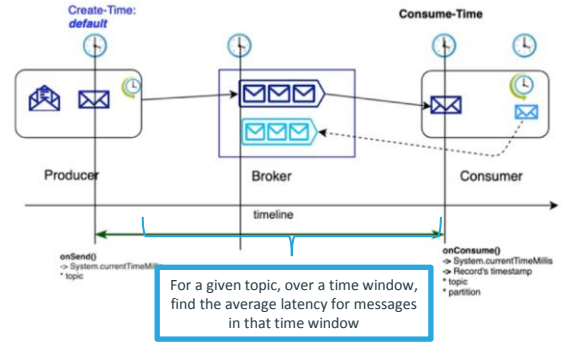
Example: Alerting on Micro-Service Consumer Group with High Lag



New DevOps Monitoring Capability with End to End Latency View

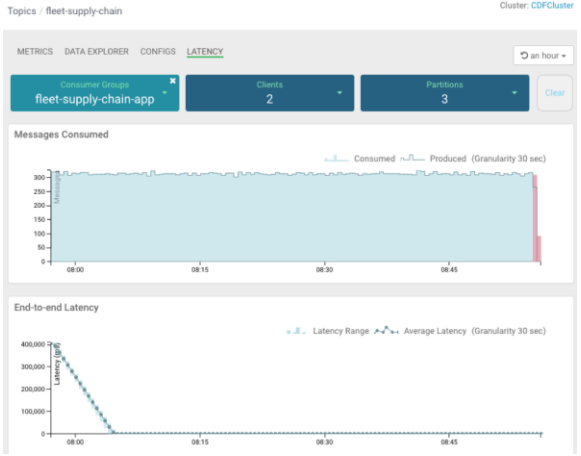
Problem Statement / Requirements

- Kafka DevOps teams need the ability to see an end to end latency view of messages produced and consumed across producers to topics/brokers to consumers
- Common DevOps questions include:
  - For the last five minutes, what is the average latency of messages consumed after being produced.
  - For the last hour, which topics have under consumption?

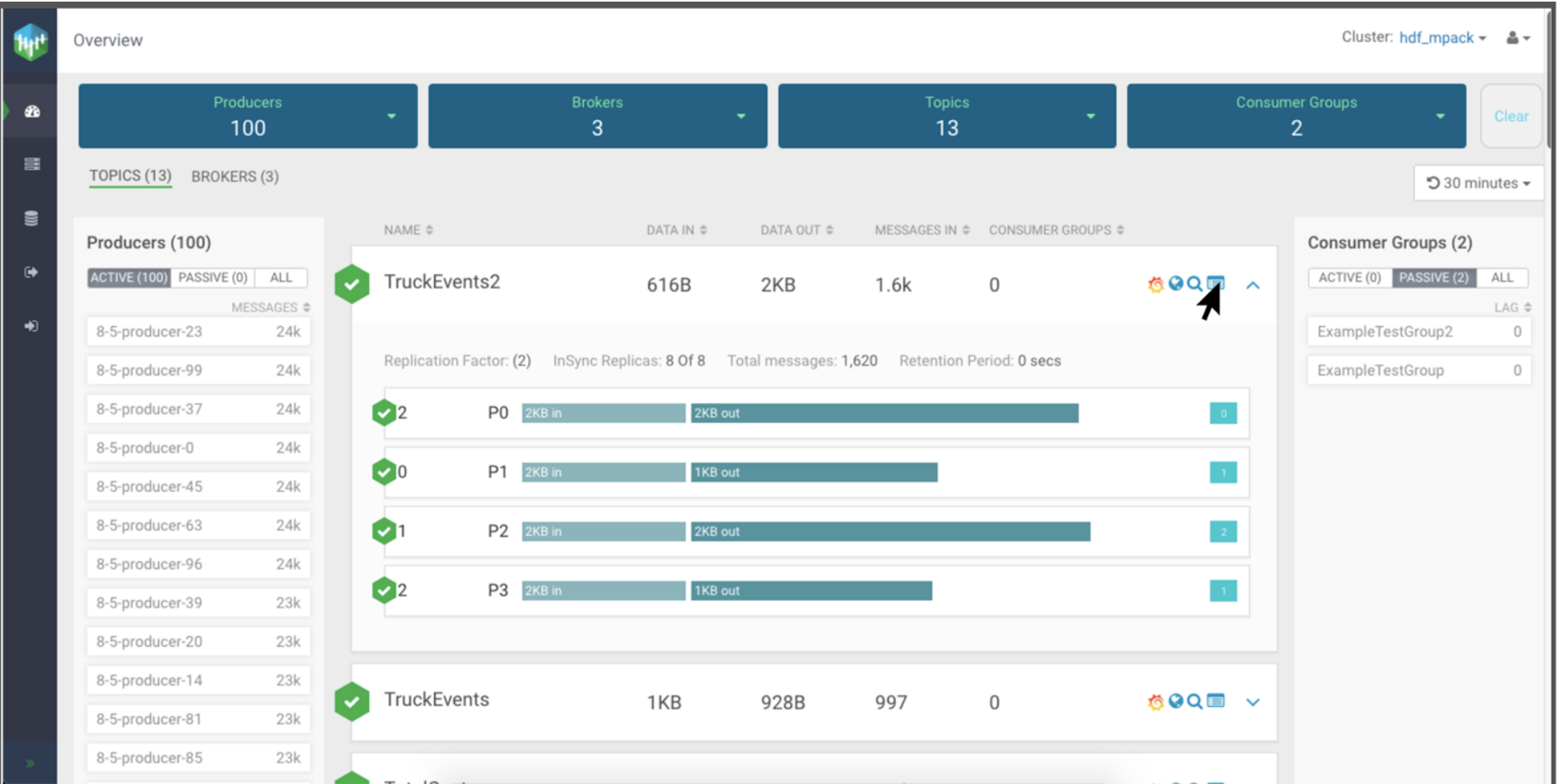


Solution / Benefit

- SMM 2.0 provides new monitoring view for end to end Latency
- End to End Latency view is powered by new embedded Kafka streams application that calculates end to end latencies and new interceptors that can be used to instrument producer and consumer clients.



Select a Topic of Interest

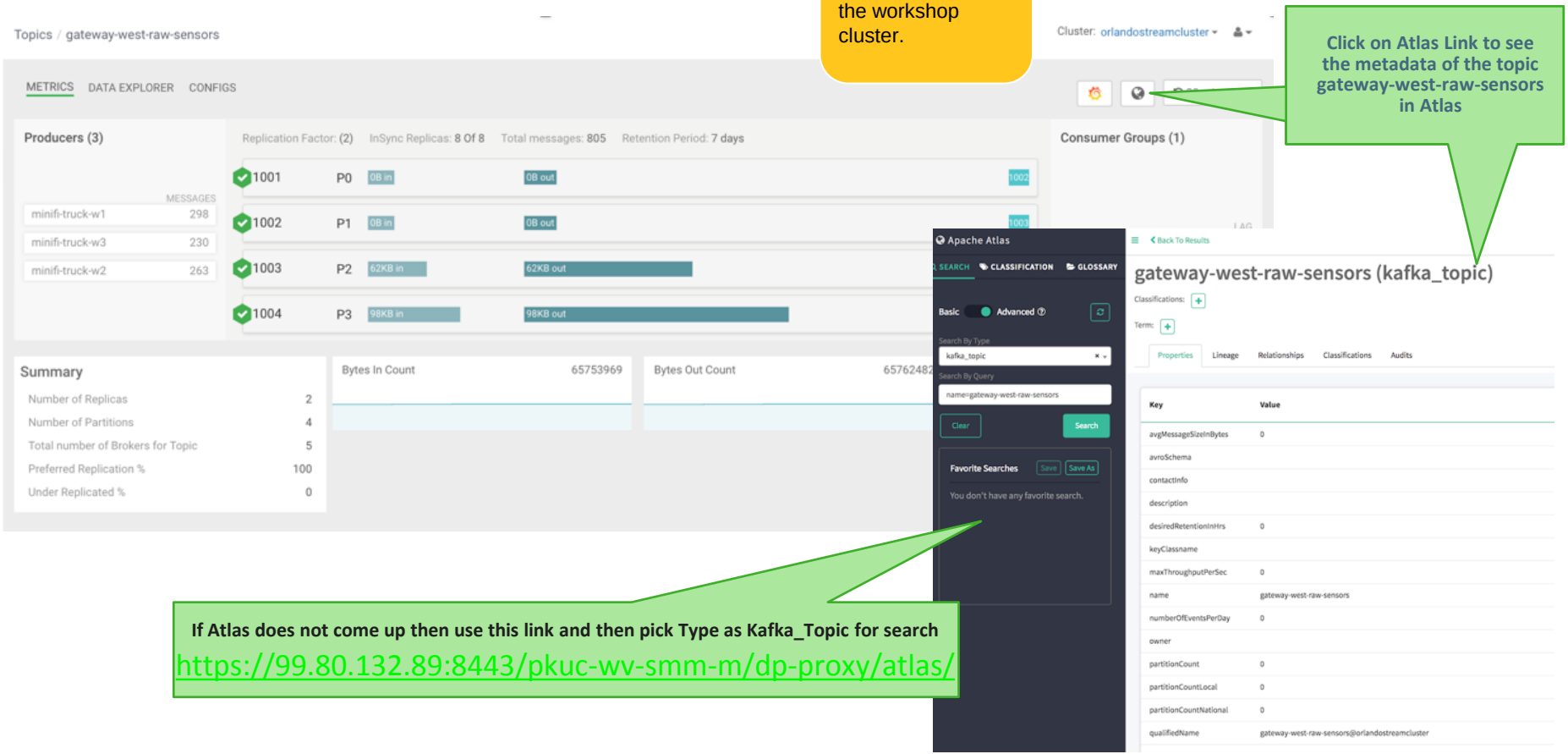


Select the End to End Latency Tab

SMM Atlas Integration

Explore Metadata about the Topic in Atlas

Atlas Integration is not available on the workshop cluster.

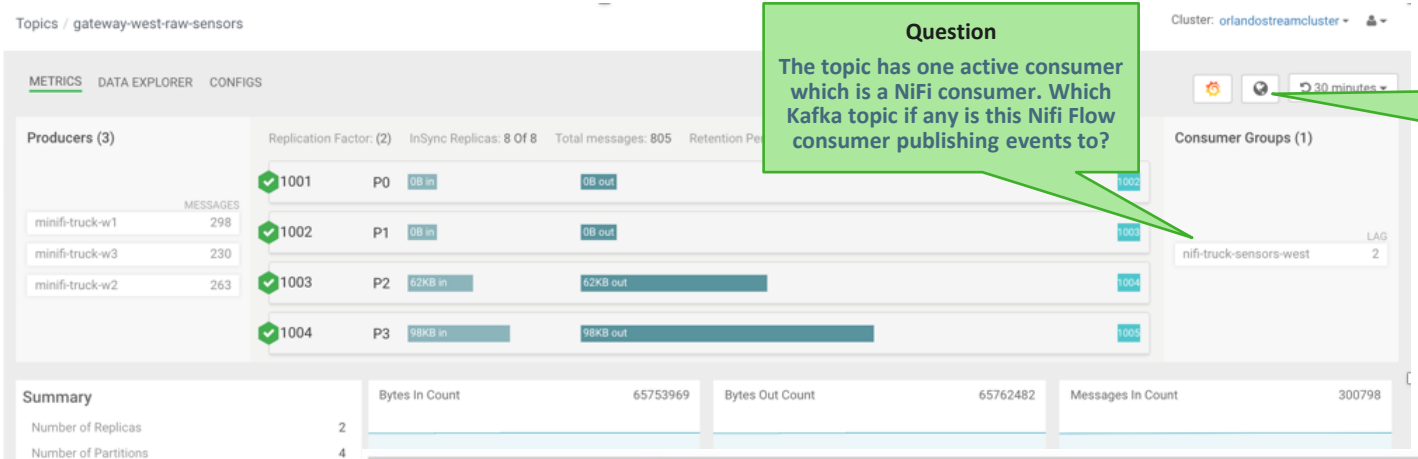


Click on Atlas Link to see the metadata of the topic gateway-west-raw-sensors in Atlas

If Atlas does not come up then use this link and then pick Type as Kafka_Topic for search

https://99.80.132.89:8443/pkuc-wv-smm-m/dp-proxy/atlas/

Traverse the flow of data across multiple Kafka Topics using SMM and Atlas Integration



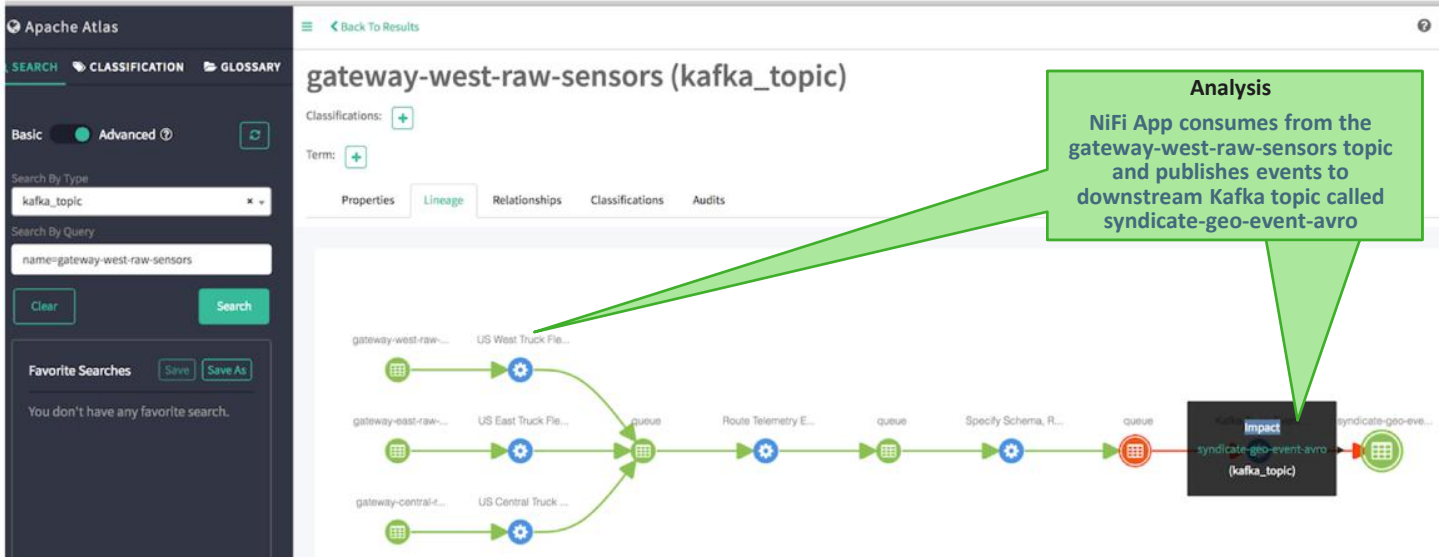
Question

The topic has one active consumer which is a NiFi consumer. Which Kafka topic if any is this Nifi Flow consumer publishing events to?

Step 1

Click on Atlas Icon to see lineage of the the topic gateway-west-raw-sensors

Atlas Integration is not available on the workshop cluster.



Analysis

NiFi App consumes from the gateway-west-raw-sensors topic and publishes events to downstream Kafka topic called syndicate-geo-event-avro

SCHEMA REGISTRY

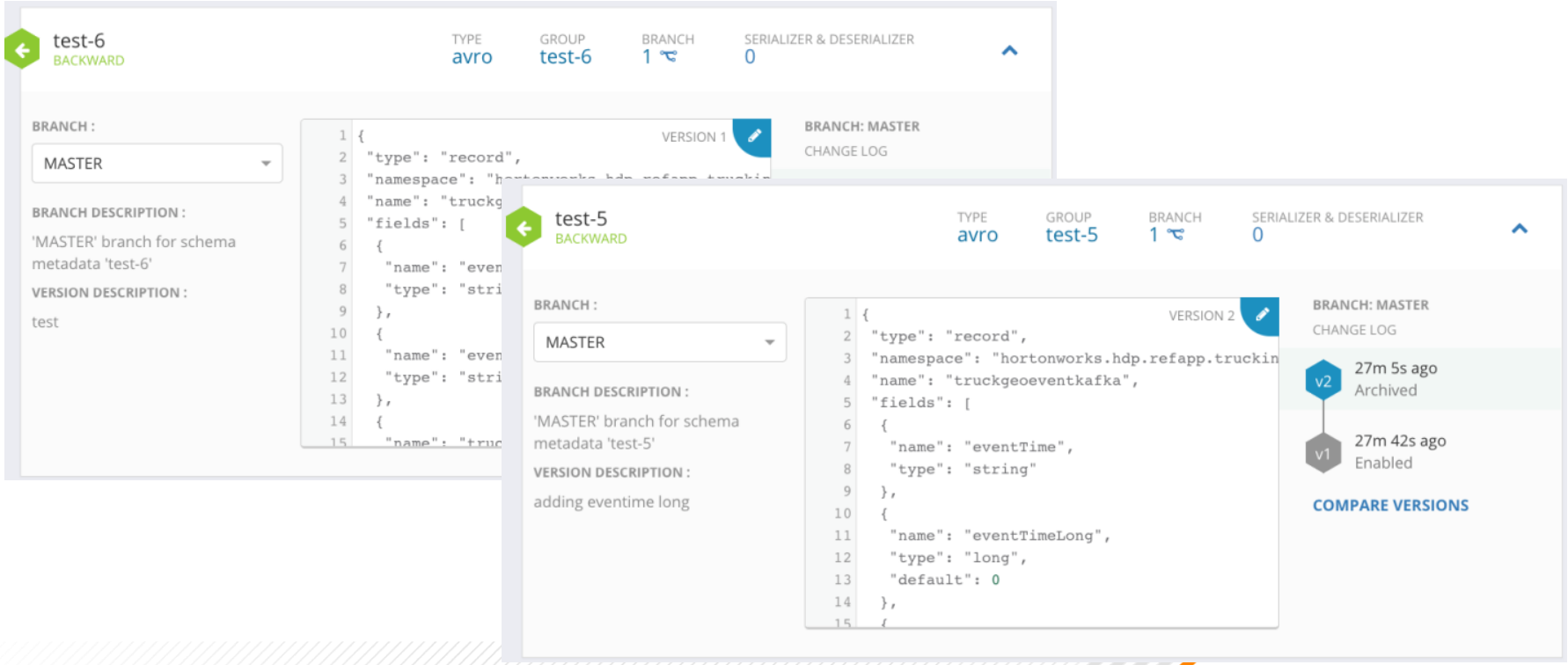
WHAT VALUE DOES SCHEMA REGISTRY PROVIDE?

- **Data Governance**
  - Centralized registry to provide reusable schema
  - Version management to define relationship between schemas
  - Validation to enable generic format conversion and generic routing

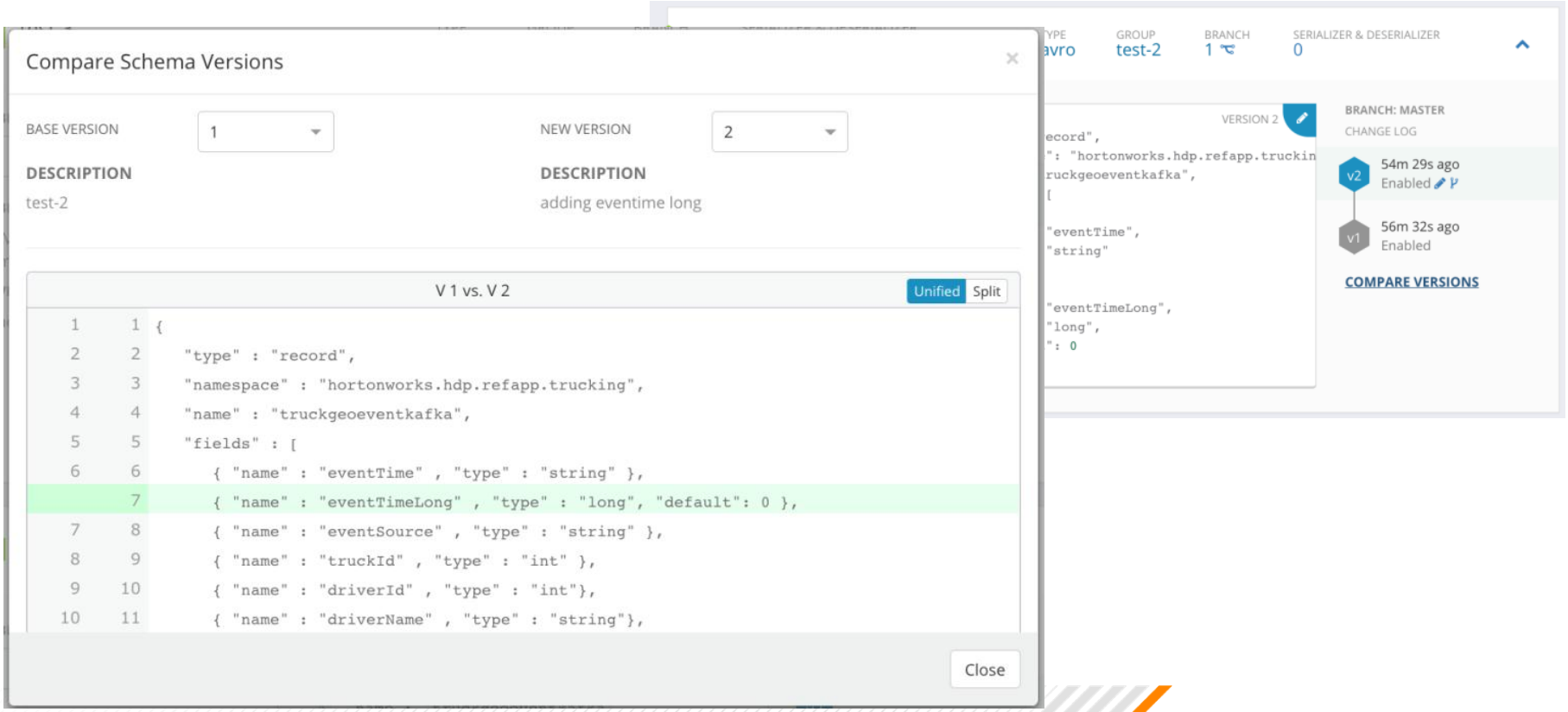
- **Operational Efficiency**
  - Centralized registry to avoid attaching schema to every piece of data
  - Version management to enable consumers and producers can evolve at different rates
  - Validation to ensure data quality

SCHEMA REGISTRY SUPPORT FOR DIFFERENT “STATES”

