intelbyte

Designing a Scalable Factory Equipment Health and Maintenance Platform

Architecture, Data Model, and Algorithms

Franklin Aryee

Table of Contents

intelbyte

intelbyte

1 Executive Summary

This document presents the technical design for a Factory Equipment Health and Maintenance Platform developed for Intelbyte. The objective of the platform is to provide clear, real-time visibility into the health and operational status of equipment across multiple manufacturing plants, while preserving a complete historical record of machine telemetry, operator inputs, and maintenance activity for analysis, reporting, and audit purposes.

The proposed solution uses a scalable, event-driven data architecture that separates high-volume telemetry ingestion from day-to-day operational decision-making. Machine sensor readings such as temperature, vibration, throughput, and fault codes are captured as timestamped events for long-term retention, while a derived current equipment state is maintained for each machine. This enables supervisors to view a single, up-to-date record per asset without querying large historical datasets, even as data volumes grow.

The platform supports multiple plants, production lines, and machines, along with operator shift reports, manual inspections, and status overrides. Maintenance requests are generated automatically based on configurable thresholds, fault patterns, and operator-reported issues. A machine health scoring process combines recent telemetry signals and fault history into a single, easy-to-interpret score, while a clear precedence model resolves the current machine status across telemetry, operator inputs, and active work orders.

Overall, the design is practical, transparent, and scalable. It delivers immediate value for day-to-day operations and supports timely maintenance decisions, while leaving room to add features such as predictive maintenance, deeper analysis, and AI based insights as the platform grows.

intelbyte

2 Problem Context and Objectives

Manufacturing environments operate large numbers of machines across multiple plants and production lines, generating continuous streams of sensor data alongside operator reports and inspections. While this information is essential for monitoring equipment condition, its volume and fragmentation make it difficult to use effectively for day-to-day operational decisions.

A central challenge is maintaining a clear, up to date view of current equipment status while preserving complete historical data for analysis, reporting, and audits. Supervisors require a simple snapshot of machine health and status, whereas engineers and analysts need access to detailed historical records to identify trends, recurring issues, and long term performance patterns. Without a unified approach, teams often rely on manual processes, disconnected data sources, and reactive maintenance practices.

The objective of this platform is to address these challenges by providing a unified, scalable foundation for equipment health and maintenance management. At a high level, the platform is designed to:

- Provide a single, reliable view of the current status and health of each machine
- Ingest and retain high volume telemetry data for long term analysis and audits
- Capture operator reports, inspections, and status changes alongside sensor data
- Automatically trigger maintenance requests based on thresholds, fault patterns, and reported issues
- Support growth across plants, machines, and data volumes without major redesign

intelbyte

3 Requirements Analysis

The following requirements are defined based on the scenario provided, which assumes the availability of machine telemetry, operator inputs, and maintenance workflows typical of modern manufacturing environments.

3.1 Core Functional Requirements

These define the essential capabilities the platform provides to support equipment monitoring, maintenance, and operational decision-making. They are organized by functional area for clarity.

3.1.1 Asset and Equipment Management

- Support multiple plants, production lines, and machines
- Maintain a clear hierarchical relationship between plants, lines, and machines

3.1.2 Telemetry and Data Ingestion

- Ingest continuous machine telemetry, including temperature, vibration, throughput, and fault codes
- Associate each telemetry record with a machine identifier and timestamp
- Retain telemetry data as a complete historical record for analysis, reporting, and audits

3.1.3 Operator and User Inputs

- Allow operators to log shift reports, inspection results, and issue descriptions
- Record manual machine status updates with user attribution
- Store operator comments and inspection notes alongside machine and telemetry data

3.1.4 Maintenance Management

- Automatically create maintenance requests based on configurable thresholds, fault patterns, and operator reported issues

- Track maintenance requests through their lifecycle, including status updates and resolution details
- Link maintenance history to machines and triggering events

3.1.5 Health and Status Monitoring

- Calculate a machine health score using recent telemetry signals and fault history
- Determine the current machine status by reconciling telemetry data, operator inputs, and active maintenance work orders
- Provide a Current Equipment Status view showing one row per machine using the most recent available data

3.1.6 Reporting and Dashboards

- Provide dashboards for supervisors to monitor equipment health, status, and maintenance activity
- Support filtering by plant, production line, machine, status, and health indicators

3.2 Operational Constraints and Quality Considerations

Beyond core functionality, the platform is designed to operate reliably in a production environment where data volume and operational complexity increase over time. It is expected to scale to thousands of machines generating high-frequency telemetry without requiring major redesign or manual intervention.

Operational views used by supervisors and maintenance teams must remain highly available and reflect the latest known state of each machine in near real-time. To support audits, investigations, and long-term analysis, all telemetry and operational events are preserved as immutable historical records. The platform enforces role based access control to ensure users can only view or perform actions appropriate to their role, protecting sensitive operational data and workflows. System components are loosely coupled so ingestion, processing, analytics, and user facing features can be updated or extended independently, supporting continuous improvement without disrupting ongoing operations.

intelbyte

4 System Architecture

The platform follows an event-driven architecture that separates high-volume telemetry ingestion from operational workflows and user facing views. Machine telemetry is captured as append-only event data to support long-term retention, analysis, and auditability. In parallel, a derived current equipment state is maintained to enable fast operational queries that present one up-to-date record per machine.

At a high level, the system consists of a web application for operators and supervisors, backend services that manage operational workflows and business rules, a telemetry ingestion path optimized for scale, and a database layer that stores both historical telemetry and current machine state. Maintenance requests are generated through configurable rules and recorded as work items linked to the machines and events that triggered them.

