



ReductStore

A data backbone for robotics
and industrial IoT

WHITE PAPER · 2026

Capture everything raw at the edge. Reduce on the way to the next tier.

Extract to any format or run SQL on top.

www.reduct.store

Executive summary

IN ONE PARAGRAPH

ReductStore is a high performance storage and streaming solution for data acquisition in robotics and industrial IoT. The pattern is always the same: capture everything raw at the edge, reduce on the way to the next tier, then extract to any format or run SQL on top. Sensor blobs, video frames, ROS messages, MQTT payloads, control logs, model outputs. None of it fits well in a time series database. None of it fits well in an object store either. ReductStore is built for the gap. One simple model. One query language. One pipeline from the device to the cloud. The same building blocks across robotics, industrial IoT, and drones.

CONTENTS

The problem at the edge	3
Why ReductStore is different	3
The mental model	5
How to integrate	6
The reduction strategy	7
The cloud tier	8
Robotics	9
Industrial IoT	10
Drones	12
Vision: a data backbone for robotics and industrial IoT	12
In summary	13

The problem at the edge

Robotics and industrial teams collect data faster than their infrastructure can handle. Lidar. Images. Point clouds. Control logs. MQTT payloads. Raw vibration time waves. Terabytes per fleet per month. Hard to store. Hard to label. Hard to retrieve.

Most teams cope with manual workflows. Hard drive swaps. Custom scripts. Bandwidth caps that block full uploads to the cloud. Connectivity drops that force buffering. Recording files that grow to tens of gigabytes and then have to be copied, opened, and filtered by hand before anyone can see what is inside. Basic tasks like keeping raw data on the device and replicating a smaller version to the cloud become projects.

Time series databases handle numbers well but struggle with images and binary payloads. Object stores handle blobs well but have no concept of time as a first class index. Most stacks end up with two databases side by side. One for numbers. One for blobs. Plus glue code to keep them in sync. Fragile, expensive, hard to operate.

The names change by domain. Robotics teams work with ROS topics, rosbag, and mcap. Industrial teams work with MQTT payloads, PLC tags, and vibration waveforms. Drone teams work with telemetry logs and imagery collected off network. The problem is the same in all three. The use case sections at the end of this paper cover each one with its own formats and protocols.

Why ReductStore is different

Six things set ReductStore apart from what you find in the marketplace.

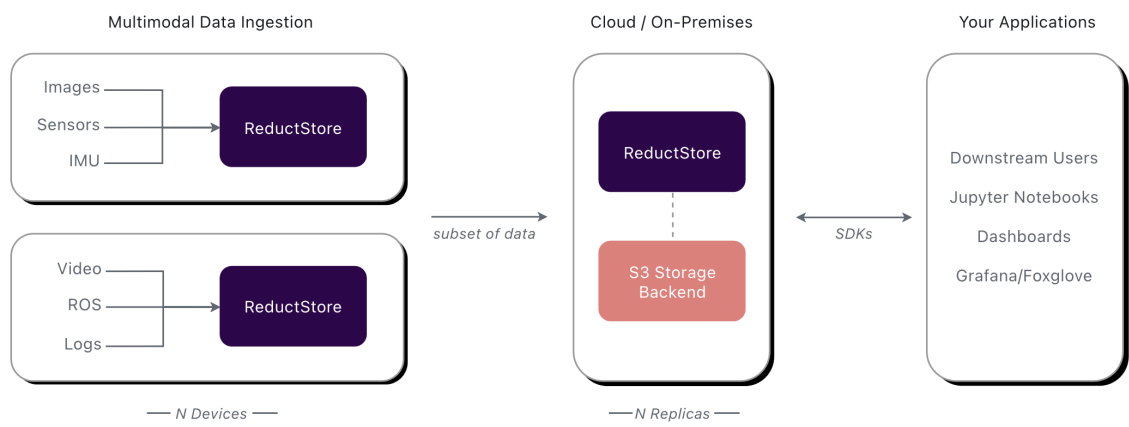


Fig. 1: Multimodal data ingestion pipeline using ReductStore.

A storage engine built for robotics and IIoT workloads. The engine is purpose built, not retrofitted from a general purpose database. Published benchmarks show ReductStore winning at every record size from 1 KB to 1 MB. The 100 KB example captures the typical gap:

REDUCTSTORE VS	READ SPEED	WRITE SPEED
MinIO	+1552%	+1288%
TimescaleDB	+603%	+924%
MongoDB	+260%	+420%

Full per database and per blob size results, including comparisons against InfluxDB and IoTDB, are linked at the end of this section.