Enable, disable, archive

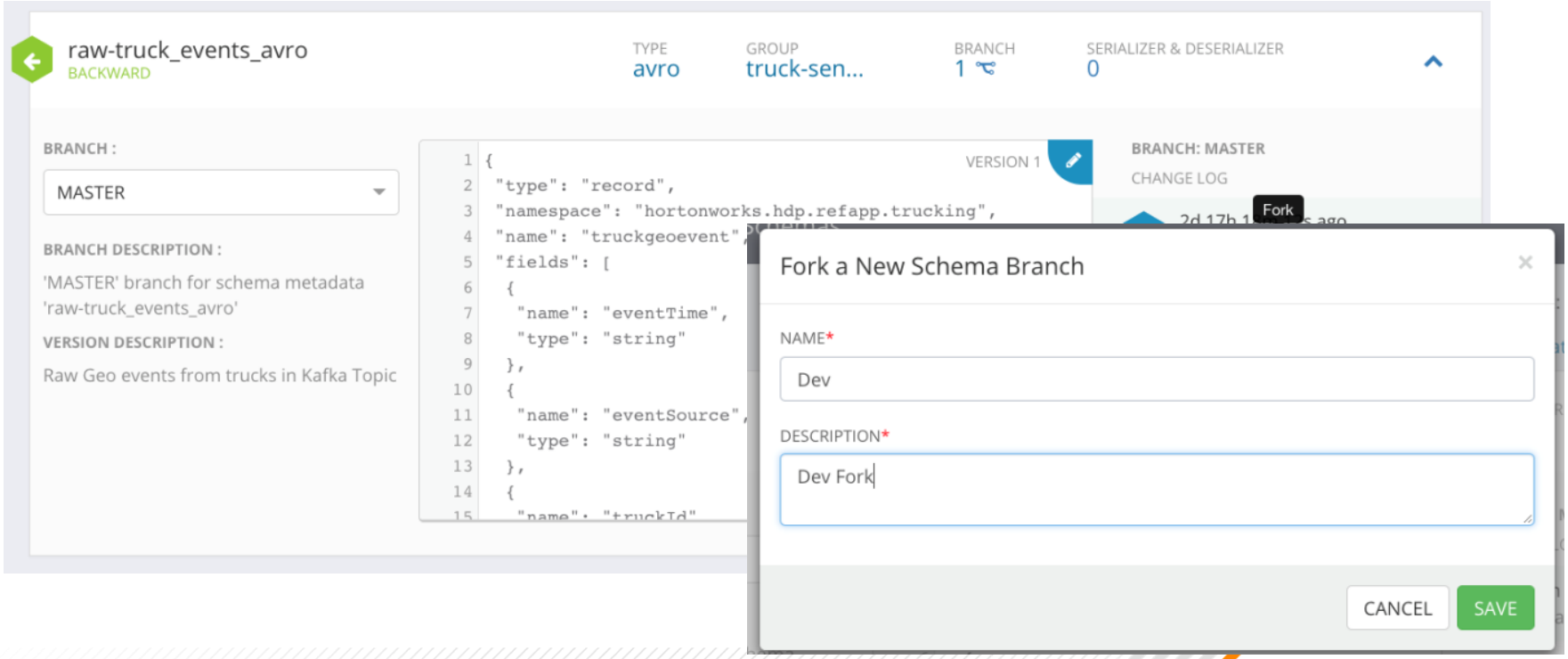


COMPARING DIFFERENT SCHEMA VERSIONS



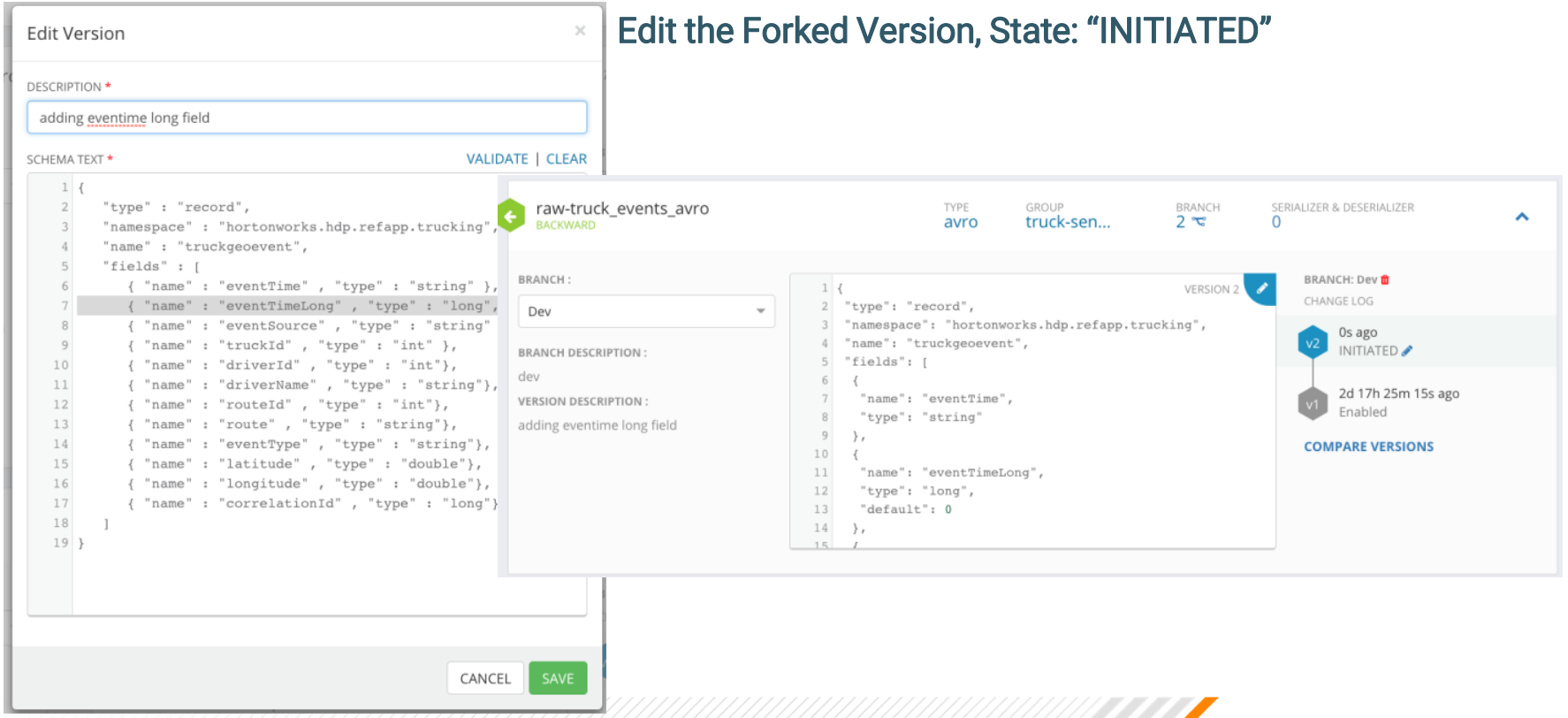
LIFECYCLE ACTION 1 - ACTION

Fork Schema Version to Branch called Dev



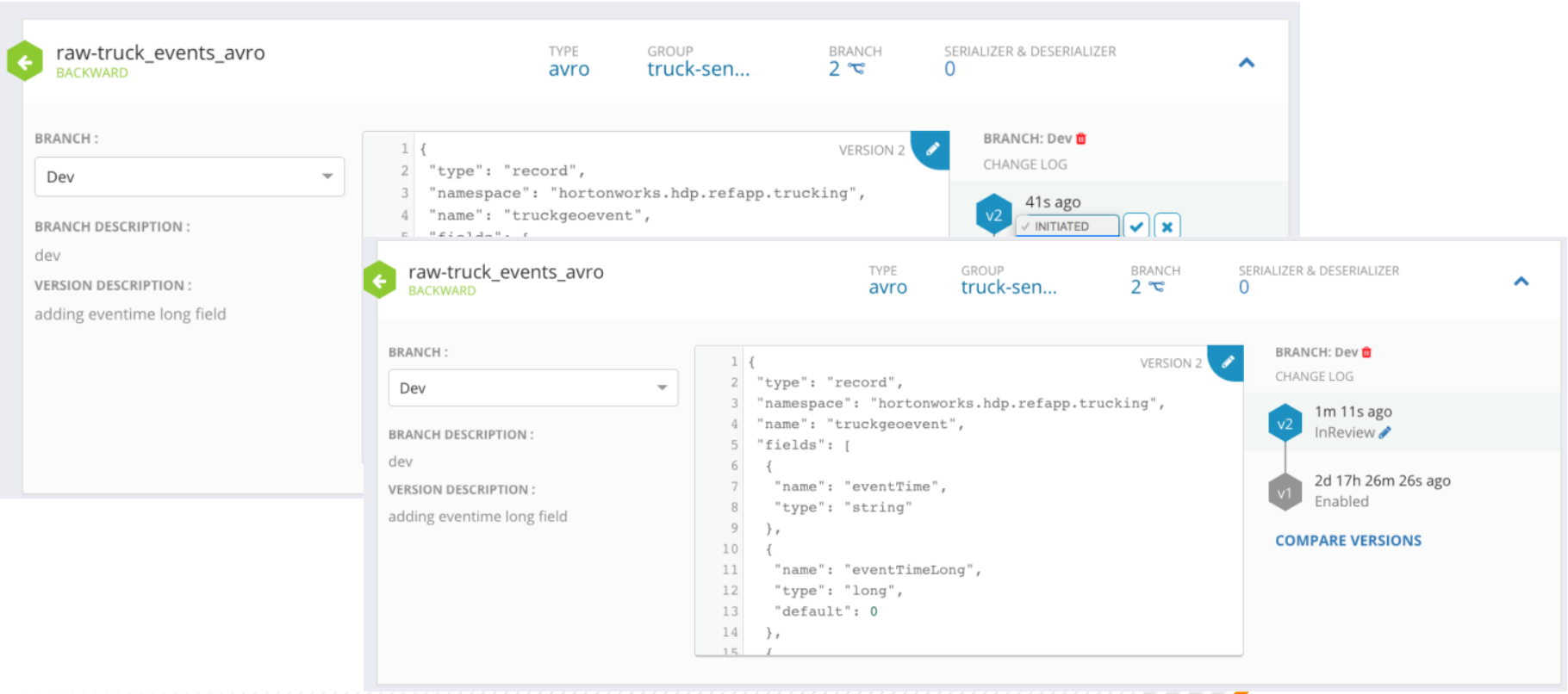
LIFECYCLE ACTION 2 - ACTION

Edit the Forked Version, State: “INITIATED”



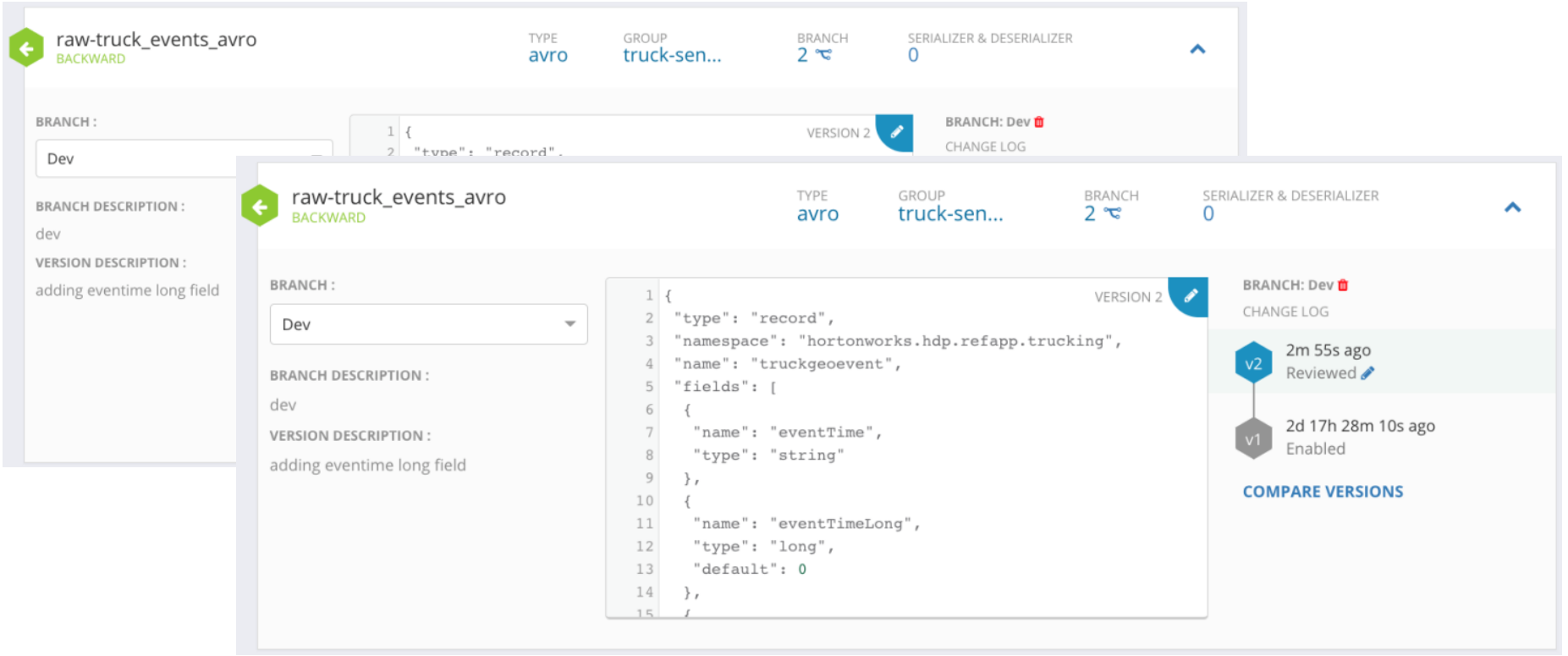
LIFECYCLE ACTION 3 - ACTION

Start Review, State: “InReview”

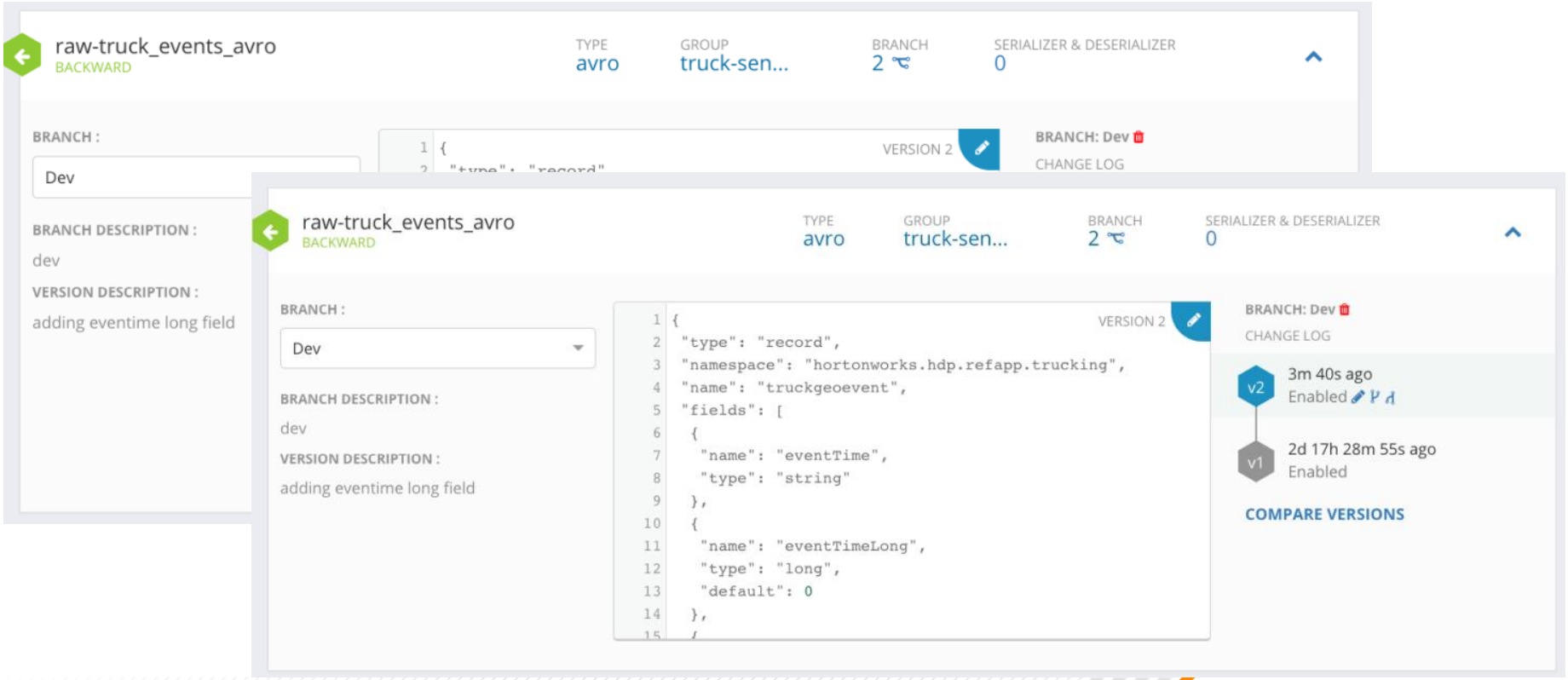


LIFECYCLE ACTION 4 - ACTION

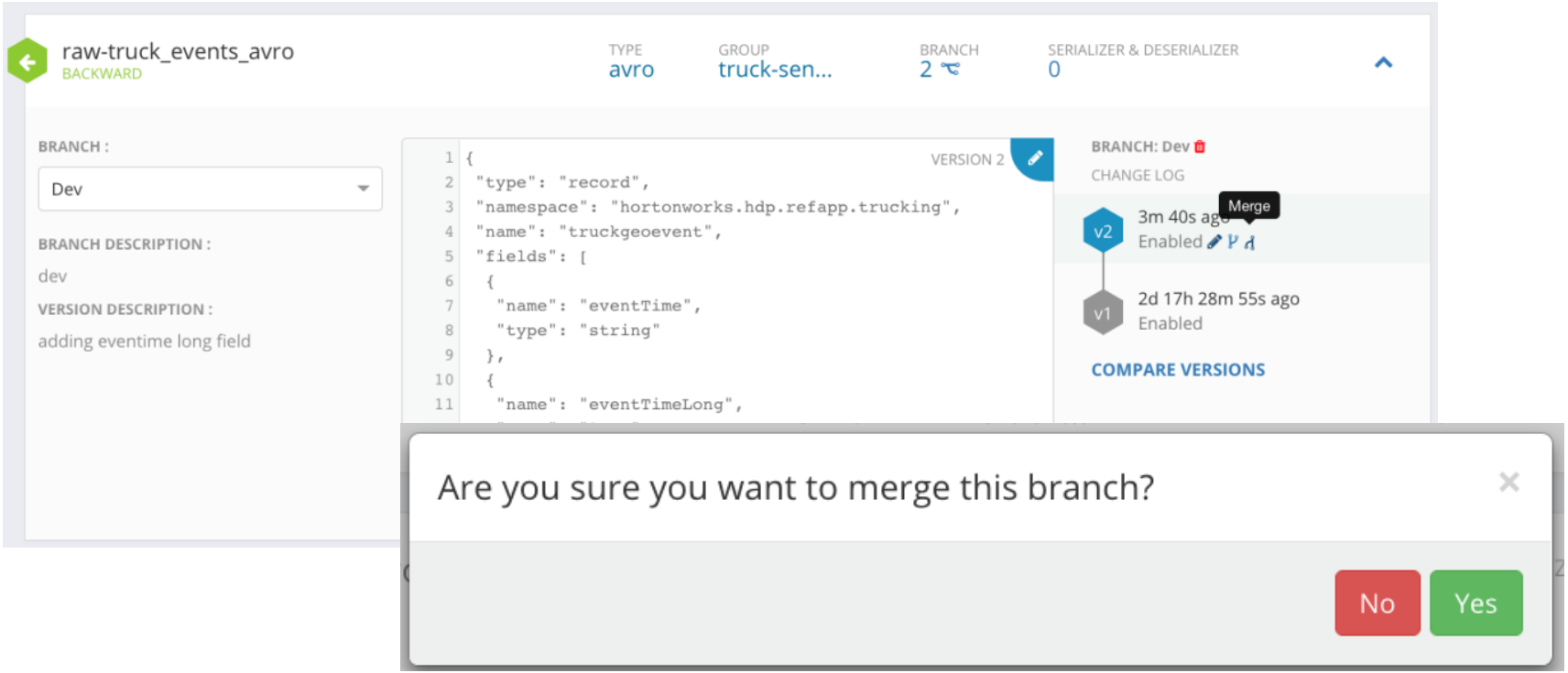
Finish Review, State: “Reviewed”



LIFECYCLE ACTION 5 - ACTION: ENABLE, STATE: “ENABLED”

