IoT-Driven Real-Time Predictive Maintenance

Team Learner

1. Abhilasha Kawle – abhilashak@iisc.ac.in
2. Sangram Kumar Y – sangramyerra@iisc.ac.in
3. Siddaraju D H – siddarajuh@iisc.ac.in
4. Venturi Naveen – naveen1@iisc.ac.in

Definition

Industry 4.0 enhances efficiency, flexibility, and competitiveness in manufacturing by using technologies like IoT, AI, big data, and automation. As markets demand faster, more customized, and higher-quality production, smart factories help monitor and optimize operations in real time. This reduces downtime and costs through predictive maintenance, while enabling faster decisions and sustainable, agile production—making it essential for staying competitive

Problem Motivation

Predictive maintenance is crucial in asset-heavy industries where equipment failure can cause major losses. It relies on thousands of IoT sensors generating high-speed, real-time data, which is analyzed by AI models to predict failures in advance. Predictive maintenance is now required across varied industries making this a scalable Big Data storage and Processing problem statement.

- *Automotive Industry*: Real-time monitoring of robots, motors, conveyors, assembly lines
- *Electronics & Semiconductors*: Monitor cleanroom equipment, chip manufacturing machines
- *Pharmaceuticals & Healthcare*: Monitoring production lines, HVAC systems, and sterilization equipment
- *FMCG & Food Processing*: Monitor mixers, packaging lines, and refrigeration units.
- *Energy & Utilities*: Monitor turbines, transformers, pipelines, and compressors

Steps involved in Predictive Maintenance

- Sensors monitor machine temperature, vibration, oil levels, etc.
- Data is streamed in real-time to a processing system
- Models analyze patterns to predict failures before they happen.
- Alerts are generated to schedule maintenance, avoiding costly downtime.

Goals

Develop a Storage, streaming and Processing architecture for varied sensor data inputs and with outputs as monitoring parameters on dashboards and alerts for predictive maintenance.

Features

Showcase scalability of above architecture with

- Varied sensor Inputs to support varied Industries
- Adaptability of Storage, Processing based on input size

Approach

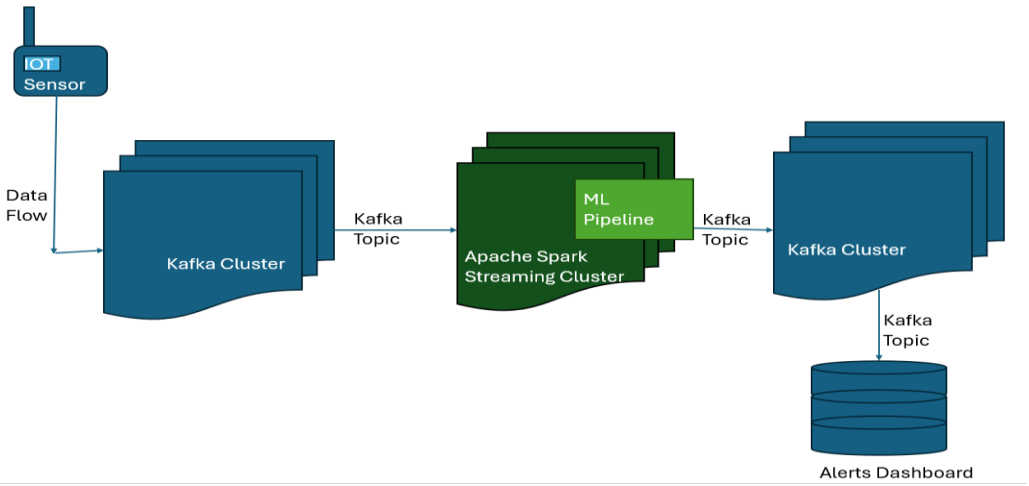
Proof of Concept

For implementation, we are show casing predictive maintenance of 7 water pump machines. Each water pump is fitted with 52 sensors. The project involves monitoring real time data from these sensors and pre-trained ML predictive maintenance algorithms predictive healthy hours of the water pump and generating appropriate alerts when pumps have healthy hours of <24hrs. The high level block diagram and big data platforms for implementation at each architecture stage is as shown in figure1.1. Scalability would be showcased by proving that system scales when a new machine is added. And re-training of ML model would support scalability towards sensor type.

