International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 10 | Oct - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

An Open-Source Industrial Digital Twin Platform for Explainable Predictive Maintenance: Architecture and Single-Node Performance Characterization

Hemalatha, Jeyamoorthi S, Maanasa Sadagopan, Malini M, Jishnu S, Koushigan S
Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, Coimbatore, Tamil Nadu, India
jeyamoorthis24cs@srishakthi.ac.in

Abstract—Predictive maintenance is usually deployed on proprietary industrial platforms, so the engineering substrate beneath the published models — acquisition, current-state storage, history, explanation delivery and scheduling — is seldom described in enough detail to reproduce and is almost never measured. This paper presents the architecture of a digital twin and predictive-maintenance platform assembled entirely from permissively licensed components, and reports what it costs to operate on one commodity machine. A discrete-event simulator reproduces ten machines of five types using eight degradation models and publishes 81 sensor channels as Sparkplug B over MQTT, while also serving OPC UA and Modbus TCP. An ingest service decodes the payloads, writes them to a TimescaleDB hypertable, patches per-asset state in Eclipse Ditto, evaluates 33 alarm rules and republishes updates to subscribed browsers. Scheduled workers run anomaly, failure-mode and remaining-useful-life inference, attach attribution-based explanations that are audited against the numeric result before release, and raise work orders that a constraint solver places against technician availability and electricity tariffs. On a laptop granting 7.57 GiB to the container runtime, fourteen services hold 3.66 GiB resident and sustain 80.7 readings per second; live updates reach a subscribed client in 74 ms at the median. Columnar compression stores 7.08 million readings at 11.91 bytes each, a ratio of 15.2:1, and an hourly continuous aggregate answers a fleet-wide query 90 times faster than the raw table. All 450 automated tests pass.

Keywords—digital twin, explainable artificial intelligence, industrial internet of things, predictive maintenance, time-series databases

I. Introduction

Unplanned stoppages are expensive in ways that are easy to underestimate: lost output, idle labour, expedited spare parts and, on process lines, scrapped material. Maintenance practice has moved from repairing after failure, through fixed-interval servicing, towards acting on the measured condition of each machine. The third approach is attractive because it spends maintenance effort where the evidence points, but it depends on a chain of working software rather than on a single model.

Most published work on predictive maintenance optimises one link in that chain. A model is trained on a benchmark corpus and compared against other models on the same corpus [3], [4]. The result is useful, but it says nothing about the system the model would live in. Between a vibration sensor and a scheduled repair sit protocol decoding, a store for current machine state, a store for history, alarm evaluation, authentication, an explanation a technician will accept, a scheduler that respects who is on shift, and an interface someone will use during a night shift. In industry these parts are supplied by commercial platforms whose internals are not published and whose licences are priced per asset.

Two consequences follow. Researchers cannot reproduce a described deployment, because the substrate is a black box. Smaller manufacturers cannot adopt the method at all, because the entry cost is set by licensing rather than by hardware. Neither group has published figures for what a complete pipeline actually consumes.

This paper addresses the substrate rather than the model. We describe a platform that covers the path from sensor acquisition to a scheduled work order, built only from components under permissive or similarly open licences, and deployable on a single machine with one command. We then measure it: ingest throughput, delivery latency on both the persistence and the live path, storage efficiency, query latency, server-side request latency, resident memory per service, and the result of the automated test suites. We deliberately make no claim about prediction accuracy; establishing that requires field data from instrumented machines and is a separate study.

The contributions are as follows. First, an end-to-end architecture spanning acquisition, twin state, time-series history, inference, explanation, scheduling and analytics, running as fourteen coordinated services on one node. Second, a physics-grounded machine simulator that emits labelled degradation data over the same industrial protocols a real line would use, so the platform can be exercised and taught without plant access. Third, an explanation path constructed so that generated prose cannot assert a quantity the underlying computation did not produce. Fourth, an actuation model that binds every natural-language command to a risk tier with read-back confirmation and a second factor for commands that move machinery. Fifth, a measured characterisation of the whole stack on commodity hardware.

Section II reviews related work. Section III describes the architecture and Section IV the acquisition path and simulator. Section V covers the analytics and decision-support design and Section VI the access-control and actuation model. Section VII reports the measurements, Section VIII discusses what they imply and where the work is limited, and Section IX concludes.

II. Related Work

A. Digital Twin Architectures

Kritzinger et al. separate three levels that are often conflated [1]. A digital model is a representation with no automatic data exchange; a digital shadow receives data automatically from the physical asset but does not act back on it; only a digital twin closes the loop in both directions. The distinction matters here because write-back to a machine is the step that introduces safety obligations, and it is the step we treat explicitly in Section VI. Tao et al. survey industrial practice and note that most reported implementations rest on vendor platforms [2].