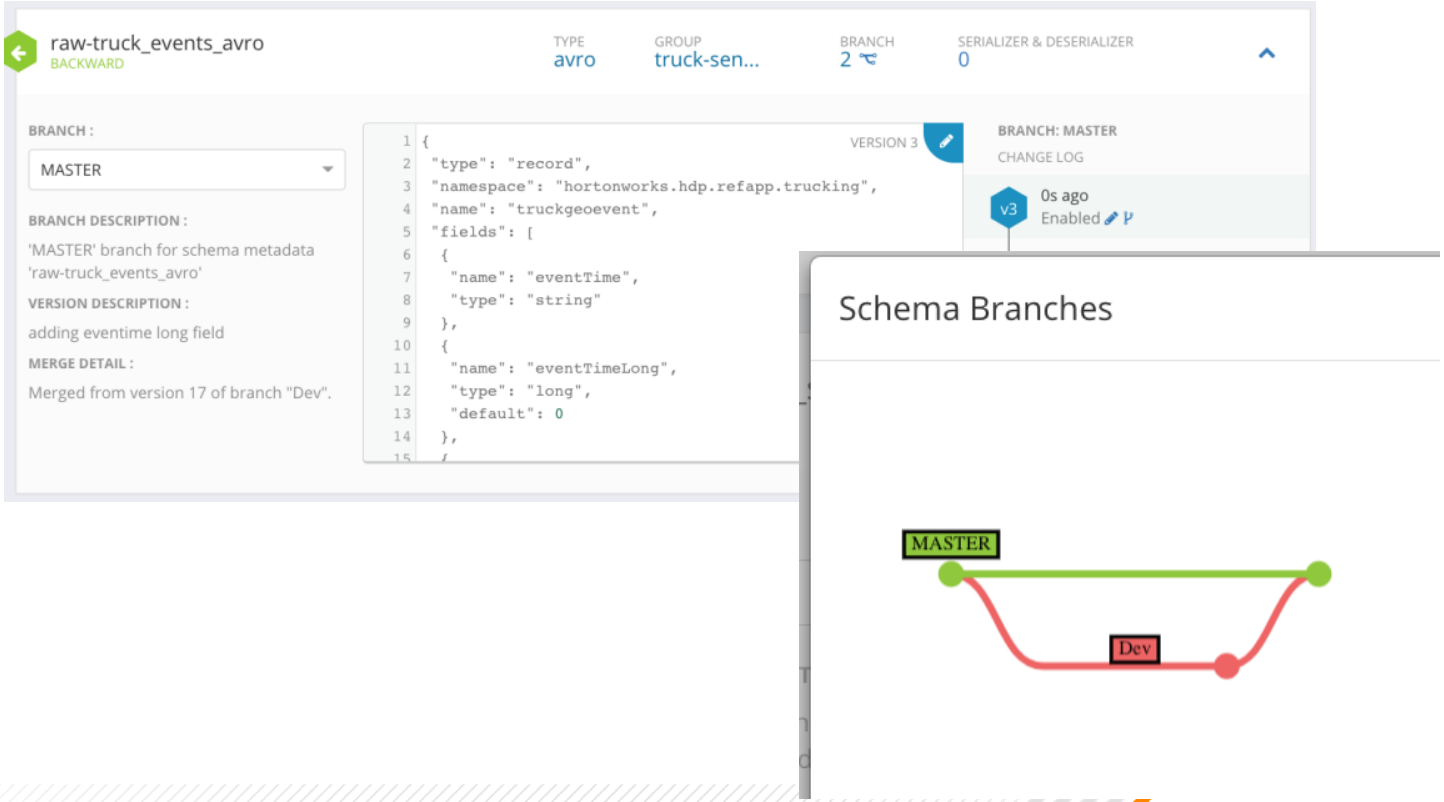


LIFECYCLE ACTION 6 - ACTION: MERGE BACK TO MASTER



COMPLETION OF SCHEMA LIFECYCLE

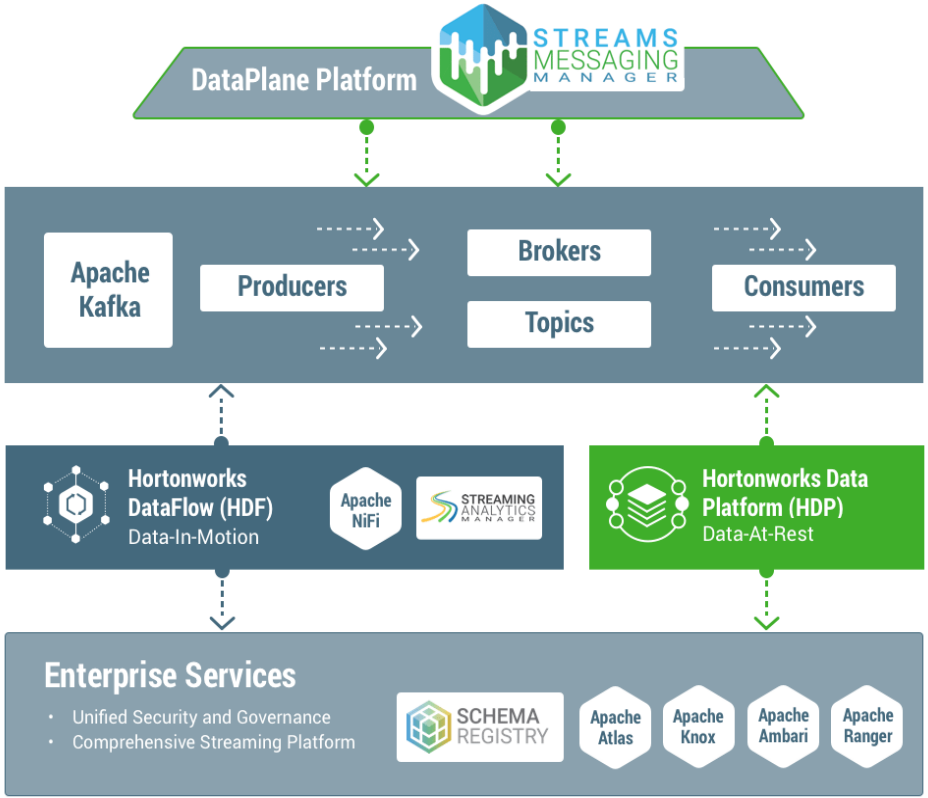
Merged Schema from Dev Branch to Master

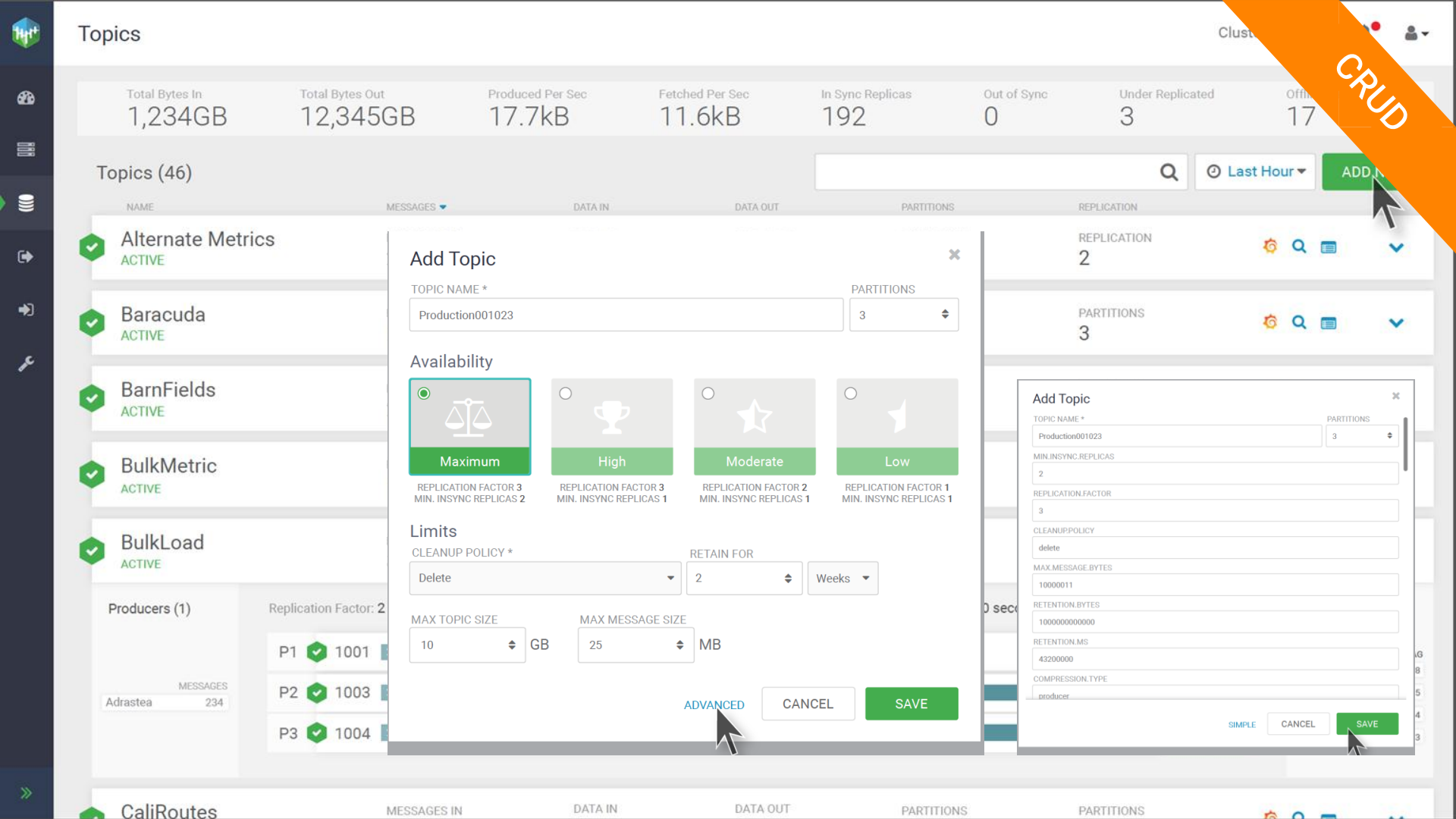


APACHE KAFKA TOPIC CREATION

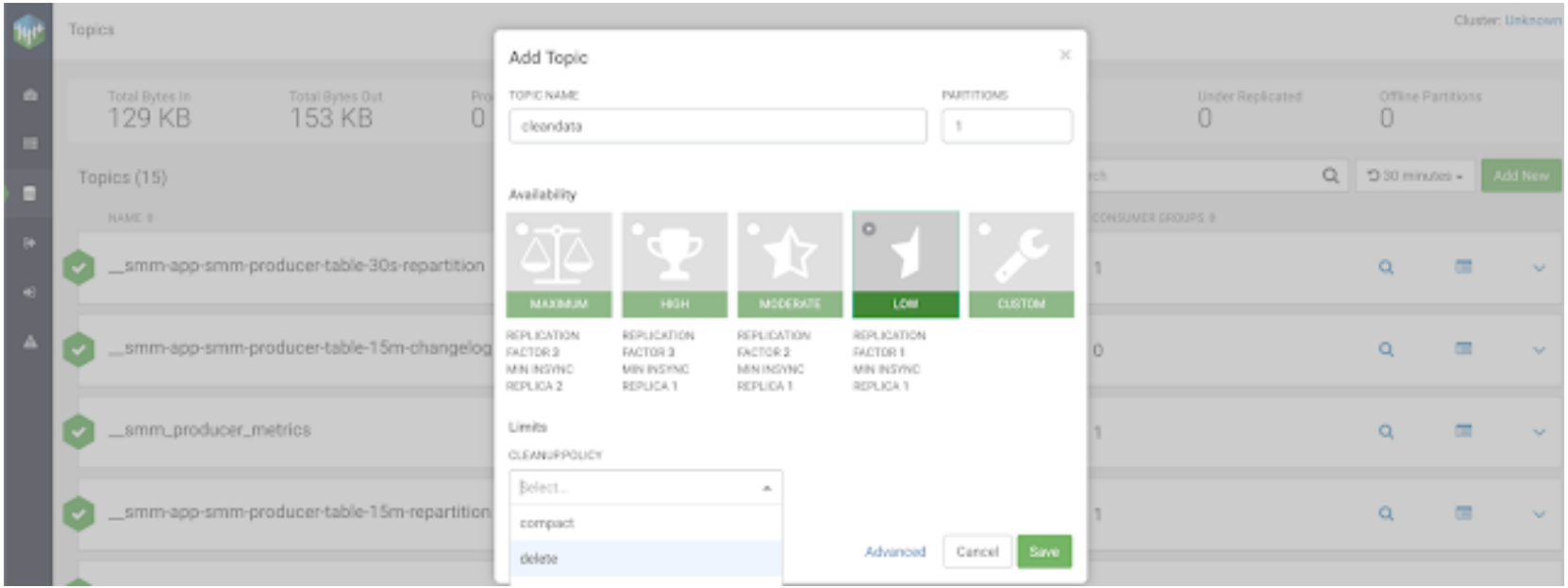
Streams Messaging Manager (SMM)

- Open Source tool to Cure the “Kafka Blindness”
- Single Monitoring Dashboard for all your Kafka Clusters across 4 entities
- Notification on available metrics
- Designed for the Enterprise
  - Support for Secure Kafka cluster
  - Rich Access Control Policies (ACLS)
  - Supports multiple Kafka Clusters
- REST as a First Class Citizen

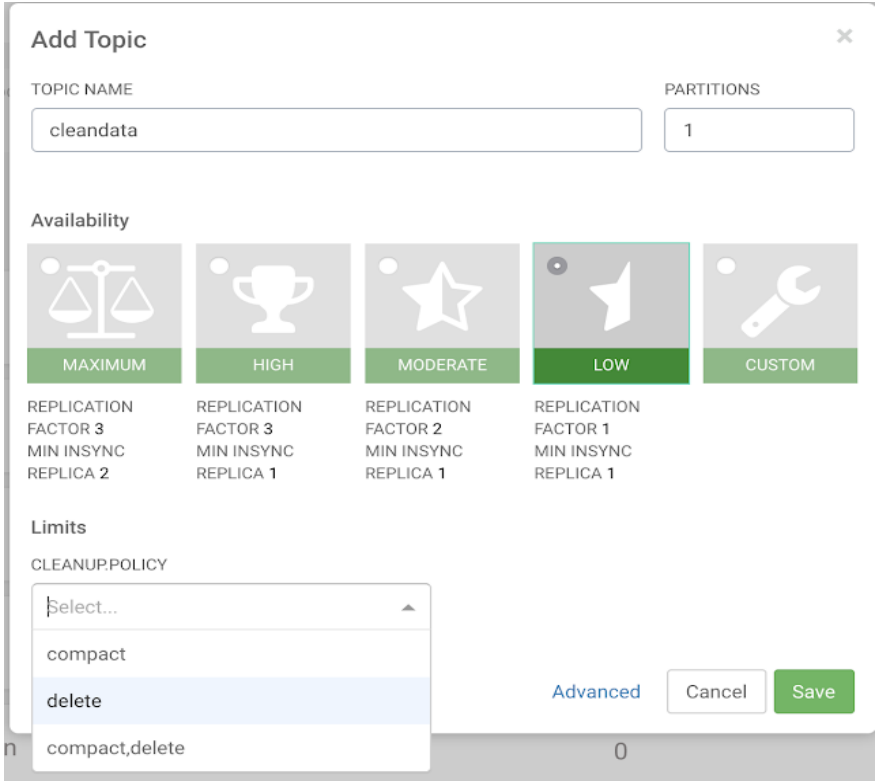



Topics
CRUD
Total Bytes In
1,234GB
Total Bytes Out
12,345GB
Produced Per Sec
17.7kB
Fetched Per Sec
11.6kB
In Sync Replicas
192
Out of Sync
0
Under Replicated
3
Topics (46)
Last Hour
NAME
MESSAGES
DATA IN
DATA OUT
PARTITIONS
REPLICATION
Alternate Metrics
ACTIVE
REPLICATION
2
Baracuda
ACTIVE
PARTITIONS
3
BarnFields
ACTIVE
BulkMetric
ACTIVE
BulkLoad
ACTIVE
Producers (1)
Replication Factor: 2
MESSAGES
Adrastea 234
P1 1001
P2 1003
P3 1004
Add Topic
TOPIC NAME *
Production001023
PARTITIONS
3
Availability
Maximum
REPLICATION FACTOR 3
MIN. INSYNC REPLICAS 2
High
REPLICATION FACTOR 3
MIN. INSYNC REPLICAS 1
Moderate
REPLICATION FACTOR 2
MIN. INSYNC REPLICAS 1
Low
REPLICATION FACTOR 1
MIN. INSYNC REPLICAS 1
Limits
CLEANUP POLICY *
Delete
RETAIN FOR
2
Weeks
MAX TOPIC SIZE
10
GB
MAX MESSAGE SIZE
25
MB
ADVANCED
CANCEL
SAVE
Add Topic
TOPIC NAME *
Production001023
PARTITIONS
3
MIN.INSYNC.REPLICAS
2
REPLICATION.FACTOR
3
CLEANUP.POLICY
delete
MAX.MESSAGE.BYTES
10000011
RETENTION.BYTES
1000000000000
RETENTION.MS
43200000
COMPRESSION.TYPE
producer
SIMPLE
CANCEL
SAVE
CaliRoutes
MESSAGES IN
DATA IN
DATA OUT
PARTITIONS
PARTITIONS

Create a Kafka Topic



Create a Kafka Topic



- Topic Name
- Partitions (1 or more)
- Pick a Template with replicas and replication factor
- Carefully select a Cleanup Policy **DELETE**
  - (your messages need a key)

Create Kafka Topics For Kafka Streams Application

- Topic Name: **streams-plaintext-input**
- Partitions: 1
- Replicas: 1
- Delete

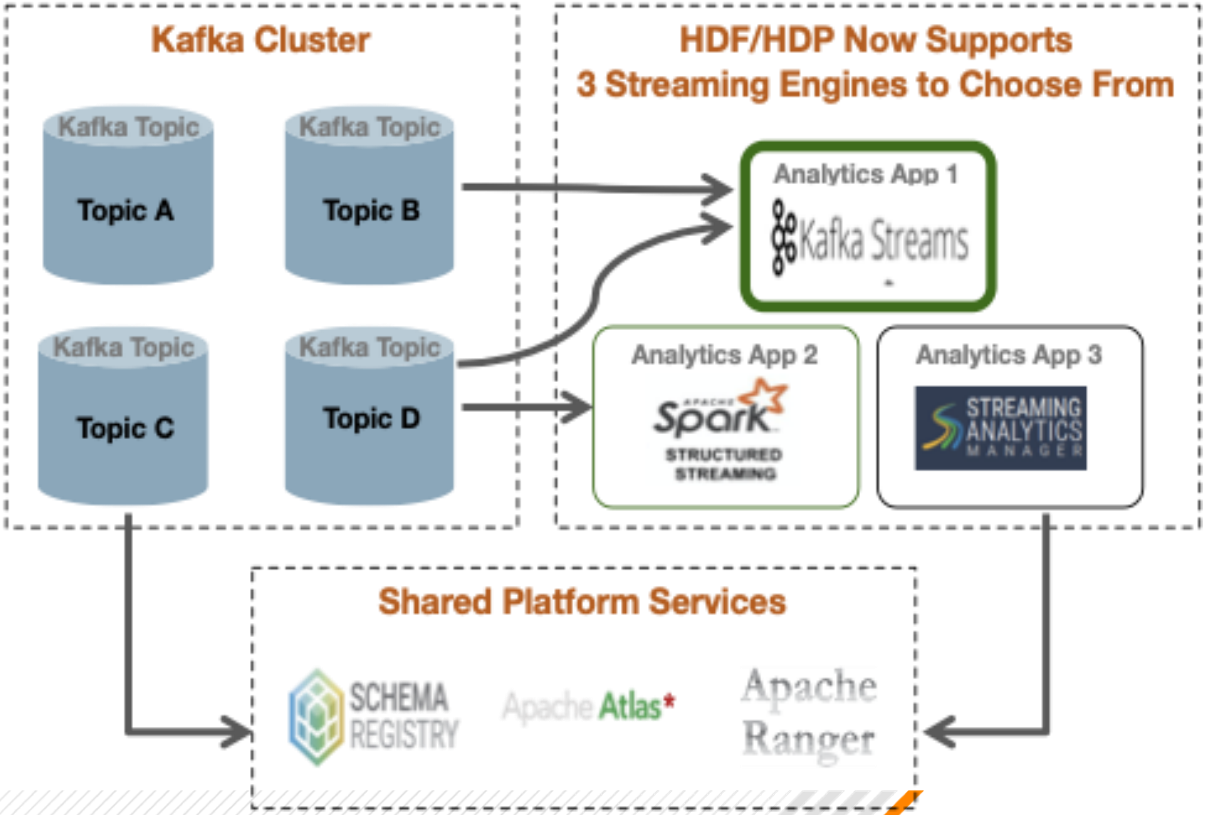
- Topic Name: **streams-wordcount-output**
- Partitions: 1
- Replicas: 1
- Delete

KAFKA STREAMS

Secure and Governed Microservices with Kafka Streams Support

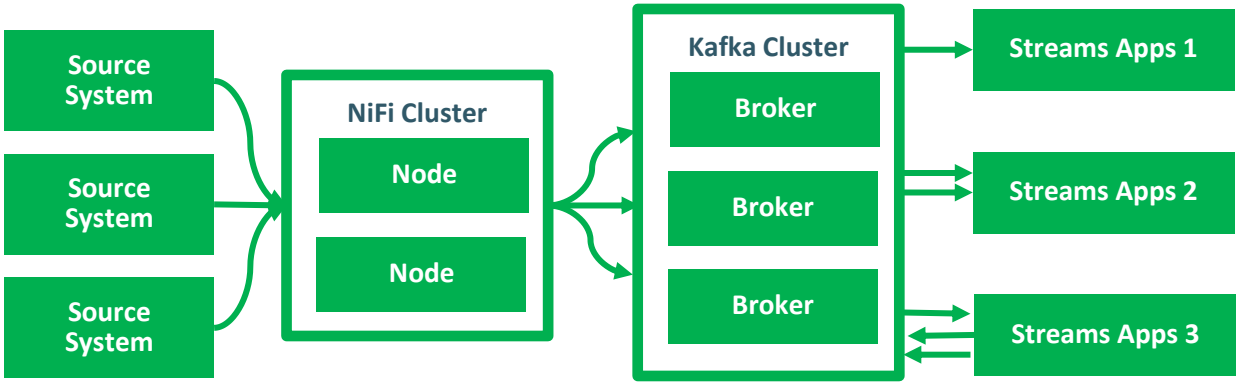
APACHE kafka Streams

API for building real-time applications and microservices on top of Kafka



Kafka Streams

- A client library for applications and microservices on top of Kafka
- Written in Java or Scala
- Elastic, highly scalable, fault-tolerant
- Supports At-Least-Once and Exactly-once semantics inside Kafka
- Deploy to containers, VMs, bare metal, cloud, K8, devices.



Kafka Streams Library

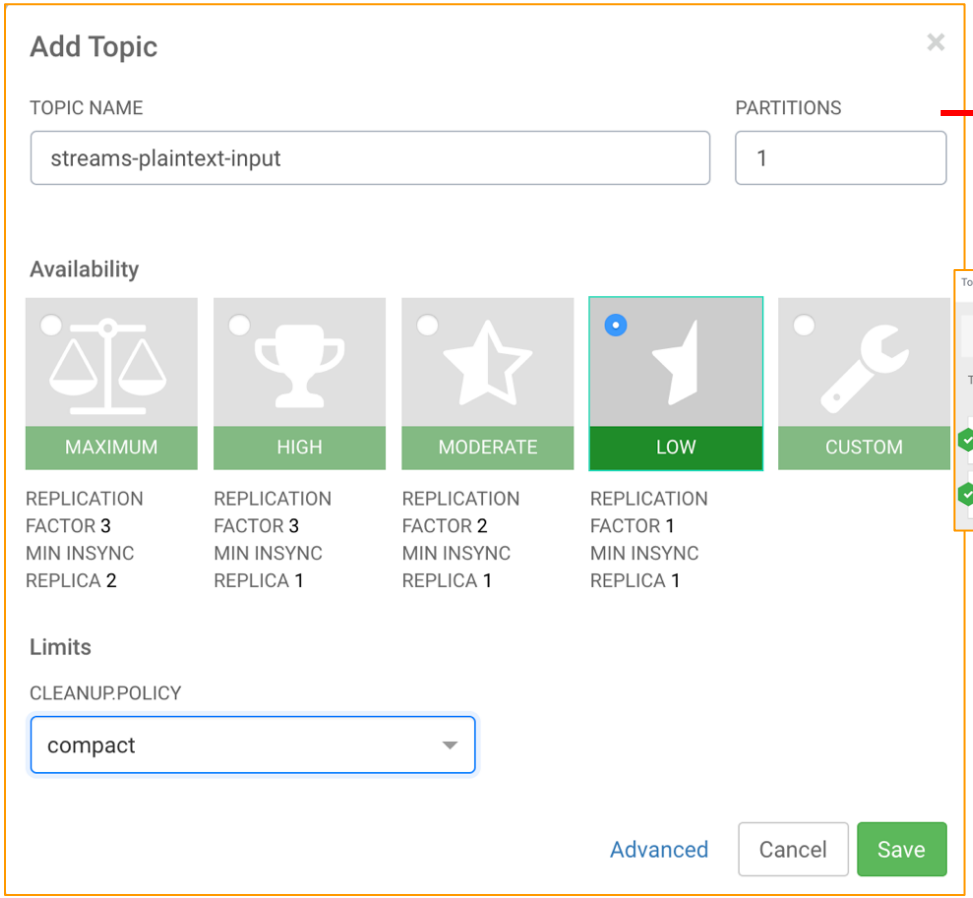
Stream Processing Style	Event a time
Delivery Guarantee	Exactly Once (within Kafka)
State Management	Implicit support with RocksDB
Fault Tolerance	Implicit support with internal Kafka topics / ZK
Advanced feature: Joins/ Aggregations/ Windowing	Stream/Stream joins, Stream/Table joins, joins with only message key, no OOO aggregation processors
Advanced Feature: Watermarking (late arriving data)	Not fully supported
Latency	Low
Throughput	Medium
APIS	Simple APIs (Compositional)

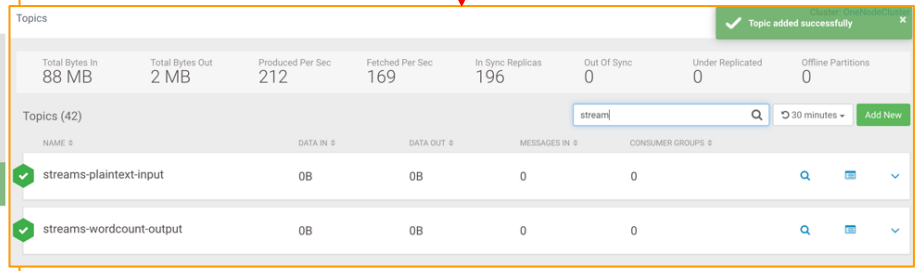
Kafka Streams Library

SQL DSL	Nothing in Apache Kafka.
Lambda Architecture	No Supported
GUI Tooling / Code-Less Approach	Not Supported
Maturity	Relatively New
Use Cases	Lightweight eventing based microservices, ETL
Cluster Requirement	No

KAFKA STREAMS WALK THROUGH

Topic - Create





Setup Your Development Environment

```
sudo su
yum install maven curl wget unzip zip git -y
mkdir /opt/demo
chmod -R 777 /opt/demo
cd /opt/demo
```

Generate a Kafka Streams Project

```
mvn archetype:generate \
        -DarchetypeGroupId=org.apache.kafka \
        -DarchetypeArtifactId=streams-quickstart-java \
        -DarchetypeVersion=2.2.0 \
        -DgroupId=streams.examples \
        -DartifactId=streams.examples \
        -Dversion=0.1 \
        -Dpackage=myapps
```

Update Word Count Kafka Connection

Edit the file
/opt/demo/streams.examples/src/main/java/myapps/WordCount.java

```
Properties props = new Properties();
props.put(StreamsConfig.APPLICATION_ID_CONFIG,
"streams-pipe");
props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG,
"<YOURAWSINTERNALIPNAME>:9092");
```

Build a Kafka Streams Project

```
rm /opt/demo/streams.examples/src/main/java/myapps/Pipe.java
rm
/opt/demo/streams.examples/src/main/java/myapps/LineSplit.java
mvn clean package
```

KAFKA TOPIC INTERACTION

Produce Kafka Messages

/opt/cloudera/parcels/CDH/lib/kafka/bin/kafka-console-producer.sh --broker-list <YOURAWSINTERNALIPNAME>:9092 --topic streams-plaintext-input

Instructions: Then type messages with a return.

