**Open Standards Analytics & Anomaly Detection Stack**

A professional, vendor‑neutral reference implementation for zero‑copy cataloging, SQL access, and anomaly detection over Parquet data using Apache Iceberg, Spark Thrift Server, and Apache Superset.

# Table of Contents

* Summary
* Who This Is For
* Architecture Overview
  + Anomaly detection (integrated)
* Quickstart
* Repository Layout (Example)
* Prerequisites
* Configuration
  + Spark & Iceberg
  + Superset
  + Entrypoint & Runtime Knobs
  + Iceberg HadoopCatalog layout
* Using the Stack
  + Catalog & Query
  + Anomaly Detection Job
  + Exploration in Superset
  + Sharing With Customers
* Services & Images
* Operating the Stack
* Troubleshooting
* Performance Tuning
* Security Notes
* Validation & QA
* Version Matrix
* Open Standards & Interoperability (Deep Dive)
* Adoption & Migration Guide
* Glossary
* License & Credits

# Summary

**Problem.** Many organizations sit on siloed, on‑prem datasets in proprietary systems (MetricDB). Sharing insights with customers or partners often means brittle exports, custom ETL, or expensive re‑platforming.

**Solution.** This stack operationalizes an **open‑standards data plane** that layers **Apache Iceberg** (table format) over your existing **Parquet** (file format) — **without copying data**. It provides:

* **Zero‑copy cataloging** (Iceberg HadoopCatalog) of folders of Parquet files.
* **SQL access** via Spark Thrift Server (compatible with Hive clients, PyHive).
* **BI & dashboards** through Apache Superset (or any tool that speaks to Spark/Trino/Presto).
* **Built‑in anomaly detection** jobs in PySpark (z‑score, IQR, k‑means, KNN) to surface outliers by system\_id and timestamp.

**Why it matters.** You move away from closed systems toward **portable, interoperable** data that’s easier to productize and share — **without vendor lock‑in**.

**Key value:**

* **Open by default.** Parquet + Iceberg + SQL → engine‑agnostic, durable access.
* **Zero‑copy onboarding.** Register existing Parquet folders as Iceberg tables. No re‑ingest or duplication.
* **Customer enablement.** Package a read‑only catalog and expose it via SQL/BI so customers can self‑serve.
* **Fast time‑to‑insight.** Outlier detection surfaces suspicious behavior on day one.
* **Portable deployment.** Runs locally via Docker; adaptable to S3/GCS & managed catalogs.

# Who This Is For

* **Data platform engineers** modernizing file‑backed lakes into table formats.
* **Analytics/BI teams** needing governed, SQL‑native access to file data.
* **Customer/field teams** delivering shareable, vendor‑neutral datasets.
* **External consumers** who want open data products without bespoke integrations.

# Architecture Overview

A screenshot of a computer program

AI-generated content may be incorrect.

**Key idea:** You never copy raw data into Iceberg. The importer registers each Parquet file as a DataFile in the Iceberg catalog so queries run **in place**. The anomaly script writes results as Parquet under ./preds/, which you can optionally promote as Iceberg tables for downstream analytics.

# Quickstart

Goal: Stand up the stack, catalog your Parquet data, run anomaly detection, and explore results — **without copying data**.

1. **Prepare data**
   * Place Parquet under ./data/<table\_name>/... (each subfolder becomes nyc.<table\_name>).
   * If needed, normalize dataset to compatible data types:
   * python sanitize\_parquet.py \

--in ./data/System\_Interface\_Counters \

--out ./data\_casted/System\_Interface\_Counters --overwrite

* + mv ./data\_casted/System\_Interface\_Counters ./data/System\_Interface\_Counters

1. **Launch the stack**

docker-compose up -d --build

This starts Postgres (Superset metadata), Spark Thrift (SQL endpoint + importer), and Superset.