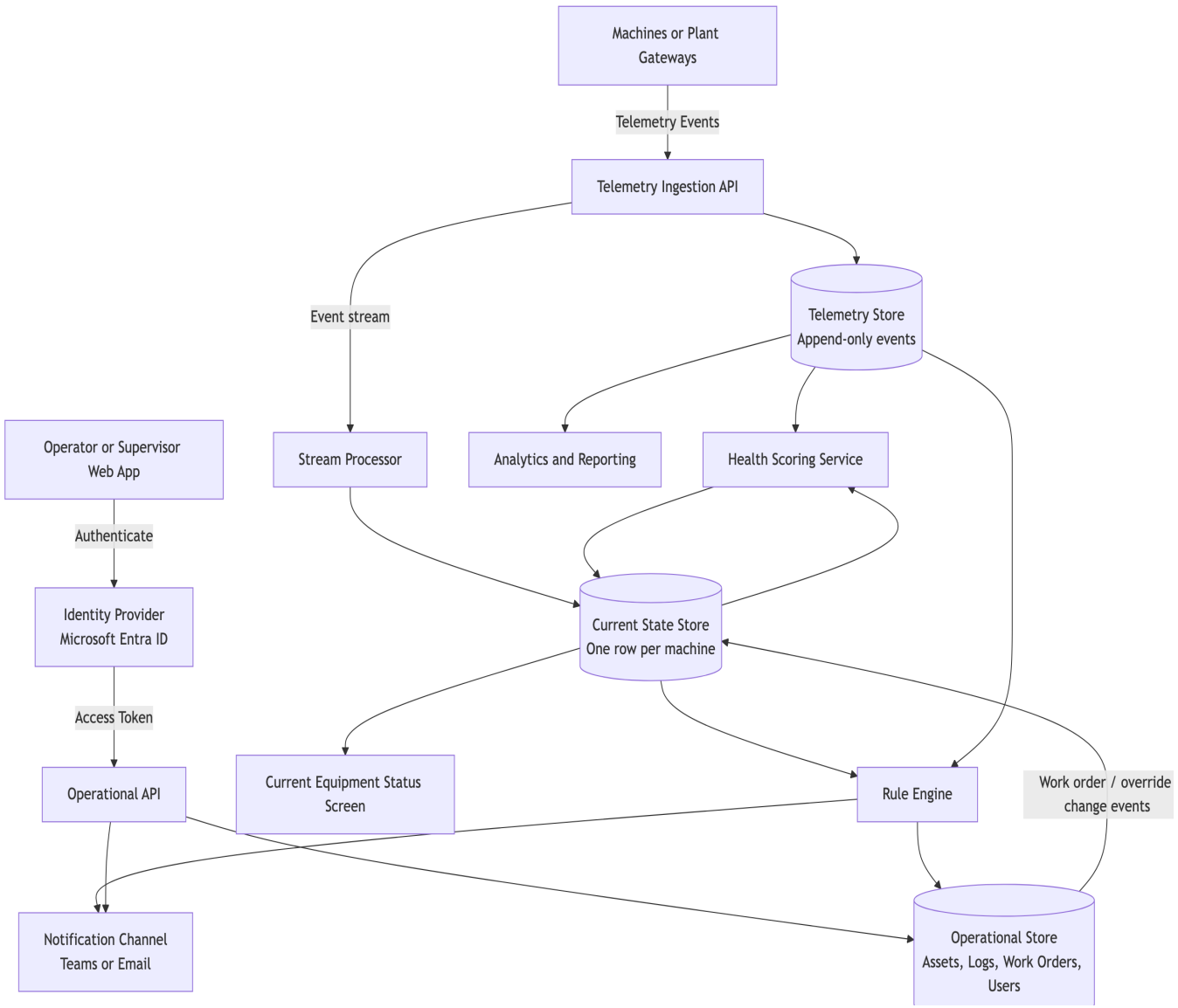

Figure 1: System architecture and data flow

intelbyte

4.1 Components and Functions

4.1.1 Frontend (Operator and Supervisor Interfaces)

- Operator and Supervisor Web App (Power Apps, Copilot): provides a unified interface for operators, supervisors, and maintenance teams to view current equipment status, review machine details, log shift reports and inspections, submit maintenance requests, and review, prioritize, assign, and track maintenance work orders.
- Analytics Dashboards (Power BI): visual dashboards showing equipment health, status, and maintenance activity by plant, production line, and machine

(a) Power Apps

(a) Power BI

(a) Copilot

(a) AI Builder

4.1.2 Backend Services and APIs (Application and Processing Layer)

- Operational API (Azure Functions): manages core business operations including plant and machine metadata, operator inputs, status updates, and maintenance workflows
- Telemetry Ingestion API (Azure IoT Hub): receives sensor data from machines or plant gateways and checks that required information is present

- Event Stream (Azure Event Hubs): temporarily buffers incoming telemetry to allow the system to scale and continue receiving data even during peak loads

- Stream Processor (Azure Stream Analytics): continuously processes incoming telemetry, determines the current condition of each machine using predefined rules, and keeps the operational view up to date.
- Rule Engine (Azure Functions): evaluates business rules against machine state and history to decide when maintenance actions are required, creating work orders and triggering notifications.

- Health Scoring Service (Azure Functions): calculates a simple health score for each machine using recent telemetry and fault history

4.1.3 Data Layer (Data Storage and Access)

- Operational Store (Dataverse or Azure SQL Database): stores structured business data including assets, users, operator reports, and maintenance records


(a) Azure Functions


(a) Azure IoT Hub


(a) Azure Stream Analytics


(a) Azure App-Services

- Telemetry Store (Azure Data Explorer): stores all raw telemetry data as a long-term historical record for analysis and audits
- Current State Store (Azure SQL Database): stores the latest status and health indicators for each machine to support fast operational views


(a) Dataverse

(a) Azure SQL Database


(a) Azure Data Explorer


(a) Azure Cosmos DB

4.1.4 Integrations (Access and Notifications)

- Identity Provider (Microsoft Entra ID): manages user login, roles, and access permissions
- Notification Channel (Power Automate with Teams or Email): sends alerts when maintenance requests are created or critical issues are detected


(a) Microsoft Entra


(a) Power Automate


(a) Teams


(a) Outlook

4.2 Data Flow and Communication

Telemetry and operator inputs follow different paths, but converge in the current equipment state used for operational dashboards.