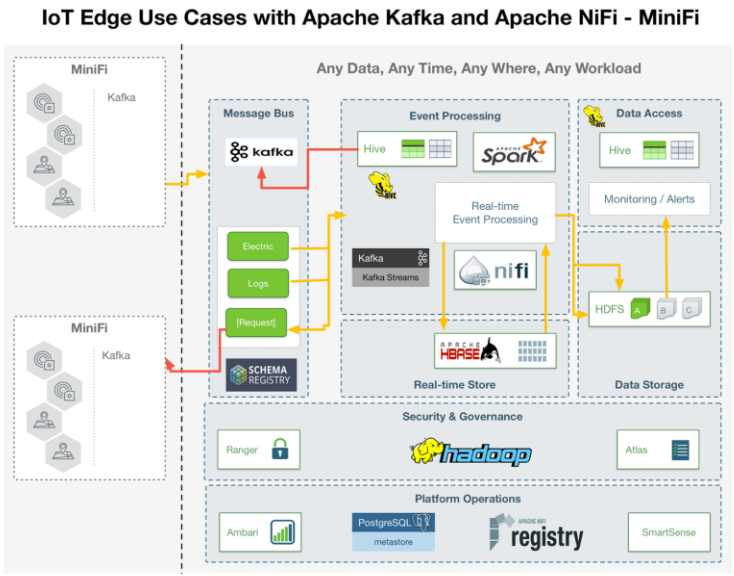
Run Kafka Streams Java Application

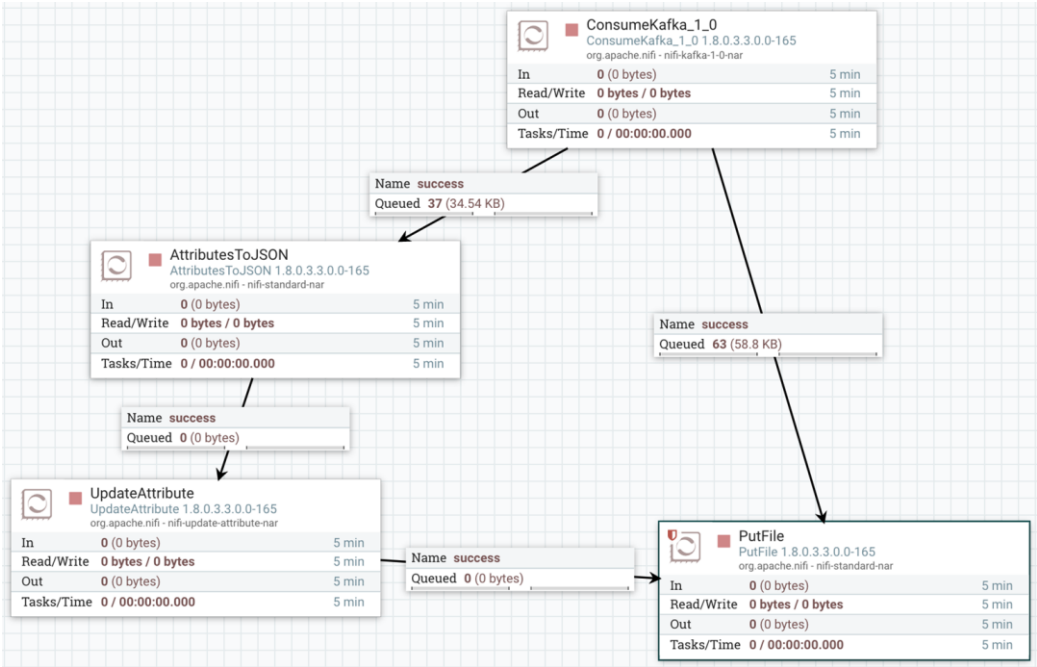
```
mvn exec:java -Dexec.mainClass=myapps.WordCount
```

Consume Kafka Messages

```
/opt/cloudera/parcels/CDH-6.3.0-1.cdh6.3.0.p0.1279813/lib/kafka/bin/kafka-console-consumer.sh --bootstrap-server <YOURAWSINTERNALIPNAME>:9092 \
  --topic streams-wordcount-output \
  --from-beginning \
  --formatter kafka.tools.DefaultMessageFormatter \
  --property print.key=true \
  --property print.value=true \
  --property key.deserializer=org.apache.kafka.common.serialization.StringDeserializer \
  --property value.deserializer=org.apache.kafka.common.serialization.LongDeserializer
```

CONSUME AND PRODUCE KAFKA MESSAGES

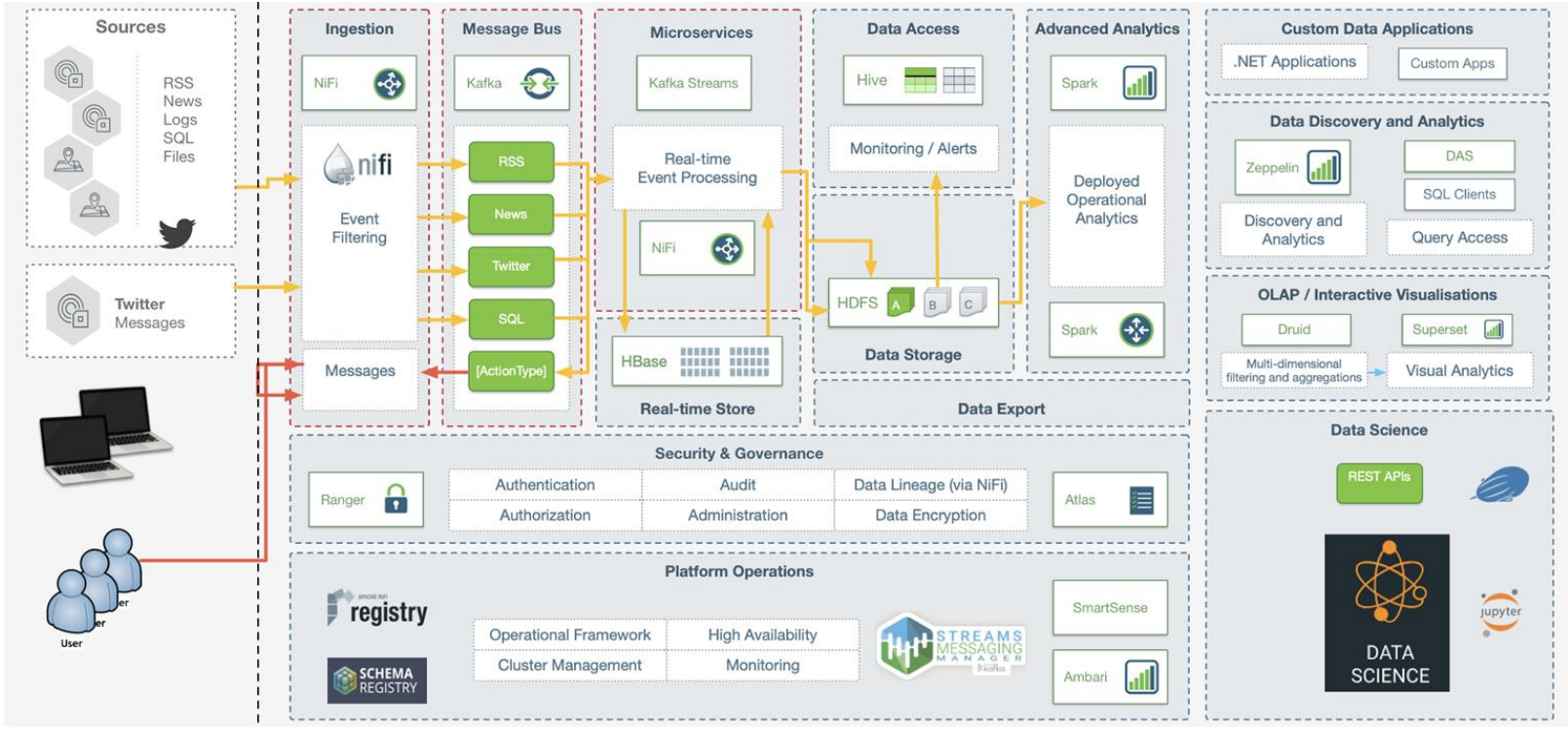




https://community.cloudera.com/t5/Community-Articles/IoT-Edge-Use-Cases-with-Apache-Kafka-and-Apache-NiFi-MiniFi/ta-p/249232

KAFKA STREAMS ADDITIONAL EXAMPLE

Kafka Streams Example Architecture



https://github.com/tspannhw/kstreams

Kafka Streams

```
Timer timer = new Timer();
timer.schedule(new DisplayStatus(), 0, 120000);

final StreamsBuilder builder = new StreamsBuilder();
KStream<String, String> source = builder.stream("bme680");

source.foreach((key, value) -> processValues(key, value));
source.to("bme680out");

final Topology topology = builder.build();
final KafkaStreams streams = new KafkaStreams(topology, props);
final CountDownLatch latch = new CountDownLatch(1);
```

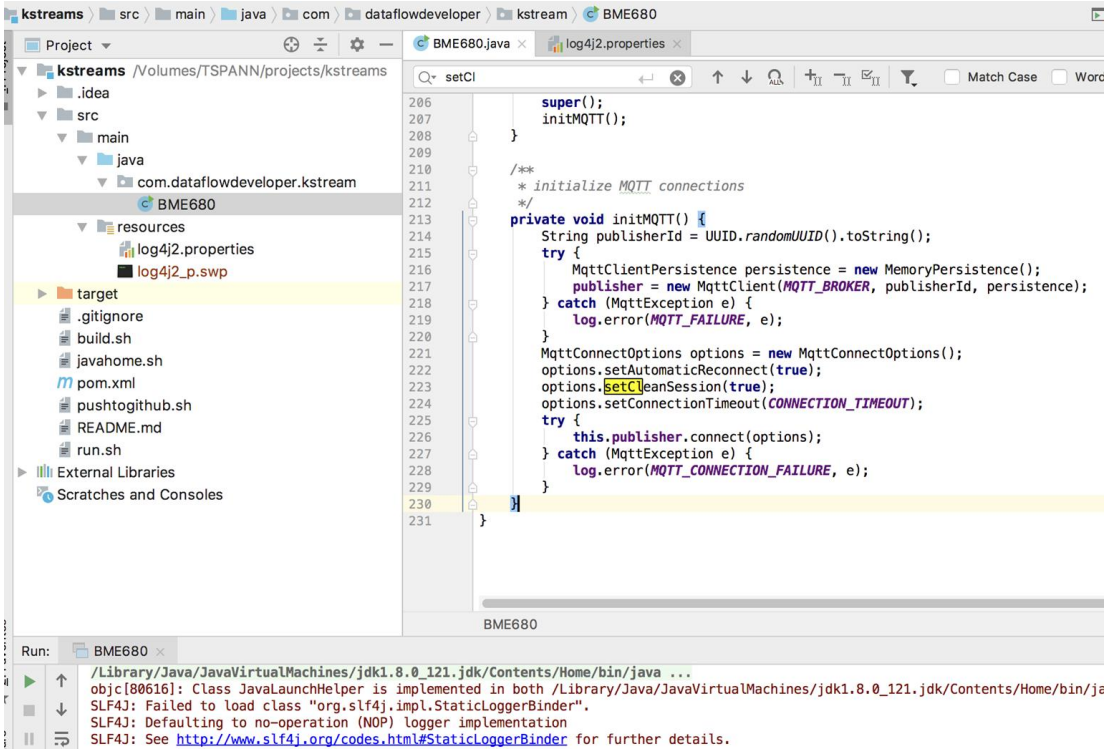
https://www.datainmotion.dev/2019/03/iot-series-sensors-utilizing-breakout_28.html

Kafka Streams

```
Runtime.getRuntime().addShutdownHook(new Thread(STREAMS_SHUTDOWN_HOOK) {
        @Override
        public void run() {
            log.error(SHUTDOWN);
            streams.close();
            latch.countDown();
        }
    });

    try {
        streams.start();
        latch.await();
    } catch (Throwable e) {
        System.exit(1);
    }
```

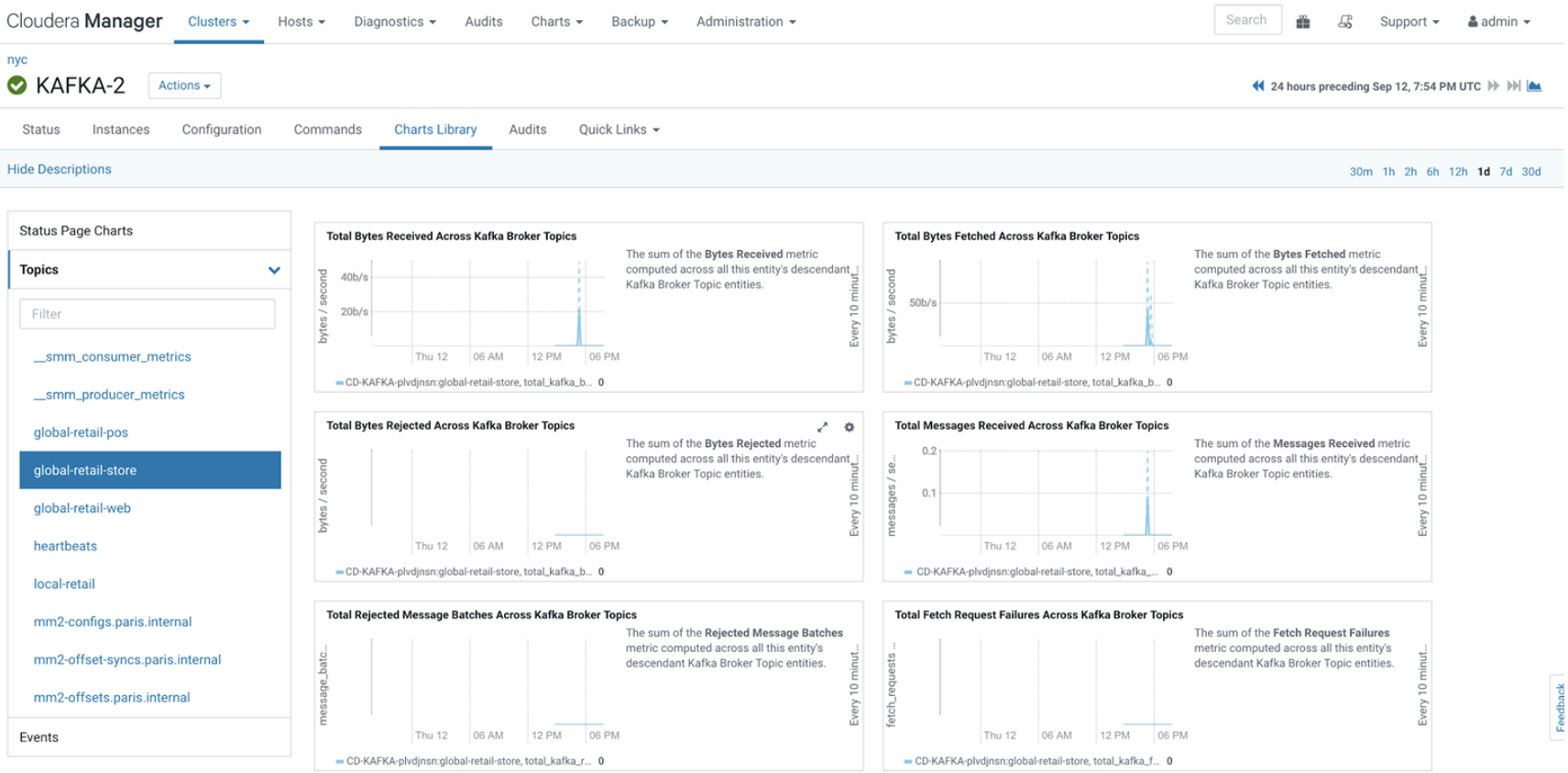
https://github.com/tspannhw/kstreams

Kafka Streams Example 2



https://github.com/tspannhw/kstreams

Cloudera Managed Kafka



CLOUDERA

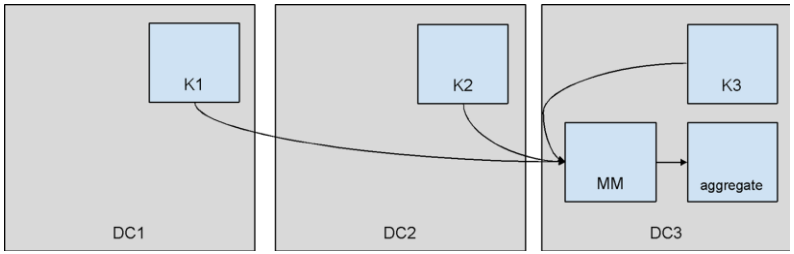
Kafka Replication

Kafka Replication Use Cases

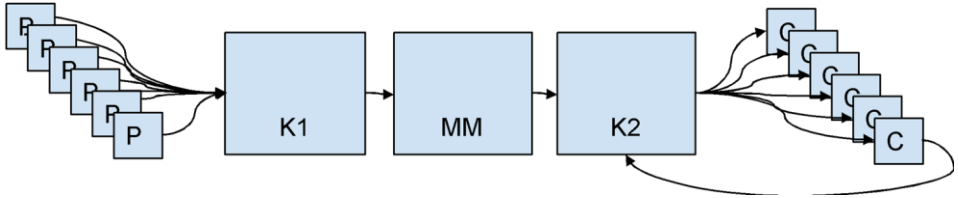
Disaster Recovery	Geo-Locality	Data Movement / Deployment
In an event of a partial or complete datacenter disaster, providing failover/failback to a secondary cluster in a different region / DC	Active-active geo-localized deployments allows users to access a near-by data center to optimize their architecture for low latency and high performance.	Use Kafka to synchronize data between on-prem applications and cloud deployments
Centralized Analytics	**Workload Isolation**	**Legal / Compliance**
Aggregate data from multiple Kafka clusters into one location for organization-wide analytics	Creation of different envs for SDLC: Dev, Test, Prod. Clusters for specific use case cases (ETL, ingestion, analytics, etc)	Since different regions have different data storage and security requirements, clusters need to be created in region but data still needs to be shared.

Various Replication Deployments Based on Use Cases

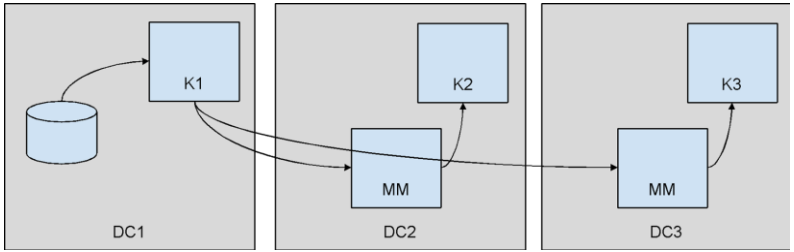
Aggregation



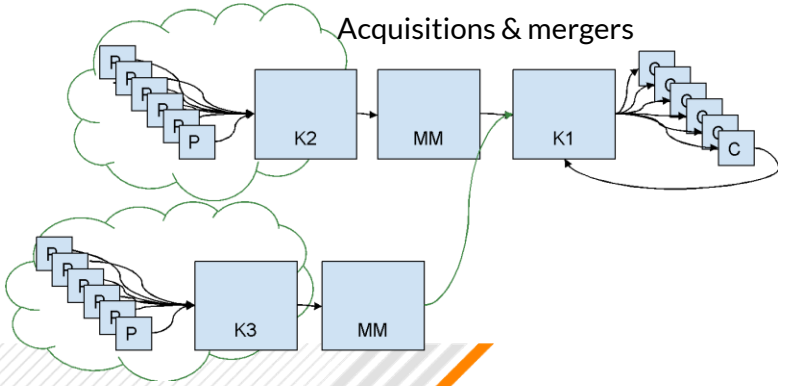
Segmentation



Data Deployment



Acquisitions & mergers



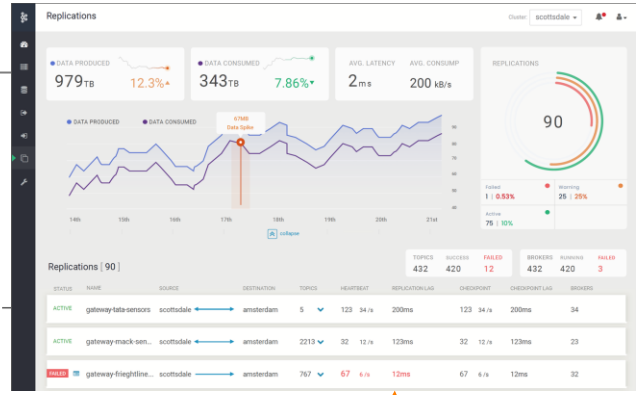
Legacy MirrorMaker

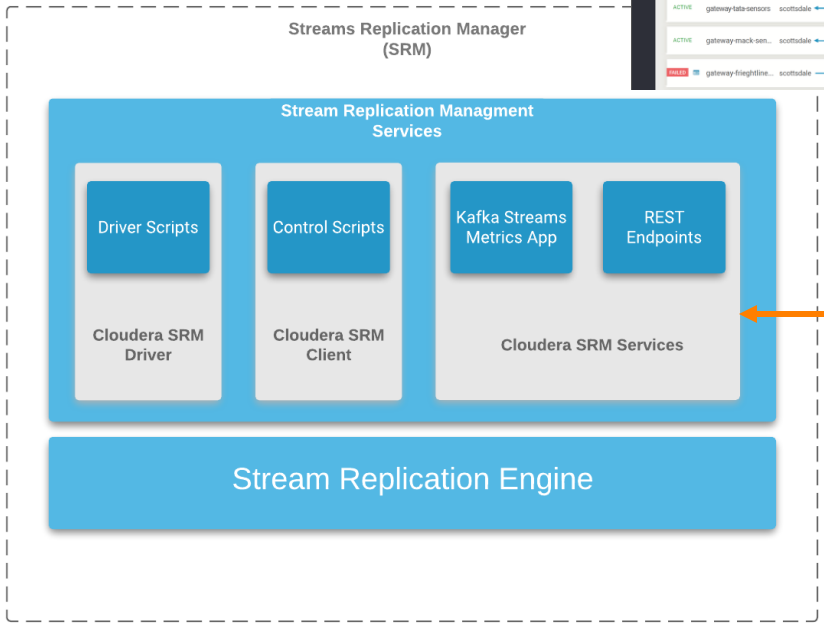
Limitations with Legacy MirrorMaker(MM) for Replication Use Cases

Challenge / Limitation	Description
Static Whitelists and Blacklists	Changes to replication jobs require restart of MirrorMaker cluster.
No Configuration/Metadata Synch	Topic configuration changes in the origin cluster are not detected and propagated to the destination cluster. New Topics are not detected
Scalability and Throughput Limitations due to Rebalances	Cannot scale replication processes as Kafka traffic increases
Lack of Monitoring and Operational Support	No tooling to to install and manage replication cluster. Difficult to monitor replication lag, consumer and producer metrics for replication workflows
No Disaster Recovery, Migration, Failover	MM doesn't support active/active without capabilities such as topic renaming, etc. No support for client failover/failover out of the box.