1. **Verify catalog & query**
   * Open Superset → [http://localhost:8088](http://localhost:8088/) (admin/admin).
   * Connect database
     1. Select Apache Spark SQL
     2. URI: hive://spark-thrift-server:10000/default?auth=NOSASL
   * In **SQL Lab**, choose **Iceberg via Spark Thrift** and run:
   * USE nyc;
   * SHOW TABLES;
   * SELECT COUNT(\*) FROM <your\_table>;
2. **Run anomaly detection (local)**
3. # defaults
4. python anomalyDetection.py
5. # common overrides
6. Z\_THRESH=3 MIN\_POINTS=50 TOPK=100 \
7. KMEANS\_K=3 KM\_PR\_CUTOFF=0.995 \
8. KNN\_MODE=time KNN\_K=8 KNN\_PR\_CUTOFF=0.995 \
9. python anomalyDetection.py

**CLI flags (pass on the command line)**

* -i, --input <PATH|GLOB> — Parquet input directory or glob
* -f, --feature <COLUMN> — numeric feature column to analyze
* --timestamp-col <COLUMN> — timestamp column name
* --system-col <COLUMN> — system/group id column name
* -o, --out <DIR> — output directory for Parquet results

*(Each of the above also honors an env var default: INPUT\_PATH, FEATURE\_COL, TIMESTAMP\_COL, SYSTEM\_COL, OUTPUT\_DIR.)*

**Environment variables (set before running)**

* Z\_THRESH (float, default 2)
* MIN\_POINTS (int, default 20)
* TOPK (int, default 50)
* KMEANS\_K (int, default 2)
* KM\_PR\_CUTOFF (float, default 0.99)
* KNN\_MODE (time | lsh, default time)
* KNN\_K (int, default 5)
* KNN\_PR\_CUTOFF (float, default 0.99)
* LSH\_BUCKET (float, default 1.0) — LSH only
* LSH\_THRESH (float, default 1.5) — LSH only
* MAX\_GROUP\_ROWS (int, default 250000) — LSH safety cutoff
* PRINT\_LIMIT (int, default 100) — console preview rows
* SPARK\_DRIVER\_MEMORY (e.g., 6g, default 6g)
* SHUFFLE\_PARTITIONS (int, default 64)

Outputs land in ./preds/ as Parquet (e.g., temp\_outliers\_zscore/, temp\_outliers\_kmeans/).

1. **(Optional) Promote results to Iceberg tables**
   * Point Spark at ./preds/... and either **CREATE TABLE AS SELECT** or move folders under ./data/ so they’re imported like any other table.

# Repository Layout (Example)

.

├─ data/ # raw data mounted into containers

│ ├─ stats\_test/ # input for the outlier script (system\_id, timestamp, temperature)

│ └─ <table\_a>/, <table\_b>/ ... # each subfolder => nyc.<table\_name>

├─ preds/ # outputs written by outlier job (created at runtime)

│ ├─ temp\_outliers\_zscore/

│ ├─ temp\_suspects\_topk/

│ ├─ temp\_outliers\_iqr/

│ ├─ temp\_outliers\_kmeans/

│ └─ temp\_outliers\_knn/

├─ conf/

│ └─ spark-defaults.conf # Spark conf injected into Spark containers

├─ docker-compose.yml

├─ Dockerfile.spark

├─ Dockerfile.superset

├─ entrypoint-spark.sh

├─ entrypoint-superset.sh

├─ src/main/java/com/example/ImportParquetFolders.java

├─ pom.xml

├─ sanitize\_parquet.py # Normalize data (run manually)

├─ superset\_config.py

├─ pyhive\_spark\_patch.py

└─ anomalyDetection.py # anomaly detection job (run manually)

The exact file names can be adapted to your repository as needed. If you still have std\_distribution\_outliers.py, it is functionally equivalent to the newer anomalyDetection.py described below.

# Prerequisites

* **CPU/RAM:** 4+ cores; ~12–16+ GB RAM recommended to run Spark Thrift + Superset comfortably.
* **Docker & Compose:** Recent Docker Desktop or engine.
* **Open Ports:** 5432 (Postgres), 8088 (Superset), 10000 (Spark Thrift).
* **Java 17:** Provided in the Spark image (required only if building locally).
* **Python:** Needed if running the anomaly job or sanitization locally.