Lower cloud bills. The cloud tier uses object storage as the backend, keeps hot data in fast tiers, and offloads infrequently accessed data to cold storage automatically. On a typical 50 TB workload, batched blocks and lifecycle aware tiering save around \$4,200 per month against a plain S3 setup. The cost section below has the line by line breakdown.

One database for raw data and analytics. Sensor blobs, video frames, ROS messages, logs, and MQTT payloads at the bottom. Conditional and SQL queries on top through the ReductSelect extension and DataFusion: filter, aggregate, reshape, output JSON, CSV, or Parquet. Extensions chain, so a single query can decode messages and run a SQL aggregation on the result. Metrics and raw data live in the same store.

Filter and replicate at the source. Data is persisted on the robot. Replication moves only the subsets that match your labels or conditions. Downsampling, fault windows, mission essential telemetry. You decide what crosses each transfer.

Edge friendly. Lightweight binary. High throughput on small machines. Designed for devices and private servers.

Volume based retention. FIFO quotas in bytes, not in days. This guarantees the disk's capacity will never be overrun by data accumulation. Stable operation regardless of how long the system runs.

Built for many devices. ReductStore comes from industrial IoT, where the workload is many devices on many sites. Replication runs over HTTP. Devices push records to central storage. Central pushes to the cloud. The receiving side never pulls. This architecture is built to scale from a single node to thousands of devices.

READ MORE [ReductStore vs MinIO](#), [ReductStore vs TimescaleDB](#), [ReductStore vs MongoDB](#), [ReductStore vs InfluxDB on vibration data](#), and [ReductStore and IoTDB](#).

The mental model

ReductStore has three things. Buckets, entries, and records. That is the whole shape of the system.

The screenshot shows the ReductStore web console. On the left is a sidebar with navigation links: Dashboard, Query, Buckets, Replications, Security, Help, Getting Started, Guides, CLI, Community, and GitHub. The main area is titled 'Query' and contains a 'DATA SOURCE' section with 'Bucket' set to 'ros_data' and 'Entries' set to 'center_camera/*'. Below this is a 'TIME RANGE' section with 'UTC' selected and a 'Last hour' dropdown. The 'Start' and 'Stop' times are both '2026-05-05T12:32:29.867Z'. The 'CONDITIONAL QUERY' section shows a query:

```
1 {
2   "search_t": "$__interval"
3 }
4
```

. Below the query is an example text explaining the query language and a 'Fetch Records' button. At the bottom is a line graph titled 'Interval: 30s' showing the 'Number of Records' over time. The x-axis represents time from 14:55 to 15:32, and the y-axis represents the number of records from 0 to 12. The graph shows a fluctuating line with peaks around 12 records. Below the graph is a table of records.

Timestamp	Entry	Size	Content Type	Labels	Actions
+ 2026-05-05T12:54:14.370Z	center_camera/camera_info	368 B	application/cdr	gps_y: 0.0, robot: orion, gps_z: 114.6385...	Download, Refresh, Delete
+ 2026-05-05T12:54:44.372Z	center_camera/camera_info	368 B	application/cdr	gps_y: 0.0, robot: orion, gps_z: 115.4018...	Download, Refresh, Delete
+ 2026-05-05T12:55:14.447Z	center_camera/camera_info	368 B	application/cdr	gps_y: 0.0, robot: orion, gps_z: 114.6385...	Download, Refresh, Delete

Fig. 2: Filtering records via the conditional query language.

A **bucket** holds data for a system. A robot, a factory line, a drone, a fleet.

An **entry** is a stream inside a bucket. One topic, one camera, one sensor. Entry names can be hierarchical, like sensors/imu/raw or cameras/front/jpeg.

A **record** is a single time stamped payload. Any type. Any size. Microsecond precision. Records carry **labels**. Labels are the only handle you need to filter data.

One conditional query language. Labels work the same way everywhere. The same labels filter records when you read them through an SDK. The same labels filter records when you replicate them across servers. There is one conditional query language, used in both places. No translation. No drift between read paths and replication rules. It supports comparison, logical, string, arithmetic, and aggregation operators. It also supports computed labels, so a query can derive new labels for downstream filters.

Storage volume quotas. Buckets expire data by storage volume, not by date. Each bucket can have a FIFO quota in bytes. When the volume is reached, the oldest records make room for new ones. This matters at the edge, where disks are small and fixed in size. A camera that runs for ten minutes longer than expected does not crash the system.