Streams Replication Manager (SRM)

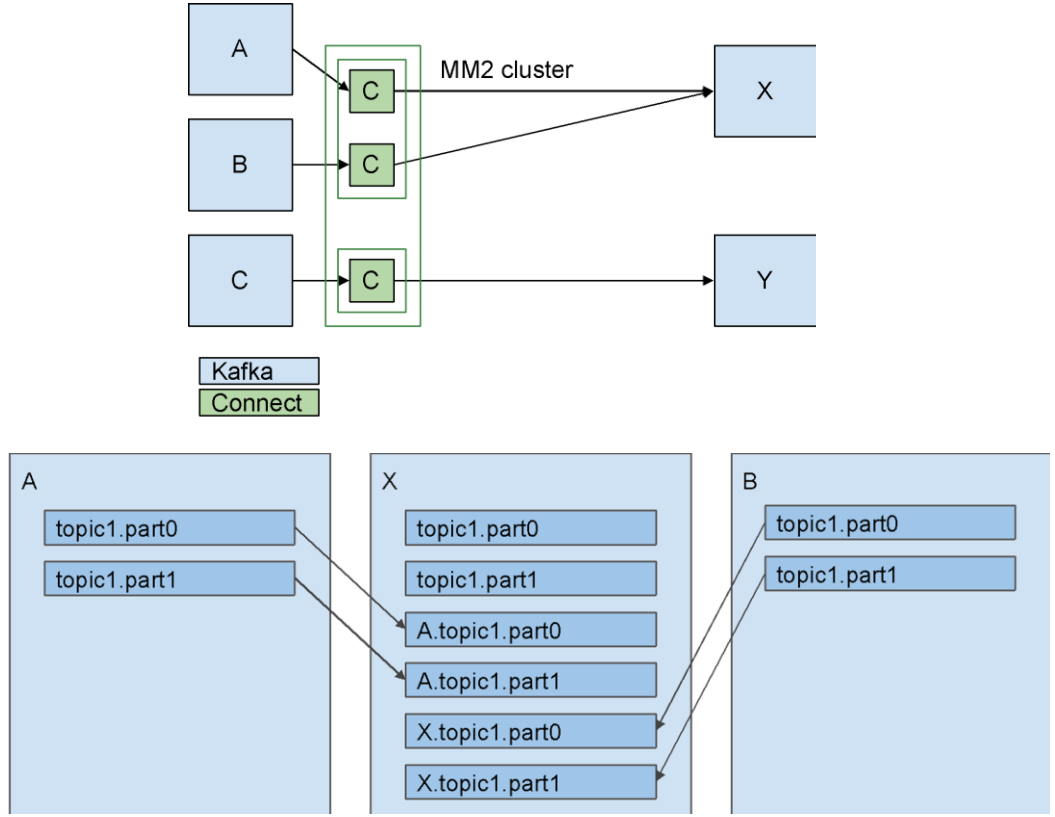
- Easy installation & automation
- Lifecycle management
- Management and monitoring of replication flows across clusters with Streams Messaging Manager
- Smart client for failover and failback





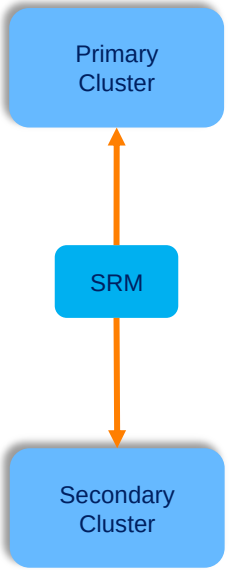
SRM Engine

- Supports active-active, multi-cluster, cross DC replication & other complex scenarios
- Leverage Kafka Connect for scalability and HA
- Replicate data and configurations (ACL, partitioning, new topics, etc)
- Offset translation



Remote topics

- Replicated topics are renamed according to `ReplicationPolicy`.
- `Default policy : <source>.<topic>`
- `Can implement custom policies`




```
topic1
topic2
secondary.topic1
secondary.topic2
```

```
topic1
topic2
primary.topic1
primary.topic2
```

4 Key Concepts in MM2

Remote Topics

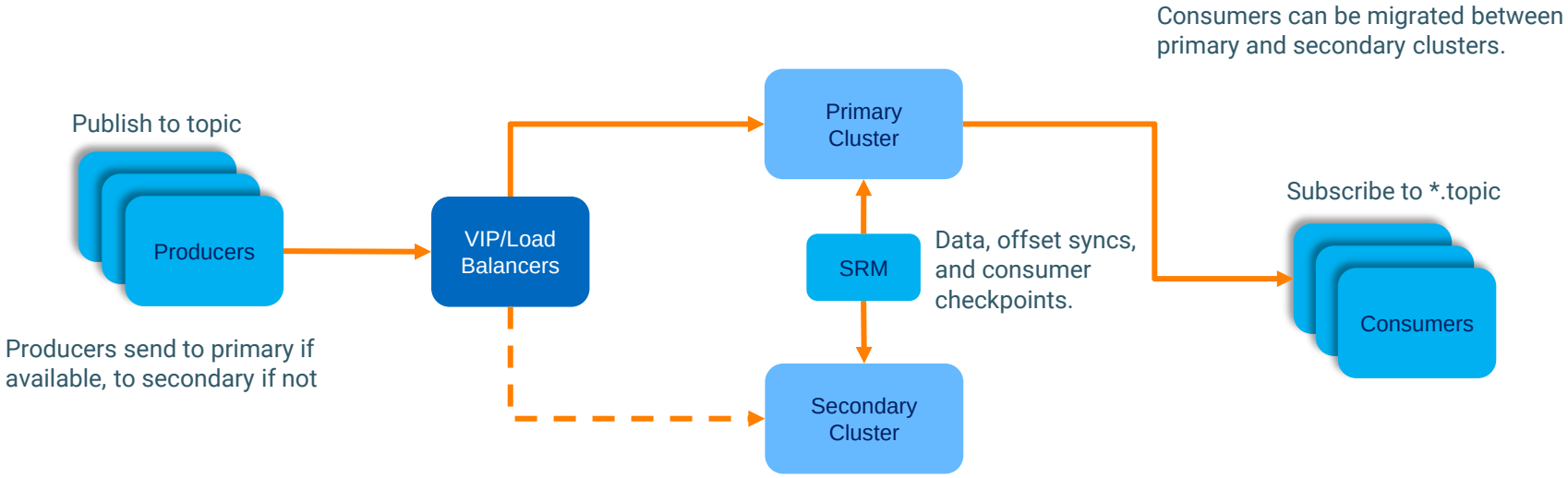
HeartBeats

Offset Syncs

Checkpoints

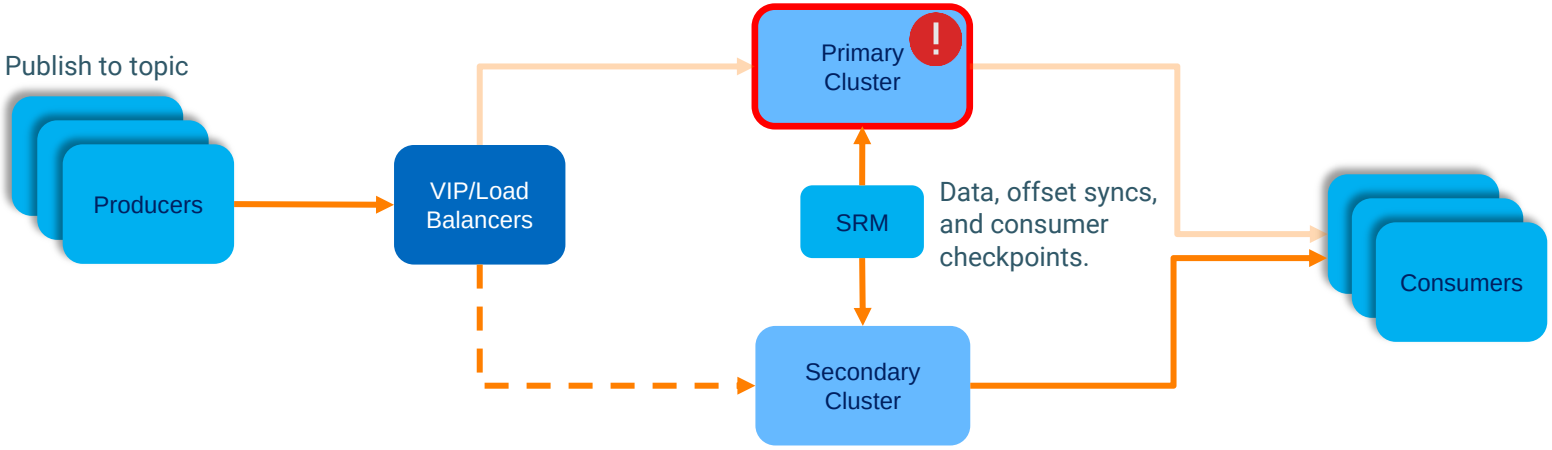
Active Passive

Active Passive



Active Passive

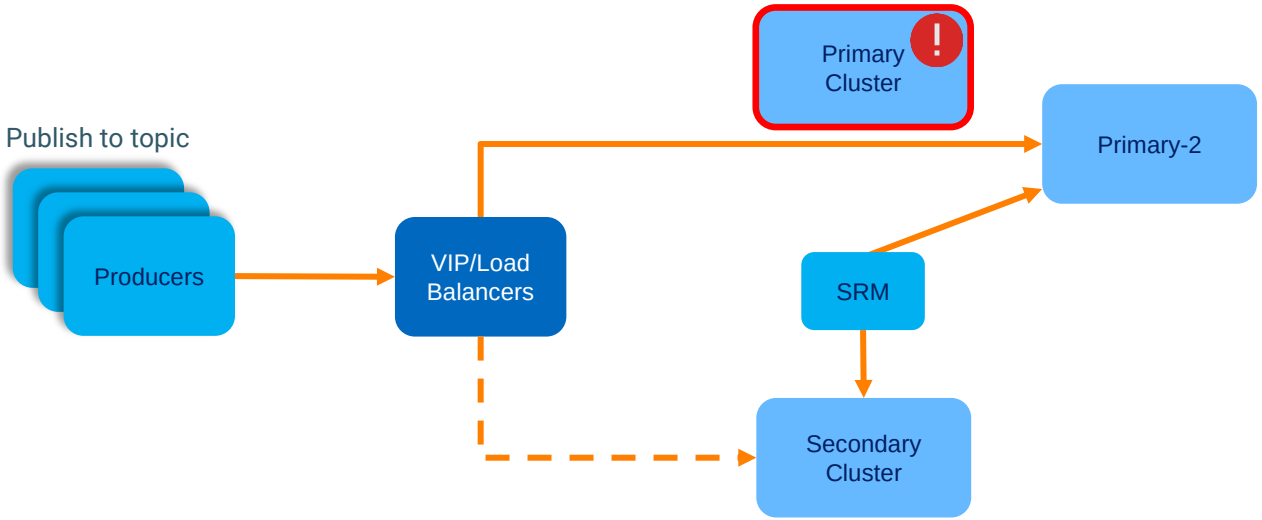
Primary down: failover and migrate consumers



$ **srm-control offsets** --bootstrap-server :9092 **--source primary** --group foo --export > out.csv
$ **kafka-consumer-groups** --bootstrap-server B_host:9092 **--reset-offsets** --group foo --execute **--from-file out.csv**

Active Passive

Primary permanently lost? Recover from secondary.



topic1
topic2
secondary.topic1
secondary.topic2
secondary.primary.topic1
secondary.primary.topic2

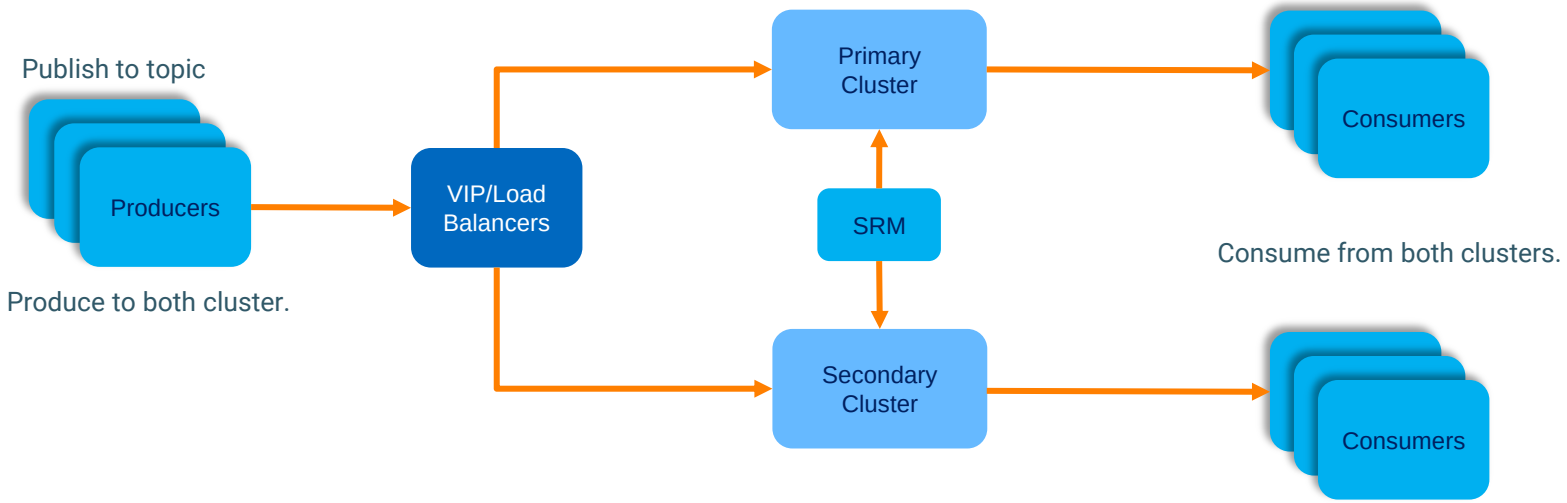
Data from old primary

topic1
topic2
primary.topic1
primary.topic2
primary-2.topic1
primary-2.topic2

Active Active

Active Active: Cross DC Consumer Groups or Cross DC Replication

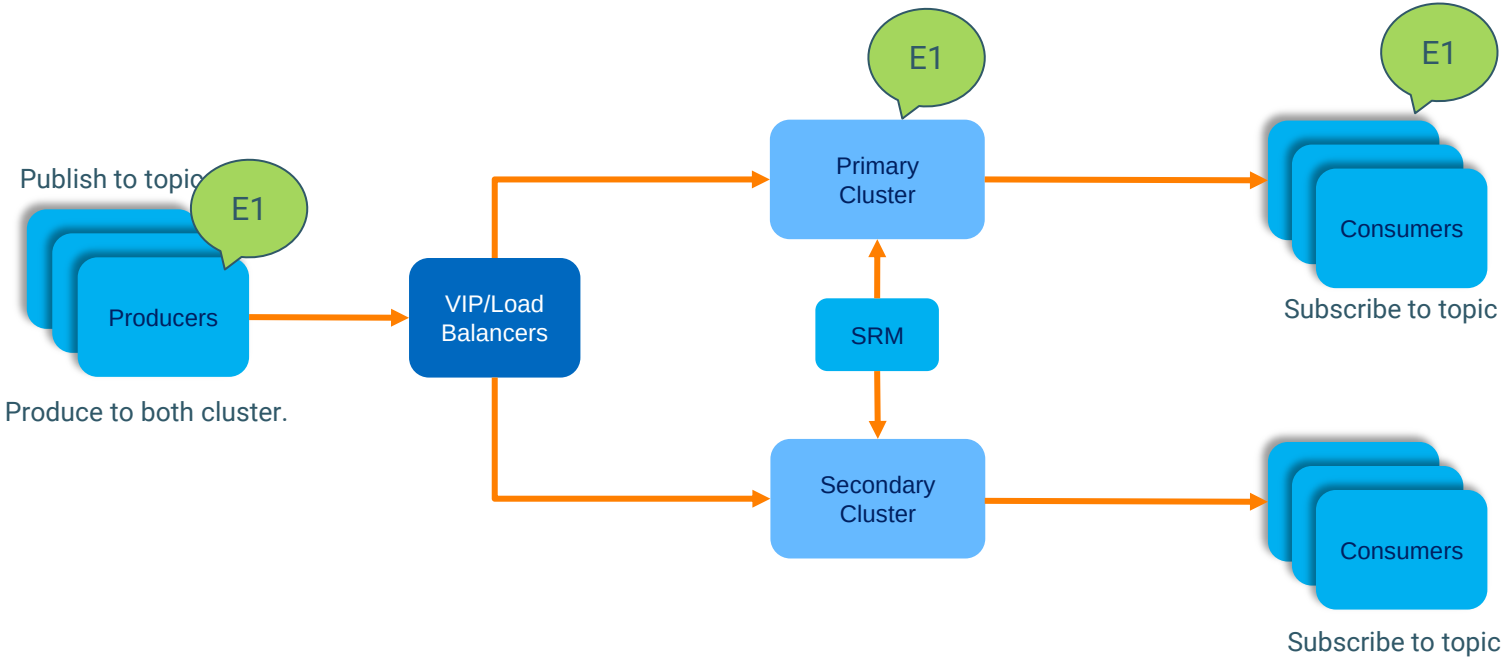
Consumer subscription defines the patterns