**Input data contract (for anomaly job)**

* Directory: ./data/stats\_test/ containing Parquet files (\*.parquet or \*.prq) with columns:
  + system\_id (STRING)
  + timestamp (Spark TIMESTAMP; epoch µs can be auto‑cast by sanitize\_parquet.py)
  + temperature (numeric)

# Configuration

**Spark & Iceberg**

**conf/spark-defaults.conf**

spark.sql.extensions=org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions

spark.sql.catalog.iceberg=org.apache.iceberg.spark.SparkCatalog

spark.sql.catalog.iceberg.type=hadoop

spark.sql.catalog.iceberg.warehouse=file:///warehouse

spark.sql.defaultCatalog=iceberg

spark.thriftserver.thrift.bind.host=0.0.0.0

spark.thriftserver.thrift.port=10000

spark.local.dir=/var/lib/spark/tmp

* Iceberg is the default catalog (iceberg).
* Thrift server binds to 0.0.0.0:10000.

**Superset**

**superset\_config.py** registers a database named **“Iceberg via Spark Thrift”** using:

hive://spark-thrift-server:10000/default?auth=NOSASL

It also includes a SQLAlchemy hook to rewrite SHOW CREATE VIEW → SHOW CREATE TABLE for Spark compatibility and extends SQL Lab timeouts for long queries.

**Entrypoint & Runtime Knobs**

**entrypoint-spark.sh** (selected highlights):

* RUN\_IMPORTER=1 → run Java importer on container start (set 0 after initial bootstrap).
* WAREHOUSE\_URI=file:///warehouse → shared volume across containers.
* Spark resources: SPARK\_LOCAL\_THREADS, SPARK\_DRIVER\_MEMORY, SPARK\_SQL\_SHUFFLE\_PARTITIONS, vectorized reader toggles, AQE, etc.

**Iceberg HadoopCatalog layout**

* Warehouse: file:///warehouse (mounted from ./warehouse).
* Namespace: nyc. Each subfolder under /data imports as nyc.<folder\_name>.
* Unpartitioned tables by default (evolve partitions later via Iceberg SQL).

**Important:** The importer appends all files it finds. If you re‑run without cleaning metadata, you may create duplicate references. Disable with RUN\_IMPORTER=0 after initial registration, or rebuild idempotently.

# Using the Stack

**Catalog & Query**

1. Launch services (docker-compose up -d --build).
2. Make sure database is successfully connected
   1. URI: hive://spark-thrift-server:10000/default?auth=NOSASL
3. In Superset → **SQL Lab**:
   1. SHOW TABLES;
   2. Should display all tables you created in data directory
4. Navigate to Dashboards tab and create your dashboard
5. Navigate to Charts tab and create your first chart using your tables
6. Drag and drop columns to populate query
7. Create chart and save to dashboard
8. Navigate to dashboard and view your charts

**Transformations**

* 1. Navigate to SQL Lab
  2. Run query to transform or join tables

**Example Query:**

SELECT

t.timestamp,

t.system\_id,

t.interface,

t.tx\_bps,

t.rx\_bps,

t.fcs\_errors\_per\_second,

t.temperature,

t.voltage,

t.pattern

FROM (

SELECT

c.\*,

s.temperature,

s.voltage,

s.pattern,

ROW\_NUMBER() OVER (

PARTITION BY c.timestamp, c.system\_id, c.interface

ORDER BY abs(unix\_timestamp(c.timestamp) - unix\_timestamp(s.timestamp)) ASC

) as rn

FROM counter\_test c

JOIN stats\_test s

ON c.system\_id = s.system\_id

AND c.interface = s.interface

AND abs(unix\_timestamp(c.timestamp) - unix\_timestamp(s.timestamp)) <= 120

) t

WHERE t.rn = 1;

* 1. Click `Save dataset` button, give your new transformed dataset a name, and save
     1. This dataset is considered a **Virtual Dataset** and will be saved under the **Datasets** tab
     2. Here, you can edit your **Virtual datasets** for any additional modifications

**Successful Query:**

**SQL Lab:**

A screenshot of a computer

AI-generated content may be incorrect.

**Datasets:**

A white background with black lines

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.**A screenshot of a data chart

AI-generated content may be incorrect.**

* 1. Once you create your **Virtual Dataset**, you can use your modified dataset to create a chart

A screenshot of a computer

AI-generated content may be incorrect.