Binary in, format out. Records are stored as raw bytes. The database does not need to know what is inside. **Extensions** parse the bytes on the way out. For instance, the ReductROS extension reads ROS messages, mcap, and rosbag files. The ReductSelect extension handles CSV and JSON. The data on disk stays untouched. Each consumer asks for the format it needs. Schema on read, not on write.

SQL on top through DataFusion. ReductSelect runs SQL queries over stored records using DataFusion. Pick columns, filter rows, aggregate, reshape output before it leaves the server. Output as JSON, CSV, or Parquet. Extensions also chain. A robotics query can extract ROS messages with ReductROS, run a SQL aggregation on the

result with ReductSelect, then write the answer out as Parquet for a downstream notebook. Time series metrics stored as raw records become queryable as a real table without a second store.

READ MORE [How Does It Work](#), [Glossary](#), [Conditional Query Reference](#), [Buckets Guide](#), [ReductSelect \(SQL via DataFusion\)](#), and [ReductROS extension](#).

How to integrate

ReductStore takes data in through several paths, and the same paths read it back. The choice depends on the device and the source.

SDKs are the main developer interface. Python, JavaScript, Go, Rust, and C++. The same operations everywhere. Create buckets, write records, query, update labels, manage replication and tokens. Single record writes and batched writes are both supported. Batching cuts overhead when ingesting at high throughput. The SDKs cover the full surface of the database, so any application can be built directly on them.

HTTP API. The SDKs are clients for it. It is a REST API with token authentication, running over HTTP/1.1 or HTTP/2, and it covers everything the SDKs do: buckets, entries, single records, batches, conditional queries, tokens, and replication. If your language is not on the list above, or you want to write records straight from a device with curl, you can call it directly.

ReductBridge is an agent for cases where you do not want to write code. It runs on the device with one TOML file and forwards data into ReductStore as it arrives. The file has three parts. Inputs produce records from a source. Pipelines route and modify them. Remotes write them to ReductStore. Documented inputs today are MQTT, ROS1, ROS2, HTTP, host metrics, and shell commands.

MQTT subscribes to broker topics and stores each message. ROS1 and ROS2 connect robotics topics to ReductStore for storage and replay. The HTTP input polls a REST endpoint on a fixed interval and stores the response, which covers gateways and devices that have no broker in front of them. The shell input runs any command on a schedule and stores the output as records. Host metrics store CPU, memory, disk, and network usage as JSON records.

Labels can be generated from the incoming payload, so JSON and Protobuf fields become filters without a separate parsing step.

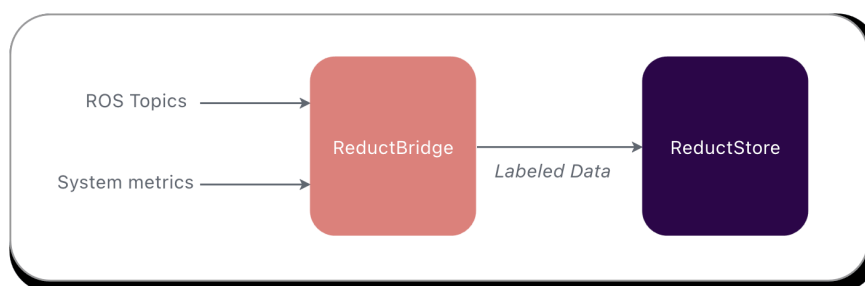


Fig. 3: Data ingestion paths (SDKs and ReductBridge).

Zenoh API. For fleets already running Zenoh, ReductStore can run a native Zenoh subscriber for writes and a queryable for reads. Records map to Zenoh keys, encodings, and timestamps with no glue code on the device. This runs alongside the HTTP API, it does not replace it. A device can publish over Zenoh while a dashboard reads the same records over HTTP.

