PlantPulse® **Industrial DataLake**

TECHNICAL WHITEPAPER

Technical Whitepaper · REV. 2026.09

Industrial data infrastructure assembled for data.

Aster engine set TSDB · CEP · MQTT · 3-tier data lifecycle · Iceberg lakehouse · time-series engine · single image, 16 components · `pd` one-line startup. Structure · figures · configuration · measurements taken as-is from the repository and running instances.



16
components · one image

7.1×
time-series storage reduction (measured)

Aster
engine set — TSDB · CEP · MQTT

4
query languages ·

September 2026

PlantPulse DataLake · on-premises / closed network · for technical reviewers

kopens®

00 · Summary

Summary — How is an industrial data lake different from a general-purpose data lake?

Factory data has tens of thousands of time-series events per second, years of retention, the context of equipment hierarchy, and the constraint of a closed network — all at once. A general-purpose lakehouse only handles the *retention* part well. PlantPulse Industrial DataLake is an infrastructure assembled into a single image that covers **ingestion → hot/warm/cold storage → time-series computation → standard querying → operations** with industrial data as the baseline. The heart of the system, the **Aster engine set** — the time-series storage **TSDB**, the real-time decision engine **CEP**, and the equipment entry point **MQTT** — is developed and maintained from source by KOPENS. The time-series engine (TSE) and the data gateway are also developed in-house and are covered separately because of their different nature (pp. 10, 15–16).

16
components, one image, one `pd start`

7.1×
time-series storage reduction — 11.9 → 1.7 B/row (measured at 20M rows)

233k
rows/s ingest (Aster TSDB single-node benchmark)

SQL:2016
Aster CEP — `MATCH_RECOGNIZE` implemented in the engine, contrasted with Flink

3
tiers — in-memory, Aster TSDB ring, Iceberg

RF 3
token-ring replication, auto-adjusted by `pd node add`

4
query languages — PQL, CQL, SQL, TQL — one gateway

12
console screens, `--json` contract, 9 exit codes

Table of Contents

01	The Problem — Three Walls	3
02	Five Design Principles	4
03	Architecture — Batch, Ports, Startup Order	5
04	Image — Source, Verification, Config Rendering	6
05	Data Lifecycle — Hot, Warm, Cold	7
06	Aster TSDB — Time-Series Database	8
07	Schema — Keyspace, Partitioning, Compression	9
08	Time-Series Engine TSE — Aggregation, TQL, Prometheus	10
09	Time-Series DB Comparison — InfluxDB, TimescaleDB	11
10	Lakehouse — Iceberg, Spark, Archive	12
11	Messaging — Kafka, MQTT, Two Buffers	13
12	Aster CEP — Real-Time Decision, EQL	14
13	Query Layer — Data Gateway	15
14	Control & Audit — Quotas, Errors, Drivers	16
15	Scaling & Replication — Token Ring, Workers	17
16	Operations — Console, pd, JSON Contract	18
17	Protection — Backup, Recovery, Upgrade	19
18	Measurements — Instances in Operation	20
19	Comparison with a General-Purpose Data Lake	21
20	Sizing, Sources, Contact	22

Audience. Data architects, infrastructure/DB operators, security reviewers. This is not a sales document — all figures come only from the repository's configuration, schema, and benchmark documentation and from measurements of the instance running on 2026-09-08 (dev 41); sources are on p. 22.

01 · Problem

Three Walls — Historian, General-Purpose Lake, DIY Assembly

There are three paths to building data infrastructure in a factory, and each hits a different wall.

WALL 1

Tag Historian

Storage and retrieval are excellent, but **events · commands · BLOBs · relationships** are kept outside. Long-term retention means "exporting to an external lake" — two copies of data, two copies of schema. Because tags are the billing unit, there is an incentive to "collect less."

WALL 2

General-Purpose Cloud Lakehouse

It handles SQL over Parquet well, but **tens of thousands of time-series ingestions per second · gap-fill · time-weighted averages · asset hierarchy** are the app's job. With usage-based billing, cost grows "the more you analyze," and it cannot enter closed networks.

WALL 3

DIY Open-Source Assembly

A PoC wiring Kafka + TSDB + Spark + a broker is fast. What remains after three years is **version combinations · configuration drift · operations knowledge tied to people**. With no standard model, schemas diverge across sites.

What Industrial Data Requires — Six Requirements

Requirement	Why General-Purpose Fails	PlantPulse DataLake's Answer (Chapter)
Tens of thousands per second, years of retention	Row-based storage grows disk linearly, and deletes·TTL become compaction load	Time-window compaction + columnar chunks 7.1× reduction, whole-window expiration (06)
Time-series computation inside the storage layer	If the app pulls raw rows and computes, network and memory become bottlenecks	21 CQL functions · gap-fill · ~30 TSE aggregators (06 · 08)
Asset context (site→line→asset→tag)	General-purpose schemas have no hierarchy, and mappings differ per app	Static column hierarchy · site_id partition · shared asset model (07)
Events · commands · images in the same place	TSDB handles only numbers, object storage is separate	Aster TSDB + RustFS in one cluster, BLOB archiver (10 · 11)
Closed network · complete on-premise	Even with a "Private" option, the app and management plane remain in the cloud	Single image · signature verification · offline bundle (04)
Operations without a dedicated team	Distributed DBs usually require a DBA team	12 console screens · `pd` · `--json` · unmeasured≠normal (14)

How this white paper "boasts." Instead of adjectives, we include configuration keys · schemas · command outputs · benchmark tables. We only write what a reviewer can **reproduce** on the same repository and the same instance.

02 · Design Principles

Five Design Principles

The principles below are enforced by code and tests, not slogans. Next to each principle, "Where it is enforced" is noted.

PRINCIPLE 1

One image, one startup order

16 components come from a single `pd/plantpulse-datalake` image, and the authoritative startup order is a single file: `services/order.txt`. "Do not define the order anywhere else" is the comment on the first line of the file. Storage and analytics act as gatekeeper services — if they don't come up, the system stops with exit code 5.

PRINCIPLE 2

Configuration from templates, rendered on every startup

27 configuration templates (`config/templates/*.template`) are rendered to runtime paths according to `render.map`. If any unsubstituted `${PP_*}` remains, the system rejects with exit code 4 — no startup with half-rendered config. Manual edits inside a container disappear on the next startup (drift 0).

PRINCIPLE 3

Open formats, standard interfaces

The warm tier uses an upstream-compatible Cassandra 6.0 on-disk format (Aster TSDB only adds "opt-in" features), and the cold tier uses Iceberg v2 + Parquet (zstd). Access is via JDBC (Aster TSDB, Spark SQL, TQL drivers), REST, MQTT, Kafka, and Prometheus remote_write. No tier requires a proprietary client.

PRINCIPLE 4

A node is the unit of scaling

Start a worker container and run `pd node add` — it raises the replication factor to match the number of live nodes and performs a repair. If it cannot read the datacenter name or node count, it "exits with 1 rather than guessing" — because a wrong ALTER could silently remove replication.

PRINCIPLE 5

Unmeasured is not normal

Every check item in the `pd --json` document has three states: `pass`, `fail`, or `na`, and `na` must always be accompanied by a `reason`. The console leaves unmeasured values in gray and does not fill them with 0 or green. "A parser that fails gracefully is exactly the flaw that draws unmeasured values as normal" (json-contract.md) — this statement runs through the console, CLI, and tests (`test-json-output.sh`).

Operational outcomes of the principles

Situation	Typical assembly stack	PlantPulse DataLake
Upgrade	Version-combination verification per component, configuration merging	Replace image tag → `pd start`. Config is re-rendered from templates
Add node	DB-specific procedures, manual RF changes, repair schedules	`pd node add` once (RF auto, repair included)
Failure diagnosis	Different thresholds per dashboard, empty graphs look normal	16 components × thresholds, 24h timeline, `na` in gray
Audit	Hard to trace which commit went to which site	Image label `io.kopens.dist-inputs`, build commit, console "Upgrade" screen

03 · Architecture

Deployment · Ports · Startup Order

A single image bootsup as either **MASTER** or **WORKER** mode. The master starts all services, while the worker starts only the storage, analytics, and admin APIs. The startup order is exactly as defined in `services/order.txt`.

Service	MASTER	WORKER	Components (startup order)	Ports
storage	●	●	valkey → postgres → cassandra → minio(MASTER only)	REDIS 6379 · POSTGRESQL 5432 · CASSANDRA 9042 · MINIO 9000
analytics	●	●	spark-master → hive → gravitino → kyuubi	SPARK-MASTER 7077 · SPARK-UI 4440 · HIVE 9083 · KYUUBI 10000 · GRAVITINO 19001
messaging	●	–	kafka(KRaft), mqtt(Aster MQTT)	KAFKA 9092 · MQTT 1883
timeseries	●	–	engine(TSE), dashboard(Grafana)	ENGINE 7800/7801 · UI 3000
cep	●	–	Tomcat (CEP engine 13 jar + war)	CEP 7400
workflow	●	–	temporal → kestra → warehouse worker	ENGINE 7233 · ENGINE-UI 8233 · BATCH 8380
data-gateway	●	–	Tomcat	DATA_GATEWAY 5500
sql	●	–	Tomcat (SQL console)	SQL 4000
admin-api	●	●	Ops console API + console static bundle	ADMIN_API 4949 · Console 4950

Gatekeeper: the `health.sh` of `storage` and `analytics` must pass before advancing to the next service (a port of the old Java launcher's "datalake ready" check to shell). The reason the timeseries engine is MASTER-only is documented: running an engine without query federation on a worker made it a "4.4 GB service copying rows the master had already written back into the same ring" (decided after measurements on 2026-09-05).

