NNMi Local Data Lake Pipeline

Project Overview

This pipeline sets up a **local data lake** on your workstation using open-source technologies:

- **NNMi (HP Network Node Manager i)** sends JSON metrics.
- **Apache Kafka** buffers and distributes the messages.
- **Apache Spark Structured Streaming** processes and converts the data.
- **MinIO** stores data as Parquet files and manages checkpoint metadata.

The result is a **queryable, reliable, scalable data lake** running locally.

Objectives

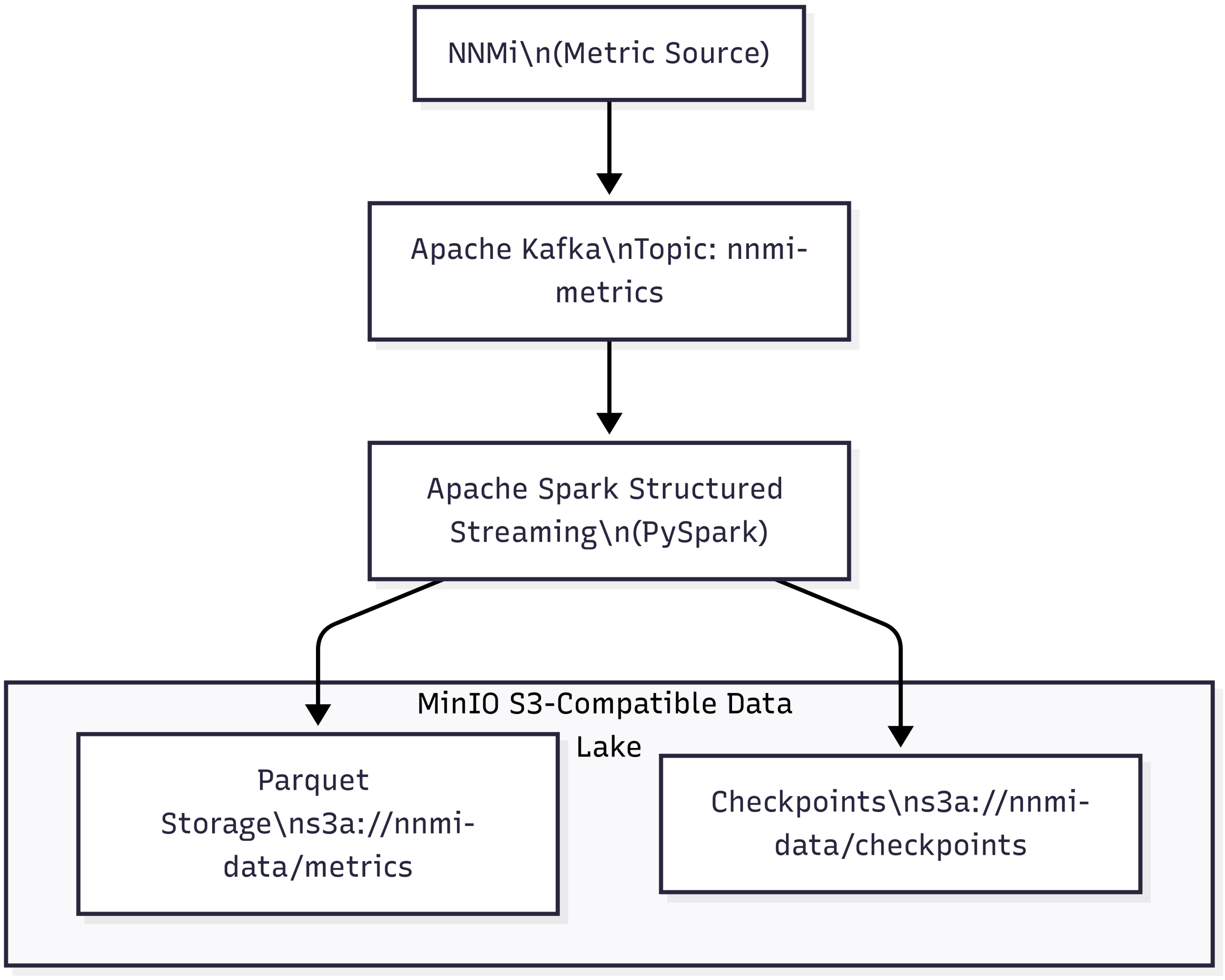
✓ Ingest metrics from NNMi via Kafka
✓ Parse JSON payloads (single or multiple records)
✓ Store columnar data in MinIO using Parquet format
✓ Use Spark checkpoints to ensure exactly-once delivery