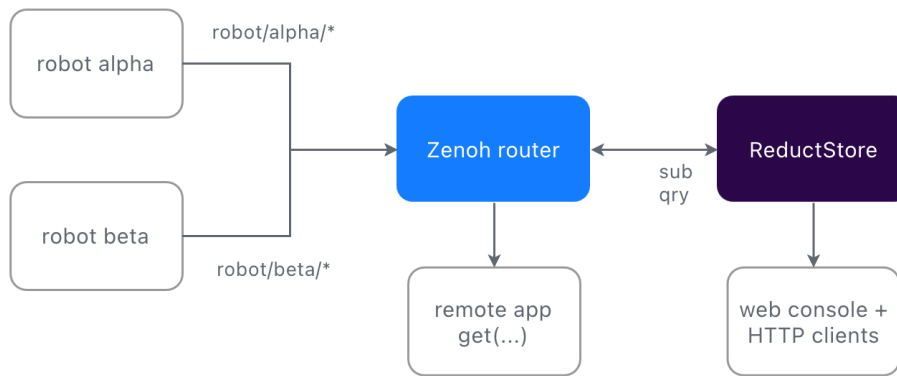


Fig. 4: Data ingestion via Zenoh API.

The choice of ingest path does not change the rest of the system. The data lands in the same buckets and entries. The same labels apply. The same conditional queries work.

READ MORE [HTTP API](#), [Python SDK](#), [JavaScript SDK](#), [ReductBridge](#), and [Zenoh API](#).

The reduction strategy

The pipeline has a second name: the reduction strategy. The idea is simple. Acquire raw at the source. Reduce as you move to the next tier. Reduce again as you move to the cloud.

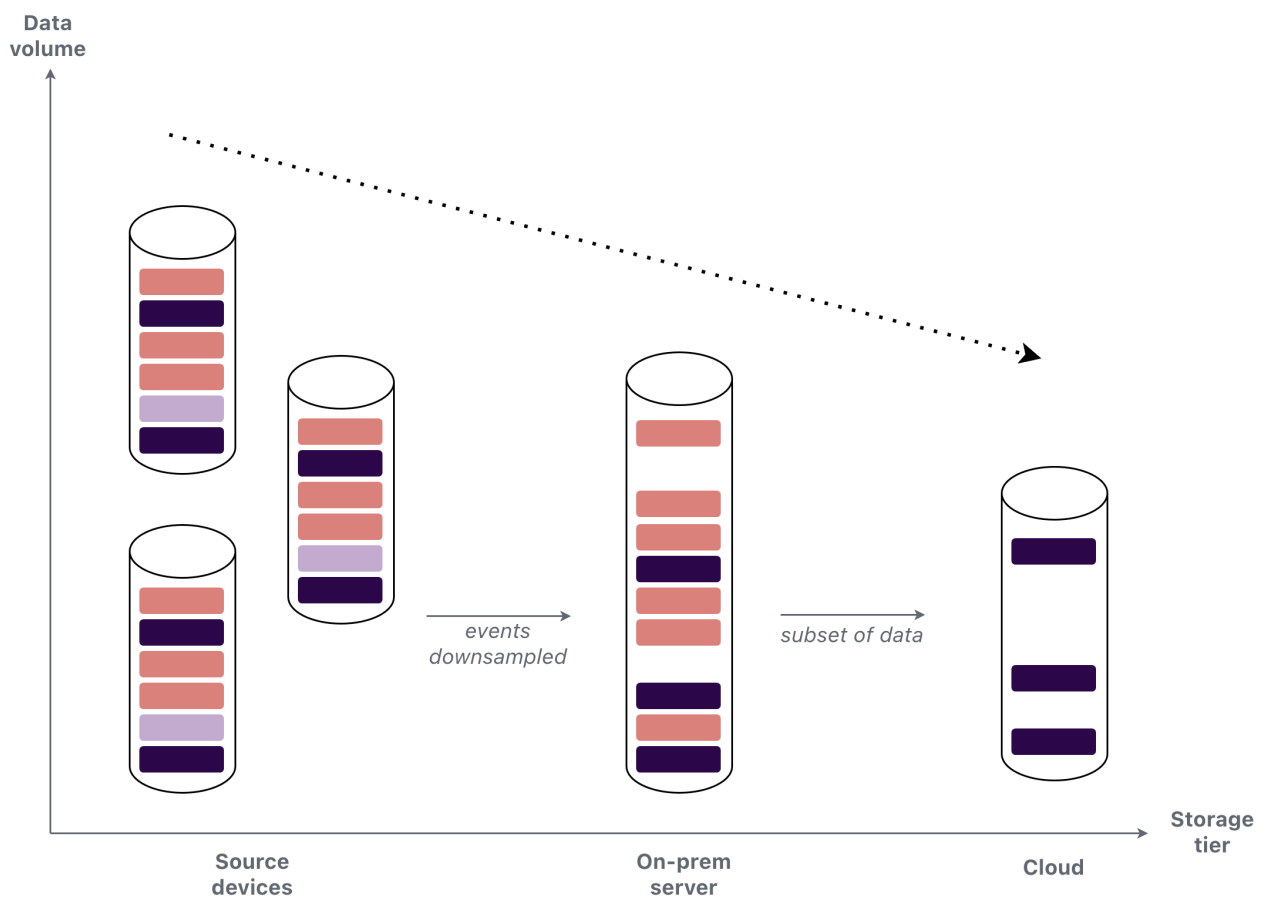


Fig. 5: The reduction strategy pipeline.