Datasets/Data Model

- Dataset - [kaggle water-pump-rul-predictive-maintenance dataset]
- Models from spark ML : Linear Regression
- Regression output – 'hours_remaining_healthy";
- Classifier output – "machine_failure_in_24hrs"
- Data Models – Linear Regression, GBTRegressor, Randomforest regressor & classifier
- Success Matrix : R2 > 95% for Regression; Accuracy > 95% for Classifier

APPROACH



IoT Sensors:

Collects real-time operational data (temperature, vibration, pressure, etc.) from industrial machines or assets. These devices send data, typically in JSON or CSV format, to the Kafka cluster. The arrows from sensors to Kafka represent streaming data transmission—the gateway may batch or relay data securely to Kafka topics.

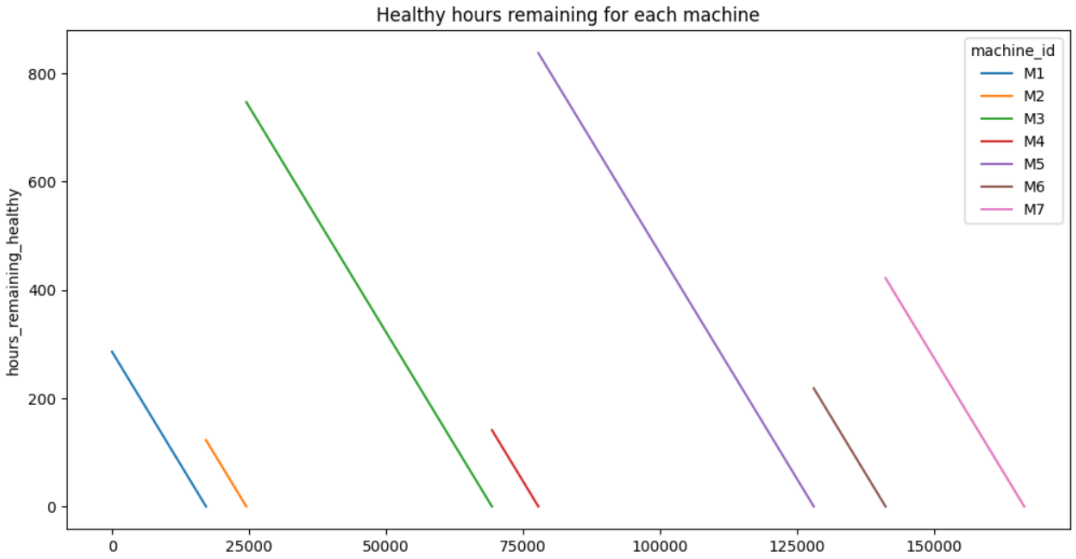
In this project we have picked dataset with 50 sensors on each water pump. And the data is tracked for 7 machines. We are selecting rows from dataset for particular machine to mimic streaming from sensors.

The sensors data is towards predicting healthy hours of the water pump before its failure. Each machine shows declining number of hours from normal hours to 0 hours. We are setting "FAILURE" alter at 24th hours of remaining healthy.

Dataset Preprocessing*:*

1. The data was inspected for NULLS – No NULLs were present
2. The datatype for sensors was "string", Converted to 'double' for use into ML training for prediction regression.
3. All the sensor data was rounded to 3 decimals for easy readability and computation
4. Added "machine_failure_in_24hrs" = 0 for hours_remaining_healthy > 24hrs
   = 1 for hours_remaining_healthy <=24hrs