Interoperability has been approached through standards rather than through a common implementation. The Asset Administration Shell defines a vendor-neutral description of an asset and its submodels [21]; OPC UA provides an information model and transport widely implemented on plant equipment [17]; the Sparkplug specification fixes payload encoding, topic structure and birth/death semantics for MQTT so that a subscriber can reconstruct state after reconnecting [18], [19]. A platform that speaks these is portable; one that exposes only a private REST interface is not.

B. Predictive Maintenance

Reviews of prognostics and health management describe a common pipeline: window the signal, extract features, detect departure from normal behaviour, classify the mode of failure and estimate remaining useful life [3], [4]. Run-to-failure data is scarce, which is why the C-MAPSS turbofan degradation corpus has become the default benchmark [5]. Isolation Forest remains a common unsupervised detector because its cost grows slowly with sample count [6], and gradient-boosted trees are a frequent choice for tabular condition data [7]. Point estimates of remaining life are of limited operational use without an interval, which conformal methods supply with coverage guarantees that do not depend on the model being correctly specified [13], [14].

C. Explanation in Maintenance Settings

Attribution methods assign contribution to input features, either by local surrogate fitting [9] or through Shapley values [8]. Glass-box models such as explainable boosting machines offer a directly readable alternative [10], and Rudin argues that in high-stakes decisions an interpretable model should be preferred to a post-hoc rationalisation of an opaque one [11]. Counterfactual explanation asks instead what would have to change for the outcome to differ [12], which maps naturally onto a maintenance decision.

What is largely missing is attention to the last step. A technician does not read a Shapley vector; they read or hear a sentence. If that sentence is produced by a language model, nothing in the attribution literature prevents it from introducing a number that the computation never produced. The design in Section V-B treats the sentence as an artefact to be audited against the numeric explanation rather than as a presentation detail.

D. Positioning

We do not propose a new estimator. The contribution is an integrated and measured substrate: an arrangement of open components that implements the full path, together with figures for what that arrangement costs on one machine.

III. System Architecture

A. Design Constraints

Five constraints shaped the design. The deployment target is a single machine with roughly eight gigabytes available to the container runtime, because that is what a maintenance office or a teaching laboratory can provide. Every runtime component must carry a permissive or comparable open licence, so that adoption is not gated on procurement. Starting the system must be a single command, since a procedure with manual steps will drift. The backend follows a fixed layering in which transport handlers contain no business rules and services contain no queries. Finally, every state change is authenticated and recorded, which is a precondition for allowing any command to reach a machine.

B. Service Topology

Table I lists the fourteen services and the role each one holds. Grouping them by purpose, there are four infrastructure services (broker, relational and time-series store, cache and message broker, identity provider), five services forming the twin store and its database, and five application services written for this platform.

TABLE I. Runtime Services and Their Roles

Service	Component	Role
postgres	PostgreSQL 16 + TimescaleDB	Relational data and telemetry history
mosquitto	Eclipse Mosquitto 2	MQTT 5 broker
valkey	Valkey 8	Pub/sub fan-out and job broker
keycloak	Keycloak 26	OpenID Connect identity provider
mongodb	MongoDB 7	Persistence for the twin store
ditto-*	Eclipse Ditto 3.9 (4 services)	Current twin state, policies, search
simulator	This work	Machine physics and protocol servers
ingest	This work	Decode, store, alarm, fan-out
api	This work	REST and WebSocket interface
worker	This work	Scheduled inference and rollups
web	This work	React single-page application

Figure 1 shows how data moves between them. The path from a sensor sample to a browser is: simulator, broker, ingest service, and then three destinations in parallel — the time-series store, the twin store, and a pub/sub channel that the interface subscribes to.