Acquire raw. Keep everything that comes off the sensor at the edge. Disks are cheap when they are local. FIFO by volume means buckets cannot be overfilled.

Reduce on replication. A replication task is a conditional query plus a destination. It picks which records are transferred. Records that match a label like event=fault go through. The rest stay on the edge until the FIFO rolls them off.

Reduce again on the next transfer. Replication is not one shot. The same pattern works between any two ReductStore instances. Each transfer applies its own filter. The further the data travels, the smaller the dataset becomes.

Push based, not pulled. Replication is initiated by the source instance, not the destination. The edge device pushes to the server. There is no central scheduler asking each device what it has. This is the pattern from industrial IoT brokers, applied to data acquisition.

ELT, not ETL. The traditional data acquisition pipeline is ETL. Extract from the sensor, transform at the source, then load the smaller dataset. The transform step usually means decimation, aggregation, or selection, and the cost is irreversible. Once a sample is dropped, it is gone. ReductStore inverts the order. Extract, load, then transform. The raw data lands in the edge bucket as is. Transforms happen on the way to the next tier through conditional queries, and on the way out through extensions. Nothing is decided early. If the question changes a week later, you can replicate with a different filter, because the raw data is still on disk.

READ MORE [Data Replication Guide and Replication API.](#)

The cloud tier

ReductStore Cloud is the last transfer in most pipelines. It is built for two things. High availability and low cost.

High availability. Instances run in active and passive roles. The active instance takes writes and reads. The passive instance follows. If the active fails, the passive takes over.

S3 and Azure Blob backends. The local disk holds a cache. The object store holds the truth. The database is elastic on storage and survivable on hardware.

Batched writes. Every write to S3 costs an API call. ReductStore batches data before writing, reducing the number of API calls. Fewer calls means a smaller bill.

Read cache. When a record is read from object storage, it is held in cache. Repeated reads hit the cache instead of going back to S3. For dashboards that refresh every few seconds, this matters.

What this means for the bill. Object storage is the cheapest and most scalable way to store unstructured data in the cloud. But small objects from robotics and data acquisition systems make the bill jump for two reasons. Most cloud storages charge per API call, and millions of PUT requests at high ingestion rates add up fast. Cold storage tiers also have minimum billable object sizes, so a 64 KB record gets charged for the minimum block size typically around 128 KB.

ReductStore lifts both problems at once. It batches records into blocks (e.g. 1024 records per block) before they reach S3, which cuts the number of operations by hundreds of times. Each block sits well above the minimum billable size, so cold tier storage is billed on the actual data.

Take a typical workload: 64 KB objects at 100 writes per second over three months, about 50 TB of data. The monthly savings against a plain S3 setup break down as:

SAVING	PER MONTH
Storage (better cold storage utilization)	~\$500
Data ingestion (fewer PUT requests)	~\$1,200
Lifecycle transitions (less object churn)	~\$2,500
Total savings	~\$4,200

READ MORE [Cloud Object Storage Backend](#), [Disaster Recovery](#), and [Cloud Solution](#).

Robotics

Robots produce a hard mix of data. Cameras at 30 frames per second. Lidar at 10 hertz. Inertial sensors at 100 hertz. ROS topics for control. Calibration files. Model outputs. Most of it has to be kept long enough to debug a fault and short enough not to fill the disk.