A/ Cross DC Consumer Groups

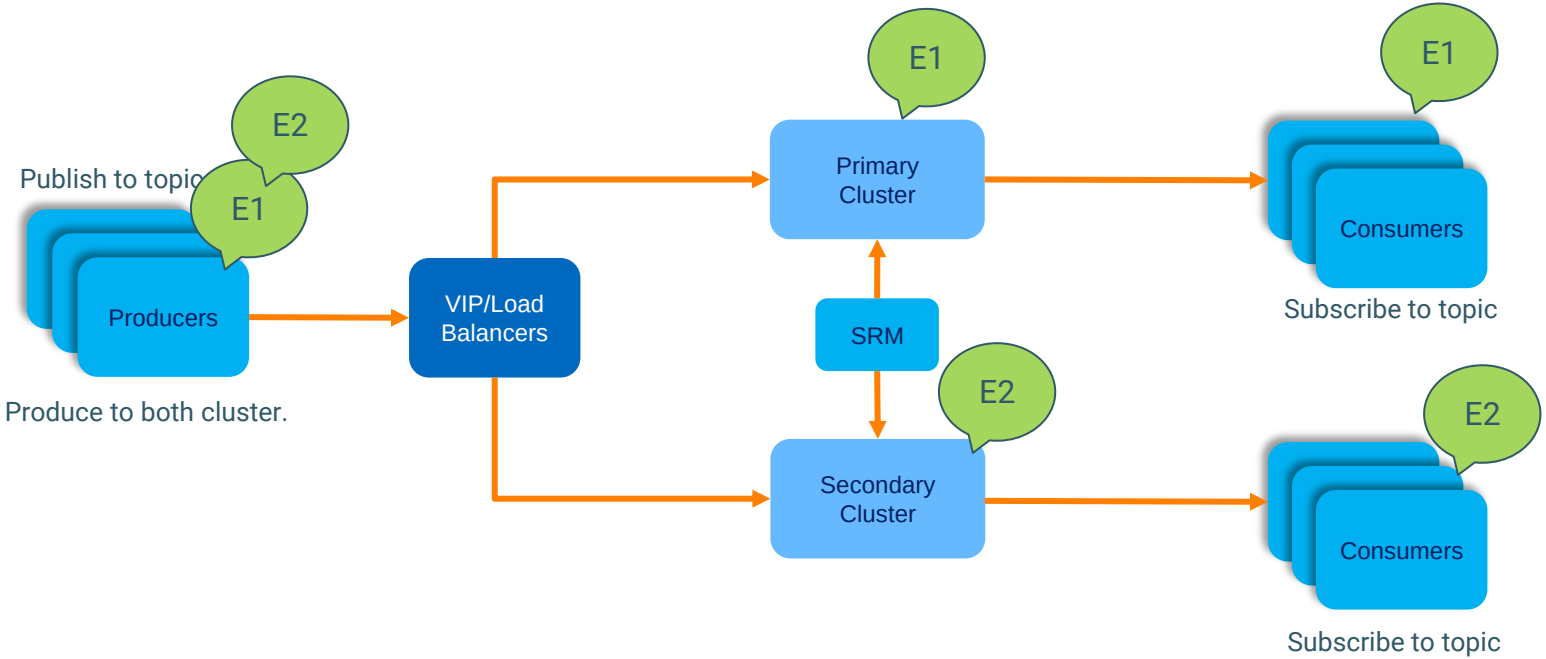
Cross DC consumer groups

Only one consumer processes each record



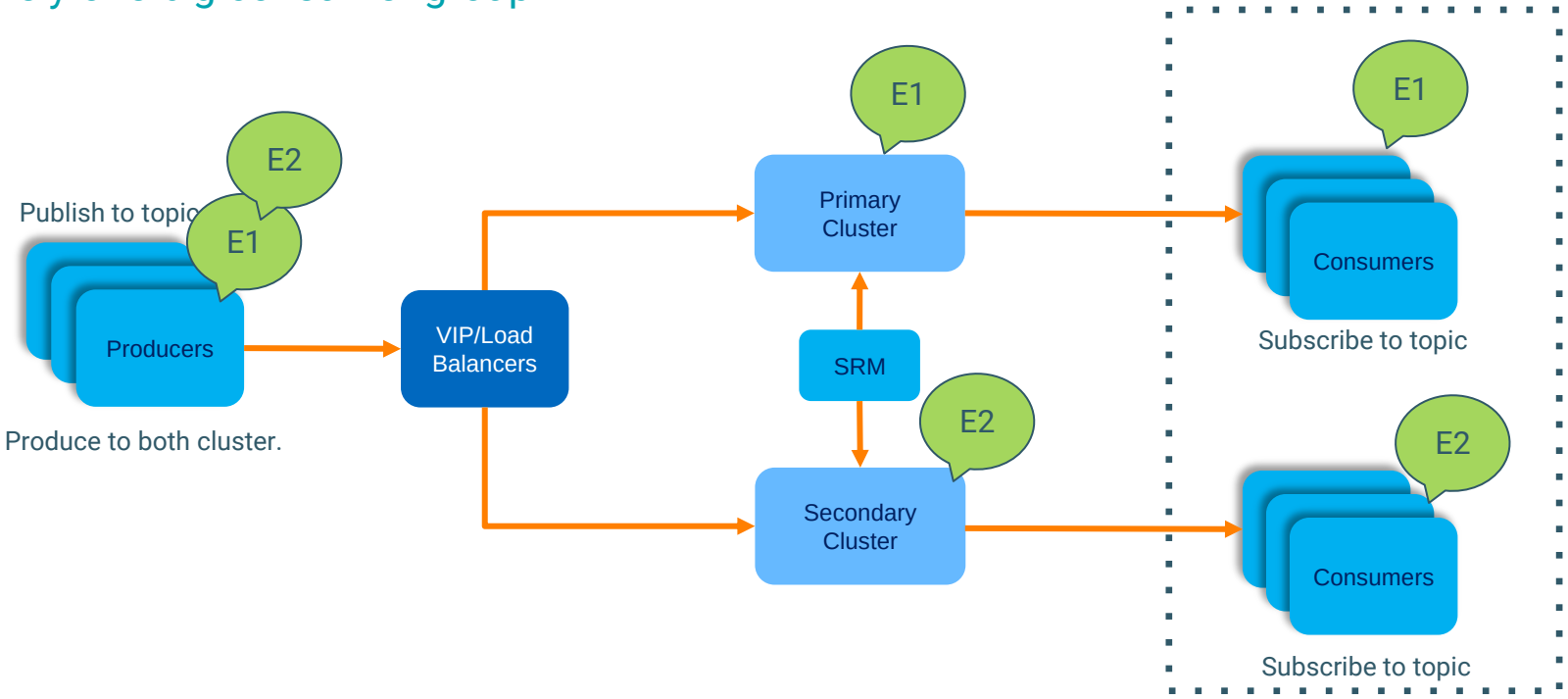
Cross DC consumer groups

Only one consumer processes each record



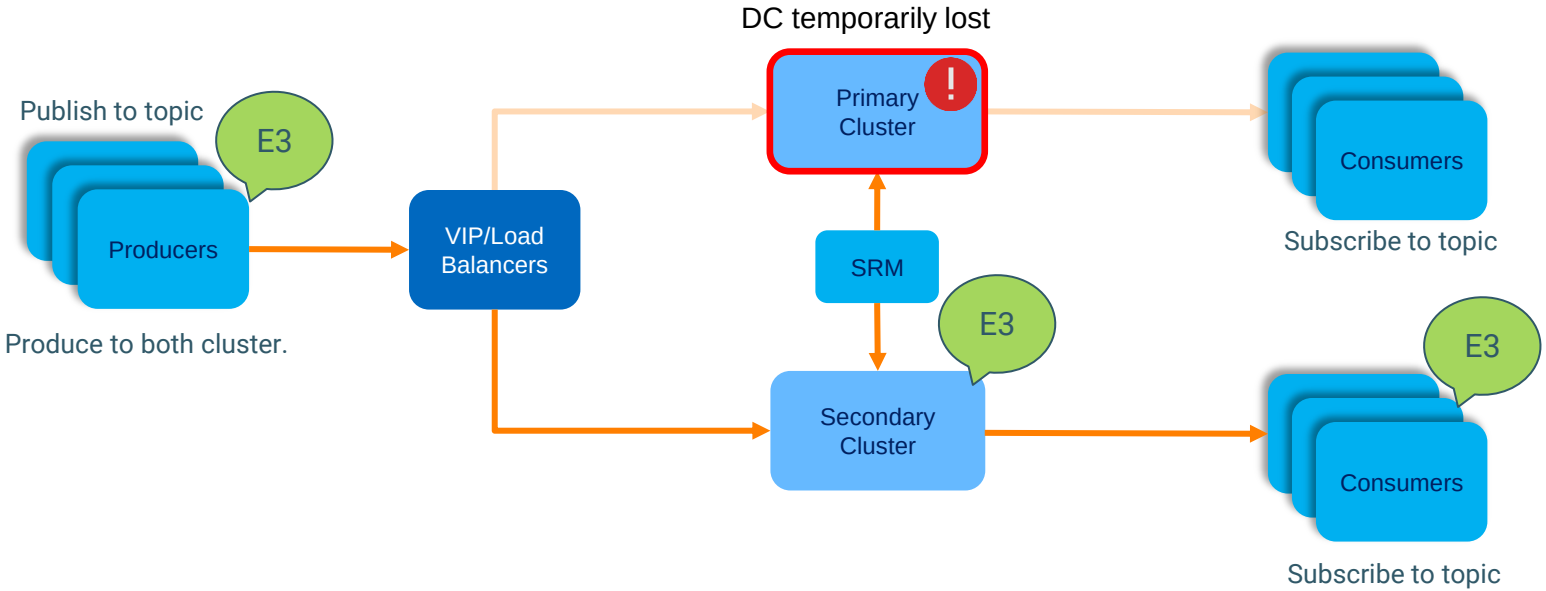
Cross DC consumer groups

Effectively one big consumer group



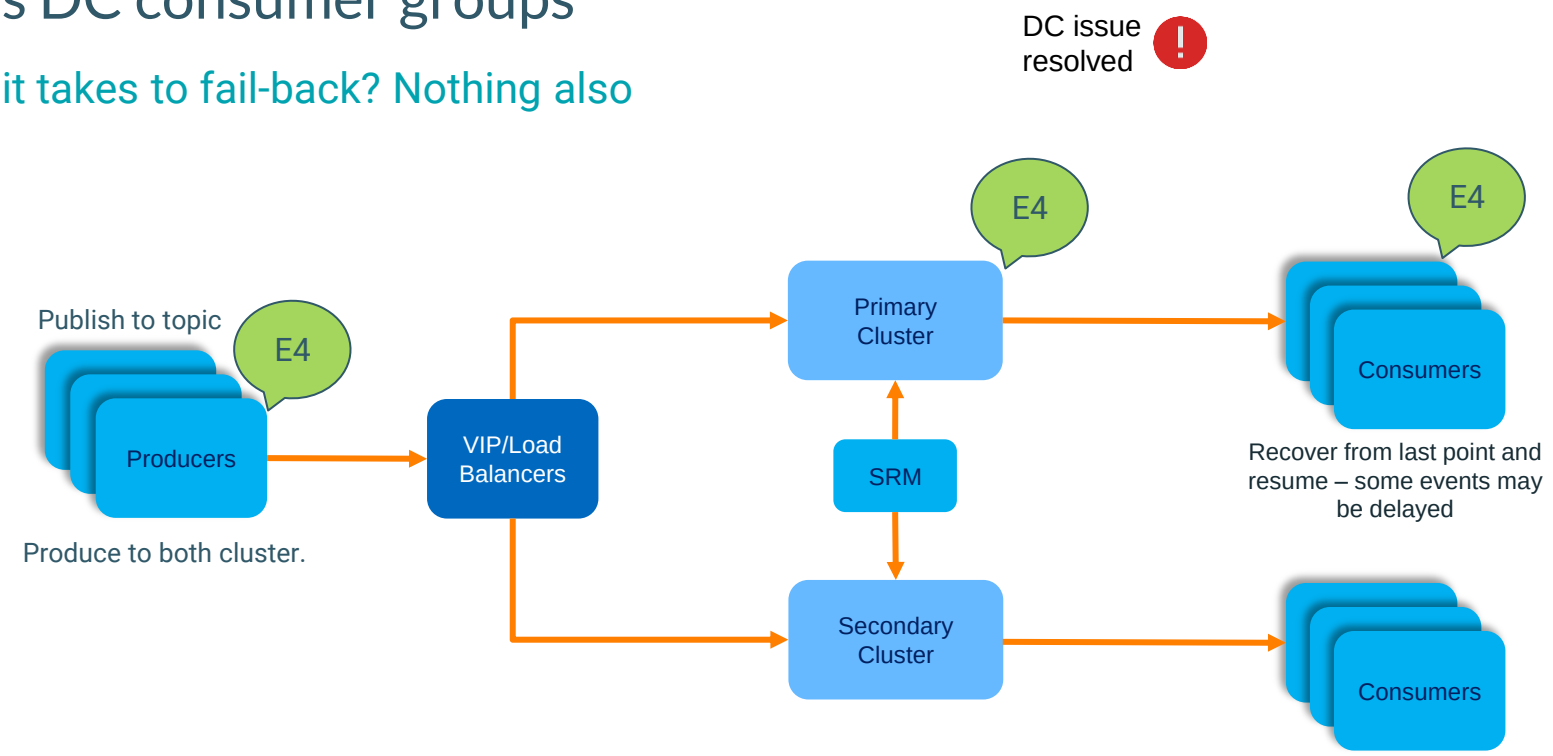
Cross DC consumer groups

What it takes to fail-over? Nothing



Cross DC consumer groups

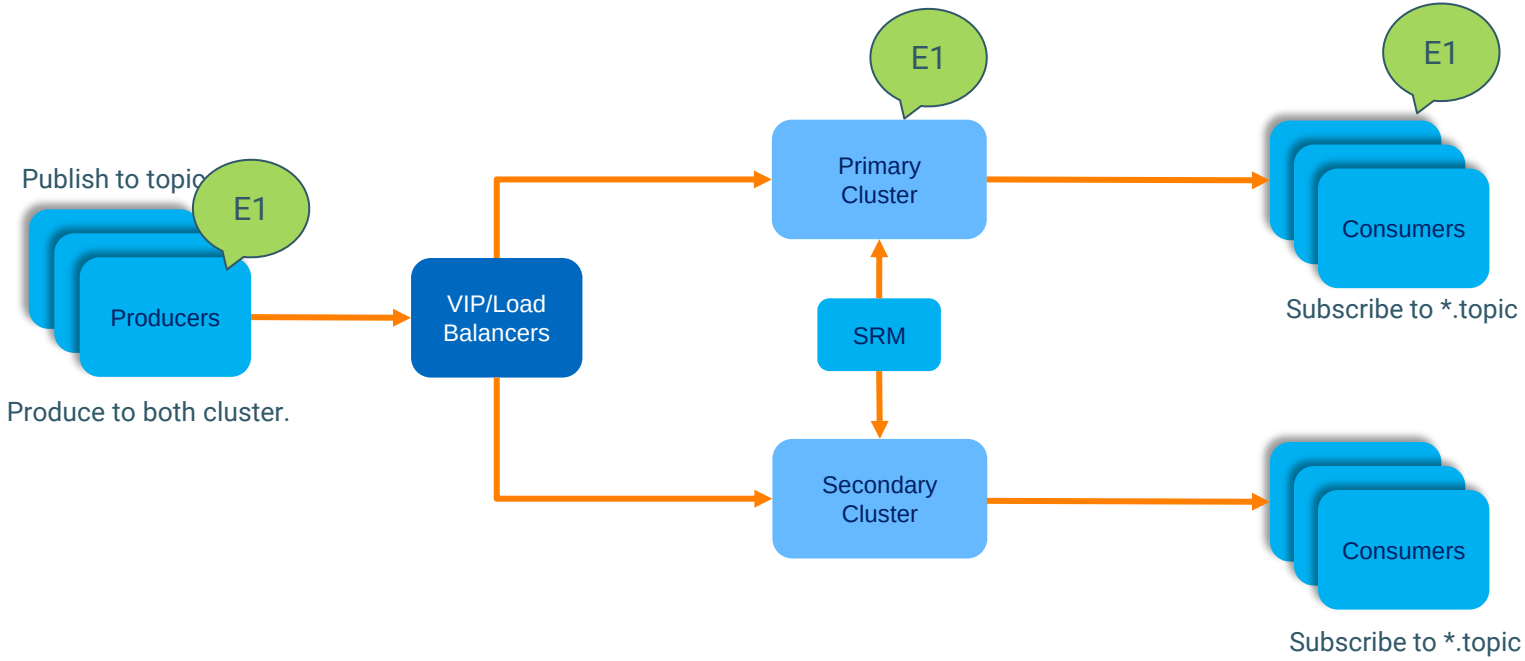
What it takes to fail-back? Nothing also



B/ Cross DC Replication

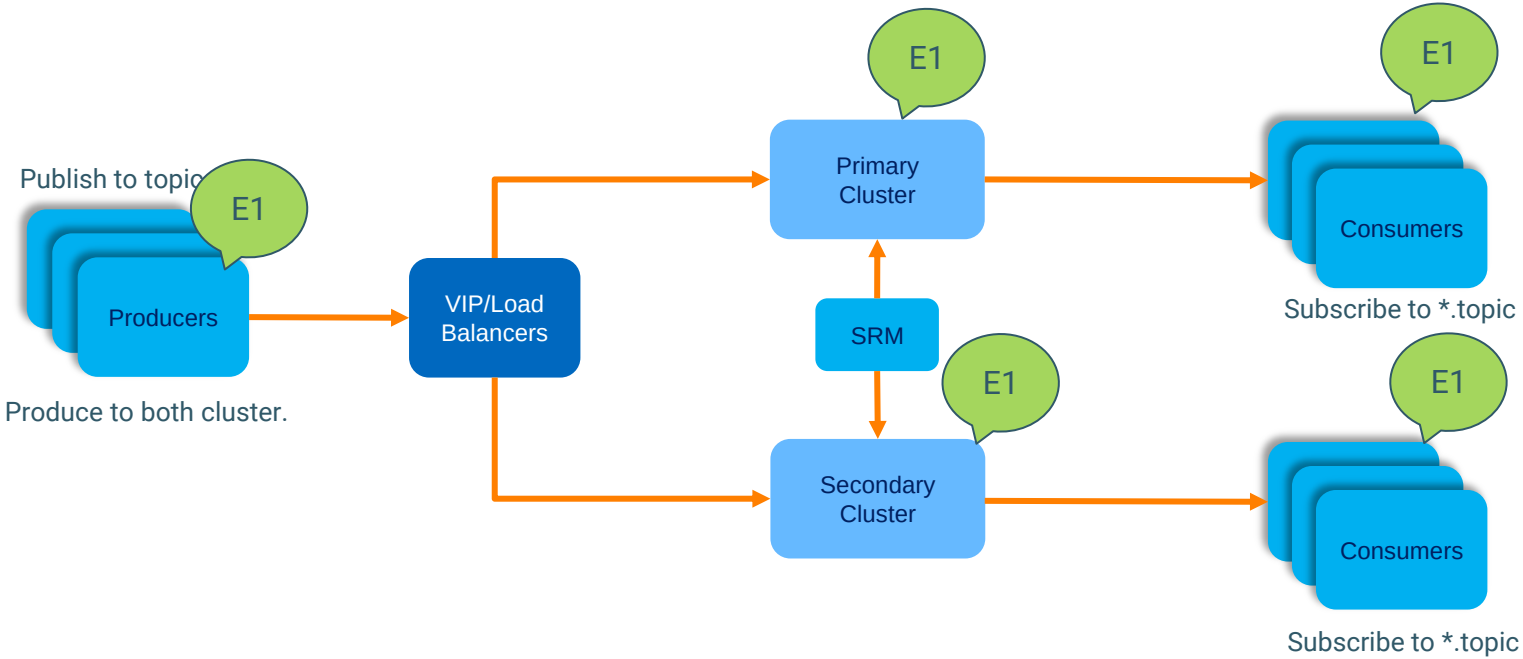
Cross Data Center Replication XDCR

All consumers process all records



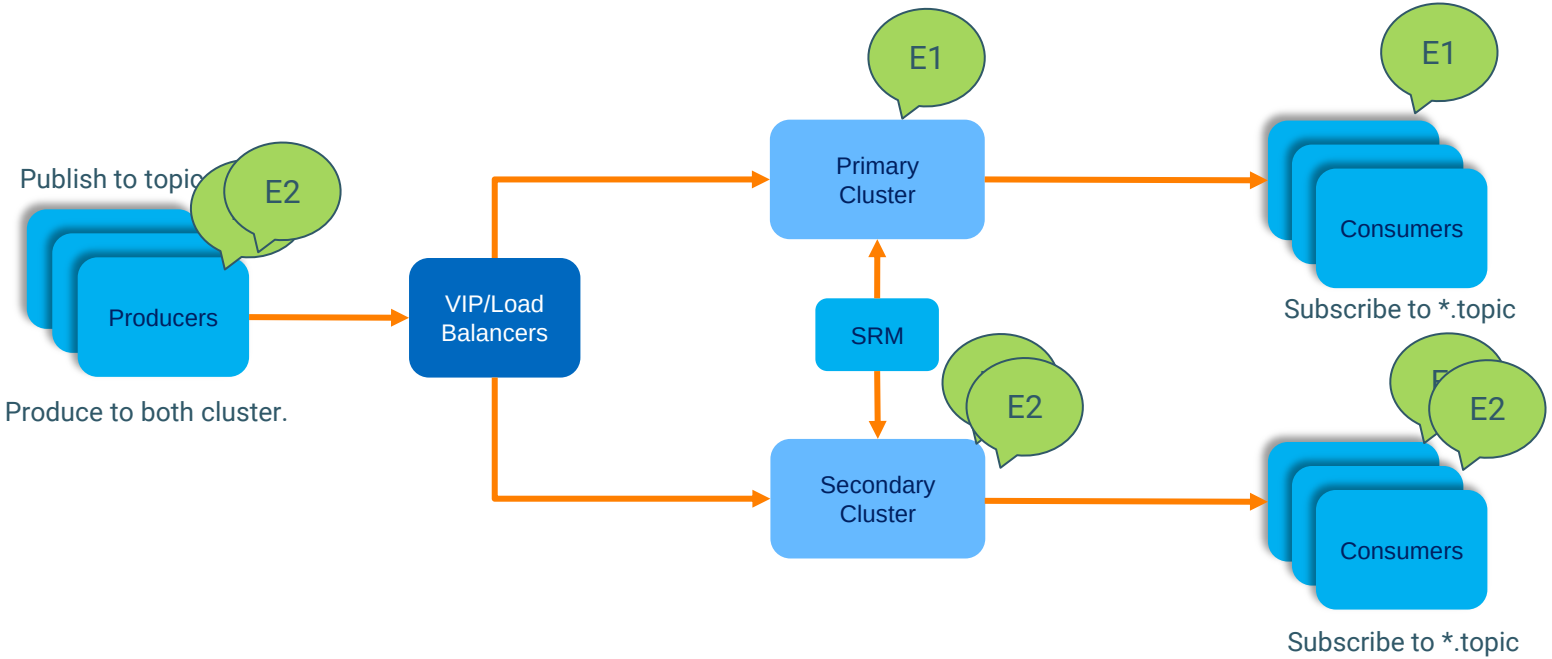
Cross Data Center Replication XDCR

All consumers process all records



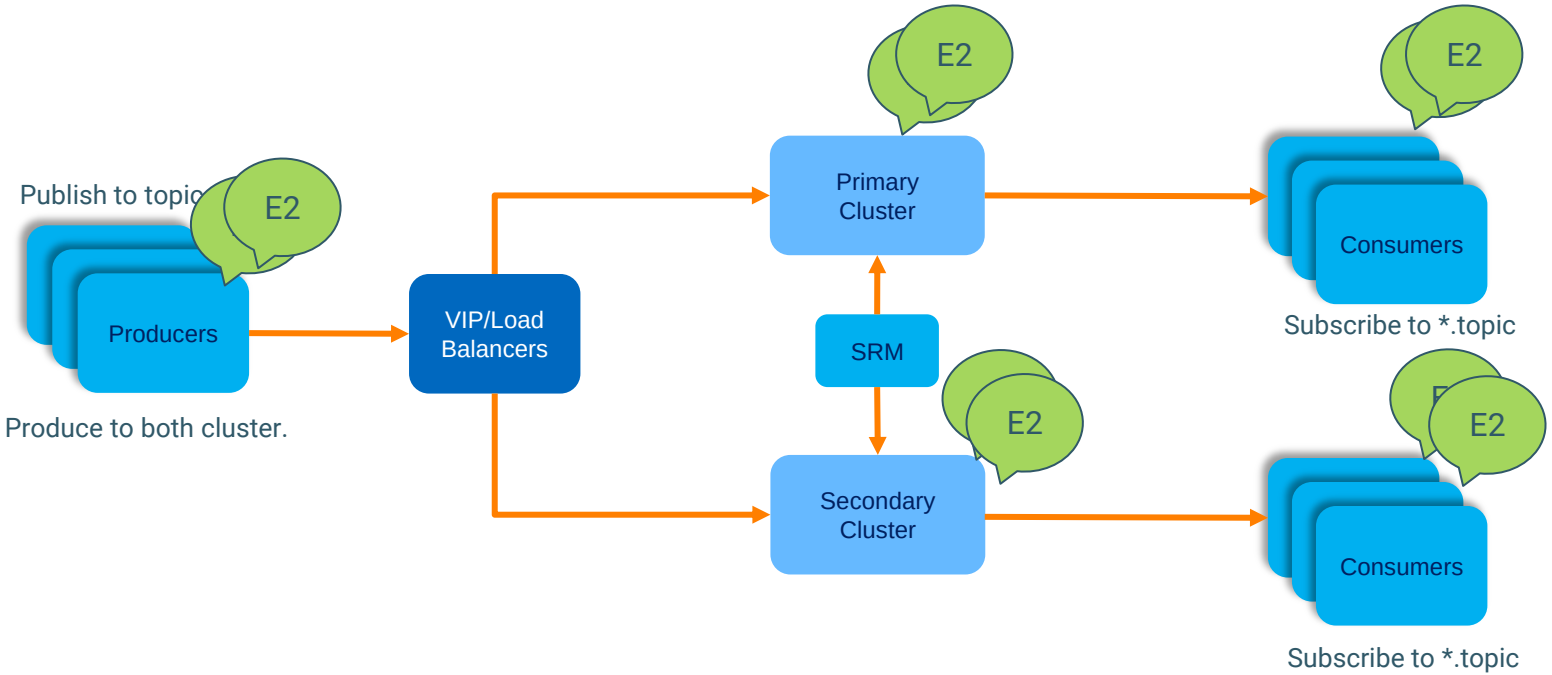
Cross Data Center Replication XDCR

All consumers process all records



Cross Data Center Replication XDCR

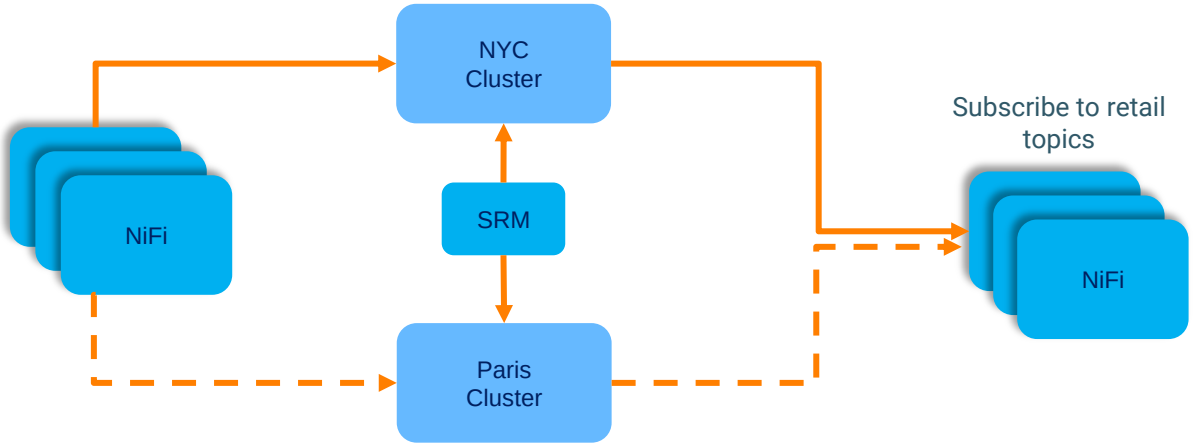
All consumers process all records



Demo DR with SRM

Use cases and scenarios

- WW retailer with critical use cases on Kafka
- Two clusters : NYC & Paris
- NiFi publish/consume events to/from Kafka
- global-retail-* topics are critical and should be replicated
- Local-retail-* are not replicated



STREAMING CORE CONCEPTS

TODAY IF A BYTE OF DATA WAS 1 GALLON OF WATER WE COULD FILL AN AVERAGE HOUSE IN 10 SECONDS, BY 2020 IT WILL TAKE ONLY 2.

< 1% OF THE DATA BEING GENERATED BY THE 30,000 SENSORS ON AN OFFSHORE OIL RIG IS CURRENTLY USED WHEN MAKING DECISIONS ABOUT MAINTENANCE PLANNING

SOURCE: 2017 McKinsey Global Institute analysis

DISTRIBUTED STREAMING PROCESSING ENGINES

Quarks | Gearpump | Concord | Heron | Pulsar | Spark | SQLStream | EsperTech | Google Cloud | S4 | Flink | DataFlow | Storm | Samza | Impetus | Kafka Streams

Apache Apex | Amazon Kinesis | MemSQL | Bottlenose Nerve Center | Striim | Azure Stream Analytics | Informatica | Oracle Stream Analytics |

WS02 Complex Event Processor | IBM InfoSphere Streams | Cisco Connected Streaming Analytics | SAS Event Stream Processing | WS02 Complex Event Processor | TIBCO StreamBase | Software AG Apama Streaming Analytics | MongoDB | Ignite | XAP

Summary

Already using Spark?

Want unified Batch/Stream?

Need highest Throughput?

Don't need low latency?

Don't need advanced Time/State features?

Like Kafka?

Only need Microservices?

Want Low Latency?

Want Just Kafka?

Don't need advanced Time/State features OOTB?

Like Flink?

Want Flexibility?

Want best-in-class Technology?

Want Low Latency?