4.2.1 Telemetry ingestion flow

1. Machines or plant gateways send telemetry events to the Telemetry Ingestion API
2. Events are validated and written to the Telemetry Store as immutable, append-only records
3. A stream processor consumes telemetry events and updates the Current State Store for the affected machine
4. The Health Scoring Service recalculates machine health scores based on new data or on a defined schedule
5. The Rule Engine evaluates thresholds and fault patterns and creates maintenance requests when conditions are met

4.2.2 Operator input flow

1. Operators submit shift logs, inspections, comments, or manual status updates through the web application
2. The Operational API records these inputs in the Operational Store with user attribution
3. Inputs that affect machine status are reflected in the Current State Store
4. Operator-reported issues may directly trigger maintenance requests or contribute to rule evaluation

4.2.3 Operational dashboard flow

- The Current Equipment Status screen queries the Current State Store to retrieve one row per machine for fast operational monitoring.
- Historical analysis and audits query the Telemetry Store and join with operational data as needed for reporting and compliance.

4.3 Architecture Pattern and Rationale

The architecture follows an event-driven pattern with a derived current-state view to handle high-frequency telemetry that grows rapidly over time. Telemetry is stored as immutable events to preserve a complete historical record for analysis and audits, while a separate current equipment state is maintained to support fast operational queries without scanning large historical datasets.

By decoupling telemetry ingestion from operational workflows, the system improves scalability and reliability. Telemetry capture can continue even if downstream processing is delayed, and user-facing views can rely on a lightweight current-state store for near real-time visibility. This design also supports incremental evolution, allowing more advanced analytics and predictive maintenance features to be added without restructuring the core architecture.

intelbyte

5 Data and Database Design

This section explains how the platform stores, organizes, and serves data in a way that balances **accuracy, scalability, and usability**.
The design supports both technical needs such as telemetry ingestion and analytics, and business needs such as fast operational views, traceability, and maintainable workflows.

At a high level, the solution separates **historical data** from **operational state**. Telemetry and logs are preserved in full for analysis and audits, while a lightweight current state model ensures that dashboards and user interfaces remain fast and simple.

5.1 Data Model Overview

The data model is built around three ideas.

First, the platform uses a clear **asset hierarchy** so machines can always be understood in context. Every machine belongs to a production line, and every production line belongs to a plant. This hierarchy enables filtering, reporting, and access control by site and line.

Second, **telemetry and events are immutable**. Sensor readings and machine signals are written once and never changed. This preserves data integrity and makes the system reliable for trend analysis, root cause investigation, and compliance.

Third, the system maintains a **derived current state** for each machine. Instead of recalculating status from millions of telemetry records, the platform keeps one up-to-date row per machine that reflects its latest condition. This is what operational screens and supervisors rely on.

5.2 Entity Relationship Diagram

The Entity Relationship Diagram below shows the core entities such as assets, telemetry, human inputs, and maintenance workflows are connected while keeping responsibilities clearly separated. See Table 1 for field level details.