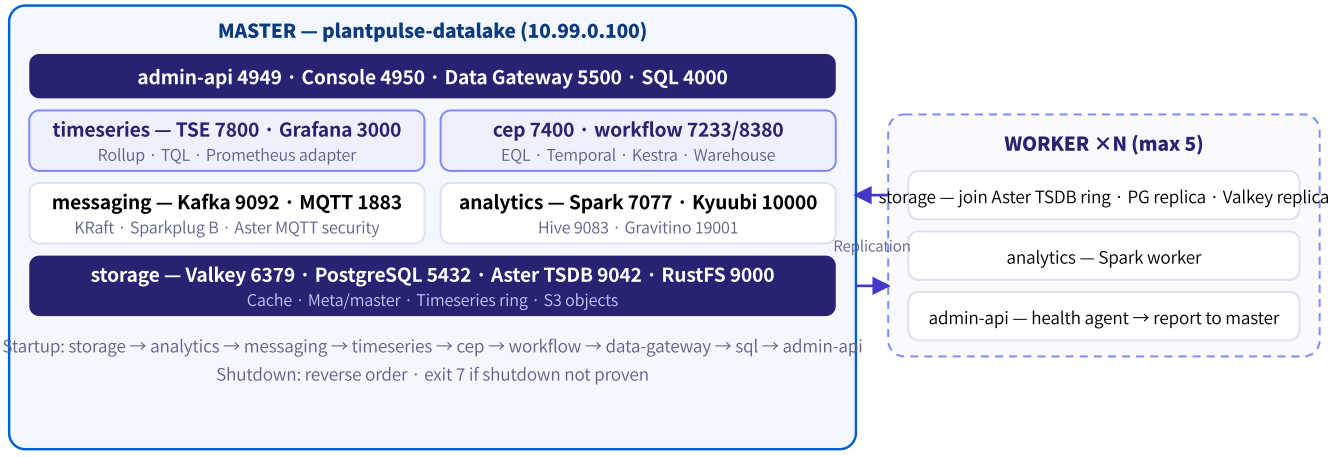


Figure 1. Master · worker deployment and ports. A worker runs only storage, analytics, and admin-api, joining the master's Aster TSDB ring, PostgreSQL streaming replication, and Valkey replica (Chapter 13).

04 · Images

Images — Four Sources, Verification Set, and One Render

The data lake image is assembled by `plantpulse-datalake-docker`. What comes from where is declared in files, and verified again inside the baked image.

Four Sources — Each has a Single Source of Truth

Source	Carried Items	Declaration File
tree/ (git)	Upstream distribution — Kafka · Aster TSDB · Tomcat · Spark · Temporal … 2026-09-01 snapshot	`modules.txt` (10 modules)
Nexus	PlantPulse artifacts — 3 wars · `-lib.zip` · fixed jars · console zip. If not fetched, **the build fails** (no skip flag)	`nexus-overlay.txt`
src/ (module git)	Scripts · templates · systemd units — full `pd`, storage starter, time-series launcher	`src-overlay.txt`
dist (module CI artifacts)	Distributions of source-only modules — `plantpulse-backup`. If the artifact commit ≠ the cloned main HEAD, **rejected**	`dist-inputs.txt`

Verification Set

- `sha256sum -c` **inside the image** — 173 src overlay files and the dist manifest are checked against the source after baking. If they differ, CI won't push the tag.
- **Signature · checksum** — cosign signature on the image; closed-network bundles are verified by checksum + signature before incoming. If verification fails, installation stops.
- **Source labels** — `io.kopens.dist-inputs` (e.g., `plantpulse-backup=<sha>`), build commit, and bake time are displayed on the console "Upgrade" screen.

Configuration Render — 27 Templates, One render.map

```
# config/render.map — <template>|<target relative to PP_HOME>|<modes>|<cluster>
cassandra.yaml|plantpulse-storage/db/cassandra/conf/cassandra.yaml|MASTER,WORKER|cluster
jvm-server.options|plantpulse-storage/db/cassandra/conf/jvm-server.options|MASTER,WORKER|cluster
postgresql.conf|plantpulse-storage/db/postgres/conf/postgresql.conf|MASTER,WORKER|cluster
spark-defaults.conf|plantpulse-analytics/spark/conf/spark-defaults.conf|MASTER,WORKER|cluster
kafka.properties|plantpulse-messaging/kafka/config/kafka.properties|MASTER|-
gravitino-iceberg-rest-server.conf|plantpulse-analytics/gravitino/conf/...|MASTER|-
```

WORKER renders the same-named template from `config/templates/cluster/` (the `cluster` column). Rendering happens on every `pd start`; if an advertised address is loopback (`127.0.0.1`), it's rejected with exit 8 — this prevents the broker from advertising its own address as 127.0.0.1 and becoming unreachable from outside.

Where Operators Make Changes — One Place

The host's `/etc/kopens/conf` is mounted over `$PP_HOME/plantpulse-datalake-cli/config/templates`. Templates edited here are reflected in the next render, and `pd config diff` shows the difference between "settings currently running" and "settings to be rendered" (exit 1 if different).

05 · Data Lifecycle

Hot · Warm · Cold — How Data Ages

The same tag IDs and asset models span all three tiers. Movement between tiers is handled by the scheduler (Temporal) and Aster TSDB tiering; humans only set the retention period.



Warm Tier Retention — Factory Default (`plantpulse-storage.properties`)

Table	TTL	Purpose
tm_asset_data (real-time asset snapshots)	10 days	Current-value series read by screens and rules
tm_tag_point (raw tag points)	**3650 days**	10 years of raw data for immediate query — the largest storage axis, so sites usually reduce it
tm_tag_point_aggregation_* · sampling_* · snapshot	93 days	1 min to 1 hour aggregations · 10 sec to 1 hour sampling · snapshots
tm_tag_point_archive	1 year	Buffer for reading directly from the ring even after moving to cold
tm_system_log · tm_monitor_* · tm_tag_point_validation	10 days · 1 day · 1 day	Operational diagnostics
tm_batch_job_run · tm_batch_audit · tm_warehouse_audit	31 days	Batch execution history · audits

In-Engine Tiering — Aster TSDB tiering policy (Chapter 06)

Aging happens "within" the warm tier as well. Per-table JSON policy `hot_window` (period kept as rows, e.g., 7d) → the re-encoder converts the past window into columnar chunks at `<table>__chunks` every `interval` (e.g., 1h) → chunks are dropped entirely when `cold_window` expires. **SELECT does not change** — querying the base table automatically merges hot rows and chunks. During chunking, cell TTLs drop, so for tables using `default_time_to_live`, the rule is to set `cold_window` to the same value.

06 · Own Time-Series DB

Aster TSDB — An Industrial Time-Series Database Built by KOPENS

The storage engine for the warm tier is **Aster TSDB** (Aster Timeseries Database, `github.com/kopens/aster-tsdb`, Apache-2.0), developed and maintained directly by KOPENS. Built on Apache Cassandra 6.0, it incorporates **compression, retention, and time-series computation into the database**. The on-disk format and CQL syntax remain upstream, so existing 6.0 data is read as-is, and all time-series features are **opt-in per table**.

+54k
lines — engine 26k · tests 28k (169 files vs. upstream)

7.1×
storage reduction — 11.9 → 1.7 B/row (measured on 20M rows)

21
server-side CQL time-series functions · gap-fill · regression

93 + 49
release-gate integration assertions · 3-node cluster assertions

COMPACTION

TimeSeriesCompactionStrategy (TSCS)

Window-ordered sorting · delegates to UCS within the current window · freezes past windows into **a single SSTable** · windows past the retention boundary are **dropped whole** without compaction. Late data pushed by edge devices (backfill) is isolated into its own past window at flush/streaming boundaries.

CHUNK FORMAT V4

Columnar Chunk Codec

One window = shared timestamp axis + independent section per column. `double` uses ALP/ALP-RD, integers use FOR/delta bit-packing, boolean 1-bit, text uses dictionary/raw. Unchanging columns are `CONSTANT`, all-null columns are `ALL_NULL` — O(1). Codecs were chosen via **measured comparison** against Gorilla and Chimp128.