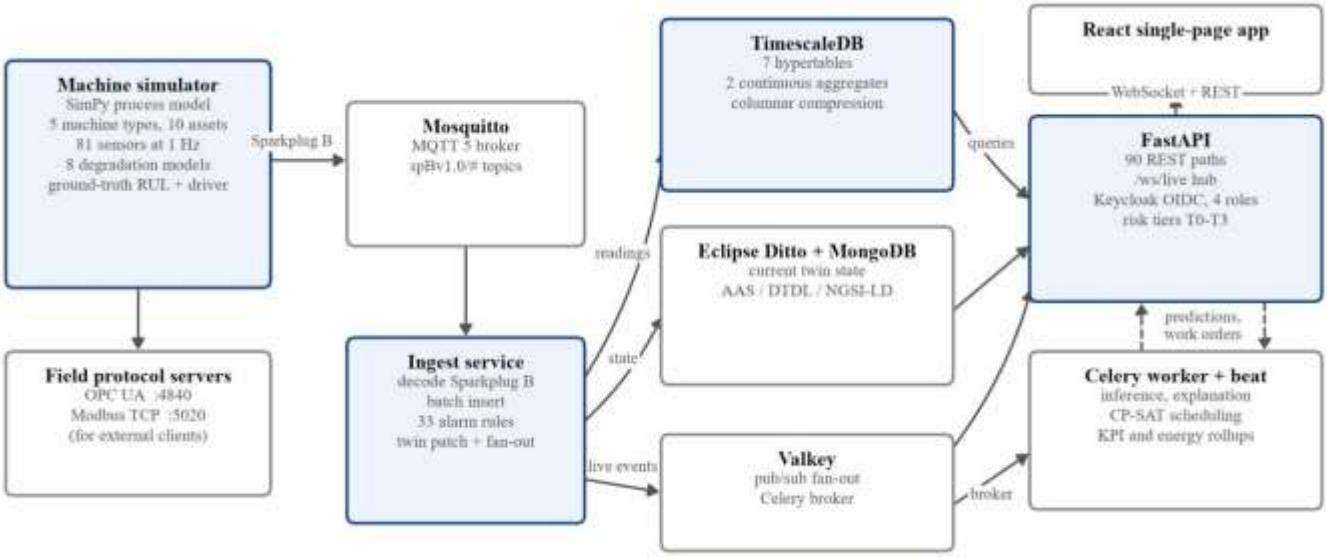


Fig. 1. Runtime data flow. Solid arrows carry data; dashed arrows carry derived records. Shaded boxes are components written for this platform; unshaded boxes are open-source infrastructure.

C. *Separating Current State From History*

The platform keeps two stores with different jobs. Eclipse Ditto holds what each asset is doing now, as a document per asset with access policies and change notifications; the TimescaleDB hypertable holds every reading that has ever arrived. The split follows the access patterns. Answering "what is the spindle temperature of CNC 01" is a point lookup against a small, frequently overwritten document. Answering "how did it behave last Tuesday" is a range scan over millions of immutable rows. Forcing both onto one engine makes one of them awkward. Keeping the twin in Ditto also supplies, without additional code, the standard serialisations an external system expects — Asset Administration Shell submodels, digital twin definition documents and NGSI-LD entities — along with per-asset authorisation policies. The cost of that convenience is quantified in Section VII-F, and it is the single largest line in the memory budget.

IV. Data Acquisition and Simulation

A. *Degradation Physics*

Each machine is a discrete-event process whose components carry a damage value in the unit interval. Damage advances under a process model chosen for the failure being represented. Fatigue crack growth in a spindle bearing follows a Paris-law formulation, in which crack length grows as a power of the stress intensity range [15], giving the accelerating trajectory characteristic of spalling. Monotone stochastic wear uses Wiener and gamma processes. Temperature-accelerated ageing uses an Arrhenius relation, so running hot shortens life in the way operators expect. Cutting-tool wear follows a Taylor tool-life relation against cutting speed. Separate models cover motor efficiency loss and the impulse trains that a defective rolling element injects into a vibration waveform.

Sensor channels are then functions of load, speed, ambient temperature and accumulated damage, perturbed by a noise model with independent Gaussian noise, slow drift, sporadic spikes, dropouts and timing jitter. Two properties make the output useful for supervised work: every sample is emitted together with the true remaining useful life of the component and the identity of the channel that is driving the degradation. A dataset with these labels cannot be obtained from a production line without waiting for machines to fail.

The fleet is configured as ten assets of five types across two lines, running three shifts over a six-day week with a per-shift production plan, which makes utilisation and energy consumption vary the way they do in a real schedule. Seven scenario files drive the fleet into specific conditions, such as a bearing failure on a named machine or a gradual rise in energy consumption.

B. *Protocol Surface*

Telemetry is published as Sparkplug B over MQTT 5, with one edge node per production line and the topic namespace the specification prescribes [18]. Using Sparkplug rather than ad hoc JSON means a subscriber that joins late can rebuild state from birth certificates rather than waiting for every channel to change. The simulator additionally exposes an OPC UA server and a Modbus TCP server carrying the same values, so that third-party industrial clients can attach to the fleet exactly as they would to real equipment.