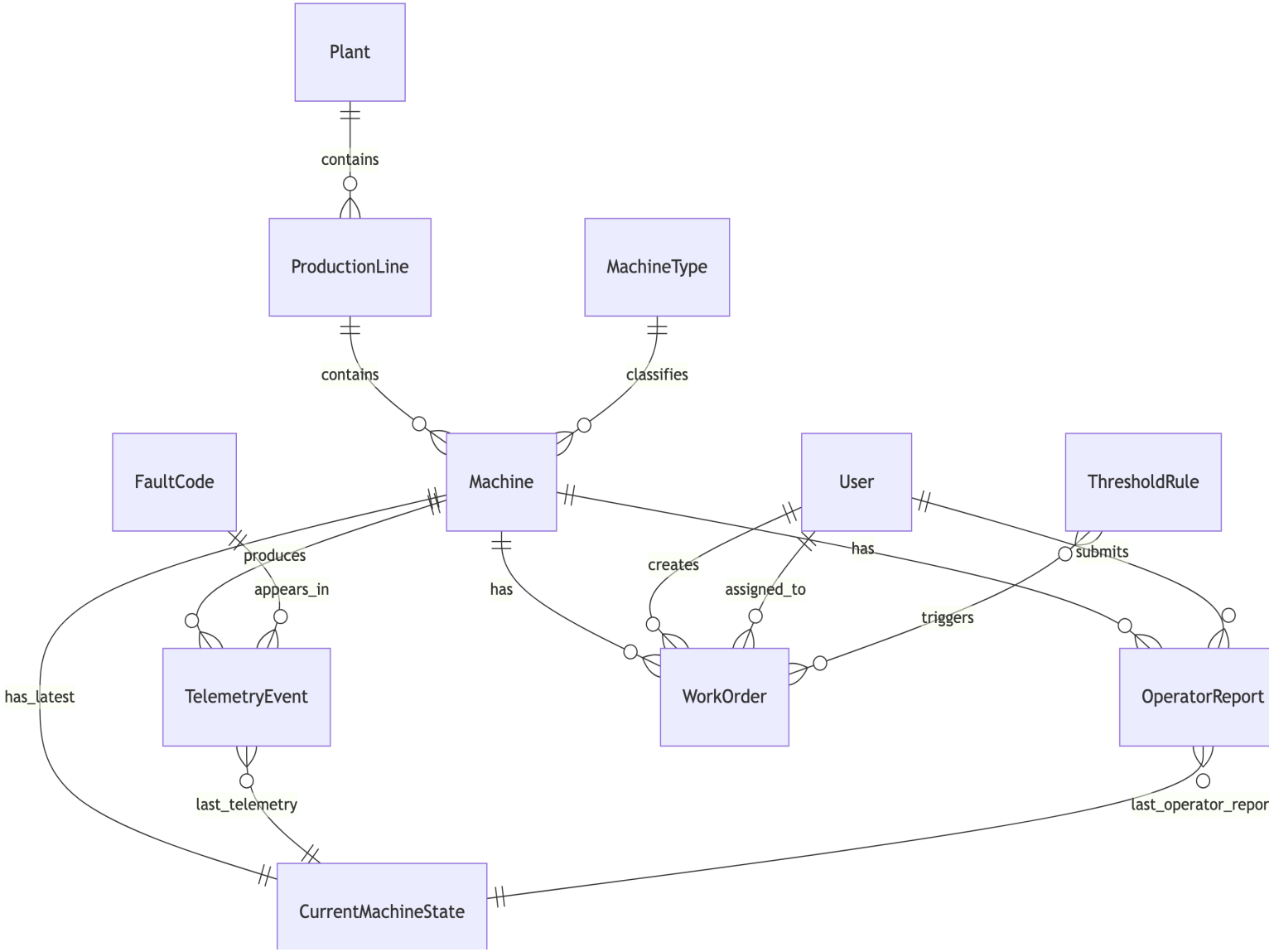

Figure 18: High level ERD for assets, telemetry, operator inputs, and maintenance

5.3 Entities and their Functions

The platform relies on a small but complete set of entities.

- Plants, production lines, machines, and machine types define the physical structure of the factory.
- Telemetry events capture raw machine signals over time.
- Operator reports capture human observations, inspections, and manual overrides.
- Work orders represent maintenance actions from creation through closure.
- Fault codes and threshold rules standardize how issues are detected and interpreted.
- Current machine state provides a fast operational snapshot for each machine.

Together, these entities allow the system to explain ***what happened***, ***why it happened***, and ***what action*** was taken.

5.4 Field Level Details

The table below lists the proposed tables with their primary keys, foreign keys, and essential fields required for the MVP.

Table 1: Field level details for the MVP schema

Table	Primary Key	Related References	Key Fields
Plant	plantId		plantCode, plantName, region, timezone, isActive, createdAt
ProductionLine	lineId	plantId (Plant.plantId)	lineCode, lineName, area, isActive, createdAt
MachineType	machineTypeId		typeCode, typeName, manufacturer, model, createdAt
Machine	machineId	lineId (ProductionLine.lineId), machineTypeId (MachineType.machineTypeId)	machineCode, machineName, serialNumber, installDate, isActive, createdAt
FaultCode	faultCodeId		faultCode, faultName, severity, description
TelemetryEvent	telemetryEventId	machineId (Machine.machineId), faultCodeId (FaultCode.faultCodeId)	eventTimestamp, ingestedAt, temperature, vibration, throughput, statusRaw, payloadJson
User	userId		displayName, email, role, isActive, createdAt
OperatorReport	operatorReportId	machineId (Machine.machineId), userId (User.userId)	reportTimestamp, reportType, statusOverride, comment
WorkOrder	workOrderId	machineId (Machine.machineId), createdByUserId (User.userId), assignedToUserId (User.userId)	workOrderNumber, priority, status, createdAt, closedAt

Table	Primary Key	Related References	Key Fields
ThresholdRule	thresholdRuleId	machineTypeId (MachineType.machineTypeId)	ruleName, metricName, operator, thresholdValue, windowMinutes, severity, isActive
CurrentMachineState	machineId	machineId (Machine.machineId), lastTelemetryEventId (TelemetryEvent.telemetryEventId), lastOperatorReportId (OperatorReport.operatorReportId)	resolvedStatus, healthScore, lastUpdateAt, openWorkOrderCount

5.5 Telemetry and Current Machine State Strategy

Telemetry data is modeled as an immutable event stream. Each new reading is inserted and never updated. This guarantees a complete and trustworthy history that can support audits, analytics, and future use cases without redesign.

Operational views do not query this history directly. Instead, the `CurrentMachineState` table is continuously updated as telemetry arrives, operator reports are submitted, and work orders change status. Each row represents the latest resolved view of a machine, including status, health score, and maintenance indicators.

This separation between append only history and derived operational state is the key scaling decision in the design. It keeps dashboards fast and predictable while preserving full historical detail for deeper analysis and long-term insight.

intelbyte

6 Core Algorithms

The platform maintains an operational view of each machine using near real time, event driven processing. As telemetry events arrive, the system updates the current machine state within seconds by resolving machine status and recalculating health indicators. Dashboards query this current state store and refresh on a short interval, providing timely and consistent operational visibility without scanning historical telemetry.

6.1 Machine Health Scoring

The Machine Health Scoring algorithm converts raw machine signals into a single, easy-to-understand indicator that reflects how healthy a machine is at any moment. The goal is not to predict failure perfectly in the MVP, but to provide a **clear, consistent signal** that helps supervisors and maintenance teams prioritize attention.

Industry platforms often normalize machine health score to a **0–100** scale to simplify interpretation for operations and maintenance teams (Uptime (2022)). The score combines **three ideas** that are common in industrial monitoring systems:

- **Normal operation** keeps the score high
- **Abnormal sensor readings** reduce the score gradually
- **Faults and maintenance states** reduce the score immediately and visibly

This balance ensures the score is stable enough for dashboards while still reacting quickly to real issues.

6.1.1 How Often the Score is Recalculated

The health score is recalculated **each time new telemetry is processed for a machine**. We assume telemetry arrives about **every 60 seconds**, so a **15-minute rolling window** provides **about 15 samples per metric**. Hence, This window reduces short lived noise while still reacting to sustained changes in machine condition.

Rather than recomputing the score from full historical data, the system updates the score incrementally using the previously stored value. In practice, this is done using an **Exponentially Weighted Moving Average (EWMA)**, which balances responsiveness with noise reduction in streaming telemetry (Montgomery (2020)).