This processed data is used for streaming and ML training

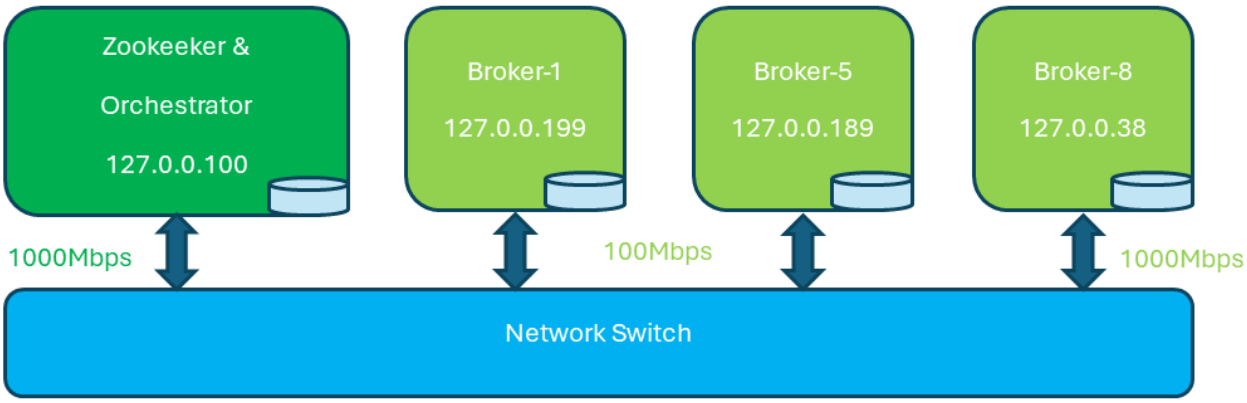


Data for “Healthy hours remaining” for each Machine

KAFKA CLUSTER:

The Kafka cluster is setup using 3 Intel XEON systems running RHEL 8+. The machines are connected through a backend network-switch of varied connection speeds ranging from 100Mbps to 1000Mbps.

Apache Kafka v3.6.2 has been chosen for OS & library compatibility reasons. For providing resilience on broker machines, a systemd service has been created for launching and registering the Kafka Brokers with the cluster. The service has been enabled to run on bootup, right after initializing the networking services.

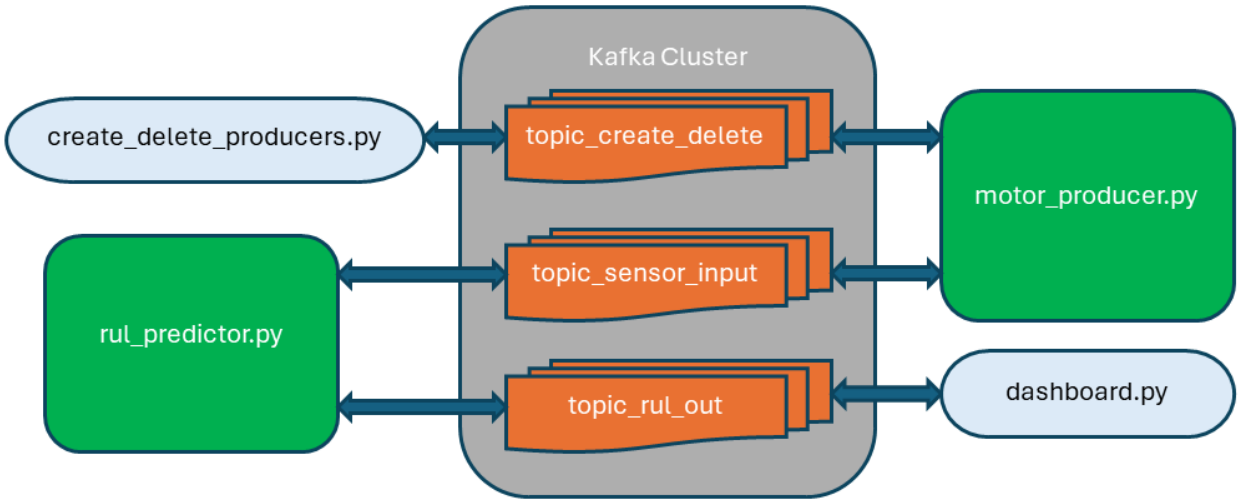


Kafka-cluster setup

Running the producer perf test confirms that the maximum producer throughput to the cluster will be bottlenecked by the slowest link that handles the entire data stream.