C. *Ingest and Alarm Evaluation*

The ingest service decodes each payload, writes the readings to the hypertable in batches, patches the corresponding twin document, evaluates the alarm rules

and publishes a per-asset message to the fan-out channel. Rules compare a channel against a threshold with hysteresis and a deadband, which prevents a signal sitting on its limit from producing a stream of alarms. Alarms carry an explicit lifecycle from active, through acknowledged or shelved, to cleared, so that an operator decision is a recorded transition rather than a disappearance. Metric, asset and rule metadata are cached and refreshed on a sixty-second cycle, so that an engineer editing a threshold does not have to restart the service.

V. Analytics and Decision Support

A. Prediction Pipeline

Inference runs on a fixed cycle for every asset that has a model promoted to production. Features are computed over a sliding window and combine statistical descriptors — mean, standard deviation, extrema, root mean square, kurtosis, skewness and peak factor — with spectral descriptors such as spectral centroid and peak frequency, the latter being where bearing defects announce themselves before temperature responds. An Isolation Forest scores each window for departure from the learned normal region [6], and the score is mapped onto a bounded health index so that the number shown to an operator is comparable across machine types.

A gradient-boosted classifier assigns probability across the known failure modes [7]. Its outputs are calibrated by isotonic regression, because a probability that is only ordinally meaningful cannot be used as a threshold for raising work automatically. Remaining useful life is produced by a regressor wrapped in a conformal procedure that returns a prediction interval [13], [14]; the interval width, rather than the point estimate alone, is what the scheduler consumes. Models are versioned in a registry and move through candidate, staging and production stages, and only the production stage is served.

B. Explanation with Numeric Grounding

Every prediction can be expanded into an explanation object before any text exists. Tree Shapley values give per-feature contributions [8]. An explainable boosting machine trained on the same features provides an independent reading, and when the two disagree — measured as the overlap of their top three features falling below a fixed threshold — the disagreement is surfaced rather than suppressed. Counterfactual search is restricted to the variables an operator can actually change, so the advice returned is actionable rather than merely true [12].

Text is then generated from deterministic templates in English or Hindi. Where a language model is configured, it may only paraphrase that template, and its output is checked against the template before release: the ranking of cited features must agree, the direction of each effect must agree, every quantity must fall within a tight tolerance of the computed value, the vocabulary must stay within the known feature set, and any recommendation must be one the knowledge base contains. Text that fails any check is discarded and the template is used. The effect is that the sentence delivered to a technician cannot contain a number the pipeline did not compute. Operator feedback is captured alongside the explanation, and feedback naming a suspect channel raises a data-quality flag on that sensor in the same transaction.

C. Risk-Aware Scheduling

A work order raised from a calibrated failure probability has to be placed in time. The platform fits a Weibull distribution whose median equals the predicted remaining life and whose shape reflects the width of the conformal interval [16]; its cumulative distribution evaluated at a candidate slot is the probability that the machine fails before the repair starts. That quantity appears in three places at once: the colour of the bar an operator sees, the objective the solver minimises, and the figure returned when someone asks what delaying by a day would cost.

Scheduling itself is a constraint model solved by CP-SAT [22] over quarter-hour slots. Constraints cover technician availability and skills, one active order per production line and one per technician, and completion within the planning horizon. The objective combines priority-weighted waiting downtime, the probability of failure multiplied by its cost, the energy cost of the slot under the applicable tariff, and labour. A planner may drag an order manually, which triggers a conflict re-check rather than silently producing an infeasible plan.

D. Production and Energy Analytics

Availability, performance and quality are computed according to ISO 22400 and combined into overall equipment effectiveness, alongside mean time between failures and mean time to repair [20]. Each stored result carries the inputs it was derived from, so a figure on a dashboard can be traced to the counts and durations that produced it, which is what makes it defensible in a review. Energy analysis fits an ordinary least-squares baseline of consumption against units produced and flags one-sided residuals beyond three standard deviations, separating genuinely abnormal consumption from the expected rise when output rises. Aggregation is expressed in SQL against the time-series tables, so a month-long window is never transferred into application memory.

VI. Access Control and Tiered Actuation

Authentication is delegated to an OpenID Connect provider with four roles. Read access is broad, write access is restricted to engineering and administrative roles, and closing a work order is deliberately permitted to technicians, since that is the role that actually performs the repair. Every state change is written to an audit record.

The natural-language interface is where access control and safety meet. Thirteen intents are recognised, and each is bound to a risk tier. Tier zero covers questions and executes immediately. Tier one runs a simulation and changes nothing real. Tier two alters records — acknowledging an alarm, raising a work order — and executes only after the system reads the action back and the user confirms. Tier three commands a machine, and additionally requires a personal identification number and

an explicit deployment-level flag, so that a system installed for monitoring cannot be talked into actuation.