If telemetry is missing for too long, the previous score is no longer reliable. As a safeguard, when the time since the last telemetry event exceeds **twice the rolling window** (in this case, more than **30 minutes** for the 15-minute window), the smoothing state is reset and the score is reinitialized from the current raw telemetry based score. See Section 6.1.4 for the health scoring pseudocode.

intelbyte

> The machines are assumed to publish telemetry approximately every **60 seconds**. A **15-minute rolling window** provides **15 samples per metric**. The health score is recalculated whenever new telemetry arrives and is typically updated once per minute.

6.1.2 Where the Score is Stored

Health scoring is performed by the Health Scoring Service (Azure Functions), which computes the score as telemetry arrives (approximately every minute). The score is stored in the `CurrentMachineState` table, which represents the latest operational view for each machine:

- Table: `CurrentMachineState`
- Primary key: `machineId`
- Related references:
  - `machineId -> Machine.machineId`
  - `lastTelemetryEventId -> TelemetryEvent.telemetryEventId`
  - `lastOperatorReportId -> OperatorReport.operatorReportId`

The scoring service keeps health scoring independent from status resolution and maintenance workflows by updating only the fields it owns:

- `healthScore`
- `lastUpdateAt`

6.1.3 Scoring Logic

At each update, the score is computed using recent machine behavior, not the full telemetry history. The scoring logic combines:

- The latest telemetry readings, aggregated as rolling averages over a 15-minute window
- Current machine status (`Running`, `Idle`, `Fault`, `UnderMaintenance`)
- Any active fault codes and their severity (`High`, `Medium`)
- Operator flags from inspections (`IssueObserved`, `MinorConcern`)

The pseudocode in Section 6.1.4 shows how each signal contributes to the final score and how EWMA smoothing stabilizes the result for operational dashboards.

6.1.4 Pseudocode for Scoring

```
def compute_health_score(previous_score, minutes_since_last_telemetry):
    rawScore = 100                                                          ①
    smoothing_param = 0.2                                                   ②

    # sensor penalties (computed from last 15 minutes of data)
    rawScore -= temperature_penalty                                         ③
    rawScore -= vibration_penalty
    rawScore -= throughput_penalty

    # fault severity penalties
    if fault_severity == "High":                                            ④
        rawScore -= 60
    elif fault_severity == "Medium":
        rawScore -= 30

    # apply operator input only if a report exists
    if operator_flag == "IssueObserved":                                    ⑤
        rawScore -= 20
    elif operator_flag == "MinorConcern":
        rawScore -= 10

    # maintenance state cap
    if status == "UnderMaintenance":                                        ⑥
        rawScore = min(rawScore, 60)

    # initialization / reset rule (no previous score or stale telemetry)
    if previous_score is None or minutes_since_last_telemetry > 30:         ⑦
        healthScore = rawScore
    else:
        # smoothing (EWMA)
        healthScore = (                                                     ⑧
            smoothing_param * rawScore +
            (1 - smoothing_param) * previous_score
        )

    # ensure the final score stays between 0 (worst) and 100 (best)
    healthScore = max(0, min(100, healthScore))                             ⑨

    return healthScore
```

① Start from a baseline score and compute a fresh rawScore for this update.
② Set the EWMA weight so recent data influences the score without causing jumps.

③ Apply penalties based on 15-minute rolling window sensor summaries
④ Apply a strong penalty when an active fault is detected, scaled by severity.
⑤ Reduce the score when operators flag issues in inspections or shift reports.
⑥ Cap the score during maintenance to reflect reduced availability.
⑦ If there is no previous score (MVP start) or telemetry is stale (>2× rolling window), reinitialize from rawScore.
⑧ Otherwise smooth using EWMA to reduce noise while keeping responsiveness.
⑨ Clamp the final score to stay between 0 and 100.

6.2 Latest Machine Status Resolution

The platform resolves `CurrentMachineState.resolvedStatus` by evaluating multiple signal sources in a fixed order of precedence. This ensures the resulting status is predictable, explainable, and aligned with real operational decision-making.

6.2.1 Status Precedence

Status precedence ensures maintenance and safety decisions take priority, while allowing the system to automatically return to telemetry driven status when conditions normalize. It is applied as follows (highest wins):

1. **UnderMaintenance**
   If one or more maintenance work orders are active for a machine, the machine is considered `UnderMaintenance` regardless of telemetry or operator inputs. This condition is represented by `openWorkOrderCount > 0`, derived from `WorkOrder.status`.
2. **Manual operator override**
   If an operator submits a report containing `statusOverride`, the most recent override is applied while it is still valid. This allows short term safety or inspection decisions to override automated signals.
3. **Telemetry derived status**
   When neither maintenance nor a manual override is active, the status is derived from the latest telemetry event. If `faultCodeId` is present the status is `Fault`; otherwise the status falls back to the device reported operating state stored in `TelemetryEvent.statusRaw` (for example `Running` or `Idle`).

> ℹ `WorkOrder.status` drives `openWorkOrderCount`. Any non-zero count forces `CurrentMachineState.resolvedStatus = 'UnderMaintenance'` until all active work orders are closed and `openWorkOrderCount` returns to `0`.

> Operator overrides are intentionally time boxed (example: 4 hours). After the window expires, status resolution automatically falls back to telemetry without requiring manual cleanup.