TIERED STORAGE

Transparent Reads — SELECT Is Unchanged

The re-encoder moves windows past the `hot_window` into chunks, and base-table `SELECT` **automatically merges** hot rows and chunks — for range, point lookups, aggregations, gap-fill, and all `LIMIT` / `DESC` cases. Operated via `nodetool retier` and virtual tables.

MEMTABLE · SEARCH

TimeSeriesMemtable · SAI Full-Text Search

The base data structure is array-based, cutting heap per row **5.5×** (511 → 92 B), with window assignment at write time. SAI `LIKE` + `index_analyzer` enables **Korean substring search** without `ALLOW FILTERING`.

21 Server-Side CQL Time-Series Functions — Eliminating the Round-Trip of Apps Pulling Raw Rows and Computing

`time_bucket` · `first` / `last` · `delta` / `rate` / `derivative` · `counter_delta` / `counter_rate` (reset-aware) · `percentile` · `time_weighted_average` · `integral` · `variance` / `stddev` · `histogram` · `approx_count_distinct` · bivariate regression (`regr_slope`, `regr_r2` …) · gap-fill `time_bucket_gapfill` + `locf()` / `interpolate()`

```
SELECT time_bucket_gapfill(1h, timestamp, '2026-07-01', '2026-07-02'), locf(avg(latency))
  FROM pp.tm_tag_point WHERE tag_id='TAG-001' AND timestamp >= '2026-07-01' AND timestamp < '2026-07-02'
GROUP BY tag_id, time_bucket_gapfill(...) ORDER BY timestamp ASC;   -- fill empty buckets with the last observation
```

Benchmark — Operational Schema with 20M Rows, Tiering Off vs. On (host 234 · Xeon Silver 4114T · 2026-08-04)

Metric	Before Tiering	After Tiering (v4)	Effect
Storage footprint	237.8 MB	33.3 MB	**7.1× reduction**
count(*) (40k-row partition)	303 ms	50 ms	**6.1×**
time_bucket aggregation	150–270 ms	31–64 ms	**3–6×**
gap-fill (locf / interpolate)	162–284 ms	30–53 ms	**5.4×**
90 tags p95 · hourly average (3.6M rows)	14.2 s · 14.9 s	2.6 s · 4.2 s	**5.4× · 3.5×**
Throughput	ingest **233k rows/s**(single node · 12 loaders × 100-row batches) · chunk re-encoding **108k rows/s** · full chunk scan 740 µs / 3,600 rows		

All measured queries were equal or faster after tiering. Drop-in · Upstream-tracked. Build artifacts keep upstream filenames, so swapping the jar is the entirety of installation, and identity is declared in the manifest (`Implementation-Title: Aster TSDB` · `Aster-Upstream-SHA`) — CI fails if it is missing. **Constraints are documented too:** full-scan aggregations without a partition key are not used on tiered tables, `DELETE` / `SET null` on chunked data is rejected (removal only via `cold_window` expiry), `counter` and non-frozen collections are unsupported. Throughput is for RF 1 (lower under RF 3).

07 · Schema

Keyspace · Partition · Compression · Index

The schema canonical form is `plantpulse-database-model` — Aster TSDB DDL(**104 tables**) · PostgreSQL DDL(43 tables, Flyway) · Iceberg DDL(Spark SQL). With polyglot persistence, storage differs by data nature.

Keyspace and node configuration

```
CREATE KEYSPACE pp WITH replication = {
  'class': 'NetworkTopologyStrategy', 'datacenter1': 3 };

# cassandra.yaml.template (after rendering)
cluster_name: 'PP_CLUSTER'
num_tokens: 16
allocate_tokens_for_local_replication_factor: 3
endpoint_snitch: GossipingPropertyFileSnitch
authenticator: PasswordAuthenticator
authorizer: CassandraAuthorizer
concurrent_reads: 256 / concurrent_writes: 256
memtable_heap_space: 2GiB / memtable_offheap_space: 2GiB
memtable_allocation_type: offheap_objects
memtable_flush_writers: 8
compaction_throughput: 320MiB/s
commitlog_sync: periodic (10000ms)
storage_compatibility_mode: NONE
# jvm-server.options.template
-Xms16G -Xmx16G -XX:+UseZGC
```

Table prefixes

Prefix	Purpose
tm_asset_*	Asset-level time series (second · minute · hour · day rollup)
tm_tag_*	Tag points · aggregation · sampling · count · alarm
tm_opc_* · tm_site_*	Connection status · site summary
tm_system_* · tm_monitor_*	System logs · cluster metrics
ts_*	Time-series engine (TSE) storage
tw_*	Iceberg archive targets (cold)

Partition key strategy

Pattern	Partition key	Purpose
High frequency	(id, date, hour)	Real-time sensors
Medium frequency	(id, date)	Minute-level aggregation
Low frequency	(id)	Hourly · daily aggregation
Archive	(site_id, year, month, day, hour)	Long-term retention
Audit	(yyyymmdd) + timestamp + id	append-only

Hierarchy metadata that changes infrequently (`site_id · area_id · line_id`) is stored as `static` columns — once per partition.

Rollup family — raw → aggregate → sampling

Table group	Interval
tm_tag_point_aggregation_*	1 · 5 · 10 · 30 min · 1 hr — avg min max sum count stddev median first last
tm_tag_point_sampling_*	10 · 30 sec · 1 · 5 · 10 · 30 min · 1 hr
tm_asset_data_based_*	timestamp · second · minute · hour · day

Compaction · compression · indexes (from ddl.cql)

- **UnifiedCompactionStrategy** two profiles — high frequency: `T8 · base_shard 8 · target 512MiB · min 128MiB` , normal: `T4 · base_shard 4 · target 1GiB` , expired SSTable check 300s
- **Compression** — 16 hot tables `LZ4 (4 KB chunk)` , 8 cold-natured `Zstd level 2–3 (16–64 KB chunk)`
- **SAI(Storage Attached Index)** 54 — secondary condition search without `ALLOW FILTERING`

08 · Time-series Engine

TSE — Aggregation · Rollup · TQL · Prometheus in a single process

The PlantPulse TimeSeries Engine is a proprietary time-series engine in the KairosDB family. It uses Aster TSDB as its backend and offers **REST · TQL/JDBC · Grafana datasource · Prometheus compatibility** — four faces in one JVM.

Runtime

Language · GC	Java 21 · generational ZGC (`-XX:+UseZGC -XX:+ZGenerational`)
Server	Jetty 9.4 — HTTP 7800 / HTTPS 7801 simultaneously
Ingest	File-based **persistent queue (bigqueue)** → batch 200 · memory queue 100,000
Authentication	JAAS file/LDAP · Basic Auth · SSL/TLS · OAuth (optional)
Storage	Aster TSDB (java-driver 4.19) · H2 (development) · remote datastore transfer
UI	React 19 time-series viewer — no CDN, self-hosted fonts (air-gap)
Client	Java SDK — **store-and-forward** buffering, connection pool 1024

Aggregators — ~30 types (measured at source)

Avg · Sum · Min · Max · Count · First · Last · Percentile · Std · Range · Rate · Diff · Divide · Scale · Sma · Ema · Integral · LeastSquares (regression) · Sampler · Filter · Trim · Limit · Drop · **Lttb** (downsampling) · **Hampel** (outlier filter) · **Anomaly** · Score · DataGapsMarking · SaveAs, etc. Rollup definitions are stored and auto-executed via `/api/v1/rollups`.

Prometheus-compatible adapter

In-process inside TSE's Jetty — receives `remote_write` (snappy+protobuf 1.0) without a separate Prometheus server, and Grafana connects as a "Prometheus datasource" to use **PromQL** directly. Lossless label↔tag mapping, same Basic Auth on `/prometheus/*` paths.

TQL — time-series queries for anyone who knows SQL

```
SELECT AVG('1 MINUTES') >> MAX('1 DAYS')     -- aggregation cha
ining
  FROM CHANNEL1.PUMP1.TEMP_0001
 WHERE timestamp WITHIN '7 DAYS'
   AND tag['site_id'] IN ('SITE_A','SITE_B')
   AND value > 80
 GROUP BY _HOST, MM:SITE_ID
 ORDER BY timestamp DESC LIMIT 1000;

SELECT PERCENTILE('1 HOURS','START','0.95')
  FROM M WHERE timestamp BETWEEN ? AND ?;   -- PreparedStatemen
t
```

Time conditions	**WITHIN '30 MINUTES' · SINCE '2026-01-01 00:00:00.000' · BETWEEN … AND …**
Aggregation	AVG MIN MAX SUM COUNT FIRST LAST DEV PERCENTILE GAPS LEAST_SQUARES
Transformation	RATE DIFF SAMPLER DIV SCALE FILTER TRIM
Other	INSERT … TAGS · DELETE · DESCRIBE · METRICS · ROLLUPS · PING
JDBC	`jdbc:ts://host:7800` · `jdbc:tss://` (TLS) · SPI auto-registration · result is a single JSON column

```
Connection c = DriverManager.getConnection(
    "jdbc:ts://192.168.0.41:7800", "tse", "...");
ResultSet rs = c.createStatement().executeQuery(
    "SELECT AVG('1 MINUTES') FROM CHANNEL1.PUMP1.TEMP_0001 "
  + "WHERE timestamp WITHIN '1 HOURS'");
while (rs.next()) json = rs.getString(1);
// {"metric":..,"timestamp":..,"value":..,"tags":{..}}
```

Why two time-series layers. The storage engine (Aster TSDB) handles bucketing, gap-filling, and regression **within CQL**, while TSE handles rollups, anomaly detection, downsampling, and Prometheus compatibility **on top of the metric API**. Raw data is stored only once per ring.