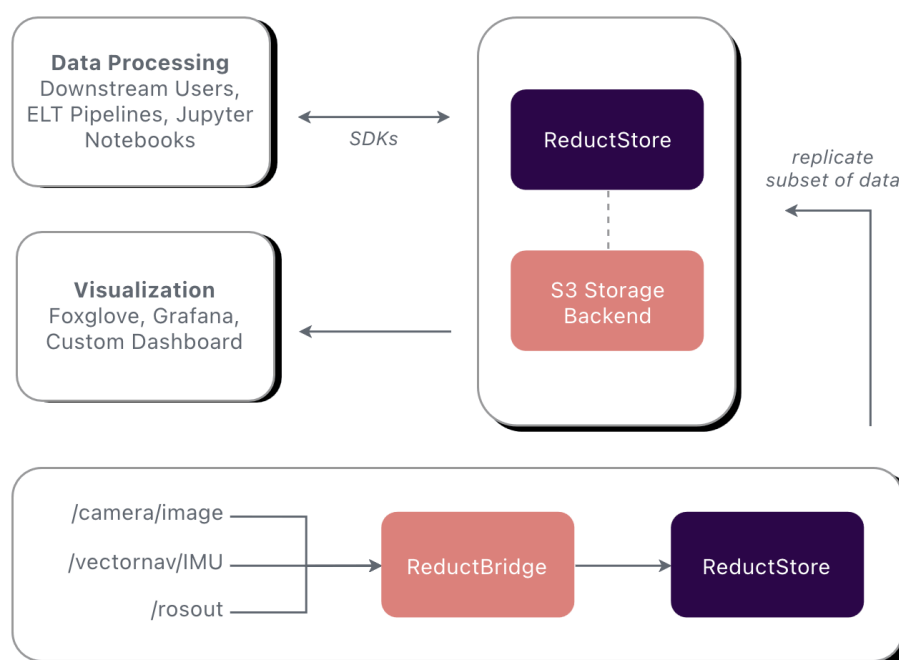


Fig. 6: The ReductStore robotics data flow.

Ingest. ReductBridge subscribes to ROS1 or ROS2 topics with one TOML file. Each topic becomes an entry. Each message becomes a record. Schemas live in the entry's \$meta attachment, so the data is self describing.

Storage and on the fly export. Records can be stored as ROS messages, mcap chunks, or rosbag files. The ReductROS extension reads any of these on the way out: filter by topic and time, return JSON, or assemble a smaller mcap or rosbag episode on demand for offline analysis. Extensions also chain. A query can extract ROS messages, run a SQL aggregation on the result through ReductSelect and DataFusion, then output JSON, CSV, or Parquet for a downstream notebook or dashboard. Raw ROS message to JSON to SQL to JSON, CSV, or Parquet, in a single conditional query. The data on disk does not change. Each downstream tool gets the format it expects. mcap export in particular is a first class output, useful when handing data to visualization or analysis tools that already speak the format.

Replication. From robot to fleet server to cloud. Each transfer has its own conditional query. Robots replicate flagged episodes to the fleet server. The fleet server replicates curated datasets to the cloud. Raw data stays on the robot until FIFO removes it.

Visualization. The official ReductStore Grafana plugin queries the database directly. ROS data can flow either directly into Grafana or be exported as an mcap file that can be opened with Foxglove or rerun.

READ MORE [How to Store and Manage Robotics Data](#), [The Missing Database for Robotics Is Out](#), [3 Ways to Store ROS Topics](#), [Comparing Data Management Tools for Robotics](#), [ReductROS extension](#), [ReductSelect \(SQL via DataFusion\)](#), [Visualize Robotics Data in Grafana](#), and [Grafana plugin](#).

Industrial IoT

Factories collect data from a different mix of sources. PLCs over MQTT. Vibration data at high sample rates. Cameras on the line for quality control. Each shop floor device has a small computer. Each plant has a server. The cloud is far and expensive.