6.2.2 Initialization at Launch

This runs once to create the `CurrentMachineState` at MVP launch. After this, the system stays current via event driven upserts. See (**upserts?**).

```
-- Physical table (read model)
CREATE TABLE IF NOT EXISTS CurrentMachineState (
  machineId               BIGINT PRIMARY KEY,
  resolvedStatus          VARCHAR(32) NOT NULL,
  healthScore             NUMERIC(5,2) NULL,
  openWorkOrderCount      INT NOT NULL DEFAULT 0,
  lastTelemetryEventId    BIGINT NULL,
  lastOperatorReportId    BIGINT NULL,
  lastUpdateAt            TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);
```

6.2.3 Event Driven Updates

After the initialization, the platforms performs small targeted updates:

- When telemetry arrives for a machine, update that machine's row
- When a work order opens or closes for a machine, update that machine's row
- When an operator submits an override for a machine, update that machine's row

The query below shows per machine update executed when telemetry, work order or operator override is received for a specific machine.

> Note
>
> The SQL code block is executed with parameters supplied by the `Stream Processor`. These parameters are taken directly from the incoming telemetry message from `Telemetry Ingestion API`. They include `:machineId`, `:telemetryEventId`, `:eventTimestamp`, `:statusRaw`, `:faultCodeId`.

intelbyte

```
-- Runs for ONE machine whenever ANY of these events occurs:
-- 1) telemetry event, 2) work order opened/closed,
   3) operator override submitted, 4) health score updated.

WITH Incoming AS (
  SELECT
    :machineId                :: BIGINT        AS machineId,

    -- Telemetry event fields (present on telemetry events)
    :telemetryEventId         :: BIGINT        AS telemetryEventId,
    :eventTimestamp           :: TIMESTAMP     AS eventTimestamp,
    :statusRaw                :: VARCHAR(32)   AS statusRaw,
    :faultCodeId              :: BIGINT        AS faultCodeId,

    -- Work order fields (emitted when a work order is created/updated/closed)
    :workOrderId              :: BIGINT        AS workOrderId,
    :workOrderCreatedAt       :: TIMESTAMP     AS workOrderCreatedAt,
    :workOrderClosedAt        :: TIMESTAMP     AS workOrderClosedAt,
    :openWorkOrderCount       :: INT           AS openWorkOrderCount,
    :workOrderCreatedByType   :: VARCHAR(32)
          AS workOrderCreatedByType,
    :workOrderCreatedById     :: BIGINT
          AS workOrderCreatedById,

    -- Operator report fields (emitted when an operator submits a report/override)
    :operatorReportId         :: BIGINT        AS operatorReportId,
    :reportTimestamp          :: TIMESTAMP     AS reportTimestamp,
    :statusOverride           :: VARCHAR(32)   AS statusOverride,

    -- Health score event fields (present on scoring events)
    :healthScore              :: NUMERIC(5,2)  AS healthScore,
    :healthScoreAt            :: TIMESTAMP     AS healthScoreAt
),

Existing AS (
  SELECT *
  FROM CurrentMachineState
  WHERE machineId = (SELECT machineId FROM Incoming)
),

Merged AS (
  SELECT
    i.machineId,

```

intelbyte

```
    -- Keep lastest telemetry
    COALESCE(i.telemetryEventId, e.lastTelemetryEventId) AS lastTelemetryEventId,
    COALESCE(i.eventTimestamp,   e.lastUpdateAt)         AS lastTelemetryAt,
    i.statusRaw                                          AS statusRaw,
    i.faultCodeId                                        AS faultCodeId,

    -- Keep latest work order signals
    COALESCE(i.openWorkOrderCount, e.openWorkOrderCount, 0) AS openWorkOrderCount,
    COALESCE(i.workOrderId, e.lastWorkOrderId) AS lastWorkOrderId,
    COALESCE(i.workOrderCreatedByType, e.lastWorkOrderCreatedByType) AS lastWorkOrderCreatedBy
    COALESCE(i.workOrderCreatedById,   e.lastWorkOrderCreatedById)   AS lastWorkOrderCreatedBy

    -- Use createdAt/closedAt from the incoming work order event to bump freshness.
    GREATEST(
      COALESCE(i.workOrderCreatedAt, TIMESTAMP '1970-01-01'),
      COALESCE(i.workOrderClosedAt,  TIMESTAMP '1970-01-01'),
      COALESCE(e.lastUpdateAt,       TIMESTAMP '1970-01-01')
    ) AS lastWorkOrderChangeAt,

    -- Keep latest operator override pointer
    COALESCE(i.operatorReportId, e.lastOperatorReportId) AS lastOperatorReportId,
    COALESCE(i.reportTimestamp,  e.lastUpdateAt)         AS lastOverrideAt,
    i.statusOverride                                     AS statusOverride,

    -- Keep latest health score
    COALESCE(i.healthScore,   e.healthScore)             AS healthScore,
    COALESCE(i.healthScoreAt, e.lastUpdateAt)            AS healthScoreAt
  FROM Incoming i
  LEFT JOIN Existing e ON TRUE
)

INSERT INTO CurrentMachineState (
  machineId,
  resolvedStatus,
  healthScore,
  openWorkOrderCount,
  lastTelemetryEventId,
  lastOperatorReportId,
  lastWorkOrderId,
  lastWorkOrderCreatedByType,
  lastWorkOrderCreatedById,
  lastUpdateAt
)
SELECT
```

```
  m.machineId,

  CASE
    WHEN m.openWorkOrderCount > 0 THEN 'UnderMaintenance'

    WHEN m.statusOverride IS NOT NULL
      AND m.lastOverrideAt >= CURRENT_TIMESTAMP - INTERVAL '4 hours'
      THEN m.statusOverride

    WHEN m.faultCodeId IS NOT NULL THEN 'Fault'
    ELSE COALESCE(m.statusRaw, 'Idle')
  END AS resolvedStatus,

  m.healthScore,
  m.openWorkOrderCount,
  m.lastTelemetryEventId,
  m.lastOperatorReportId,
  m.lastWorkOrderId,
  m.lastWorkOrderCreatedByType,
  m.lastWorkOrderCreatedById,

  GREATEST(
    COALESCE(m.lastTelemetryAt,         TIMESTAMP '1970-01-01'),
    COALESCE(m.lastWorkOrderChangeAt,   TIMESTAMP '1970-01-01'),
    COALESCE(m.lastOverrideAt,          TIMESTAMP '1970-01-01'),
    COALESCE(m.healthScoreAt,           TIMESTAMP '1970-01-01')
  ) AS lastUpdateAt
FROM Merged m

ON CONFLICT (machineId) DO UPDATE
SET
  resolvedStatus              = EXCLUDED.resolvedStatus,
  healthScore                 = EXCLUDED.healthScore,
  openWorkOrderCount          = EXCLUDED.openWorkOrderCount,
  lastTelemetryEventId        = EXCLUDED.lastTelemetryEventId,
  lastOperatorReportId        = EXCLUDED.lastOperatorReportId,
  lastWorkOrderId             = EXCLUDED.lastWorkOrderId,
  lastWorkOrderCreatedByType  = EXCLUDED.lastWorkOrderCreatedByType,
  lastWorkOrderCreatedById    = EXCLUDED.lastWorkOrderCreatedById,
  lastUpdateAt                = EXCLUDED.lastUpdateAt;
```