Two details matter more than the tier table. A pending confirmation is stored as a row rather than held in process memory, so it survives a page reload, a second browser tab and a restart of the interface, and it expires on its own. Parameters are re-validated at the moment of confirmation rather than at read-back, because the machine state that made a command safe may have changed during the seconds in between. Both choices follow from treating actuation as a distributed transaction rather than a user-interface flow.

VII. Experimental Evaluation

A. Environment and Method

All measurements were taken on one laptop with an Intel Core i5-12500H processor (12 cores, 16 threads) and 15.6 GB of memory, running Windows 11 with a container runtime granted 16 virtual CPUs and 7.57 GiB of memory. All fourteen services ran concurrently on that host during every measurement; nothing was moved aside to improve a figure.

Throughput was obtained by counting inserted rows across a measured wall-clock window. Query latency is the median of seven executions reported by the database's own analyser, with the first discarded to exclude cache warming. Request latency was measured from inside the container network with twenty-five samples per endpoint; Section VII-E explains why measuring from the host would have been misleading. Resident memory is the runtime's own per-container accounting at steady state. The corpus held 7.08 million readings captured over nine days, with a peak of 2.03 million in a single day.

B. Ingest Throughput and Delivery Latency

Table II reports the acquisition path. The fleet sustains 80.7 readings per second across 81 channels, with a separate energy stream of ten rows per second. Two delivery paths diverge after ingest. A reading becomes visible in the database after 508 ms at the median, because writes are batched. The same reading reaches a subscribed browser in 74 ms at the median and 136 ms at the 95th percentile, because the fan-out path does not wait for the write to commit. The live path is therefore roughly seven times faster than the durable one, which is the correct ordering for an operator display.

The fan-out also coalesces: the 81 individual channels arrive at the client as 10.1 messages per second, one per asset, with a median payload of 484 bytes carrying about nine metrics. That eightfold reduction in message count is what keeps a browser responsive while showing a whole line.

TABLE II. Ingest Throughput and Delivery Latency

Measurement	Value
Sustained telemetry rate	80.7 readings/s
Active sensor channels	81
Energy stream	9.99 rows/s
Database visibility, median	508 ms
Database visibility, range	155–1035 ms
Live push to client, median	74 ms
Live push, 95th percentile	136 ms
Live push, maximum (n = 401)	254 ms
Client message rate	10.1 msg/s
Median client payload	484 bytes

C. Storage Efficiency

Table III reports storage. The telemetry hypertable holds 7.08 million readings in 80.1 MiB, an average of 11.91 bytes per reading including indexes. Columnar compression, applied by policy to chunks older than seven days, reduced 1162 MB to 77 MB, a ratio of 15.2:1 over the eight chunks eligible at the time of measurement.

The contrast with the energy table makes the point sharply. That table was left uncompressed and holds 871,679 rows — eight times fewer — in 178.5 MiB, which is 2.2 times more disk, at 214.77 bytes per row. Extrapolating the compressed figure, continuous operation of this fleet would accumulate roughly 30 GB per year, which a single disk absorbs without a retention policy. Uncompressed, the same corpus would require about 540 GB, which would not.

TABLE III. Storage Footprint

Table	Rows	Size	B/row
telemetry (compressed)	7,080,235	80.1 MiB	11.91
energy_readings (uncompressed)	871,679	178.5 MiB	214.77
telemetry_1m aggregate	117,936	16 kB	—
telemetry_1h aggregate	2,592	24 kB	—
Whole database	—	295 MB	—

D. Query Latency

Table IV and Figure 2 compare scanning the raw hypertable against reading a continuous aggregate for the same question. Over the full retained range, a fleet-wide average takes 151.26 ms against the raw table and 1.68 ms against the hourly aggregate, a factor of 90. Over a 24-hour window the minute-level aggregate gives a factor of 46.6, and for a single channel over a week the factor is 34.9.

The pattern is that the benefit scales with the number of rows the raw query would have to touch, which is exactly the regime a dashboard operates in when an engineer widens a time range. Indexing alone does not produce this; the aggregate wins because the rows it reads were reduced before the query arrived.