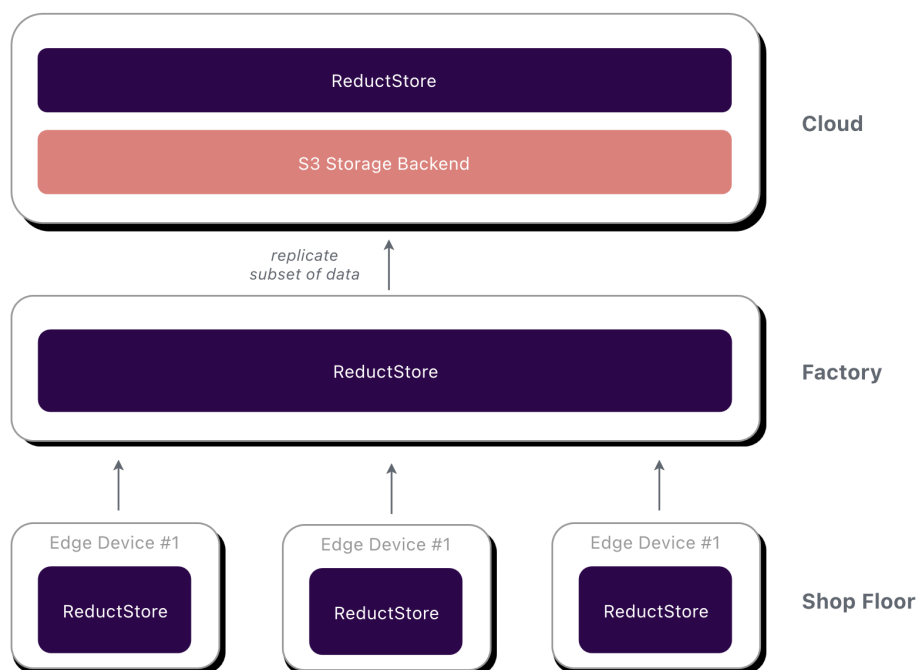


Fig. 7: The Industrial IoT three tier architecture.

Ingest. MQTT is the main path on the shop floor. ReductBridge subscribes to MQTT v3 and v5 topics over `mqtt://` or `mqtt://` and writes each message as a record. Labels can be pulled from the payload or from the MQTT properties, so a device id, a line number, or a quality flag becomes a filter later. For gateways and historians that expose a REST endpoint instead of a broker, the HTTP input polls the endpoint on a fixed interval and stores the response. For equipment reached through a vendor CLI, the shell input runs the command on a schedule and stores the output. If you would rather write the ingest code yourself, the Python, Node, and Rust SDKs each have a working pattern documented. Zenoh is supported on devices that already use it.

Storage formats. Vibration time waves are stored as one second binary chunks of the raw waveform, with labels like RMS and peak to peak written alongside. PLC payloads stay as MQTT body bytes, whether the line sends JSON, Protobuf, or a packed binary struct. ReductBridge can parse JSON and Protobuf on the way in to pull out labels, and the payload itself is stored untouched. Quality control images stay as JPEG.

Three tiers. Edge device, plant server, cloud. The edge device keeps a few days of raw data with FIFO by volume. The plant server keeps a few weeks of filtered data for the line engineers. The cloud holds the long term dataset for predictive maintenance models and audits.

Replication. A conditional query selects what crosses each transfer. Vibration chunks where RMS exceeds a threshold. Quality control frames flagged as defects. Logs around fault events. The rest stays on the previous tier.

Analytics. SQL queries run directly on stored data through ReductSelect and DataFusion. Aggregate vibration RMS over a shift, count defective frames per line. Output JSON, CSV, or Parquet for the next tool. No copy to a separate analytics store.

Visualization. The official ReductStore Grafana plugin powers dashboards on temperature, vibration, throughput, and image counts.



Fig. 8: Industrial IoT dashboards powered by Grafana.

READ MORE [Building a Data Acquisition System for Manufacturing](#), [DAQ shop Floor to Cloud](#), [How to Choose the Right MQTT Database](#), [How to Store Vibration Sensor Data](#), [Key Component of a Manufacturing Data Lakehouse](#), [Grafana plugin](#), and [ReductSelect \(SQL via DataFusion\)](#).

Drones

Drones change the rules. Most of the mission has no network. Battery limits everything. The data comes off the aircraft once it is back at the station. The chain of custody matters for inspection, mapping, and any regulated workload.

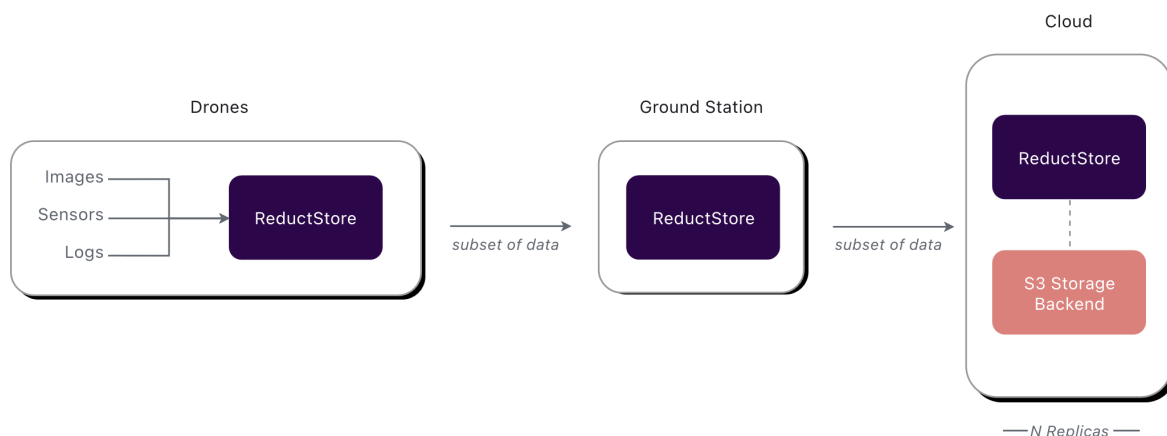


Fig. 9: Data flow for drone operations.

