Storage & Data Pipelines for IoT

This lecture addresses the core challenge of IoT: transforming raw, voluminous device data into actionable intelligence. We will explore the specialized storage, processing, and visualization technologies required to handle IoT's unique data characteristics from the edge to the cloud.

1

IoT Data Characteristics

Understanding time-series data, cardinality, and velocity.

2

Specialised Storage

Time-Series Databases (TSDBs) vs. NoSQL vs. Relational.

3

Stream Processing

Ingesting and processing high-velocity data with tools like Kafka.

4

Data Lifecycle Management

Implementing retention, downsampling, and archival policies.

5

Edge Processing Paradigms

Architecting for low latency, bandwidth efficiency, and resilience.

6

Orchestration & Management

Deploying and managing applications at the edge.

7

Visualization & Analytics

Building effective dashboards and implementing real-time alerting.

The Unique Nature of IoT Data

IoT data possesses distinct characteristics that demand specialized data management strategies, diverging from traditional web or transactional data. Understanding these attributes is crucial for selecting the appropriate architectural components.

Time-Series Nature

Data is a sequence of data points indexed in time order. The timestamp is the primary key and the basis for most queries.

High Cardinality

A massive number of unique data series, driven by the combination of `device_id`, `sensor_type`, `location`, and other tag sets.

High Velocity & Volume

Millions of devices can generate terabytes of telemetry data daily, requiring high write throughput and scalable storage.

Predominantly Append-Only

Data is written once and rarely updated; reads are typically for time-range queries and aggregations rather than individual record lookups.

> **Implication:** Traditional Relational Database Management Systems (RDBMS) struggle with the write-throughput, storage efficiency, and specific query patterns of this data, leading to the rise of Time-Series Databases (TSDBs).

Time-Series Databases (TSDBs)

TSDBs are specifically engineered for storing and retrieving time-stamped data, offering significant performance and efficiency advantages for typical IoT workloads compared to general-purpose databases.

Optimised for Temporal Data

Efficient Compression

Algorithms such as Gorilla compression for floats and delta-of-delta encoding for timestamps dramatically reduce the storage footprint and increase query speeds.

Specialised Query Languages

TSDBs employ languages (e.g., InfluxQL, Flux) optimized for native time-series operations, such as calculating moving averages (`MOVING_AVERAGE()`) or differences (`DIFFERENCE()`).

Time-Partitioning (Sharding)

Data is automatically organized by time (e.g., daily partitions). This makes large time-range queries and data retention policy enforcement (purging old data) highly efficient.

Continuous Queries & Rollups

The ability to pre-compute aggregations in the background (e.g., rolling up 1-second raw data into 1-minute averages) is key for optimizing long-term storage and dashboard performance.

Conclusion: TSDBs (e.g., InfluxDB, TimescaleDB) are the de facto standard for storing raw telemetry from sensors, devices, and applications due to their optimization for high-volume, time-indexed data.

NoSQL vs. Relational Trade-offs for IoT

An IoT data architecture often necessitates a polyglot persistence approach, using different database types optimized for distinct data use cases beyond just raw telemetry.

Time-Series DB (InfluxDB)	High write throughput, efficient time-range queries, excellent compression.	**Primary telemetry storage.** Sensor data, operational metrics.
Document Store (MongoDB)	Flexible schema, support for complex JSON structures, horizontal scaling.	**Device metadata.** Storing variable device profiles, configuration settings, and state.
Wide-Column Store (Cassandra)	Linear scalability, high availability, excellent write performance at global scale.	**Event data & time-series at massive scale.** Ideal for global, multi-region deployments requiring extreme resilience.
Relational DB (PostgreSQL)	ACID transactions, complex joins, strong consistency, mature tooling.	**Business data integration.** Linking device data to customer, order, and physical asset records in an Enterprise Resource Planning (ERP) system.

The selection of the database type is dictated by the specific query and write patterns required for each layer of the IoT solution, moving from raw telemetry to aggregated and contextual data.

Message Queues & Stream Processing

Before data reaches its permanent storage destination, it must be reliably ingested and processed in real-time. This crucial stage is managed by highly scalable message queues and dedicated stream processing engines.

Apache Kafka as a Data Backbone

Kafka serves as the central nervous system for high-velocity data streams in modern IoT architectures:

Decoupled Ingestion

Devices and gateways publish data to Kafka topics. Downstream consumers (storage, analytics, microservices) subscribe independently, ensuring system resilience and modularity.

Buffering & Backpressure

Kafka acts as a massive, durable buffer. It absorbs sudden traffic spikes (bursts of device reporting) and prevents slower downstream systems from becoming overwhelmed by maintaining a persistent log.

Stream Processing Patterns

Utilising tools like Kafka Streams or Apache Flink, data is processed in motion:

1

Filtering & Enrichment

Filtering out redundant noise (e.g., zero-change readings) and enriching data with static metadata (e.g., adding a machine's manufacturing date to a sensor reading).

2

Windowed Aggregations

Computing real-time statistical summaries over defined time windows, such as calculating the "average temperature per machine over the last 5 minutes" using sliding or tumbling windows.

3

Complex Event Processing (CEP)

Detecting specific sequences of events or correlations across multiple, heterogeneous data streams to trigger immediate, high-value business or operational alerts.

Data Lifecycle: Retention, Downsampling & Archival

Storing every raw data point indefinitely is cost-prohibitive. Effective IoT architectures implement tiered data management to optimize cost, performance, and utility across the data's lifespan.