TABLE IV. QUERY LATENCY: RAW HYPERTABLE VERSUS CONTINUOUS AGGREGATES

Query	Raw	Aggr.	Gain
Fleet, full retained range	151.26 ms	1.68 ms	90.0×
Fleet, 24 h window	27.04 ms	0.58 ms	46.6×
Single sensor, 7 d	2.44 ms	0.07 ms	34.9×
Single sensor, 24 h	1.95 ms	0.21 ms	9.3×
Fleet, full range, 1 min aggregate	151.26 ms	67.33 ms	2.2×

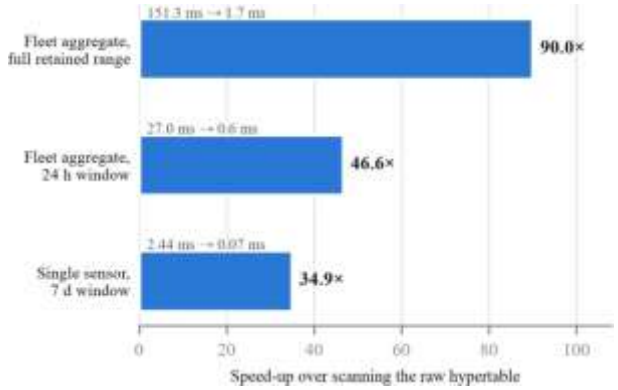


Fig. 2. Speed-up obtained by answering a query from a continuous aggregate instead of the raw hypertable. Absolute latencies are shown above each bar.

E. Request Latency

Table V reports server-side latency. Record-oriented endpoints answer in 13 to 28 ms at the median. Assembling the asset hierarchy costs 65.88 ms, since it walks the plant structure and merges twin state. The dependency health check, which contacts six services, returns in 24.73 ms.

The informative result is the gap between two endpoints that answer nearly the same question. Computing overall equipment effectiveness on demand over a seven-day window takes 180.74 ms, while reading the pre-computed rollup for the same period takes 16.72 ms — 10.8 times faster. The energy summary, which also aggregates on demand, behaves like the former at 182.91 ms. This is direct evidence for scheduling rollups rather than computing indicators per request, and it identifies precisely which endpoints should be converted as windows grow.

One measurement artefact is worth recording because it would otherwise corrupt every figure in this table. Measured from the Windows host rather than inside the container network, every endpoint showed a floor near 220 ms. Decomposing the timings showed that approximately 210 ms of it was connection establishment through the desktop runtime's port-forwarding proxy, with server processing accounting for about 28 ms. The overhead is per connection, not per request, so it disappears when a client reuses connections, as the single-page application does. Reporting host-observed numbers would have overstated the platform's latency by an order of magnitude.

TABLE V. SERVER-SIDE REQUEST LATENCY (N = 25 PER ENDPOINT)

Endpoint	Median	95th pct
Work orders	13.43 ms	14.51 ms
Downtime Pareto, 7 d	15.61 ms	26.24 ms
Key indicators, 7 d (pre-rolled)	16.72 ms	19.77 ms
Asset list	17.42 ms	20.43 ms
Alarms	20.31 ms	36.97 ms
Identity (token verification)	21.58 ms	28.41 ms
Health, six dependencies	24.73 ms	33.23 ms
Reliability, 7 d	27.82 ms	37.16 ms
Asset hierarchy	65.88 ms	82.87 ms
Equipment effectiveness, 7 d	180.74 ms	276.26 ms
Energy summary, 7 d	182.91 ms	279.02 ms

F. Memory Footprint

The fourteen services occupy 3.66 GiB resident at steady state, within the 7.57 GiB granted to the runtime. Table VI groups the total and Figure 3 breaks it down by service. The distribution is the most consequential measurement in this paper.

The four clustered Java services that make up the twin store account for 2.00 GiB, or 54.8 per cent of the total, each pinned to a 512 MiB ceiling and running at 97 to 99 per cent of it. The identity provider adds a further 591 MiB. Together, twin state and authentication consume 70.9 per cent of memory. The five services that carry out the actual domain work — acquisition, ingest, inference, scheduling and the interface — total 632 MiB, or 17.3 per cent, and the three data stores take 11.6 per cent.

For a platform intended to run on constrained hardware, this is where attention should go. The twin store earns its place through standard serialisations and policy handling rather than through computation, and substituting a lighter current-state representation would free more memory than optimising every service written for this platform combined.