intelbyte

`CurrentMachineState.lastUpdateAt` represents the most recent time **any part of the machine's current state changed**. This includes:

- a status change driven by telemetry, work orders, or operator overrides
- an update to `openWorkOrderCount`
- a new `healthScore` written by the Health Scoring Service

Because both status resolution and health scoring update the same read model, `lastUpdateAt` reflects overall **state freshness**, not just telemetry arrival time.

7 Dashboards and User Experience

The dashboards provide a near-real time operational view of machine health, status, and maintenance activity across the fleet. Updates to the underlying data model are event driven rather than tied to a fixed polling schedule.

Telemetry is assumed to arrive approximately every 60 seconds per machine. Each telemetry event is processed as it arrives and immediately triggers an update to the affected machine's current status and health score. Operational events, such as work order creation or closure and operator status overrides, may occur at any time and similarly trigger immediate updates to the corresponding machine's current state.

All event driven updates are written directly to the `CurrentMachineState` table, which serves as the single read model for dashboard views. Dashboards poll this table on a short, configurable interval, typically between 30 seconds and one minute. This polling interval is chosen to balance responsiveness with visual stability for human operators, while ensuring that operational changes are reflected shortly after they occur.

The `lastUpdateAt` field records the most recent time any current state attribute changed, including telemetry driven status resolution, health score updates, work order activity, or operator overrides. This allows users to assess data freshness, distinguish active machines from stale ones, and detect potential reporting or connectivity issues.

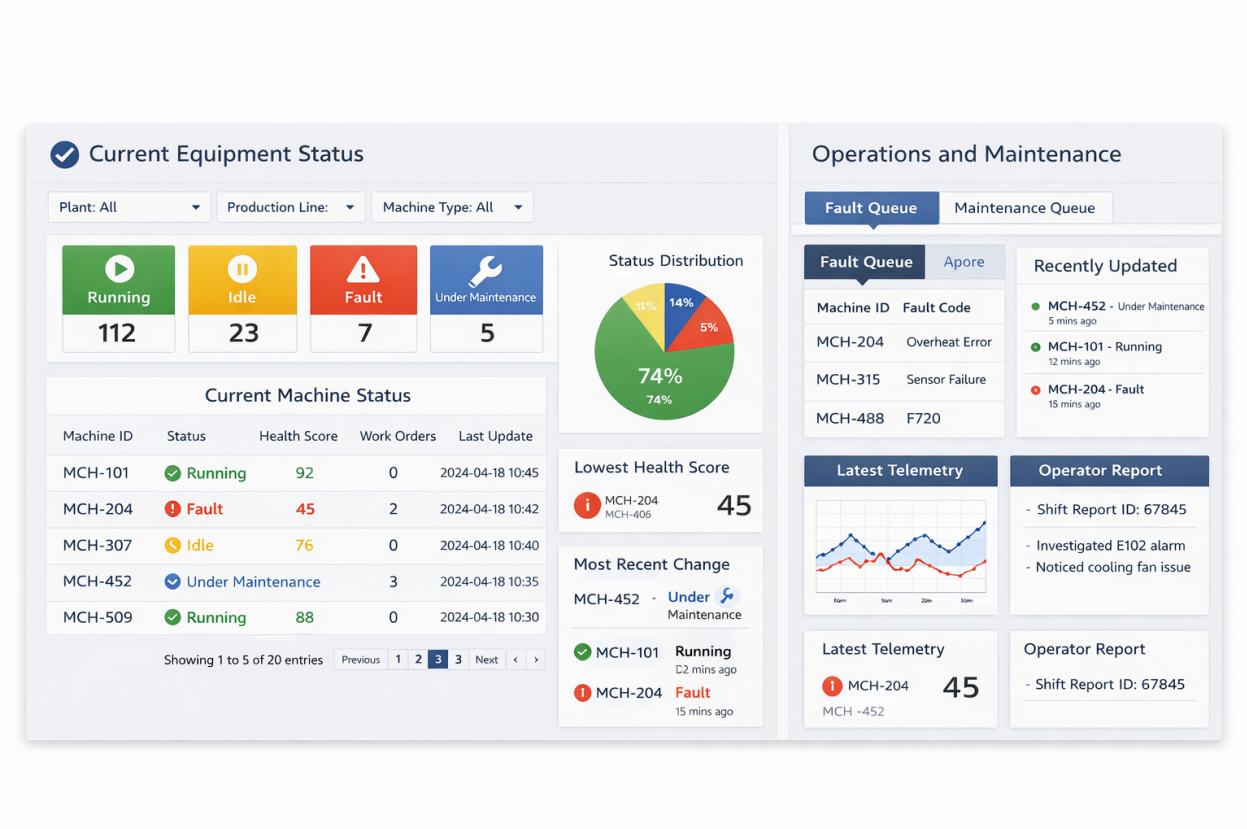

Figure 19: Combined dashboard mockup showing fleet wide equipment status, health scores, work orders, and maintenance actions

7.1 Current Equipment Status Screen

The Current Equipment Status screen is the primary operator view. It answers one question quickly: what is happening right now across the line.