# Anomaly Detection

**Script:** anomalyDetection.py

**Inputs & safety**

* Reads from ./data/stats\_test/\*.prq (or any directory/glob you supply).
* Ensures timestamp is a true TIMESTAMP (casts from string if needed).
* Caches an intermediate scored DataFrame reused across detectors.

**Tunables (env overrides)**

Z\_THRESH=2 # z-score threshold

MIN\_POINTS=20 # min rows per system

TOPK=50 # always surface N strongest |z|

KMEANS\_K=2 # k-means clusters (1-D over z)

KM\_PR\_CUTOFF=0.99 # top 1% farthest by centroid distance

KNN\_MODE=time # 'time' (adjacent neighbors) or 'lsh'

KNN\_K=5 # neighbors to average

KNN\_PR\_CUTOFF=0.99 # top 1% by knn\_dist

# LSH-only

LSH\_BUCKET=1.0

LSH\_THRESH=1.5

MAX\_GROUP\_ROWS=250000 # groups above this fall back to 'time'

**Methods**

* **Z‑Score (per system)** — z=(x−mean)/std; emits rows with |z|>Z\_THRESH and n≥MIN\_POINTS; also writes a **TopK**suspects list by global |z|.
* **IQR (per system)** — flags rows outside [q1−1.5·IQR, q3+1.5·IQR]; writes both **bounds per system** and **detailed outliers** (with distance from bound).
* **K‑Means (1‑D over z)** — VectorAssembler → StandardScaler → KMeans; scores centroid distance; marks tail by percent\_rank() ≥ KM\_PR\_CUTOFF.
* **KNN (per system)**
  + **time mode (default):** compares to K leading/lagging neighbors (linear memory profile).
  + **lsh mode:** approximate neighbors via BucketedRandomProjectionLSH for small groups; big groups fall back to time mode for safety.

**Outputs (all Parquet, coalesced per folder)**

* ./preds/temp\_outliers\_zscore/ → system\_id, timestamp, temperature, z
* ./preds/temp\_suspects\_topk/ → strongest global |z|
* ./preds/temp\_outliers\_iqr/ → IQR outliers (compact)
* ./preds/iqr\_bounds/ → per‑system q1,q3,iqr,low,high,n,n\_outliers
* ./preds/iqr\_outliers\_detailed/ → detailed IQR outliers with distance to bound
* ./preds/temp\_outliers\_kmeans/ → KMeans (cluster, km\_dist, km\_pr)
* ./preds/temp\_outliers\_knn/ → KNN (knn\_dist, knn\_pr)

**Examples**

# Default run

python anomalyDetection.py

# Target a directory on specific data and feature

python anomalyDetection.py -i ./data/stats\_test -f temperature

# Glob pattern & column overrides

python anomalyDetection.py -i "./data/stats\_test/\*.prq" -f cpu\_temp --timestamp-col ts --system-col device --out preds\_cpu

**Exploration in Superset**

1. Create a **Dataset** from an Iceberg table.
2. Build charts (e.g., tail KNN distances by system\_id, top‑K |z| by day).
3. Assemble a dashboard for internal reviews or customer‑facing use.

**Sharing With Customers**

* Ship the catalog + data as a **read‑only bundle** (mount Parquet and the Iceberg warehouse) — or
* Host in **object storage** (S3/GCS) with a managed catalog (e.g., AWS Glue) and grant consumer access.

# Services & Images

* **Postgres 15** — metadata DB for Superset (volume: pgdata).
* **Spark Thrift Server** — built from Dockerfile.spark; mounts ./warehouse & ./data; runs the importer, then Thrift on :10000.
* **Superset 5.0.0** — built from Dockerfile.superset; connects via PyHive to the Thrift endpoint; configured via superset\_config.py and a small PyHive dialect patch.

# Operating the Stack

**Start / Stop**

docker-compose up -d --build # start

# logs

docker logs -f spark-thrift-server

docker logs -f superset-iceberg

# stop

docker-compose down

**Query from CLI (optional)**

* Connect any Hive‑compatible client to localhost:10000 (NOSASL).

# Troubleshooting

**Iceberg NotFoundException / missing local Parquet**

* **Symptoms:** Queries fail due to missing file paths in Iceberg metadata.
* **Causes:** files moved/renamed/deleted after import; different /data layout on subsequent importer runs.
* **Fixes:** restore files to original paths; or drop & re‑import; or run Iceberg maintenance (e.g., REWRITE DATA FILES) after restoring paths.