09 · Time-Series DB Comparison

On the same basis as InfluxDB · TimescaleDB

Both are excellent time-series DBs. The difference becomes apparent when positioned as the "warm tier" of an industrial data infrastructure. Competitor product details are based on public documents dated 2026-09-09 (source p.20).

Item	PlantPulse (Aster TSDB + TSE)	InfluxDB 3 (Core / Enterprise)	TimescaleDB (PostgreSQL extension)
Storage Model	Aster TSDB (proprietary engine based on Cassandra 6.0) — transparent merge of rows (hot) + columnar chunks (warm), token ring distribution	Arrow · DataFusion · Parquet columnar, object storage (S3)-based "diskless"	PostgreSQL hypertables · chunks, columnar compression
Scaling · HA	● Token ring · RF 3 · Node addition = linear, max 5 workers, automatic `pd node add`	◐ **Core is single-node**. Multi-node (separate ingest · query · compactor) is **Enterprise**	◐ **Multi-node support ended in 2.14** (2.13 is the last). Vertical scaling + PG streaming replicas
Compression	● Chunk codec v4 (ALP · FOR/delta) **7.1×** (11.9 → 1.7 B/row, measured on 20M rows) + SSTable LZ4/Zstd	● Parquet columnar (numbers not disclosed)	● Columnar compression "90–95% reduction" (public source)
Long-term Query · Retention	● Table-level TTL, whole-window expiry, cold tier auto-archives to Iceberg — no query range limit	◐ **Core has a 72-hour query range limit** (432 Parquet files = 10-min blocks × 72h). Lifted by a compactor only in Enterprise	● Retention policy · chunk drop, tiered storage is Timescale Cloud
Time-series Functions	● 21 CQL functions (gap-fill · regression · time-weighted average · counters) + ~30 TSE aggregators (LTTB · Hampel · outlier)	● SQL · InfluxQL (Flux excluded from 3.x)	● `time_bucket` · continuous aggregates (auto-refreshing MV) · gap-fill · hyperfunctions
Equipment Context · Events · BLOB	● Asset hierarchy as static columns · alarms · events · commands · BLOBs in the same ring/object storage	○ Numeric · string time-series, objects stored separately	◐ Relational, so events are possible, BLOBs recommended outside PG
Ingestion Standards	● Kafka · MQTT/Sparkplug B · REST · Prometheus remote_write · 42 industrial protocols via Edge	◐ Telegraf (plugins) · line protocol	◐ PG client · external ingesters
License	Aster TSDB Apache-2.0 (open) + product license (based on configuration)	Core: MIT/Apache-2 · Enterprise: commercial (free for home use exception)	Apache-2 core + **TSL** (compression · continuous aggregates · retention under community license, managed service provision prohibited)
Operations	● Console 12 screens · `pd` · backup (Medusa) · threshold-based health · 24h timeline	◐ CLI · UI (Enterprise), compactor operations	● Same PostgreSQL operational tools (pgBackRest, etc.)

When to Choose PlantPulse

When you need linear scaling of multi-year retention and tens of thousands of tags **within one ring**, when events · commands · images must sit beside the time-series, and when you need completion from console to closed network. When you want to run time-series functions inside the storage to keep applications light.

When Influx · Timescale Fits

For single-node metrics · monitoring (Influx Core), or for teams already centered on PostgreSQL where relational joins are key (Timescale). Both products can integrate with PlantPulse via **Kafka · Prometheus remote_write · JDBC**.

10 · Lakehouse

Iceberg · Spark · Archive — The Actual Shape of the Cold Tier

Table Definition (spark/ddl.sql Excerpt)

```
CREATE TABLE spark.plantpulse.tm_tag_point_warehouse (
  tag_id STRING, tag_name STRING, value STRING, quality INT,
  site_id STRING, area_id STRING, line_id STRING, asset_id STRI
NG,
  opc_id STRING, timestamp TIMESTAMP, tuuid STRING, type STRIN
G,
  attribute MAP<STRING,STRING>, error_code INT, latency INT,
  year INT, month INT, day INT, hour INT, minute INT, second IN
T)
USING iceberg
PARTITIONED BY (site_id, year, month, day)
TBLPROPERTIES (
  'format-version'='2',                          -- row-level del
ete
  'write.target-file-size-bytes'='268435456',  -- 256 MB
  'write.data.compression-codec'='zstd',
  'write.metadata.previous-versions-max'='5',
  'write.metadata.delete-after-commit.enabled'='true',
  'read.split.target-size'='134217728');       -- 128 MB
```

Why Iceberg

- **ACID** — No half-written files visible during archive
- **Schema & Partition Evolution** — No rewriting of existing files when columns are added/changed
- **Time Travel** — Re-query the table as of a snapshot · **Hidden Partitioning** — Partition columns unnecessary in queries
- **Engine Neutral** — Spark, Kyuubi, and other Iceberg readers like Trino and Flink read it directly

Analytics Tier Versions (UPSTREAM-LOCK Field Verification)

Spark	4.1.3 (Scala 2.13) — Generational ZGC, `spark.sql.shuffle.partitions=128`, executor 8 cores
Kyuubi	1.12.0 — Multi-user SQL serving · JDBC 10000
Gravitino	1.3.0 — Iceberg REST catalog 19001
Hive · Hadoop	2.3.9 metastore 9083 · 3.3.4
RustFS	1.0.0 — S3 API 9000 · Console 9001 (Apache-2.0)

8 Archive Jobs — SPI Plugin (plantpulse-archive-job)

jobType	Aster TSDB → Iceberg
ARCHIVE_TAG_POINT	tm_tag_point_archive → tw_tag_point
ARCHIVE_ASSET_DATA	tm_asset_data → tw_asset_data
ARCHIVE_ASSET_AGGREGATION	tm_asset_aggregation → tw_asset_aggregation
ARCHIVE_ASSET_ALARM	tm_asset_alarm → tw_asset_alarm
ARCHIVE_ASSET_EVENT	tm_asset_event → tw_asset_event
ARCHIVE_ASSET_CONTEXT	tm_asset_context → tw_asset_context
ARCHIVE_ASSET_COMMAND	tm_asset_command → tw_asset_command
ARCHIVE_BLOB	tm_blob / tm_blob_object → tw_blob + S3 original

```
// plantpulse.warehouse.s3.spi.ArchiveTableJob — discovered vi
a ServiceLoader
String jobType();  String sourceTable();  String targetTable();
long archive(SparkSession spark, ArchiveRequest request);
```

The jar is not baked into the image; it goes in `$PP_HOME/plantpulse-workflow/warehouse/jobs` or `$PP_DATA_DIR/jobs` — sites can add jobs without an image rebuild. If no provider is found, it "fails stating where it looked" (not a 0-row success). Execution log: `[ARCHIVE-JOB] loaded ARCHIVE_ASSET_ALARM ... tm_asset_alarm -> tw_asset_alarm from file:/.../jobs/plantpulse-archive-job.jar`

Schedule — Temporal Workflow

Workflow	Schedule (Asia/Seoul)	Execution
ArchiverWorkflow	Daily 02:20	`archive.sh` → `spark-submit`, 8 jobs
OptimizerWorkflow	Sunday 03:00	`rewrite_data_files` (min 10 files · 128 MB) · `expire_snapshots` (3 days, minimum 1) · `remove_orphan_files`

Task Queue `PP-S3-QUEUE`. The optimization target uses the "tables written by discovered archive jobs" directly — if you kept two lists, you'd silently end up with "tables that archive but don't get optimized".

Kafka · MQTT — The Four Incoming Doors

Kafka 4.3.1 (KRaft — no ZooKeeper)

```
# kafka.properties.template (excerpt)
num.partitions=4
log.retention.ms=3600000          # 1h — a buffer, not a store
log.retention.check.interval.ms=60000
log.segment.bytes=10485760         # 10 MB segments
message.max.bytes=157286400        # 150 MB — BLOBs (images · drawings) pass through
num.network.threads=3 / num.io.threads=8
default.replication.factor=1
```

Kafka is an **ingestion buffer**. The 1-hour retention expresses the design "storage is Aster TSDB's job," and the console "Storage" screen shows Kafka disk and retention separately (measured at 191 MB).