7043 records sent, 1408.6 records/sec (11.00 MB/sec), 1455.0 ms avg latency, 1456.0 ms max latency.
7041 records sent, 1408.2 records/sec (11.00 MB/sec), 1455.0 ms avg latency, 1456.0 ms max latency.

To address linear scalability, each motor produces its sensor data into a unique partition on the topic_sensor_input. With this approach, the consumer gets benefited by exploiting Kafka’s built-in load-balancer by joining a common consumer group and subscribing to the rebalance-listener events. This also helps isolate consumer failures to specific motor instead of randomly affecting all motors datapath.



Application interface with Kafka-cluster

About the disk usage and topic log retention, the lowest disk size available on the brokers is about 460GB. The longest message that flows on the topic_sensor_input is about 1300 bytes and accounting 1msg/second from a single motor will total about 750MB for the default log retention duration of 7days.

Spark Streaming Cluster

Apache Spark Structured Streaming powers the real-time analytics engine for IoT-driven predictive maintenance. It continuously consumes live sensor data in micro-batches, performs ML-based inference from pre-trained models loaded from HDFS and streams actionable results for monitoring and alerting with low latency.

Key Streaming Workflow

- Ingest real time sensor messages from incoming Kafka Topic
- Decode JSON payload using a pre-defined Schema
- Apply Stateful Running averages per machine to handle missing values
- Generate ML Model friendly feature vectors using Vector Assembler
- Load pre trained Spark ML Model using HDFS:
    - Regression Model -> Predicts remaining_healthy_hours
    - Classification Model -> Predicts Machine Status
- Post -process output by mapping binary predictions into labels
- Stream predictions into out going Kafka topic for dashboard and alerts

Fault Tolerance & Reliability

- Kafka + Spark Streaming guarantees at-least-once data delivery
- Checkpointing enabled for recovery and exactly-once state consistency
- Configure 5-second streaming trigger interval for near real-time responsiveness

Scalability Features

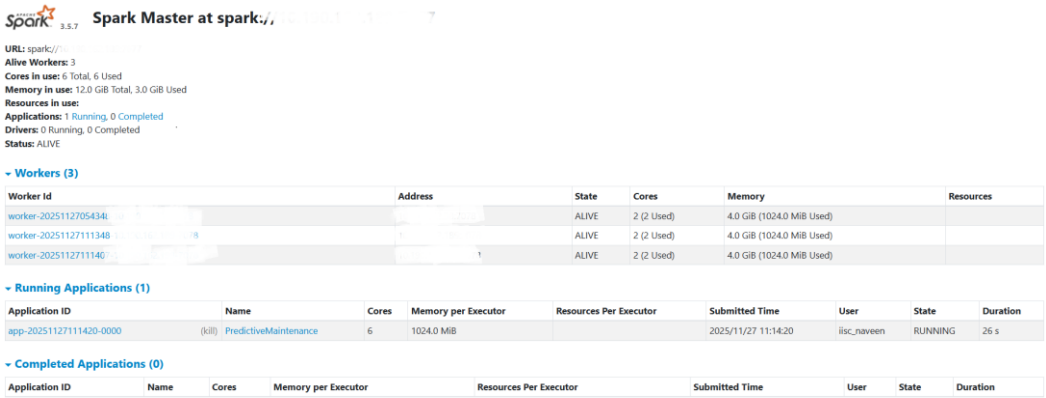
- Distributed processing architecture supports increasing number of machines
- Easily adapts to different industries and new sensor configurations
- Computation is horizontally scalable across Spark Worker Nodes for higher throughput
- ML Model updating can be done without any down time of the application as model can be loaded on the fly

Spark Performance

Spark Structured Streaming provides detailed performance insights for every micro-batch through its built-in Streaming Query Progress events. For each batch, Spark emits a structured JSON snapshot containing metrics such as input rows, processing time, batch duration, and throughput. These metrics help analyze how the system scales with the number of producers and worker nodes.