TABLE VI. RESIDENT MEMORY BY SERVICE GROUP

Group	Memory	Share
Twin store (4 Ditto services)	2.00 GiB	54.8 %
Identity provider	591 MiB	16.2 %
Application services (5)	632 MiB	17.3 %
Data stores (3)	425 MiB	11.6 %
Message broker	3.7 MiB	0.1 %
Total	3.66 GiB	100 %

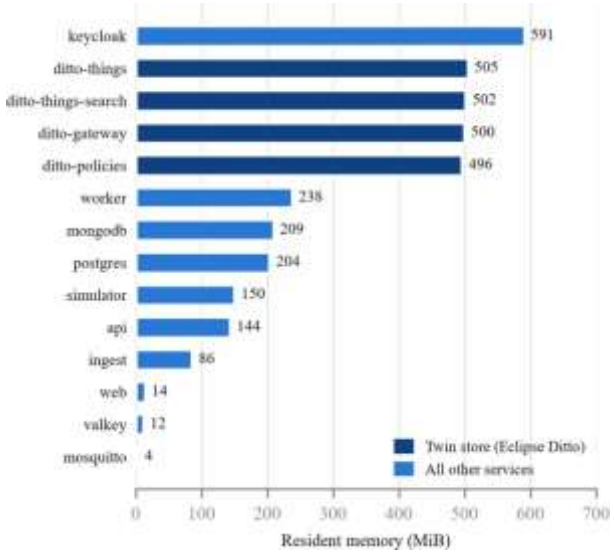


Fig. 3. Resident memory per service at steady state. The four clustered twin-store services hold more memory than every other service combined.

G. Functional Verification

Table VII reports the automated suites, all executed on the machine described above. The 221 backend tests run against a real PostgreSQL and TimescaleDB instance rather than an in-memory substitute, with migrations applied first, so schema and query behaviour are exercised as deployed; only the external services are replaced by stubs at the dependency boundary. All 450 tests pass.

Two of the nine simulator test modules could not be collected inside the production image, because they require packages that are deliberately absent from a runtime image, and were excluded from the run; the remaining seven modules supplied the 85 results reported.

TABLE VII. AUTOMATED TEST RESULTS

Suite	Tests	Result
Backend API (real database)	221	pass
Machine simulator	85	pass
Language understanding	56	pass
Sparkplug B codec	48	pass
Explanation library	40	pass
Total	450	pass

H. Implementation Scale

The platform comprises 29,322 lines of Python and 11,892 lines of TypeScript across 430 tracked files. The backend exposes 90 REST paths carrying 107 operations across eleven modules, over a schema of 50 tables built by nine migrations, of which seven are hypertables and two are continuous aggregates. The frontend is organised into ten feature areas. These figures are given so that the measurements above can be read against the size of the system that produced them.

VIII. DISCUSSION

A. What the Measurements Imply

The first implication is that a single commodity machine is sufficient for a small plant. At ten machines and 81 channels the limiting resource is not throughput, which has considerable headroom, but memory held by infrastructure that performs no analysis.

The second is that compression changes the retention decision rather than merely saving disk. At 11.91 bytes per reading, keeping a year of raw history is a 30 GB commitment, so the question of what to discard, which normally arrives early in such a project, can be deferred.

The third is that pre-computation, not indexing, is what keeps an interface responsive as the window widens. The 90-fold difference in Section VII-D and the 10.8-fold difference in Section VII-E are the same phenomenon at two layers, and both argue for materialising what will be asked repeatedly.

The fourth is methodological. The 210 ms of connection overhead introduced by the desktop runtime would, if left unattributed, have been reported as application latency. Measurements of containerised systems should be taken inside the deployment boundary, with the client-side path characterised separately.

B. Limitations

The data is simulated. The degradation models are drawn from established formulations and produce trajectories of the expected shape, but no claim is made that any distribution matches a specific machine in a specific plant. Validation against instrumented equipment is necessary before the prediction path can be trusted operationally.

This paper characterises the platform and does not evaluate prediction accuracy. That separation is deliberate, and we prefer to report it rather than present figures obtained on data the platform generated itself. We also record, for reproducibility, that the backend container image omits a shared library required by the gradient-boosting runtime, so the failure-mode and remaining-life components will not load until that library is added to the image; this is a packaging defect found during measurement and does not affect any figure reported here.

Capture was intermittent rather than continuous: the corpus covers nine days of operation rather than nine consecutive days, so the storage and query figures describe a working corpus rather than a year of production. The deployment is single-site and single-tenant, and no horizontal scaling was evaluated. Finally, several planned subsystems are specified but not built, among them a three-dimensional plant view, a speech front end for the language interface, an edge inference runner and an automated benchmark harness.

IX. CONCLUSION AND FUTURE WORK

We have described an industrial digital twin and predictive-maintenance platform built entirely from open components, covering acquisition over standard industrial protocols, separated stores for current state and history, an inference path whose explanations are audited against their own numbers, risk-aware scheduling by constraint solving, and an actuation model that binds natural-language commands to explicit risk tiers. The system runs as fourteen services on one commodity machine from a single command.