BLOB Transfer Contract (plantpulse-event-client-blob)

```
topic: blob   key = JSON, value = byte[]
{ "_event_type":"BLOB", "blob_id":"BLOB_ID_001",
  "timestamp":1772041960266,
  "attribute":"{\"site_id\":\"SITE_00001\",\"asset_id\":\"ASSET_DJ_M_01_1\"}",
  "_file_path":"/data/images/vision.jpg",
  "_file_size":102400, "_mime_type":"image/jpeg" }
```

Vision images and drawings come in under the same asset ID as tags, then travel one path: `tm_blob` → RustFS → `tw_blob` archive.

MQTT — Aster MQTT

This is the equipment door. On top of standard MQTT, we bind connections to platform accounts with our own authentication/ACL, and mirror the asset tree into topics via Sparkplug B (UNS).

What operations fixed — 🔴 **startup is refused when no authenticator is present** (if the security extension did not load, the broker looked "normal" while **left open**). **Seven counters per drop reason** separate capacity · backpressure · policy, and we closed **15 public vulnerabilities** in the network library.

Why Two Buffers

Buffer	Role
Kafka	Bulk ingestion · BLOB · external DB sync. Partition-level parallelism, 1-hour disk retention
Valkey stream	**Events for real-time detection**. Prefix `PP:CEP-IN-STREAM`, **16 shards**, read by explicit position (stream ID)

They do not compete — the storage path selects Kafka or Redis via `batch.data.source`, and real-time detection (Chapter 12) reads the Valkey stream.

Ingestion Path Summary

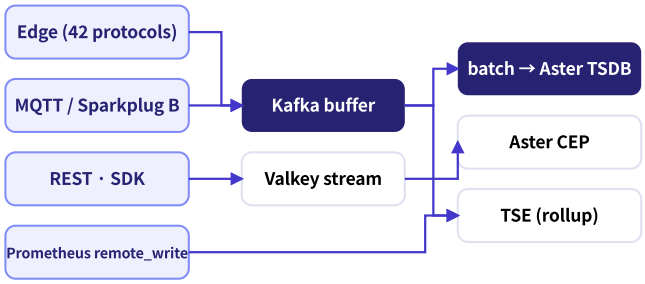


Figure 2. Four doors, two buffers, three consumers. Storage (batch-web → Aster TSDB) can select Kafka or Redis via `batch.data.source`, and external DB sync (TimescaleDB · InfluxDB · QuestDB · M3DB) is handled by the same consumer.

> **Retention is not the buffer's job.** Kafka's 1 hour and the Valkey stream's max length are headroom for a "consumer that fell behind," not an archive. Permanent retention is handled by the Aster TSDB ring and Iceberg (Chapter 10).

The Real-Time Buffer Contract — No Loss, But Possible Duplicates

at-least-once

There is no way to atomically couple "moving the position" with "committing the checkpoint." Move it **after** the checkpoint and a crash in between redelivers; move it **before** and you lose data. **We chose the former** — rather than losing an event, the events between the last checkpoint and the failure may arrive twice. This is the same ceiling Kafka consumers have without transactions.

The client holds the position

We read by **explicit stream ID**, not the server-side list of a consumer group. If the server held the position, a checkpoint could not restore to it. An explicit ID is client-side state, so **it checkpoints exactly like a Kafka offset**.

A truncated gap is "reported"

Streams are truncated at max length, so if a consumer has been down a long time, it returns to a position **older than anything that remains**. Reading from there yields items after the gap **with no error at all**. So instead of resuming silently, we **count the gap and report it** — if it is not zero, those items disappeared before any consumer could see them.

The output side has a contract too. Detection results go to one topic per statement, `PP:CEP-OUT-TOPIC:<name>`, and **an empty flush publishes nothing** — an empty array asserts "matched and there are no rows," and when there is nothing, the honest message is silence. Publishing succeeds even with zero subscribers, so that is where results silently vanish — hence the `published with no subscriber` warning.

12 · Real-Time Determination

Aster CEP — Real-Time Determination Engine

Asking after storing and determining while flowing are different problems. **CEP (Complex Event Processing)** checks time · logic conditions while events pass and emits alarms · triggers. Rules are written in **EQL**, and the determination is done **in-process** by KOPENS' **Aster CEP** engine within the CEP server (`:7400`) — no separate streaming cluster required.

0
Separate streaming cluster — determination happens "inside" the CEP server

2,127
Engine tests + 2,501 for the CEP layer

152
Golden EQL corpus — automatic generation of per-engine capability matrix

SQL:2016
`MATCH_RECOGNIZE` consecutive quantifier — implemented directly in the engine

Structure — language first, engine executes it

Language	**EQL 1.0** — SQL-92 core + streaming extensions (`WINDOW` · `OUTPUT ... EVERY` · patterns · absence patterns · `MATCH_RECOGNIZE`)
Intermediate representation	Engine-independent common IR · determination set `SUPPORTED` · `UNSUPPORTED` · `UNPARSEABLE`
Execution engine	**Aster CEP** — one statement = one app inside the CEP server
Oracle	Apache Flink 2.3.0 — used **solely for cross-checking** answers to the same statements. Not used in operations
Input · output	Valkey stream 16 shards → per-statement topic `PP:CEP-OUT-TOPIC:<name>`

What it means that the engine is ours

Aster CEP is an engine that KOPENS **develops and maintains at the source level**. Four modules (`aster-core` · `query-api` · `query-compiler` · `annotations`), package `plantpulse.cep.aster.*` , released together with the platform on the same calendar version (`2026.09`). **Syntax and defect fixes required on-site are incorporated without waiting for someone else's roadmap** — the row pattern recognition below was added that way.

Rules for modifying the engine — CI gates

- **Separate layers first.** What can be done with functions is layered on as a classpath extension; only **what requires changing the syntax or state machine** goes into the engine core — the fewer engine changes, the fewer regressions
- **Public API gate** — the build fails if the class set or public API changes without notice. Adapters on top of the engine don't silently break
- **Regression gate** — engine tests **2,127** and the golden EQL corpus run with every build

Unsupported configurations are "refused by name". If you skip and deploy, statements end up **running · exporting · wrong** — with nothing showing red. The engine is one per boot and **there is no fallback** — if both run together, the same input is published twice.

Row pattern recognition — SQL:2016 `MATCH_RECOGNIZE`

Determining a "start–end interval" of equipment requires **all** rows "in between" to satisfy the condition. That is, the **consecutive** quantifier `A{2,5}` is needed, and repetition that allows gaps is a different set. This cannot be emulated with functions or plugins, so **syntax and state machine were implemented directly in the engine** — something possible only because the engine is ours.

```
-- Golden corpus 230: latency exceeding 100 for 3 "consecutive" rows
SELECT tag, n FROM Point MATCH_RECOGNIZE (
  PARTITION BY tag_id
  MEASURES first(h.tag_id) AS tag, count(h.latency) AS n
  AFTER MATCH SKIP TO NEXT ROW
  PATTERN (h{3})
  DEFINE h AS h.latency > 100
)
```

Does	Refuses by name
Quantifiers `* + ? {n} {n,m} {n,}` · greedy/reluctant · `DEFINE` · `MEASURES FIRST/LAST/COUNT` · `ONE ROW PER MATCH` · `SKIP TO NEXT ROW` · `PARTITION BY`	`ALL ROWS PER MATCH` · `SKIP PAST LAST ROW` · alternation `|` · `PERMUTE` · arithmetic in MEASURES

Proof — answers cross-checked against "another implementation"

Fixture	Aster CEP	Flink 2.3.0
interval `s m* e`	1 match (1,5,2,4,3)	same
greedy · reluctant `m*?`	mid 2 · mid 0	same
overlapping inputs · `{1,2}`	4 matches · 2 matches	same
greedy requiring backtracking	**1 match (SQL:2016)**	0 matches

The last row is a **measured divergence** from the oracle, and the side following the standard is Aster CEP. Fixtures were chosen by first confirming with the oracle where the modes **actually diverge**.

Capability is not a declaration but a "product"

Each golden EQL case (152) is fed into each engine's actual compiler to **generate** the capability matrix, and CI compares it against the committed version. Of the 127 cases that reach the IR, **Aster CEP 106** · Esper 78 · Flink 53. The 21 that are not supported are all named with **rejection tokens** like `pattern:until` . The differential harness has three verdicts — `AGREEMENT` · `DIVERGENCE` · `NOTHING_COMPARED` — and if there is nothing to compare, it is a failure, not a pass.

13 · Query Tier

Data Gateway — Four Languages, One REST

Analysts, external systems, BI, and AI agents do not connect directly to the store. **Data Gateway (port :5500 · TLS :5501)** receives the four languages at a single endpoint, blocks statements that alter the schema, limits results, and **logs every query**.