Keep raw on the drone. ReductStore runs on the aircraft. It writes every payload locally with FIFO by volume. This includes telemetry and flight logs, as well as JPEG or raw images from the camera used for the aerial survey and point clouds from the lidar payload. Each one becomes a timestamped record in its own entry. The disk never runs out of space. The oldest records roll off first.

Sync on landing. When the drone is back at the station, a replication task runs. It streams the new records to the ground station bucket. The replication task is a conditional query plus a destination, so the operator can choose what to replicate. Mission essential records and telemetry data.

Rate limits when needed. Several drones share the same network connection. Replication throughput can be capped so the network stays usable for live operations.

Audit. The data is append only. Each record has a timestamp and labels written at ingest. Nothing is rewritten in place. The chain from sensor to cloud is verifiable record by record.

Air gapped operations. The full pipeline can operate entirely without the cloud. Replication remains optional and catches up automatically once a connection is restored.

READ MORE [Air Gapped Drone Data Operations, Distributed Storage in Mobile Robotics, and Importance of Data Reduction for Edge Computing.](#)

Vision: a data backbone for robotics and industrial IoT

ReductStore is becoming a reference data backbone for robotics and industrial IoT. Raw payloads at the bottom. Time stamps and labels in the middle. SQL on top through DataFusion for analytics. The same store from sensor to cloud, with raw data and metrics in the same place.

The shape stays the same. Bucket, entry, record, label. The database that holds your raw streams answers your analytics questions too. No second store. No drift between raw data and reporting.

The full pipeline runs from device to dataset. Devices push to a central storage tier. Central pushes to a private cloud or to ReductStore Cloud. From there, dataset export writes the data in open formats: Parquet, Hive, LeRobot, or whichever structure the next tool expects. The same model scales from a single edge node up to thousands of devices across multiple storage tiers. Each transfer is configurable, and you can stop at any tier that fits your operation.

Open to the rest of the stack. Grafana powers dashboards. mcap exports tools like Foxglove or rerun for visualization. Workflow orchestration runs through Airflow or any tool that can talk to the HTTP API or the SDKs. Data source plugins fit ReductStore into analytics platforms like Databricks. The goal is to slot into an existing stack instead of replacing it.

Time based retention will also run alongside FIFO by volume so a single bucket can apply different rules per use case.

Compression is on the way too, designed per topic rather than as a one size fits all step. The model is chunk based, so the algorithm runs on the storage block rather than on the individual record. A vibration time wave, an mcap chunk, and a camera stream each call for a different scheme. Vibration data compresses well with delta plus a generic algorithm. Camera streams want a video codec, where adjacent frames look almost the same. Each entry will pick the algorithm that fits its data.

The end state is a single backbone for every team that touches the data. Robotics engineers query rosbag and mcap. Process engineers run SQL on vibration. Data teams pull curated Parquet datasets for ML. All in one place, with one operator story.

READ MORE [White Paper landing](#), [Use Cases](#), and [GitHub: reductstore/reductstore](#).

In summary

One model across robotics, industrial IoT, and drones. Buckets, entries, records, labels.

One query language from the SDK to replication. No translation between paths.

One pipeline from the device to the cloud. The reduction strategy keeps raw at the edge and sends only what matters across each transfer.

Raw data and analytics in one store. ReductSelect runs SQL queries through DataFusion on top of the same records, with output as JSON, CSV, or Parquet.

A cloud tier built to keep the bill down. Batched writes. Read cache. Lifecycle aware tiering on top of object storage.

The result is a stack you can reason about. The same building blocks for every application. The same operations on every tier.

TRY IT

ReductStore Cloud has a free tier. Self hosted is available under Apache 2.0. The source code is on GitHub at github.com/reductstore.

WEB

www.reduct.store

COMMUNITY

community.reduct.store

LINKEDIN

[linkedin.com/company/
reductstore](https://linkedin.com/company/reductstore)

YOUTUBE

[youtube.com/
@ReductStore](https://youtube.com/@ReductStore)