The Three-Stage Data Lifecycle

Hot Storage (TSDB)

- **Content:** Raw, high-resolution telemetry data.
- **Retention:** Short-term (e.g., 30 to 90 days).
- **Purpose:** Real-time monitoring, detailed debugging, and immediate analysis.

Warm Storage (Downsampled TSDB)

- **Content:** Aggregated data (e.g., 1-minute averages rolled up from raw seconds).
- **Retention:** Medium-term (e.g., 6 months to 2 years).
- **Purpose:** Historical trend analysis, dashboard reporting, and machine learning model training. Reduces storage footprint by typically over 90%.

Cold Storage (Object Store)

- **Content:** Archived raw or downsampled data.
- **Retention:** Long-term archival or regulatory compliance periods.
- **Purpose:** Cost-effective, durable storage using services like AWS S3 Glacier or Azure Archive Storage, with infrequent access.

This tiered approach ensures that high-value, recent data is instantly accessible, while older data is moved to cheaper storage, effectively balancing operational needs with financial constraints.

Edge Processing: Motivations & Architecture

Edge computing moves processing power closer to the data source, satisfying stringent operational requirements that cloud-only architectures cannot meet.

Core Motivations for Edge Compute

Ultra-Low Latency

Closed-loop control systems require response times in the single-digit millisecond range, making cloud round-trips impractical for mission-critical operations.

Bandwidth Efficiency

Pre-processing and filtering massive streams of raw data at the edge significantly reduces the volume of data transmitted over expensive or constrained Wide Area Networks (WANs).

Operational Resilience

Systems must maintain core functions and data collection capabilities autonomously, ensuring continued operation even during intermittent or complete network outages.

Data Privacy & Sovereignty

Sensitive data (e.g., video, personnel tracking) can be processed and anonymized locally, complying with local data protection regulations before any derived insights are sent to the cloud.

Edge Architecture Patterns

Modern edge deployments leverage cloud-native patterns for consistency and manageability:

- **Microservices on Edge:** Applications are decomposed into small, independent services (e.g., a "vibration analysis" service, a "protocol translation" service).
- **Containerization:** Using Docker/OCI containers for packaging ensures that applications run identically in the cloud and at the edge. Management is handled by lightweight orchestrators like K3s.
- **Gateway Offload:** The edge gateway centralises functions like protocol translation (e.g., Modbus to MQTT), security, and the execution environment for containerized applications.

Lightweight Analytics & Inference at the Edge

The ability to perform analytics and Machine Learning (ML) inference varies widely based on the compute capability of the edge node.

Microcontroller (MCU)	KiloBytes of RAM, MHz clock.	Simple rule-based alerts (e.g., if temp > threshold), fast Fourier transform (FFT) for vibration analysis, or highly optimized models (TensorFlow Lite Micro).
High-End MCU / MPU (ESP32, Cortex-A)	MBs of RAM, hundreds of MHz.	Running pre-trained TensorFlow Lite Micro models for acoustic event detection or basic sensor anomaly classification.
Single-Board Computer (Raspberry Pi, Industrial PC)	GBs of RAM, multi-core GHz.	Full TensorFlow/PyTorch models for complex computer vision (e.g., object detection), lightweight stream processing using containerized functions.

Key Concept: Train in the Cloud, Infer at the Edge. Complex ML models are developed and trained on aggregated data in the cloud, then aggressively optimized, quantized, and deployed to the constrained edge environment for low-latency prediction and decision-making.

Edge Orchestration & Management Challenges

Managing distributed, diverse, and often physically inaccessible edge infrastructure introduces significant operational complexity compared to centralized cloud environments.

Critical Management Domains

1

Resource Scheduling

Tools like K3s (lightweight Kubernetes) manage containerized applications across the fleet, handling resource constraints, node failures, and zero-downtime updates in a highly decentralized manner.

2

Update Mechanisms

Requires secure, robust Over-The-Air (OTA) update processes for both the operating system and application containers, often utilizing A/B partitions for reliable, safe rollback capabilities.

3

Physical Security

As nodes are in untrusted physical locations, hardware-based security features (e.g., Trusted Platform Modules or Hardware Security Modules) are required for secure boot and device identity.

4

Remote Manageability

The ability to remotely monitor health, logs, performance, and apply patches to thousands of geographically dispersed nodes must be implemented using secure, resilient communication channels.

Achieving operational efficiency in large-scale IoT requires treating the edge fleet as an extension of the cloud infrastructure, managed uniformly via centralized orchestration platforms.

Visualization, Alerting & UX for IoT

The ultimate objective of the data pipeline is to deliver actionable intelligence. This relies on effective visualization and proactive, real-time alerting systems.

Real-Time Visualization Tools

Grafana

The industry standard for time-series dashboarding. It connects directly to TSDBs (InfluxDB, Prometheus) and supports complex query execution for visual rendering.

ThingsBoard

An integrated IoT platform that combines device management, data acquisition, rule-based alerting, and built-in visualization capabilities.

Alerting & Anomaly Detection

Rule Engines

Define simple threshold-based rules: "IF pressure > 100 PSI FOR 5s THEN trigger 'CriticalAlert' via SMS/Slack."

Anomaly Detection

Moving beyond static rules to using statistical models (e.g., z-score, standard deviation) or deployed ML models to automatically identify unusual patterns in the data stream.

User Experience (UX) Considerations

- **Dashboard Latency:** Use pre-aggregated data and appropriate refresh rates to ensure a responsive and "snappy" user experience, especially with real-time data feeds.
- **Backend for Frontend (BFF):** Employ dedicated API layers for dashboards to aggregate data from multiple microservices and databases, tailoring the data structure precisely for the specific UI requirements.