During large-scale ingestion, we observed that Spark automatically distributed load across the 3-worker cluster, and processing time per batch reduced as more executors became available. The performance logs also show consistent input rates and stable end-to-end latency, indicating efficient backpressure handling.

Spark UI was used to monitor executor-level details like task execution time, shuffle behavior, memory usage, and job DAGs. This helped validate that the cluster was effectively utilizing resources and scaling with the data volume.



The above figure depicts the spark scaling to 3 worker nodes with each running 2 cores at 4GB RAM

Throughput Observed	: 400 rows/sec
Batch Latency Observed	: 1k -> 2.5s
Executor CPU Memory	: 1GB/executor

ML Pipeline

- SparML has been used for ML model selection.
- For ML training we are ignoring the machine_id and timestamp columns so that we have uncorrelated data for ML training and validation

EDA Insights:

- We started first with understanding the correlation between sensors to target column to ensure they have correlation with the target column.
- Correlation Matrix

sensor_00 → correlation = 0.05912114471592276	sensor_26 → correlation = 0.045197749024761175
sensor_01 → correlation = 0.10540636463422501	sensor_27 → correlation = -0.0245719821589028
sensor_02 → correlation = 0.12300591002615685	sensor_28 → correlation = 0.09464004022724334
sensor_03 → correlation = 0.04041734522601546	sensor_29 → correlation = 0.2250137744266765
sensor_04 → correlation = 0.01573231841868516	sensor_30 → correlation = -0.061478137485349776
sensor_05 → correlation = -0.13682952984536964	sensor_31 → correlation = 0.043659078397549955
sensor_06 → correlation = -0.1223092882115208	sensor_32 → correlation = 0.0822291422715112
sensor_07 → correlation = -0.11858716896761821	sensor_33 → correlation = 0.06570889756690317
sensor_08 → correlation = -0.06647880648566623	sensor_34 → correlation = -0.027322528191321336
sensor_09 → correlation = -0.13483941648913783	sensor_35 → correlation = -0.07117765266982991
sensor_10 → correlation = 0.04008874762219254	sensor_36 → correlation = -0.06981219833353816
sensor_11 → correlation = -0.12257091315839416	sensor_37 → correlation = 0.17717208122736233
sensor_12 → correlation = -0.015098263031566205	sensor_38 → correlation = 0.06414954265040392

sensor_13 → correlation = -0.27694254394872486 sensor_14 → correlation = 0.09776902426077916 sensor_16 → correlation = 0.09473561596304655 sensor_17 → correlation = 0.11385662864394683 sensor_18 → correlation = 0.11758259600171464 sensor_19 → correlation = 0.09682180559531385 sensor_20 → correlation = 0.09520047730795352 sensor_21 → correlation = 0.09350522271199155 sensor_22 → correlation = 0.09467056885479816 sensor_23 → correlation = 0.03917902242453107 sensor_24 → correlation = 0.11287912332029659 sensor_25 → correlation = 0.08817721285730835	sensor_39 → correlation = 0.10373973759517595 sensor_40 → correlation = -0.05556162116656311 sensor_41 → correlation = 0.14391186031610578 sensor_42 → correlation = 0.09921633044535491 sensor_43 → correlation = 0.03481323313245766 sensor_44 → correlation = -0.07489907242852385 sensor_45 → correlation = -0.012358041788517177 sensor_46 → correlation = 0.02967172977929997 sensor_47 → correlation = -0.007274757265420465 sensor_48 → correlation = 0.028934483525286332 sensor_49 → correlation = 0.013345553534335444 sensor_51 → correlation = -0.08523234119919791

- The maximum correlation is with sensor 13 with negative sign. This means the sensor data is increasing as remaining healthy hours decrease.
- Next the skewness of the sensor data was checked.
- Some sensors showed strong skewness > 10. But the correlation factor for these sensors was low so the skew transformation was not applied

Top 5 Sensors with high correlation	Correlation factor	Skew
Sensor 13	(-)0.276	1.55
Sensor 29	0.225	-0.9
Sensor 37	0.177	-0.22
Sensor 41	0.1439	8.5
Sensor 05	(-)0.136	-2.688

ML Data transformation:

- Vector assembler of numerical columns containing sensor data. There is no categorical feature in this dataset.
- Standard scaler on vector – For normalizing the sensor data as we don't have information on type of sensors in this dataset.