Four Languages, Four Stores

Language	Connects To	What It Sees
PQL	PostgreSQL	Metadata — assets · tags · users · settings
CQL	Aster TSDB	Raw sensor readings · aggregations · BLOB metadata
SQL	Kyuubi (Spark)	Iceberg cold tier · warehouse
TQL	Time-series Engine (TSE)	Metrics · rollups

Endpoints

POST /api/v1/query	Full result + `truncated` flag (limit 100,000 rows)
POST /api/v1/query/stream	NDJSON — one line per row, last line is `end`
POST /api/v1/update	Updates (TQL is rejected)
POST /api/v1/format	Query formatting · indentation
POST /api/v1/cache	In-memory current-value lookup
/api/v1/keys	API key issuance · listing · revocation
GET /api/health	Readiness (no auth required) — DB · in-memory · query log
GET /admin/*	Operational metrics · thread dumps (auth required)

Authentication is determined by a single Spring Security chain and is **fail-closed** — paths not on the list return 401, not anonymity. It creates no session and no cookie (STATELESS).

Why Direct Connections Are Disallowed

- **Audit** — "who queried what and when" is not scattered across stores but recorded in one table
- **Limits** — row count · request size · requests per second are enforced **in one place**, not in different ways per store
- **Blocking** — schema-altering statements are blocked by the same rule regardless of language
- **Replacement** — even if the store's address · account · driver change, **consumer code remains unchanged**

That is why, for cases like BI tools that **must connect directly via JDBC**, a separate driver set (Chapter 14) is provided, and that path is controlled by the store's account system. The purpose of this tier is **not "not opening a door" but "knowing which door was opened."**

Streaming Queries

```
curl -X POST -H "X-API-Key: <key>" -H "X-Lang: PQL" \
  -H "Content-Type: application/json" \
  -d "SELECT tablename FROM pg_tables WHERE schemaname='pp'" \
  https://host:5501/api/v1/query/stream

{"row":{"tablename":"mm_asset"}}
{"row":{"tablename":"mm_tag"}}
{"end":{"count":43,"exec_ms":42,"truncated":false}}
```

The server streams via JDBC cursor without loading the entire result into memory. Authentication completes at the header stage, so **the response always starts with 200** — subsequent failures are visible only in the last line of the body, `{"error":{...}}`.

Parameter Binding

```
X-Params: <URL-encoded JSON>
{ "site_id": "SITE_00001",
  "from:ts-long": 1772041960266,
  "limit:int": 500 }

SELECT * FROM pp.mm_tag
 WHERE site_id = :site_id AND created_at >= :from LIMIT :limit
```

Appending `:type` to the key specifies conversion — `int long float double string epoch ts-long ts-string`. An unknown type suffix results in **400**, not being ignored. A malformed header (`%zz` · not JSON) also returns 400 — if a client error were reported as 500, it would be read as "the server is down."

> **AI agents enter through the same door.** The platform's MCP tools run internally under **the same permissions, the same logs, and the same limits** as this gateway. There is no path that allows an agent more visibility.

One Connection Setting Example (plantpulse-jdbc.properties)

```
jdbc.cassandra.url = jdbc:cassandra://${PP_HOST_IP}:9042/pp
jdbc.postgres.url  = jdbc:postgresql://${PP_HOST_IP}:5432/plantpuls
e
jdbc.spark.url     = jdbc:hive2://${PP_HOST_IP}:10000/
jdbc.tse.url       = jdbc:ts://${PP_HOST_IP}:7800
cache.host         = ${PP_HOST_IP}
```

14 · Control · Audit

Do No Harm — What to Block and What to Keep

Block Statements That Change the Schema

```
DROP · TRUNCATE · ALTER · CREATE · GRANT · REVOKE

/* hello */ DROP TABLE pp.mm_tag   → blocked
SELECT 1; DROP TABLE pp.mm_tag    → blocked (multiple statements)
```

We strip block and line comments **first**, then examine the first token. The first-token-only approach only holds when **one statement per request**, so if a semicolon is visible, we reject **before looking at the content**.

Limits — Along with Values

Setting	Default
query.max.rows	**100,000** rows (0 = unlimited)
rate.limit	**1,000** req/s (per client, fixed window)
rate.limit.global	0 = unlimited — ceiling on burst spikes
max.body.bytes	1 MB — **reject early with 413** on exceed
max.params.bytes	64 KB
async.timeout.ms	60,000
async.concurrency.limit	**256** — virtual threads, backpressure when limit reached
slow.query / slow.update	10,000 ms / 2,000 ms — warning log

> **Truncated results come back as "success".** HTTP is 200 and the rows are valid — if you don't check `truncated`, you may aggregate **as if the truncated result were complete**. That's why we always carry this flag on both paths. And when the limit is reached, we **stop reading right there** — not read to the end and then cut.

`X-Forwarded-For` is ignored by default. We use the first hop as the client IP only when the direct peer is on the trusted proxy list — otherwise, one could fabricate a header to bypass the limit. `/api/*` and `/admin/*` **share the same counter**.

Error Contract — Distinguish 4xx from 5xx

error_code	HTTP	Meaning
AUTH_MISSING / AUTH_INVALID	401	Key missing · mismatch
PARAM_MISSING	400	Header · language · type conversion failure
QUERY_SYNTAX	400	DDL · multiple statements · syntax
QUERY_CONSTRAINT / _DEADLOCK	409	Integrity · deadlock
QUERY_CONNECTION	503	Failed to connect to backend
QUERY_TIMEOUT	504	Backend response delay

```
{"result":"ERROR","error_code":"QUERY_SYNTAX",
 "message":"DDL statements are not allowed: DROP"}
```

Syntax · constraint · data · authentication are **client-side causes (4xx)**, while connection · timeout are **infrastructure causes (5xx)**. If we lump them together, the caller cannot tell "was my query wrong or is the store down". Server logs follow the same split: 4xx as WARN, 5xx as ERROR.

Every Query Is Logged

Table	`tm_data_gateway_log` (Aster TSDB, timeuuid PK)
What remains	Language · query text · row count · execution time · success/failure · error
Write	Queue + batch flush — does not block the query path

`/api/health` is not liveness but **readiness** — `database` · `inmemory` · `query_log` must all be true for UP, otherwise `STARTING` · `DEGRADED`. **A gateway whose audit log is not initialized is not UP.**

JDBC Driver Set — The Path to Connect "Directly" to the Store

Driver	URL · Notes
Aster TSDB JDBC	`jdbc:cassandra://host:9042/pp` — Aster TSDB CQL functions as-is, publishes immutable coordinate `YYYY.MM-YYYYMMDD`
Spark SQL JDBC	Kyuubi Thrift(10000) — thin + standalone jar, Iceberg cold tier in BI tools
TQL JDBC	`jdbc:ts://host:7800` · `jdbc:tss://` — SPI auto-registration

All three drivers are SDKs that the customer compiles, so they publish **both the moving pointer (YYYY.MM) and the immutable coordinate**, and the public API must pass `abiCheck` (ABI comparison against the previous release) before it ships. For Java applications, there is a separate gateway SDK (`plantpulse-data-gateway-client`).

15 · 확장 · 복제

토큰 링 — 워커를 꽂고 `pd node add`



그림 3. 콘솔 "토폴로지" — 마스터 10.99.0.100 + 워커 10.99.0.101, 빈 슬롯 4. 노드 상세: 링 정상 · datacenter1/rack1 · 부하 8.37 GiB · 토큰 16 · 키스페이스 `pp` 유효 소유 48.8%. *2026-09-08 실측.*

절차 — 셋

1. 워커 컨테이너 기동(`PP_MODE=WORKER` , 같은 이미지). `pd status` 로 storage · analytics · admin-api RUNNING 확인.
2. 마스터에서 `pd node status` — 새 노드가 `UN` (Up·Normal). `UJ` 면 아직 합류 중.
3. 마스터에서 `pd node add` — 살아 있는 노드 수에 맞춰 RF 조정 + `nodetool repair` . 셋째 노드가 붙으면 한 번 더.

```
$ pd node add
[INFO] [node] cassandra: change replication factor + repair ...
[INFO] [node] datacenter=datacenter1 live_nodes=2 -> replicatio
n_factor=2
[WARN] [node] the ring has 2 node(s), so RF is 2. It rises to 3
when
       the third node joins and pd node add runs again - ...
[INFO] [node] cassandra: nodetool repair ...
```

규칙: 노드 2대 → RF 2, 3대 이상 → RF 3(그 이상 올리지 않음 — RF > 노드 수면 QUORUM 질의가 전부 실패). 데이터센터 이름이나 노드 수를 못 읽으면 **exit 1**.

무엇이 어떻게 복제되나

구성요소	방식
Aster TSDB	토큰 링(`num_tokens: 16` , RF 인지 토큰 배정) · NetworkTopologyStrategy · GossipingPropertyFileSnitch · hinted handoff
PostgreSQL	스트리밍 복제 — `wal_level = logical` , 슬롯 `replica_worker_N` , `max_slot_wal_keep_size = 8GB` (슬롯이 WAL 을 무한히 붙잡지 못하게), `max_wal_size = 4GB`
Valkey	프라이머리 → 레플리카, 뒤처짐(bytes) 콘솔 표시
Spark	워커 노드가 Spark 워커로 합류(7077)
명부	`/etc/kopens/workers.roster` — 콘솔 "워커" 화면이 명부 · 링 · PG 슬롯 · Redis · 헬스를 한 줄로 판정



그림 4. 워커 화면 — 1번 워커 10.99.0.101: 명부 있음 · 링에 있음 · PostgreSQL streaming(async) · 슬롯 `replica_worker_1` · Redis online(11.8 KB 뒤처짐) · 헬스 정상.