Measured on that machine, the platform holds 3.66 GiB resident, sustains 80.7 readings per second, delivers live updates in 74 ms at the median, stores readings at 11.91 bytes each under a compression ratio of 15.2:1, answers fleet-wide queries 90 times faster from continuous aggregates than from raw history, and passes 450 automated tests. The memory distribution identifies the clustered twin store, at 54.8 per cent of the total, as the component to reconsider for constrained deployments.

Four directions follow. The prediction path should be validated against instrumented machines, using the simulator's labelled output only for pre-training. The twin store should be compared against a lighter current-state representation under the same measurement protocol. The explanation audit should be evaluated with technicians rather than by construction, measuring whether an audited narration changes what a maintenance decision looks like. Finally, moving inference to the edge would reduce the volume crossing the network and allow the architecture to be measured under intermittent connectivity.

REFERENCES

[1] W. Kritzinger, M. Karner, G. Traar, J. Henjes, and W. Sihn, "Digital twin in manufacturing: A categorical literature review and classification," IFAC-PapersOnLine, vol. 51, no. 11, pp. 1016–1022, 2018.
[2] F. Tao, H. Zhang, A. Liu, and A. Y. C. Nee, "Digital twin in industry: State-of-the-art," IEEE Transactions on Industrial Informatics, vol. 15, no. 4, pp. 2405–2415, 2019.
[3] J. Lee, F. Wu, W. Zhao, M. Ghaffari, L. Liao, and D. Siegel, "Prognostics and health management design for rotary machinery systems — Reviews, methodology and applications," Mechanical Systems and Signal Processing, vol. 42, no. 1–2, pp. 314–334, 2014.
[4] T. Zonta, C. A. da Costa, R. da Rosa Righi, M. J. de Lima, E. S. da Trindade, and G. P. Li, "Predictive maintenance in the Industry 4.0: A systematic literature review," Computers and Industrial Engineering, vol. 150, art. 106889, 2020.
[5] A. Saxena, K. Goebel, D. Simon, and N. Eklund, "Damage propagation modeling for aircraft engine run-to-failure simulation," in Proc. International Conference on Prognostics and Health Management, 2008, pp. 1–9.
[6] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in Proc. 8th IEEE International Conference on Data Mining, 2008, pp. 413–422.
[7] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "LightGBM: A highly efficient gradient boosting decision tree," in Advances in Neural Information Processing Systems 30, 2017, pp. 3146–3154.
[8] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in Advances in Neural Information Processing Systems 30, 2017, pp. 4765–4774.
[9] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why should I trust you? Explaining the predictions of any classifier," in Proc. 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 1135–1144.
[10] H. Nori, S. Jenkins, P. Koch, and R. Caruana, "InterpretML: A unified framework for machine learning interpretability," arXiv:1909.09223, 2019.
[11] C. Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," Nature Machine Intelligence, vol. 1, pp. 206–215, 2019.
[12] R. K. Mothilal, A. Sharma, and C. Tan, "Explaining machine learning classifiers through diverse counterfactual explanations," in Proc. ACM Conference on Fairness, Accountability, and Transparency, 2020, pp. 607–617.
[13] V. Vovk, A. Gammerman, and G. Shafer, Algorithmic Learning in a Random World. New York, NY: Springer, 2005.
[14] Y. Romano, E. Patterson, and E. J. Candès, "Conformalized quantile regression," in Advances in Neural Information Processing Systems 32, 2019, pp. 3543–3553.
[15] P. C. Paris and F. Erdogan, "A critical analysis of crack propagation laws," Journal of Basic Engineering, vol. 85, no. 4, pp. 528–533, 1963.
[16] W. Weibull, "A statistical distribution function of wide applicability," Journal of Applied Mechanics, vol. 18, no. 3, pp. 293–297, 1951.
[17] International Electrotechnical Commission, IEC 62541: OPC Unified Architecture. Geneva, Switzerland: IEC.
[18] Eclipse Foundation, Sparkplug Specification, Version 3.0, 2022.
[19] OASIS, MQTT Version 5.0, OASIS Standard, 2019.
[20] International Organization for Standardization, ISO 22400-2: Automation Systems and Integration — Key Performance Indicators for Manufacturing Operations Management — Part 2: Definitions and Descriptions. Geneva, Switzerland: ISO, 2014.
[21] Plattform Industrie 4.0, Details of the Asset Administration Shell, Part 1, Version 3.0, 2020.
[22] L. Perron and F. Didier, CP-SAT Solver, Google OR-Tools. Available: developers.google.com/optimization/cp/cp_solver
[23] Eclipse Foundation, Eclipse Ditto: Digital Twin Framework. Available: eclipse.dev/ditto
[24] Timescale Inc., TimescaleDB: Time-Series Database for PostgreSQL. Available: timescale.com