**Type incompatibilities (e.g., UINT64)**

* Use sanitize\_parquet.py before import or before running the anomaly job.

**Memory / OOM during KNN LSH**

* Large system\_id groups can blow up the LSH join. Prefer KNN\_MODE=time, increase driver memory, or lower MAX\_GROUP\_ROWS/KNN\_K.

**Superset timeouts**

* Extended, but long scans can still exceed browser/proxy limits. Filter, add partitions, or pre‑aggregate with materialized views.

**Port conflicts**

* Change port mappings in docker-compose.yml if 5432/8088/10000 are in use.

**Output writers slow or memory‑spiky**

* coalesce(1) writes a single file but can be slow. Remove or write a small number of partitions for large outputs.

# Performance Tuning

**Spark (entrypoint defaults)**

* SPARK\_DRIVER\_MEMORY / SPARK\_DRIVER\_OVERHEAD: increase for larger joins/LSH.
* spark.sql.shuffle.partitions: lower (16–64) for small data; raise for big.
* Adaptive Query Execution enabled; tune advisoryPartitionSizeInBytes.
* Parquet vectorized reader + filter pushdown enabled.

**Outlier job**

* Prefer KNN\_MODE=time for very large time series.
* Reduce KNN\_K or raise thresholds (e.g., KNN\_PR\_CUTOFF=0.999) to focus on extremes.
* Remove .cache() if RAM is tight.

**Importer**

* Run once, then set RUN\_IMPORTER=0 to avoid repeated appends.

**Iceberg**

* Add partition evolution for common predicates (date/time, system\_id) to accelerate scans.

# Security Notes

* Default Superset credentials are admin/admin — **change for production**.
* Thrift uses **NOSASL** and binds to 0.0.0.0:10000. Restrict access (Docker network only) or front with a gateway.
* No TLS in this local setup. For production, terminate TLS at a reverse proxy and secure DB connections.

# Validation & QA

**Sample SQL checks (SQL Lab)**

SHOW TABLES; -- Should display all tables in data directory after successful connection to database

# Version Matrix

| **Component** | **Version** |
| --- | --- |
| Spark | 3.4.1 |
| Hadoop client | 3.x |
| Iceberg | 1.9.2 |
| Superset | 5.0.0 |
| Postgres | 15 |
| Java | 17 |
| PyArrow | 16.1.0 |
| Parquet (Java) | 1.12.3 |

Update versions via Dockerfile.spark, Dockerfile.superset, and pom.xml as appropriate.

# Open Standards & Interoperability (Deep Dive)

* **Parquet (storage):** Columnar, splittable, compressed, portable across engines.
* **Iceberg (table):** Schema and partition evolution, ACID, snapshots/time travel, metadata tables.
* **Decoupled compute:** This reference uses Spark Thrift; you can add Trino/Presto, Flink, Spark SQL directly, or DuckDB for local dev (read‑only).
* **Zero‑copy:** Importer registers file URIs; data stays in place.
* **BI compatibility:** Any tool that talks to Spark/Trino/Presto (or reads Iceberg) can consume the same tables.

# Adoption & Migration Guide

1. Export source tables to Parquet (ideally partitioned by date/system).
2. Sanitize types (fix UINT64, normalize timestamps).
3. Bootstrap the catalog (importer → Iceberg tables).
4. Validate counts and schema diffs; smoke‑test with SQL.
5. Publish datasets (Superset, JDBC/ODBC, or REST proxies).
6. Iterate: add partitions, materialized views, and data‑quality checks.

# Glossary

* **Parquet:** Open columnar file format for analytics.
* **Iceberg:** Open table format providing evolution & ACID over files.
* **Catalog:** Maps table names to metadata/snapshots.
* **Spark Thrift Server:** SQL endpoint compatible with Hive clients.
* **Superset:** Open source BI for dashboards and SQL exploration.
* **LSH:** Bucketed Random Projection LSH for approximate KNN.

# License & Credits

* Apache Spark, Apache Iceberg, and Apache Superset are trademarks of their respective foundations.
* This project glues them together into a local, file‑backed analytics stack and an anomaly‑detection demo.