Model performance:

Spark ML Model	R2 / Accuracy
Ridge L2 Regression	0.444
Random Forest Regressor (Trees=50, Depth=12)	0.991
GBT Regressor (Depth=5)	0.917
Random Forest classifier (Trees=50, Depth=12)	0.99

ML Model Performance comparison

- For the given dataset, the target column shows linear characteristics. This could be the reason for R2/accuracy ~ 0.99
- This sparkML model was saved for loading into kafka topics for prediction over streaming data.

Alerts Dashboard

Real-time monitoring and visualization of Remaining Useful Life (RUL) predictions for industrial machine fleet, enabling proactive maintenance decisions and operational efficiency.

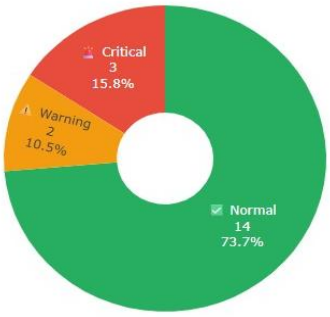
Data Source	Kafka Topic (topic_predictions_out)	Real-time RUL predictions
Backend	Python + confluent-kafka	Message consumption & processing
Frontend	Dash + Plotly	Interactive web dashboard
Storage	In-memory (deque)	Temporary data storage (1000 points)
Refresh Rate	3 seconds	Live dashboard updates

Key Features

- Real-time Data Ingestion: Consumes Kafka messages every 3 seconds
- Fleet Health Visualization: Interactive pie chart showing critical/warning/normal status distribution
- Trend Analysis: RUL progression over time for individual machines
- Prioritized Alerts: Color-coded table sorted by criticality (lowest RUL first)
- Automated Status Classification: Critical (<50), Warning (50-100), Normal (≥100)

Dashboard Components

1. Fleet Health Status (Pie Chart)



- Visual: Donut chart with status distribution
- Colors: Critical, Warning, Normal
- Special Case: Full green when all machines normal

2. Machine RUL Trends (Line Chart)

- Visual: Multi-line time series (last 10 machines)
- Features: Color-coded by status, threshold lines at 50 & 100
- Purpose: Identify deteriorating vs improving machines

3. Machine Status Table

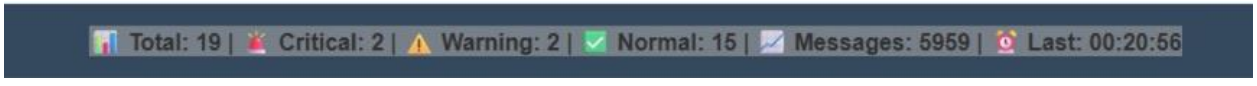
Machine Status Details

Machine ID	RUL Value	Status	Last Update
Machine 23	32.84	critical	2018-04-17 07:40:00
Machine 21	51.67	warning	2018-04-15 22:30:00
Machine 22	55.09	warning	2018-04-16 14:46:00
Machine 20	61.63	warning	2018-04-15 05:47:00
Machine 40	473.41	normal	2018-04-29 02:19:00
Machine 38	515.32	normal	2018-04-27 17:37:00
Machine 37	516.2	normal	2018-04-27 00:27:00

- Data: Machine ID, RUL value, status, timestamp
- Sorting: Automatic by RUL (critical first)
- Highlighting: Row colors match status severity

4. Statistics Bar

- Metrics: Total machines, critical count, warning count, message count
- Updates: Real-time every 3 seconds



Business Value

- Cost Savings: Prevent catastrophic failures
- Operational Efficiency: Real-time visibility into fleet health
- Data-Driven Decisions: Replace reactive with predictive maintenance