**Slide 1: Title**

**Data Quality Scoring – ML-Driven Pipeline Design**

* Detect malformed, missing, or anomalous records
* Use ML (e.g., XGBoost) to score records
* Runs inline while reading from Kafka
* Protects OpenSearch / VictoriaMetrics from bad data

**✅ Slide 2: Why It Matters**

* Cluster data pipelines ingest large volumes of telemetry, metrics, logs
* Silent errors like missing timestamps or invalid values can corrupt dashboards and models
* Scoring data quality early ensures:
  + High trust in downstream systems
  + Fewer false alerts
  + Better model performance
* Lightweight scoring can be done using ML in real-time

**✅ Slide 3: Example – Correct Record (Clean)**

json

CopyEdit

{

"timestamp": "2025-07-09T08:45:23Z",

"node\_id": "compute-01",

"cpu\_temp": 67.5,

"fan\_speed": 3800,

"power\_watts": 180,

"status": "OK"

}

**✅ Example – Incorrect Record (Bad)**

json

CopyEdit

{

"timestamp": null,

"node\_id": "",

"cpu\_temp": -273.15,

"fan\_speed": "NA",

"power\_watts": 99999,

"status": "UNKNOWN"

}

**✅ Slide 4: Where It Runs in the Pipeline**

**Processing Flow:**

1. Kafka consumer ingests data
2. Features extracted
3. Unified ML model assigns quality\_score
4. Based on score:
   * Forward to OpenSearch or VictoriaMetrics
   * Or route to bad-data topic / alert system

**✅ Slide 5: Training Phase – Step-by-Step**

1. **Collect Sample Records** from multiple Kafka topics (e.g., redfish-telemetry, job-metrics)
2. **Assign topic\_id** to distinguish topic structure
3. **Extract Features** from each record (e.g., cpu\_temp, mem\_used, status)
4. **Label Records** using a config file (quality\_rules.yaml)
   * cpu\_temp between 0–100
   * status ≠ "UNKNOWN"
   * mem\_used ≠ null  
     → label = 1 (clean) or label = 0 (bad)
5. **Create Training Dataset** with features + topic\_id + label
6. **Train a Classifier (e.g., XGBoost)**

python

CopyEdit

model.fit(X, y) # X includes topic\_id and all numeric/categorical features

**✅ Slide 6: quality\_rules.yaml (Labeling Rules Config)**

yaml

CopyEdit

redfish-telemetry:

cpu\_temp:

min: 0

max: 100

status:

not\_in: ["UNKNOWN"]

job-metrics:

mem\_used: not\_null: true

cpu\_usage:

max: 100

**✅ Slide 7: Inference Phase – Real-Time Scoring**

1. Kafka consumer receives a record
2. Extract and clean features
3. Assign topic\_id based on topic
4. Pass features into model:

python

CopyEdit

quality\_score = model.predict\_proba([features\_with\_topic\_id])[0][1]

1. Based on score threshold (e.g., 0.8):
   * If clean → forward to destination
   * If bad → route to bad-data topic, alert, or quarantine

**✅ Slide 8: Why Unified Model Works Well**

* ✅ One model for all topics — avoids training N separate models
* ✅ topic\_id helps model learn context-specific logic
* ✅ XGBoost handles missing values and mixed types
* ✅ Rule-based labeling is easy to maintain and tune
* ✅ Easy to extend to new topics by just updating the rules + retraining

**✅ Slide 9: Summary of Key Benefits**

* ML scoring ensures high-quality data across the pipeline
* Runs inline after Kafka, before sink systems
* Fast, lightweight (XGBoost or LightGBM)
* Works across telemetry, logs, metrics, etc.
* Easily maintained via rules config + retrain cycle

### ✅ Data Quality Scoring – ML-Driven Design (Training + Inference)

#### 📌 Purpose

To automatically detect malformed, missing, or anomalous records in data flowing through our cluster pipeline (e.g., Redfish telemetry, job metrics, etc.) using a unified machine learning model. This prevents low-quality data from corrupting downstream systems like OpenSearch, VictoriaMetrics, or ML models.

---

#### 🧠 Training Phase (One Model for All Topics)

1. \*\*Collect Sample Records\*\* from multiple Kafka topics (e.g., `redfish-telemetry`, `job-metrics`).

2. \*\*Assign `topic\_id`\*\* to each record to indicate its origin.

3. \*\*Extract Features\*\* (topic-specific fields like `cpu\_temp`, `mem\_used`, etc.).

4. \*\*Label Records (`label = 1 or 0`)\*\* using external rule-based logic via a config file:

- For example:

- `cpu\_temp` must be between 0–100

- `mem\_used` must not be null

- `status` must not be "UNKNOWN"

5. \*\*Build Unified Training Dataset\*\* with:

- Mixed features from all topics (missing values allowed)

- `topic\_id` to help model distinguish formats

- `label` as target (1 = clean, 0 = bad)

6. \*\*Train a Binary Classifier\*\* (e.g., XGBoost or LightGBM):

- `model.fit(X, y)` where X includes all features + `topic\_id`

---

#### 🔮 Inference Phase (Live Scoring)

1. \*\*Kafka Consumer\*\* reads new data from any topic.

2. \*\*Extract Features\*\* from the incoming record.

3. \*\*Assign `topic\_id`\*\* based on topic name.

4. \*\*Predict Cleanliness\*\*:

```python

quality\_score = model.predict\_proba([features\_with\_topic\_id])[0][1]