> **마스터 1 + 워커 최대 5.** 서비스를 세우지 않고, 이관 없이 늘리는 것이 목적입니다. 늘릴 시점은 콘솔 "스토리지"(볼륨 사용률) · "메트릭"(CPU · 메모리 · 디스크 I/O)이 미리 보여줍니다.

16 · Operations

Console · `pd` · JSON Contract

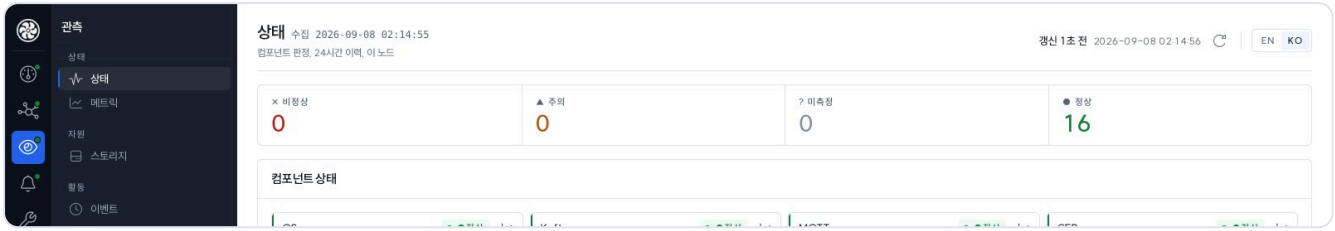


Figure 5. Console "Status" — for each of the 16 components, "what is measured and what the threshold is" (Kafka offline partitions 0/0 · PG longest statement 26 ms/5000 · Aster TSDB pending compaction 0/200 · TSE queue 626/40000).

Console twelve screens · five groups

Overview	Dashboard — one-line verdict · active alarms (threshold exceeded 3 consecutive times) · component health · 24/48h timeline
Observability	Status · Metrics (1h~7d) · Storage · Events · Logs · Diagnostics · Jobs
Operations	Config (render · diff) · Upgrade · Workers · Backup
Tools · History	8 web screen shortcuts (incl. Status) · Audit

The dot on the rail icon is "the worst state of that group". A group that could not be measured draws no dot.

Metric sources — JMX and net

Component	Path
Kafka · Aster TSDB · PG · MQTT · CEP · TSE · …	JMX (admin-api `jmx/impl`)
Spark	`:4440/metrics/master/prometheus/` (slash required — 302 without it)
Temporal · Kestra	`127.0.0.1:8244/metrics` · `:8391/prometheus` (loopback)
Valkey	RESP `INFO`

`pd` — one file per verb

```
pd start | stop | restart | status [service]
pd config render | diff [--templates] | list
pd secret rotate VAR=value ... | probe-mqtt
pd node add | status | repair | info
pd backup [list|schedule|status|run|restore]
pd recover | logs | storage | retention | flo
w | doctor | env
```

`--json` Contract (json-contract.md)

```
{ "schema":"pd.doctor/1", "verb":"doctor",
  "generated_at":"2026-09-08T02:10:58+09:00",
  "status":"ok", "reason":null, "exit_code":
0,
  "mode":"MASTER",
  "checks":[ {"id":"core ports","result":"pas
s","reason":null},
             {"id":"worker volumes","resul
t":"na",
              "reason":"docker socket not mou
nted"} ],
  "summary":{"pass":31,"fail":0,"na":2} }
```

- stdout is exactly one JSON object · secrets masked as `***` · `na` requires `reason` · `pass+fail+na == len(checks)` · `exit_code` equals the exit code

Exit codes

0 / 1 / 2	success / failure · diff present / usage error
3 / 4	missing required secrets (all names printed) / render failure (all unsubstituted `${PP_*}` printed)
5 / 6 / 7 / 8	gatekeeper service not started / some services not ready / shutdown not proven / advertised address loopback

17 · Protection

Backup · Restore · Upgrade



Figure 6. Console "Backup" — PostgreSQL (pgBackRest) last full 2026-09-06 01:00, differential daily 01:30, 8 sets (33.6–36.4 MB), "Restore" button for each set. Aster TSDB (Medusa) continues in the same shape.

Schedule — systemd timers, 10 units, inside containers

Job	Interval
PostgreSQL diff / full	Daily 01:30 / Sunday 01:00 (pgBackRest)
Aster TSDB diff / full	Daily 01:20 / Sunday 00:10 (Medusa)
purge	Sunday 03:00 — `repo1-retention-full` · Medusa `max_backup_age`

```
sudo CQL_PASS='...' backup.sh --install     # /etc/pgbackrest/pgbackrest.conf · /etc/medusa/medusa.ini (600)
backup.sh --schedule-install | --schedule-show   # JSON: units · switches
backup.sh --postgres diff | --cassandra full | --purge
pd backup run --engine postgres --type full --json
pd backup restore --engine cassandra --set full_20260906_0010 --dry-run --json
```

- `stanza-create` and `check` are gates — any failure exits 1, rerun is idempotent
- With the switch off, it does not touch tools and **records** `SKIPPED BY CONFIGURATION` · `result=skipped`
- At backup time it only asks "is the tool there" and if not, fails pointing to `install-deps.sh` — the path where dawn backups ran `dnf install` has been removed
- Default backup root `/data1/pp-backup` (`BACKUP_ROOT`) — **separate disk · NAS recommended**, external S3 sync is a design item at storage-separation time

Partial Loss — Data Recovery Tool

```
java -jar plantpulse-data-recovery.jar \
  -asset_id=ASSET001,ASSET002 -start=20250101 -end=20250105
# Reconstruct tm_asset_data from tm_tag_point tag points
# day → hour/min/sec → tag 3-step parallel, latest value per timestamp
```

Upgrade — Single Image Replacement

- Console "Upgrade" shows the replacement command with **image tag · build.version · burn commit · timestamp**
- Config is rendered from templates at every start — modifications inside containers do not persist (drift 0)
- Data lives in volumes (`PP_DATA_DIR · PP_BACKUP_DIR …`), config in `/etc/kopens/conf` — separated from the image, so **reverting to the previous tag is all that is needed**
- DB schema is applied by Flyway at startup

Supply Chain

Image signing	cosign — sign digest after registry push
Offline bundle	checksum + signature verification before import, stop install on failure
In-image verification	src overlay 173 files + dist manifest `sha256sum -c`
Input commit tracking	dist artifact commit ≠ clone HEAD → refuse build (after max 300s wait)

> **Rules learned from operational incidents have become code.** A regression test that prevents "success report with exit 0 while classpath is incomplete" (2026-09-04), and an archiver that distinguishes "missing job" and "0-row archive" in logs — many "must fail" clauses in this white paper are records of such incidents.

18 · Live Measurement

Running Instance — 2026-09-08 02:14 KST, dev 41

The following figures are exactly what the console displayed at that time. This is not a load test, but "evidence that it is running, not just documented."

16 / 16
Components healthy · Services 19/19 · Alarms 0

2 / 2
Nodes — Master + Worker, connected normally

48.8%
Keyspace pp valid ownership (Master) · Tokens 16

2.8 / 4.7 TB
Volume usage (59%) — PP_HOME · DATA · TEMP · BACKUP

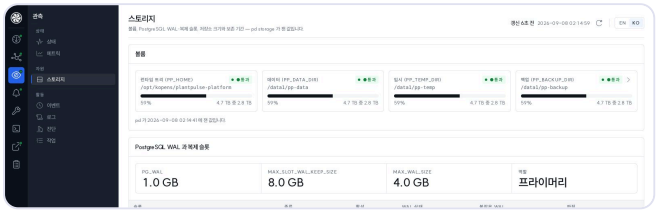


Figure 7. "Storage" — `pd storage` live measurement: PG_WAL 1.0 GB · MAX_SLOT_WAL_KEEP_SIZE 8.0 GB · MAX_WAL_SIZE 4.0 GB · Role primary · Slot `replica_worker_1` physical/active/passing.