What it shows

- Fleet level counts by `resolvedStatus` (Running, Idle, Fault, UnderMaintenance)
- A sortable table of machines with `resolvedStatus`, `healthScore`, `openWorkOrderCount`, and `lastUpdateAt`
- Quick filters (Plant, Production line, Machine type)

7.2 Operational and Maintenance Views

The Operational and Maintenance views support supervisors and maintenance teams. They focus on exceptions and action queues rather than the full fleet.

intelbyte

What it shows

- Machines currently in `Fault` with the latest fault context
- Machines currently `UnderMaintenance` with `openWorkOrderCount`
- A small “Recently changed” list using `lastUpdateAt`
- A drill down entry point using `lastTelemetryEventId` and `lastOperatorReportId`

intelbyte

8 Security and Data Protection

The platform is designed to protect operational data, ensure controlled access, and maintain trust in machine status and maintenance decisions. Security is treated as a foundational requirement rather than an afterthought, especially given the safety and reliability implications of industrial systems.

8.1 Key Security Concerns

The primary security concerns addressed by the platform include:

- Unauthorized access to operational data, such as machine health, fault history, and maintenance records
- Improper modification of machine state, which could lead to unsafe operational decisions
- Loss of traceability, where it becomes unclear who changed a status or triggered maintenance
- Exposure of sensitive telemetry or operational metadata across plants and regions

The architecture mitigates these risks by separating ingestion, processing, and presentation layers, and by enforcing access controls at every boundary.

8.2 Authentication and Authorization

Authentication and authorization are handled centrally using Microsoft Entra ID, ensuring consistent identity management across all user interfaces and services. APIs validate access tokens on every request, ensuring users can only view or modify data permitted by their role.

Authentication
All users access the platform through Entra ID using enterprise credentials. This supports single sign-on, multi-factor authentication, and centralized user lifecycle management.

Authorization
Role-based access control (RBAC) governs what actions users can perform. Typical roles include:

- Operator: view machine status, submit operator reports
- Supervisor: review machine health, approve or prioritize work orders
- Maintenance team: manage work orders and maintenance actions
- Administrator: manage users, roles, and system configuration

8.3 Data Integrity and Auditability

Trust in machine condition and maintenance decisions is critical for safe and reliable operations. This design makes it easy to review incidents after they occur, meet regulatory requirements, and continuously improve how operational rules are applied.

The platform ensures data integrity and auditability through:

- Append-only telemetry storage, preserving all raw sensor data for analysis and audits
- Explicit pointers in the current state, such as `lastTelemetryEventId` and `lastOperatorReportId`, which link the current view back to its source records
- Clear separation of concerns, where:
  - Status resolution is handled by event processing and workflows
  - Health scoring is computed independently by a scoring service
- Timestamps on all updates, allowing operators and auditors to understand when and why a state changed

intelbyte

9 Scalability and Reliability

The platform is built to scale from a small MVP deployment to a large, multi-plant industrial environment without any major redesign.

9.1 Data Growth and Performance

Telemetry data grows rapidly as machines publish frequent updates. The platform approach ensures predictable performance as the number of machines, plants, and telemetry volume increases.The architecture addresses this by:

- Separating historical and operational data
  - Raw telemetry is stored in a scalable analytics store optimized for high write volumes
  - The `CurrentMachineState` table stores only one row per machine for fast reads
- Event-driven processing

Telemetry events update only the affected machine's current state, avoiding periodic full-table scans or fleet-wide recomputation

- Read-optimized dashboards

Dashboards query the `CurrentMachineState` table directly, enabling fast refreshes even as historical data grows into millions or billions of records.

9.2 Failure Scenarios and Resilience

The system is designed to remain operational under common failure scenarios:

- Telemetry spikes or bursts: Event buffering absorbs temporary surges without dropping data.
- Temporary processing delays: If downstream services slow down, telemetry ingestion continues and processing resumes when capacity is available.
- Service restarts or redeployments: Stateless processing components can restart safely, using stored state and telemetry history to resume operation.
- Partial data loss or gaps: Rolling windows and incremental scoring ensure that short interruptions do not permanently distort machine health indicators.

intelbyte

10 Conclusion and Recommendations

This report presents a scalable and practical approach to building a Factory Equipment Health and Maintenance Platform MVP. By separating telemetry ingestion, state resolution, health scoring, and visualization, the platform achieves clarity, performance, and operational reliability.

Key strengths of the proposed solution include:

- A clear and explainable machine status model with explicit precedence rules
- Near real-time operational visibility without querying historical telemetry
- Independent and incremental health scoring suitable for streaming data
- Strong foundations for security, auditability, and role-based access

10.1 Future Enhancements

As the platform evolves beyond the MVP, the following enhancements can be considered:

- Introduce predictive maintenance models using historical telemetry and work order outcomes
- Refine health scoring using machine-type-specific models and adaptive thresholds
- Add automated anomaly detection to complement rule-based logic
- Enhance dashboards with trend views and cross-plant comparisons

Overall, the architecture provides a solid foundation that balances simplicity for the MVP with a clear path toward more advanced industrial analytics and automation.

intelbyte

References

Montgomery, D. C. 2020. *Introduction to Statistical Quality Control*. John Wiley & Sons, Incorporated. https://books.google.ca/books?id=X6JFxAEACAAJ.

Uptime, Infinite. 2022. "Understanding Machine Health Score and Its Importance in Asset Reliability." 2022. https://medium.com/@infiniteuptime/understanding-machine-health-score-and-its-importance-in-asset-reliability-4912218ea2f7.