Kafka Streams and Spark Structured Streaming

Picking the Right Streaming Engine Based on the Use Case / Requirement

Non-Functional Requirement (NFR) / Use Case	Spark Structured Streaming	Kafka Streams
Stream Processing Style	Micro-batching. Event at a time aka continuous mode is still experimental	Event a time
State Management	Implicit Support with HDFS	Implicit support with RocksDB
Fault Tolerance	Implicit Checkpoint support with HDFS	Implicit support with internal Kafka topics / ZK
Advanced feature: Joins/ Aggregations/ Windowing	Stream/Stream Joins, Stream/Table joins, joins using different keys, Windowing, OOO aggregation processors	Stream/Stream joins, Stream/Table joins, joins with only message key, no OOO aggregation processors
Advanced feature: watermarking	Supported	Not fully supported
Latency	Medium	Low
Throughput	High	Medium
APIS	Simple APIs (Declarative)	Simple APIs (Compositional)
SQL DSL	First Class feature within Spark	Nothing in Apache Kafka.
Lambda Architecture	First Class support. Write app and execute in streaming or batch mode	No Support
GUI Tooling / Code-Less Approach	Not Supported	Not Supported
Maturity	Maturing and Growing Community	Relatively New
Use Cases	complex stream processing use cases, ETL	Lightweight eventing based microservices, ETL
Cluster Requirement	Yes	No

Comparison: Technical Features

	Flink 1.9	Kafka Streams 2.3	Spark Structured Streaming 2.3	Storm 2.0 and Storm Trident
Approach / Position	Streaming First, modern class-leader	Message-Broker First, popular Streaming Add-on	Batch First, popular Streaming Add-on	Streaming First, legacy architecture
Streaming Model	1st class Stateful, (<500ms), Event-at-a-time	Stateful, (<500ms), Event-at-a-time	Stateful, (>1s), Microbatch	Natively stateless, (<1s), Event-at-a-time, Stateful plugins
Time Support	Event/Processing/Custom	Event/Processing	Event/Processing	Processing OOTB, Event developable
Processing Abstractions	Table/CEP/Graph/ML/SQL. Batch experimental	Table/SQL(KSQL). No Batch.	Table/ML/Graph. Unified APIs for Batch and Stream	Streaming only OOTB
SQL Abstraction	based on Apache Calcite, ANSI compliant	SQL-like DSL (KSQL), not ANSI compliant	ANSI SQL:2003 in Spark 2.3	Experimental since 1.2.3
Delivery Guarantee	Exactly Once Upstream, some Downstream	Exactly once, Kafka-only	Exactly Once Upstream, some Downstream	At least once (Trident exactly once Upstream)
State Mgmnt	RocksDB, Configurable Snapshots, Queryable	RocksDB, Snapshot to Kafka Topic, Queryable	OSS Async on HDFS. RocksDB Databricks Only	External, not OOTB
Fault Tolerance / Resilience	Built-in, Checkpoints, Savepoints, Redistributable	BYO Microservice	Built-in	Built-in

Comparison: Operational Features

	Flink 1.9	Kafka Streams 2.3	Spark Structured Streaming 2.3	Storm 2.0 and Storm Trident
Deployment Model	Clustered, Kubernetes, Microservices	Microservices, Kubernetes	Clustered, Kubernetes	Clustered
Documentation	Good Technical Docs, growing examples and SO	Extensive Docs and Examples and SO coverage	Extensive Docs and Examples and SO coverage	Good for 1.x
Maturity / Community	Smaller but Fastest growing community with strong research and Production deployments	Newest, strong community with strong growth	Spark community is strong, but Streaming is a small quiet corner	Oldest framework, community eclipsed by newer Engines
Use Cases	Everything	Microservice / Event driven, Embedded in another Application	Unified ETL, semi-RT processing	IOT, complex event processing
Enterprise Management	Rich OSS, enhanced vendor offerings	Minimal OSS, Some via vendor offerings	Rich OSS, enhanced vendor offerings	Some integrations
Push Button Security	Complex, some OSS support, limited vendor offerings	Simple, some OSS support, good vendor offerings	Complex, good OSS support, good vendor offerings	Complex, good OSS support, good vendor offerings
Logging / Metrics	Usual OSS integrations, some vendor offerings	BYO Microservice	Good Logging integration	Good Logging integration
Scaling Up/Down	Not yet Autoscaling, but all requirements available	BYO Microservice	Not yet Autoscaling, but all requirements available	Management Tools help, but Tuning is challenging

Streaming Analytics on CDF

- Simple operation: split, join, filtering, enrichment, etc
- Time window operators
- Complex Event Processing
- Pattern matching
- Predictive and Prescriptive Analytics

Publish subscribe high-throughput, low-latency broker

API for building real-time applications and microservices on top of Kafka

Only for HDP/CDH customers

Leverage Spark micro-batching capabilities for HDP/CDH customers

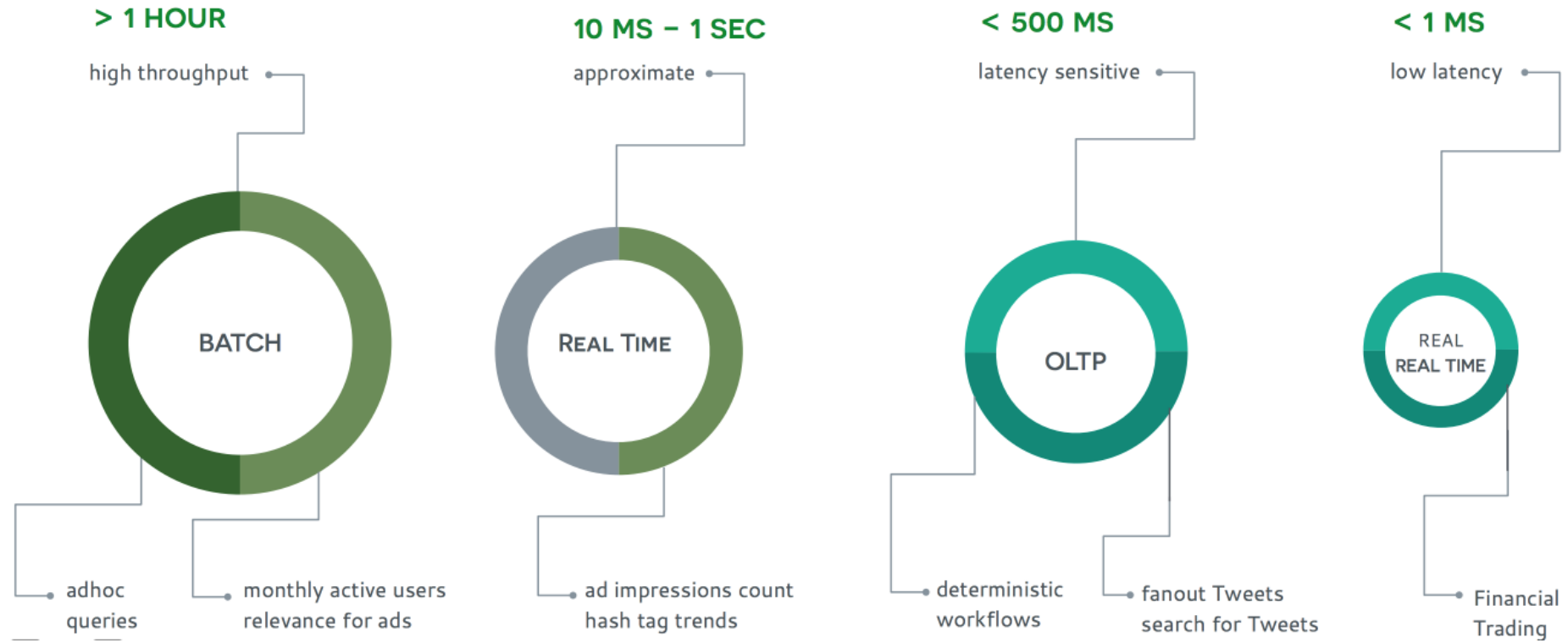
Coming Soon

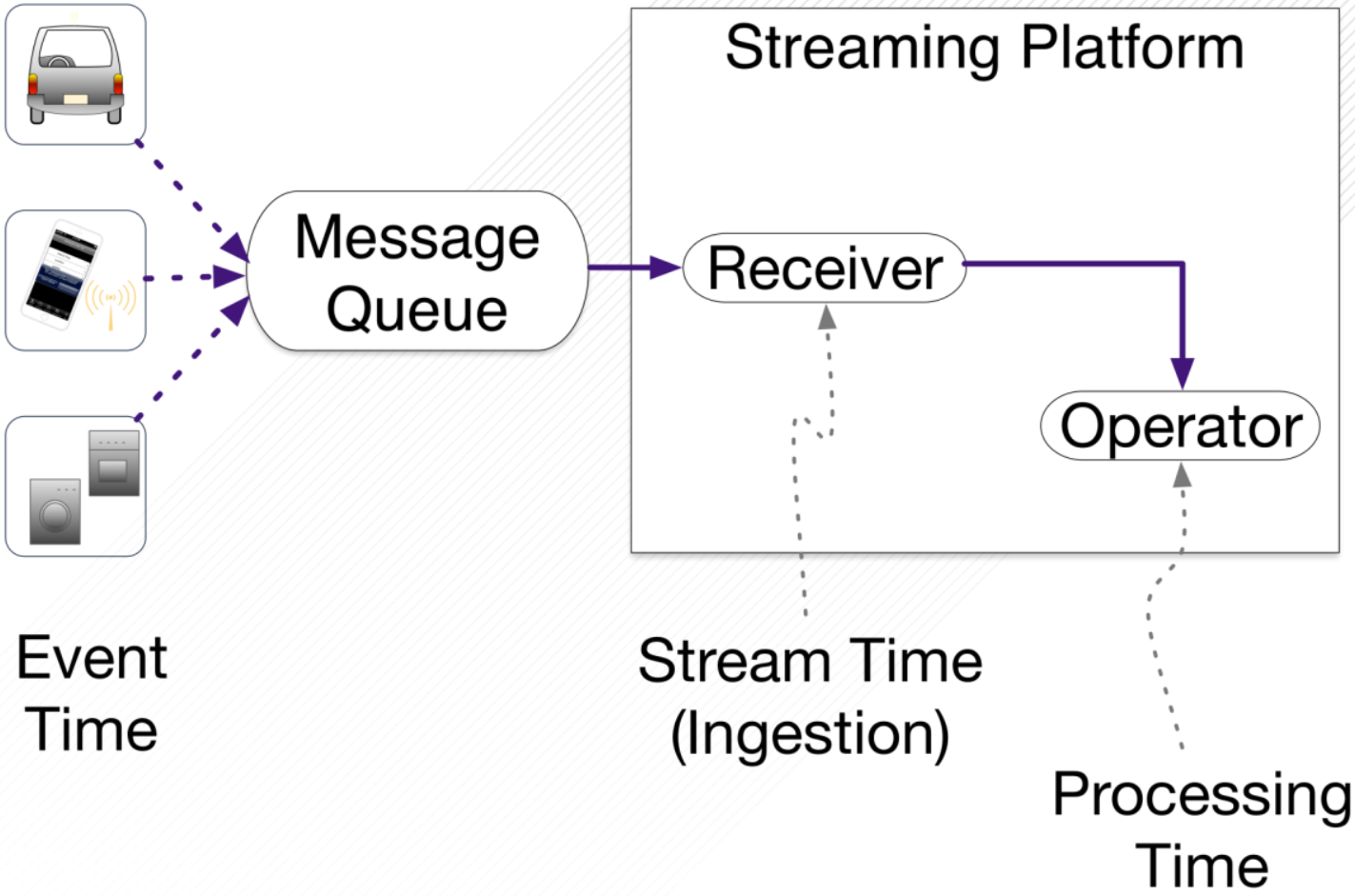
MOST SIGNIFICANT REQUIREMENTS

1. Process an instance at a time, and inspect it (at most) once
2. Use a limited amount of time to process each instance
3. Use a limited amount of memory
4. Be ready to give an answer (prediction, clustering, pattern) at any time
5. Adapt to temporal changes

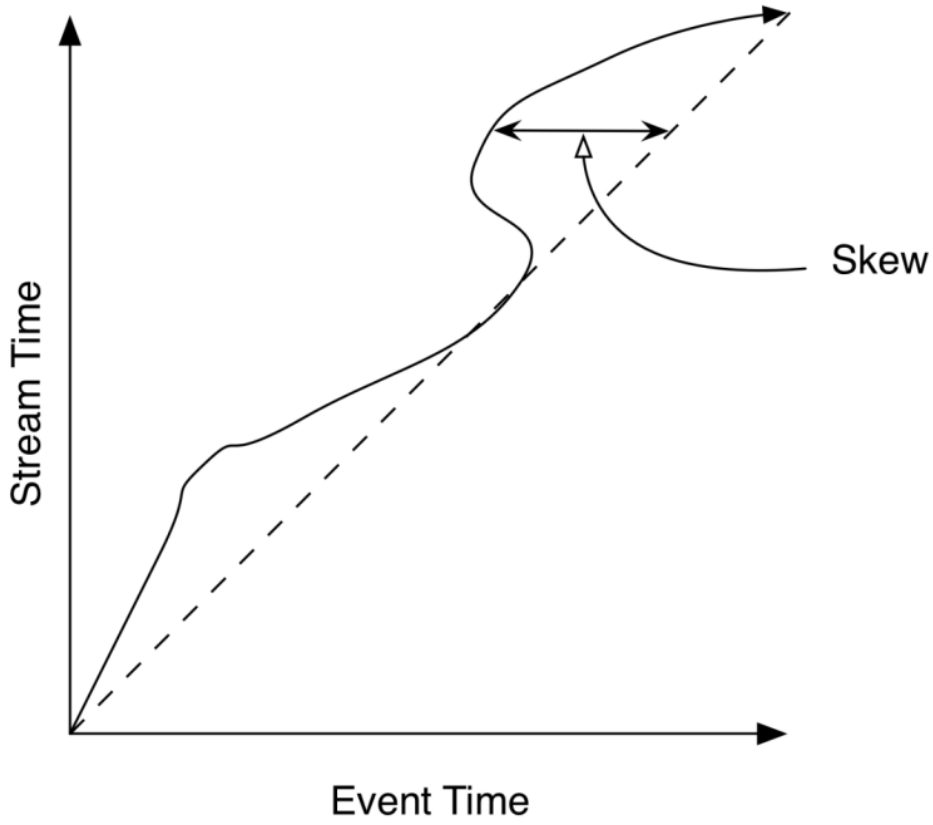
THINKING ABOUT… TIME

WHAT IS REAL-TIME?