Aster TSDB Disk	Size
Data (keyspace pp)	12.3 GB
Commit log	75.1 MB
Snapshot	3.9 GB
system · system_auth · schema …	7.8 MB · 154.9 KB · 1.2 MB

Kafka · Monitor retention	Value
kraft_combined_logs	191.2 MB · Retention 1 hour
Console monitor retention	Metrics 7 days · Events 30 days · Jobs 30 days · Health 1 day

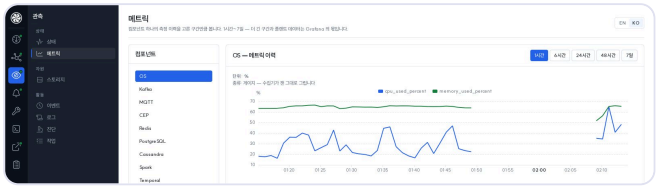


Figure 8. "Metrics" — OS 1 hour: CPU 15–47%, memory ~65%, disk write peak ~135 MB/s, disk total 5 TB / used 3 TB. The gray sections are where the server said "nothing to plot" — not connected by estimation.

Node Load (Master)	Value
CPU / MEM / DISK	36% / 65% / 59%
Aster TSDB Load · Pending compactions	8.37 GiB · 0 (threshold 200)
PostgreSQL Longest query	26 ms (threshold 5000)
TSE ingest queue	626 (threshold 40,000)
Kafka offline partitions · CEP drops	0 · 0
Temporal Longest wait · Kestra heap	0 s · 52–61%

Backup (Console)	Value
pgBackRest repository	3.4 GB · 8 sets · passing
Medusa repository	26.8 GB · Latest set marked "incomplete" — a failure the console does not hide

To be honest. This instance serves as both development and staging, so the latest Medusa set was caught as incomplete. The fact that this shows up as a red ✕ on the console is itself evidence of the principle "not to paint unmeasured or failed items green," so it is included as is.

19 · Comparison with General-Purpose Data Lakes

General-purpose Lakehouse · How It Differs from Cloud IoT Time Series

General-purpose lakehouses like Databricks · Snowflake and cloud IoT services like AWS IoT SiteWise / Timestream are excellent "analysis destinations." Below we note only the differences when positioned as an industrial "data infrastructure."

Axis	PlantPulse Industrial DataLake	General-purpose Lakehouse (Databricks · Snowflake, etc.)	Cloud IoT Time Series (AWS SiteWise · Timestream, etc.)
Deployment	● On-premises · fully isolated network as a single image, offline bundle · signature verification	◐ Cloud SaaS (on public cloud), limited on-premises options	○ Cloud-specific only
Industrial Protocol Collection	● Edge 42 types + Kafka · MQTT/Sparkplug B · Prometheus remote_write	○ None — requires a separate collection layer	◐ Gateway (OPC UA, etc.) provided, uploads to cloud
Time Series Storage · Computation	● Time series-specific storage engine (7.1×) + CQL 21 functions + TSE rollup/anomaly detection	◐ SQL over Parquet — some time series functions, ingestion is batch-oriented	● Time series-specific storage · aggregation
Asset Context · Events · BLOB	● Asset hierarchy · 6 data classes · alarms/events/commands/BLOB in one cluster	◐ Modeling in tables is possible, but a standard model is up to applications	◐ Asset model provided, but images · documents need separate storage
Long-term Archival Format	● Iceberg v2 · Parquet — readable by any engine, no "export" needed	● Delta / Iceberg open formats	◐ Service-internal storage, exported via export
Pricing Structure	● Based on configuration (nodes · sites) — not proportional to storage · query count	◐ Billing on compute · storage usage (credits)	◐ Billing on ingestion count · storage · query scan volume
Real-time Decision	● Proprietary engine Aster CEP — EQL (SQL-92 + streaming extensions) · SQL:2016 row pattern recognition, no separate cluster	◐ Streaming is a separate product · job	◐ Alarm rules (threshold-centric)
Operating Entity	● Customer infrastructure, console · `pd` · includes backup	● Vendor-managed (low operational burden, low control)	● Vendor-managed

Co-deployment Configuration

You can keep PlantPulse DataLake at the site and have a general-purpose lakehouse read the Iceberg tables in the cold layer **as is** for enterprise-wide analysis — if you share the catalog (Gravitino Iceberg REST), no copying is needed. Only aggregated metrics go to the cloud to control usage-based charges.

When a General-purpose Product Fits

Organizations where raw process data is already in the cloud, security policy permits export, and the focus is more on **relational · large-scale joins · ML pipelines** than time series. In that case, PlantPulse sits at the front end as the "on-site collection · standardization · cold archive producer."

Competitive · general-purpose product items summarize general characteristics from each vendor's public documentation (as of 2026-09-09) and may vary by version · plan. ● Built-in ◐ Option · separate · limited ○ Not applicable

Sizing Procedure and Sources

Inputs → Node Configuration

Input	What It Determines
Tag count × collection interval	Rows per second → ingest · memtable · compaction load, primary criterion for node count
Retention period (hot rows · chunks · cold)	Warm disk (`hot_window` rows + chunks) · cold object storage capacity
Concurrent users · dashboards	Query load → TSE · Kyuubi · Data Gateway concurrency (256)
CEP rule count · event rate	Real-time evaluation load → CEP heap · Valkey stream

Disk Sizing Example (based on Aster TSDB bench B/row)

```
100,000 tags × 1 sec = 8.64 × 10⁹ rows/day
Row storage (hot_window)  ≈ 11.9 B/row → ≈ 103 GB/day
After chunk (warm)        ≈  1.7 B/row → ≈  15 GB/day

Retention 90 days   : 7d hot + 83d chunk ≈ 0.72 + 1.2  ≈ 1.9 TB
(RF 3 → ×3)
Retention 365 days  : 7d hot + 358d chunk ≈ 0.72 + 5.4  ≈ 6.1 TB
(RF 3 → ×3)
Retention 3650 days : 7d hot + 3643d chunk ≈ 0.72 + 54.6 ≈ 55 TB
(RF 3 → ×3)  ← default shipping value
Cold (Iceberg zstd)   calculated separately — Parquet compression
ratio depends on value distribution
```

The above figures are an **example** obtained by simply multiplying bytes/row from the Aster TSDB benchmark (20M-row production schema); actual values vary with tag value distribution, column count, and replication factor. **Retention period is the largest knob determining capacity** — leaving the shipping default of 3650 days results in the last line of the table above, so set `storage.tag.point.ttl` first according to field requirements. We recommend measuring for a week using `pd storage --json` in a PoC before sizing.

Minimum Specifications Derived from Default Runtime Values

Aster TSDB heap 16 GB (`-Xms16G -Xmx16G`) + memtable off-heap 2 GiB, PostgreSQL `shared_buffers 4GB` (assuming ~25% of RAM), Spark driver 2 GB · executor memory per core automatic, TSE minimum 4 GB. The minimum specification for a master node is **16 vCPU · 128 GB RAM · 1 TB SSD**; the exact specification is derived from the input net above.

Sources

- **kopens/aster-tsdb** README · `doc/timeseries/` (tiering-benchmark · codec-bakeoff · production-rollout) — TSCS · chunk codec v4 · 21 functions · benchmarks · constraints
- **plantpulse-cep-aster-engine** — engine source · `MATCH_RECOGNIZE` tests (compared with Flink 2.3.0) / **plantpulse-cep** `docs/capability-matrix.md` (artifacts) · `differential-harness.md`
- **plantpulse-database-model** — `cassandra/CASSANDRA.md` · `ddl.cql` (104 tables · UCS · compression · SAI) · `spark/ICEBERG.md` · `ddl.sql`
- **plantpulse-datalake-cli** — `services/order.txt` · `config/render.map` · `config/templates/*` · `docs/PD-USAGE.md` · `docs/json-contract.md` · `docs/metrics-endpoints.md`
- **plantpulse-datalake-docker** README — source net · overlays · in-image sha256 verification · dist input
- **plantpulse-storage · timeseries · analytics · messaging · workflow** README · `UPSTREAM-LOCK.md` — version verification
- **plantpulse-timeseries-engine-server** docs — TQL JDBC · Prometheus adapter · clients · Web UI
- **plantpulse-data-gateway-web** README — endpoints · controls · error contract
- **plantpulse-archive-job · warehouse-web** README — 8 jobs · SPI · Temporal scheduling · Optimizer
- **plantpulse-backup** README · `backup.sh` — schedule · config paths · gate
- **plantpulse-datalake-admin-web** README — console screen structure · principles / console actual screens 2026-09-08 dev 41
- InfluxData blog (InfluxDB 3 Core 72h limitation · Enterprise) · TimescaleDB 2.14 release notes (multinode support discontinued) · Timescale license documentation — retrieved 2026-09-09

> **How to read.** Except for competitor and general-purpose product items, all figures are values at specific commit points of the above repositories. Products change on a monthly calendar version basis, so re-verify with `UPSTREAM-LOCK.md` at the time of review and the "Upgrade" screen in the console.

Inquiries

Phone · Email	1588-5268 · webmaster@kopens.com
Inquiry · Demo	kopens.com/ko/contact
Product · Docs	plantpulse.io · docs.plantpulse.io
Storage Engine Source	github.com/kopens/aster-tsdb (Apache-2.0)

Korea Open Solution Inc. · KOPENS, Inc. © 2026 KOPENS. PlantPulse® is a registered trademark of Korea Open Solution Inc. This document Rev. 2026.09.