Technologies Used

Layer	Technology	Purpose
Metric Producer	NNMi	Emits JSON metrics
Message Broker	Apache Kafka	Buffers and streams the messages
Stream Processor	Apache Spark Structured Streaming (PySpark)	Reads Kafka, parses JSON, writes to MinIO
Serialization	Parquet	Columnar storage optimized for analytics
Object Storage	MinIO (S3-compatible)	Stores metrics and checkpoints

---

Architecture Diagram


NNMi\n(Metric Source)
Apache Kafka\nTopic: nnmi-metrics
Apache Spark Structured Streaming\n(PySpark)
MinIO S3-Compatible Data Lake
Parquet Storage\ns3a://nnmi-data/metrics
Checkpoints\ns3a://nnmi-data/checkpoints

🔄 Data Flow

1. **NNMi** continuously pushes JSON-formatted metrics to Kafka (`nnmi-metrics` topic).
2. **Spark Structured Streaming**:
   - Reads Kafka messages.
   - Parses JSON data into a structured DataFrame.
   - Supports:
     - Single JSON object:

       ```
       {
         "timestamp": "2025-06-28T12:00:00Z",
         "server_name": "Server1",
         "cpu_utilization": 45,
         "memory_used_mb": 2048,
         "network_in_kbps": 120,
         "network_out_kbps": 80
       }
       ```

     - Or JSON array:

```
[
  {...},
  {...}
]
```

- Flattens the data.
- Writes Parquet files to:

```
s3a://nnmi-data/metrics/
```

- Records checkpoint data to:

```
s3a://nnmi-data/checkpoints/
```

3. You can query the Parquet data in Spark SQL, Presto, or other tools.

🧩 Schema

Field	Type
`timestamp`	String
`server_name`	String
`cpu_utilization`	Integer
`memory_used_mb`	Integer
`network_in_kbps`	Integer
`network_out_kbps`	Integer

💡 Key Benefits

☑ **Exactly-once delivery** using checkpoints
☑ **Columnar storage** for fast analytics
☑ **Support for both single and multiple JSON records**
☑ **Local S3-compatible storage** (MinIO)

⚙ Tips & Notes

- Keep MinIO running during ingestion (`minio server /path/to/data`).
- Use `spark-submit` with all Kafka and Hadoop dependencies: spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.4.4 your_script.py
- Spark 3.4.x works well with Python ≤3.11 (not 3.12).
- For testing, you can produce sample JSON into Kafka using `kafka-console-producer`.

- Parquet files can be easily explored using: (python) df = spark.read.parquet("s3a://nnmi-data/metrics/") df.show() You can partition data by date/hour for better performance.

🗂️ What Are `metrics` and `checkpoints`?

🎯 Overview

When you run your pipeline, Spark creates **two critical folders in MinIO**:

▨ 1 `metrics` Folder

This is **your actual data lake storage**.

☑ **What it contains:**

- Parquet files with ingested records: s3a://nnmi-data/metrics/ part-00000-.parquet part-00001-.parquet

markdown Copy Edit

- Each file holds structured tabular data.

☑ **Purpose:**

- Stores your **historical and current metrics** for analysis.
- Can be queried via Spark SQL or other engines.

☑ **Example:**

timestamp	server_name	cpu_utilization	memory_used_mb	network_in_kbps	network_out_kbps
2025-06-28T12:00:00Z	Server1	45	2048	120	80

☑ **Analogy:**

> Think of `metrics` as **your warehouse shelves** where all collected data lives.

▨ 2 `checkpoints` Folder

This is **internal metadata Spark uses to track progress**.

☑ **What it contains:**

- Files like: s3a://nnmi-data/checkpoints/ metadata offsets/0 commits/0 sources/0

yaml Copy Edit

- They track:
- Kafka offsets already read
- Which batches were processed
- Exactly-once delivery info

☑ **Purpose:**

- Ensures **Spark knows where to resume**.
- Prevents duplicates if the job stops and restarts.

✅ **Analogy:**

> Think of `checkpoints` as **your bookkeeping ledger** that says: “I processed up to offset 1234. Next time, start from 1235.”

🧩 How They Work Together

Example Workflow:

1️⃣ **First run:**

- Reads offsets 0–999.
- Writes Parquet data to `metrics`.
- Saves checkpoint info up to offset 999.

2️⃣ **Second run:**

- Starts at offset 1000 (avoids reprocessing).

✅ **Why you should keep both:**

- Deleting `checkpoints` = duplicate ingestion.
- Deleting `metrics` = data loss.

🚀 Quick Recap

Folder	What It Is	Why It Matters
`metrics`	Parquet data lake storage	Holds your usable metrics data
`checkpoints`	Spark’s tracking metadata	Enables exactly-once processing

---

💡 **Tip:**
Always backup both folders if you migrate or clean up MinIO.