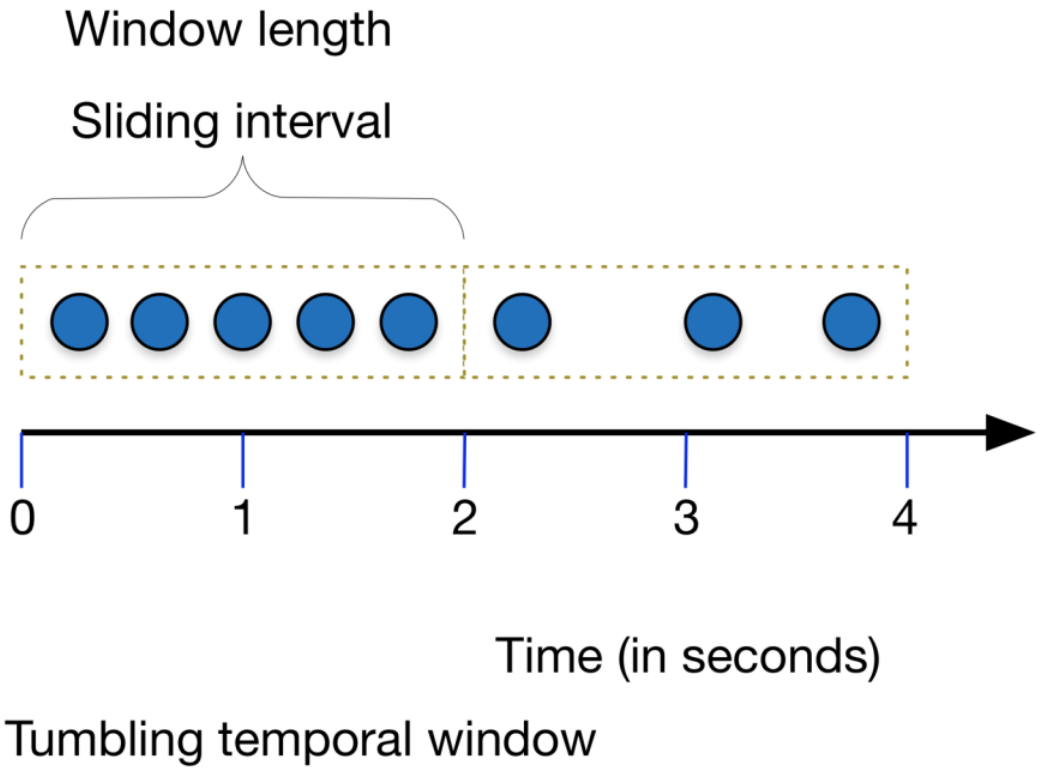
TIME SKEW



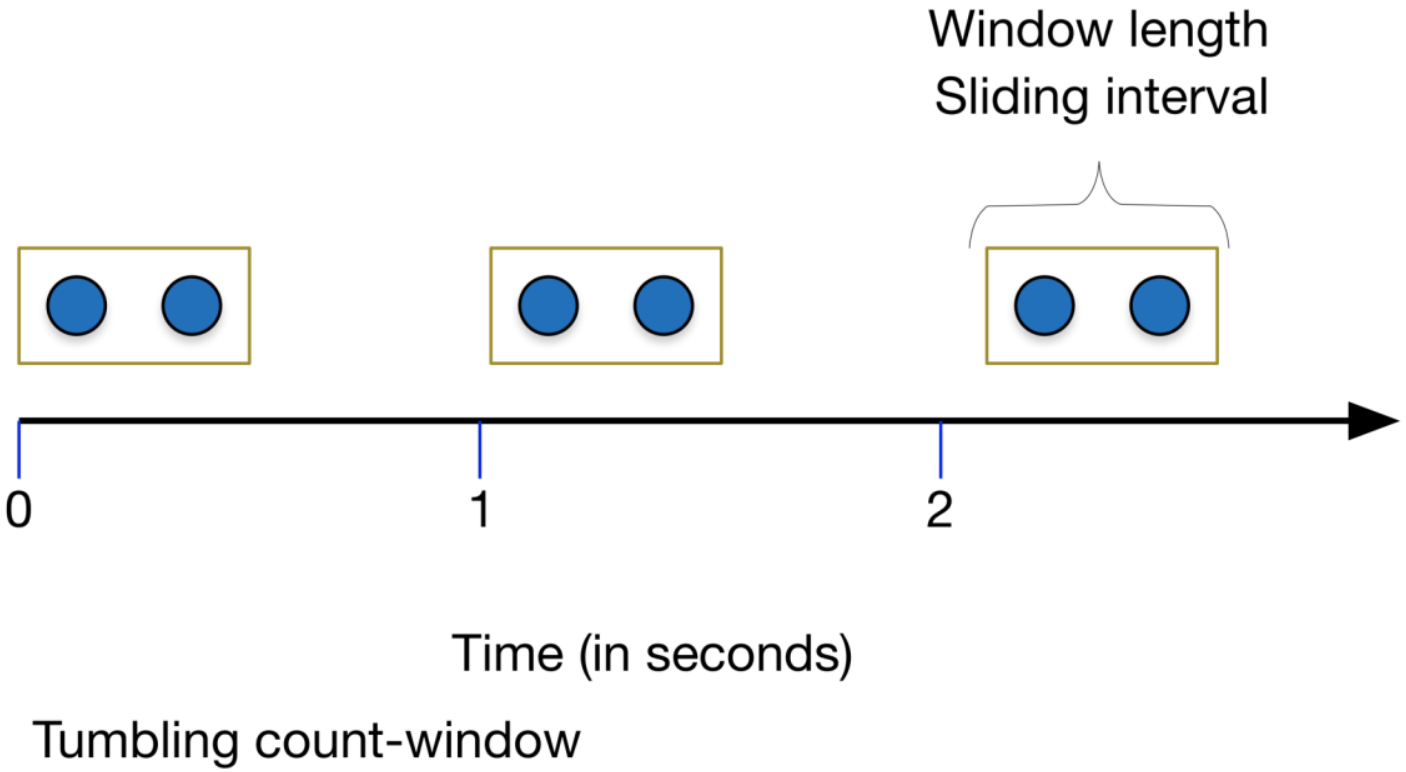
THINKING ABOUT... WINDOWS

TUMBLING WINDOWS

TUMBLING TIME WINDOWING



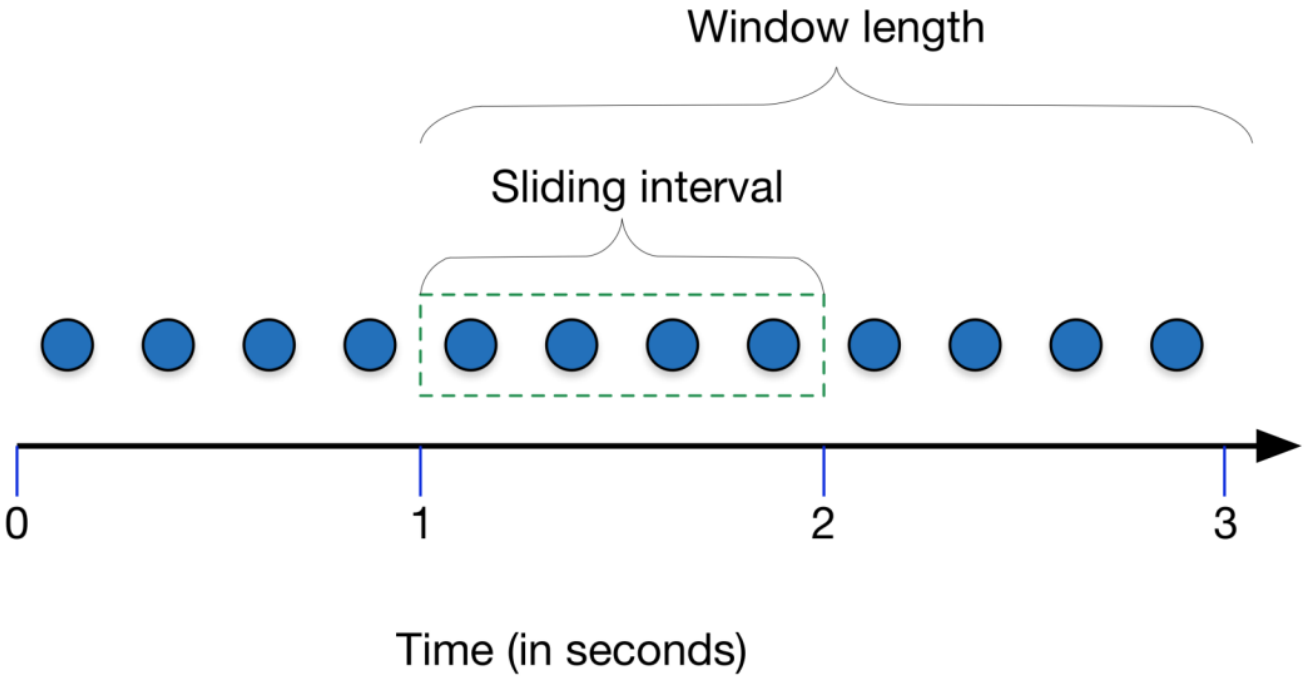
TUMBLING COUNT WINDOWING



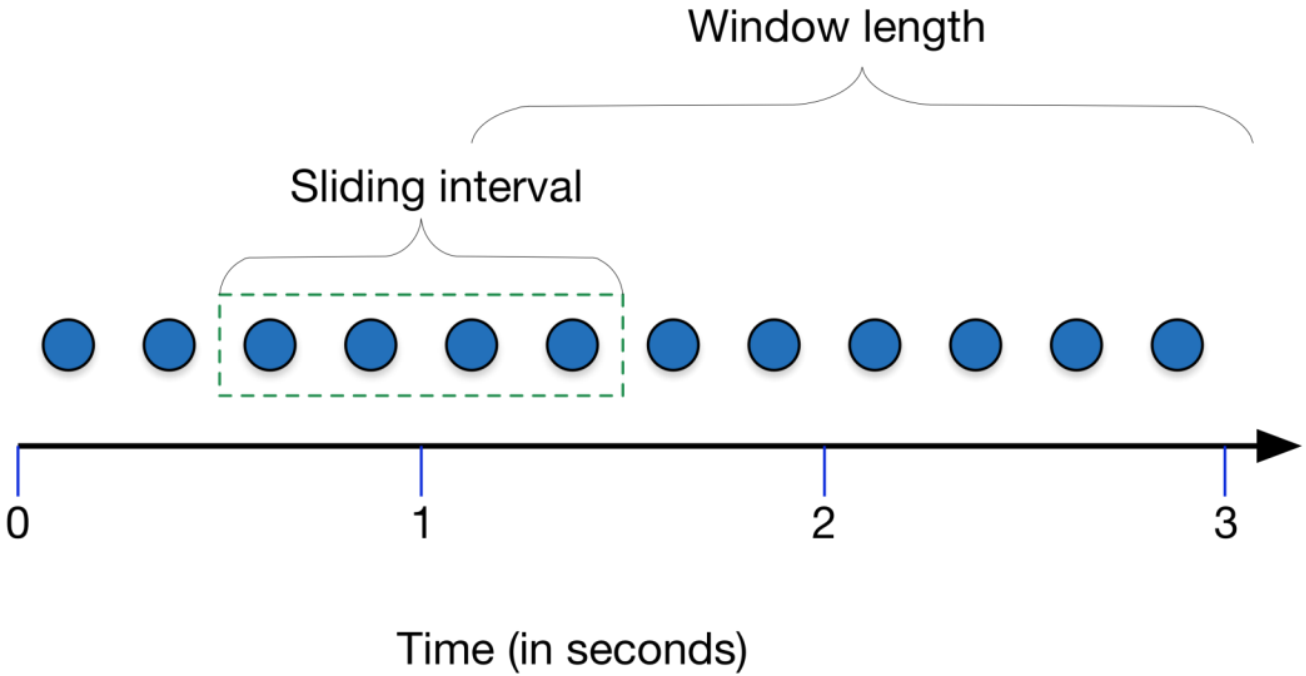
Tumbling count-window

SLIDING WINDOWS

SLIDING TIME WINDOW

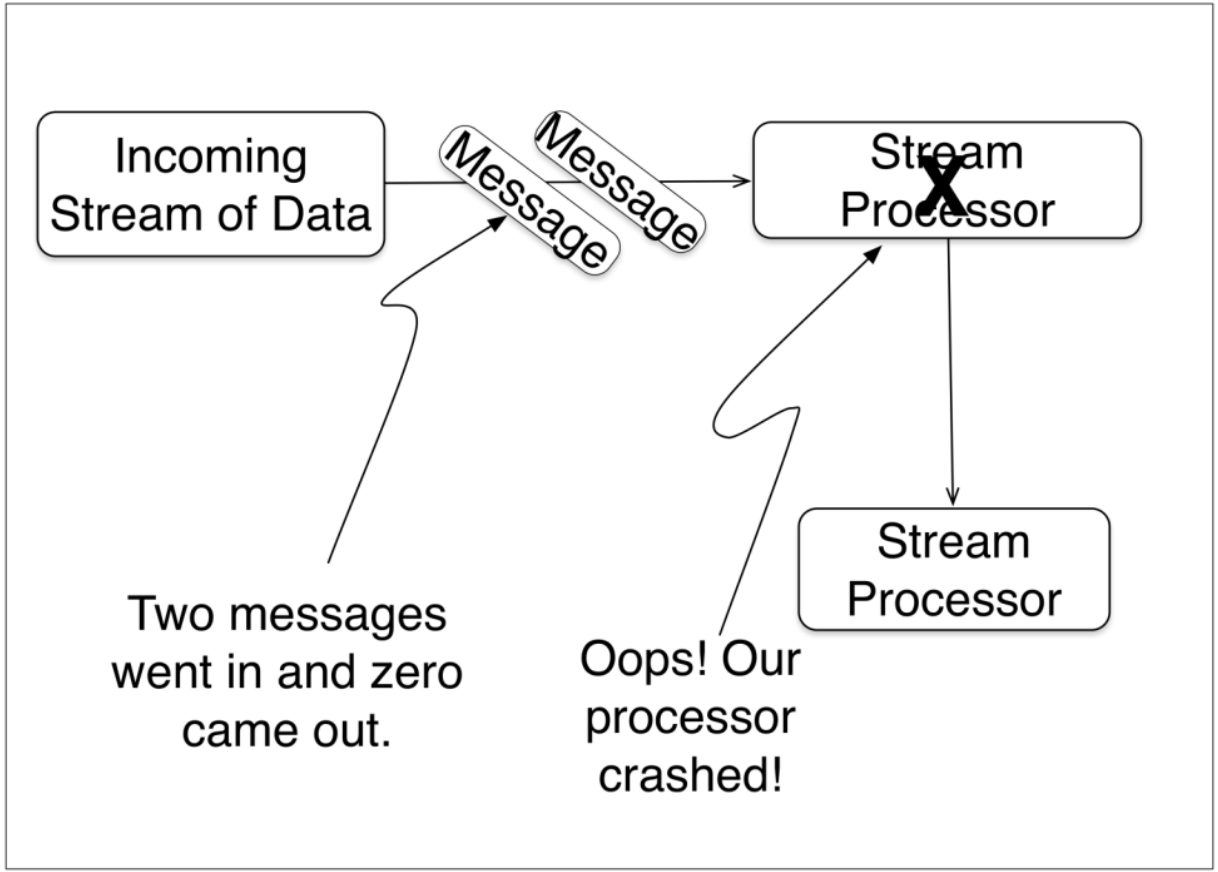


SLIDING COUNT WINDOW

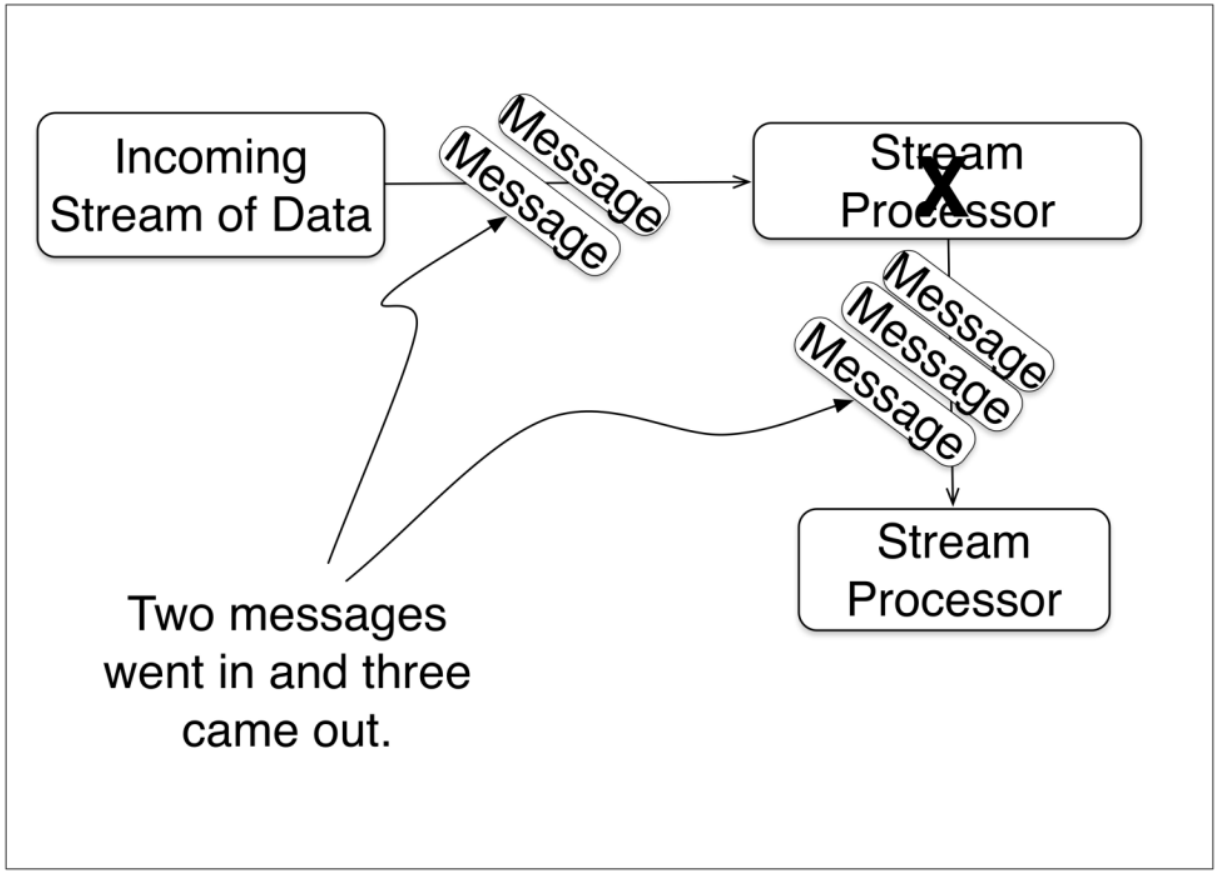


THINKING ABOUT… SEMANTICS

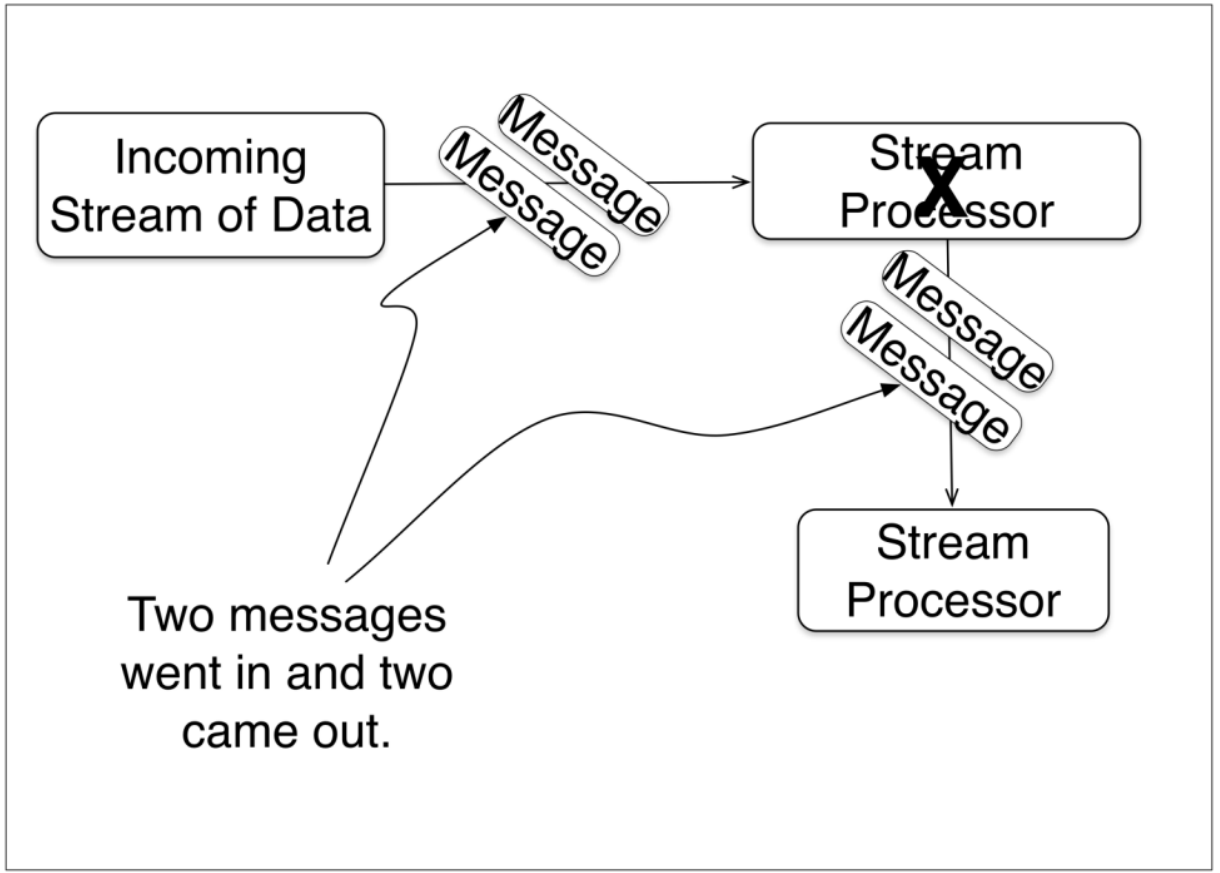
AT-MOST-ONCE



AT-LEAST-ONCE



EXACTLY-ONCE



Final Lab Architecture

PREDICTING MACHINE FAILURE

Photos by Dominik Vanyi on Unsplash

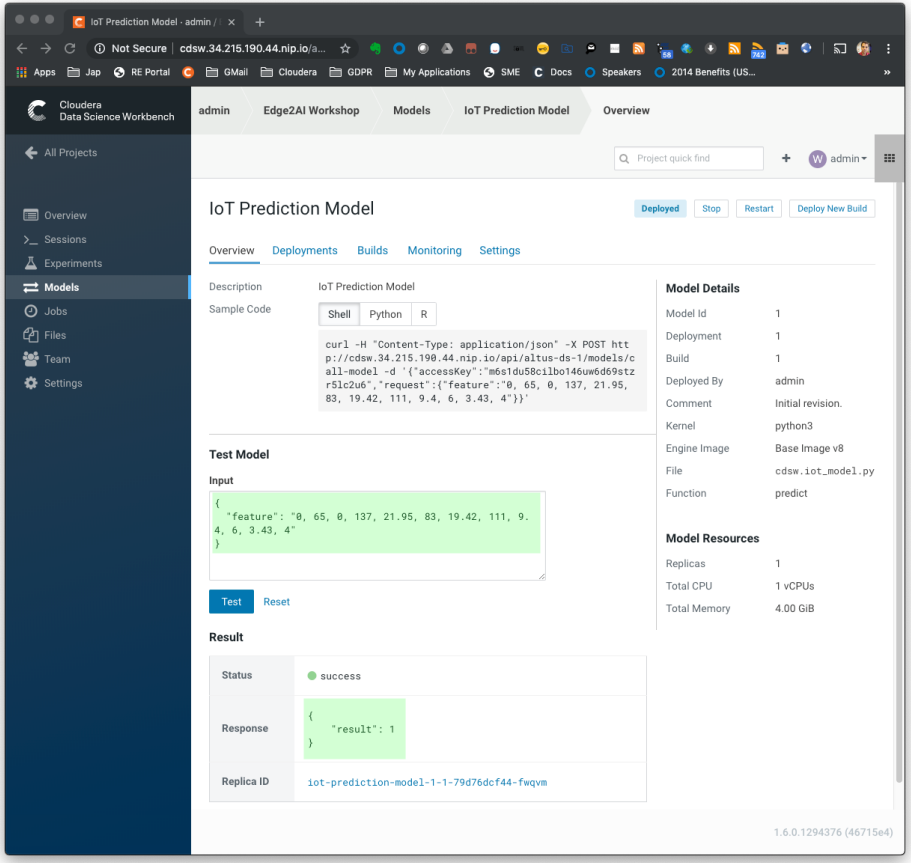
TEMPERATURE SENSORS

Photo by Kai Dahms on Unsplash

- Temperatures of different truck components
- 12 per truck
- Transmitted by radio to a local field office
- Data is published to a MQTT server

TRUCK HEALTH MODEL

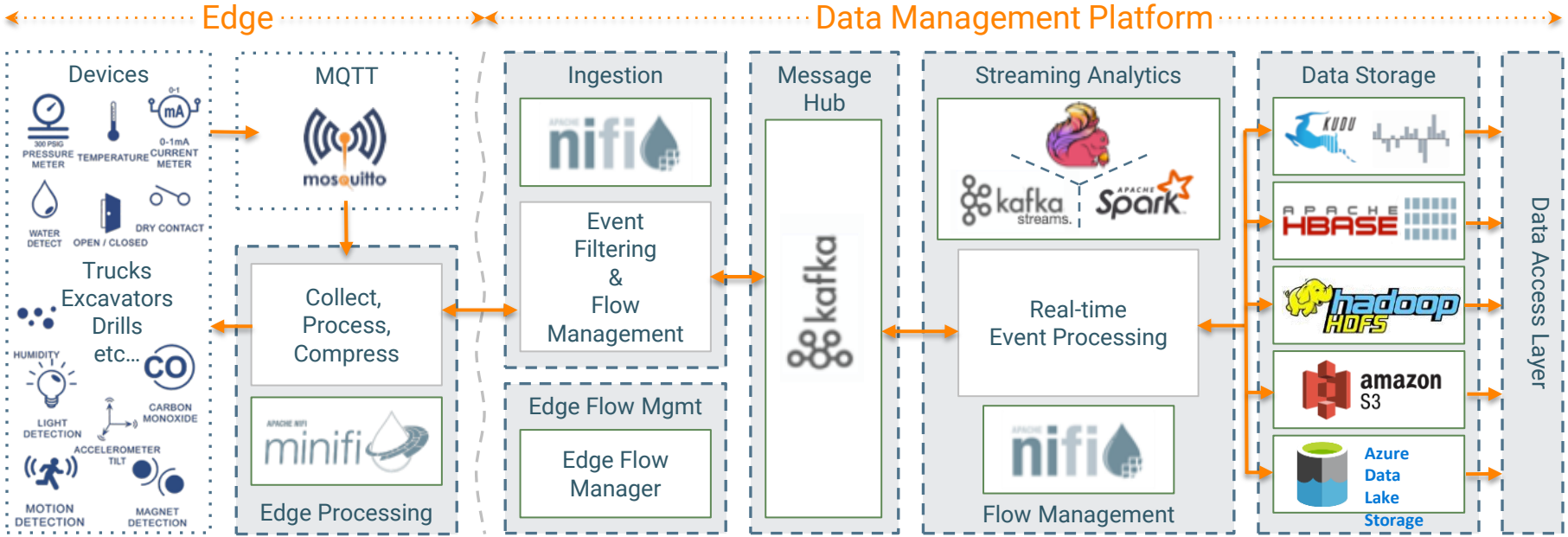
- Cloudera Data Science Workbench (CDSW)
  - Data exploration
  - Model creation, training and deployment
- REST API



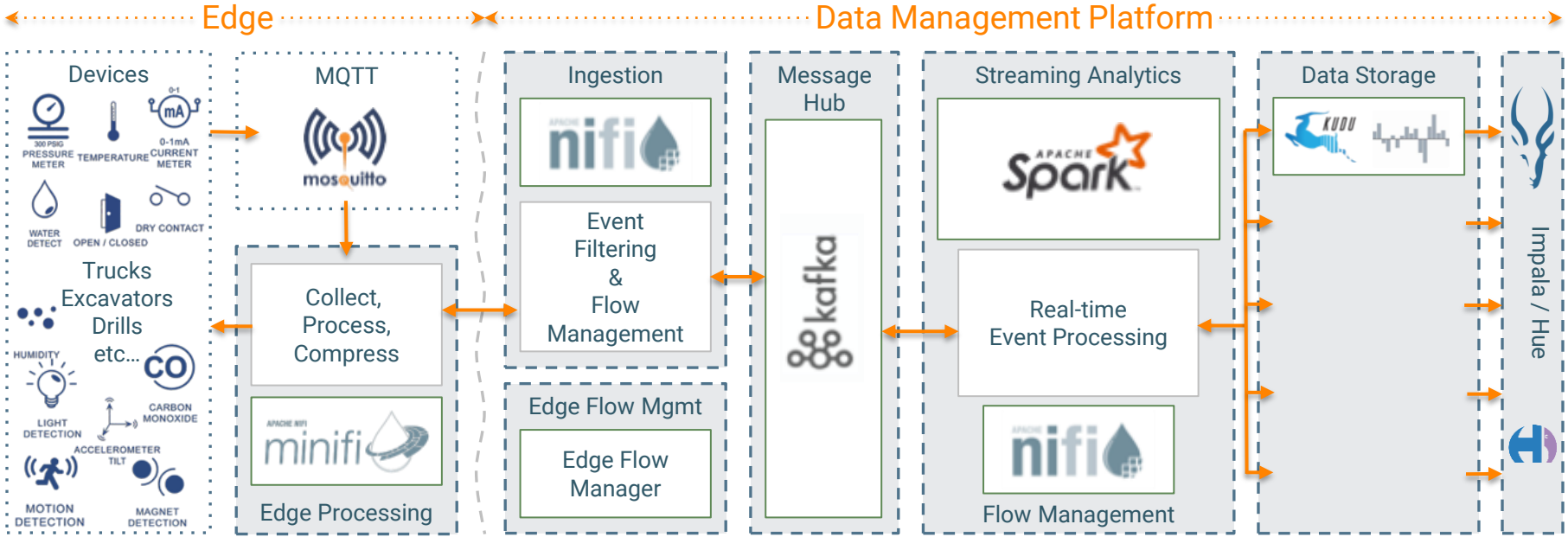
STREAMING AND PREDICTING

- Process the real-time data and it's collected from truck sensors
- Forward data from remote locations to our central Apache NiFi and to our main Apache Kafka message bus
- Read sensor readings from Apache Kafka and enrich the data
  - Call the deployed health model API to predict truck failures
- Update the prediction results in Apache Kudu
- Enable fast analytics on Apache Kudu

REFERENCE ARCHITECTURE



AND THAT'S WHAT WE DID TODAY!

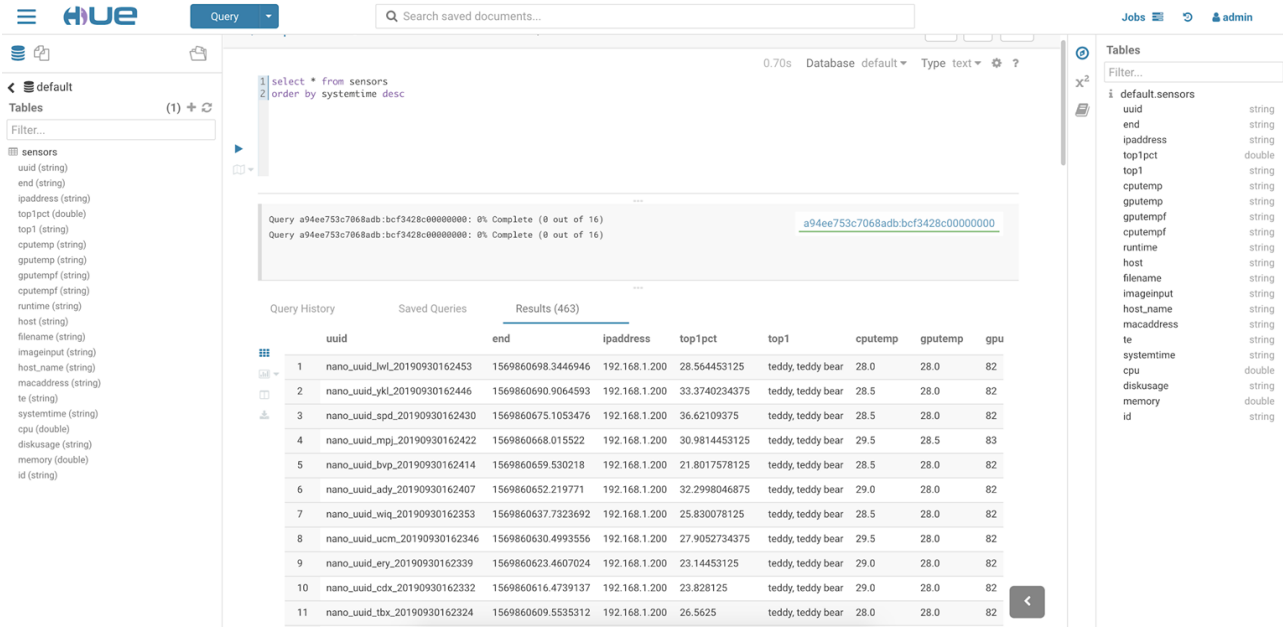


Summary

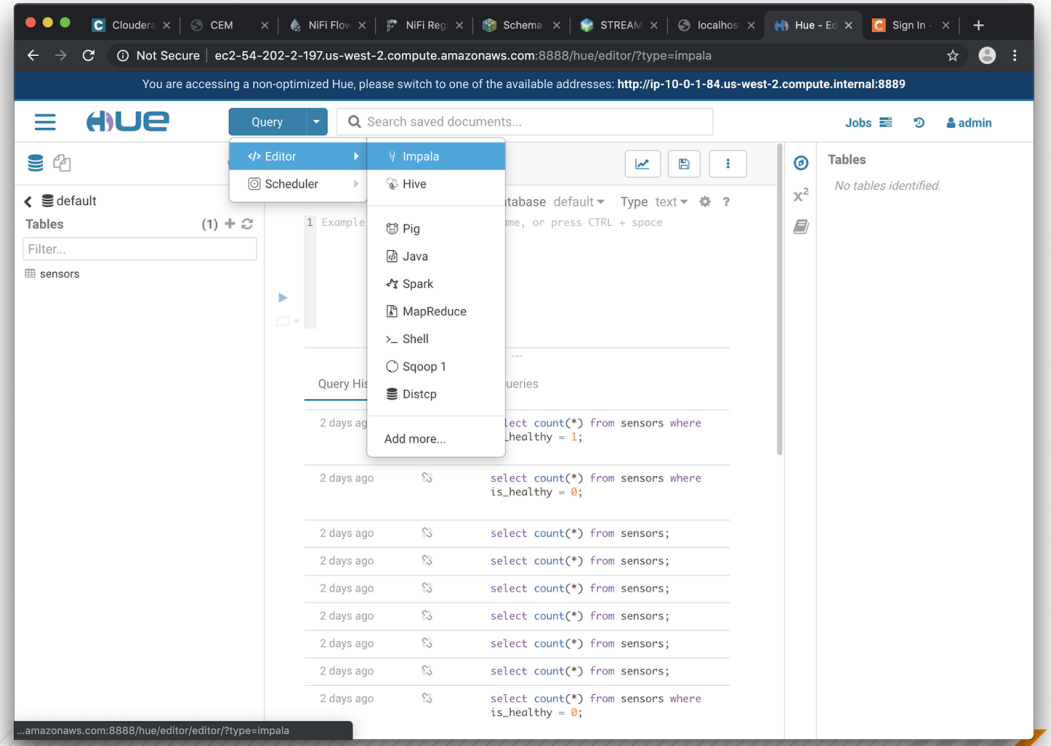
Visualization

Apache Hue

SQL and Query Editor & Performance Diagnostics Tool for the Cloudera Data Platform



Visualization



Apache NiFi - Storage - Apache Kudu

Property	Value
Kudu Masters	**7051**
Table Name	**impala::default.sensors**
Kerberos Credentials Service	No value set
Skip head line	**false**
Record Reader	**AvroReader**
Insert Operation	INSERT
Flush Mode	**AUTO_FLUSH_BACKGROUND**
FlowFiles per Batch	**1**
Max Records per Batch	**100**

REFERENCES

- https://www.datainmotion.dev/2019/05/cloudera-edge-management-introduction.html
- https://www.datainmotion.dev/2019/09/powering-edge-ai-for-sensor-reading.html
- https://www.datainmotion.dev/2019/08/google-coral-tpu-with-edge-devices-and.html
- https://www.datainmotion.dev/2019/08/rapid-iot-development-with-cloudera.html
- https://www.datainmotion.dev/2019/07/edge-data-processing-with-jetson-nano.html
- https://github.com/purn1mak/SMM-NewYork/blob/master/SMM_Kafka_CrashCourse.pdf
- https://docs.cloudera.com/cem/1.0.0/release-notes/topics/cem-whats-new.html

REFERENCES

- https://www.slideshare.net/Hadoop_Summit/kafkasmm-crash-course
- https://www.slideshare.net/Hadoop_Summit/byop-custom-processor-development-with-apache-nifi
- https://www.slideshare.net/Hadoop_Summit/intelligently-collecting-data-at-the-edge-intro-to-apache-minifi-141236290
- https://www.slideshare.net/Hadoop_Summit/apache-nifi-crash-course-131483547