Architecture & Design Document: IoT-Based Mining Site Monitoring System

1. Introduction

This document outlines the architecture and design of an IoT-based monitoring system for mining sites, covering both underground and surface operations. The system collects telemetry data from vehicles, fixed assets, environmental sensors, and personnel safety devices, enabling real-time monitoring, fault detection, and analytics. The architecture leverages AWS IoT services, edge computing, and robust communication protocols to ensure scalability, reliability, security, and observability. The system supports future integration with Digital Twin implementations using AWS IoT TwinMaker.

The design aligns with the requirements for handling hundreds of vehicles and thousands of sensors, ensuring reliable data delivery in harsh mining environments with intermittent connectivity, secure communications, and comprehensive monitoring and analytics capabilities.

2. System Overview

The system is designed to collect, process, and analyze telemetry data from a mining site, simulating realistic scenarios for testing IoT architectures. It supports four asset types and corresponding sensors:

- **Sensors**: Each sensor (child node) uses an RTC clock for ISO 8601 timestamps (e.g. 2025-08-25T20:44:46.014422Z). Connectivity options include:
    - **Vehicles**: LoRa, WiFi, Cellular. ((haul trucks, loaders, excavators): GPS, speed, engine temperature, vibration, fuel level.)
    - **Fixed Assets**: Modbus/RS485, Zigbee, WiFi, LoRa. (fans, ventilation systems, conveyors): Vibration, motor temperature, operational status, airflow rate, power consumption
    - **Environmental Sensors:** LoRa, RF433, Modbus RS485, Modbus IP(CO/CO_2, methane, dust, temperature/humidity, ground stability): Gas levels, dust concentration, temperature, humidity, ground vibration.
    - **Personnel: BLE, Zigbee.** (wearable BLE devices): CO exposure, proximity alerts, heart rate, location, fall detection.
    - Each sensor (child node) includes an RTC clock to provide accurate timestamps in JSON payloads, as shown below


```
{
  "device_id": "env_methane_000",
  "type": "env",
  "timestamp": "2025-08-25T20:44:46.014422Z",
  "metrics": {
    "co_co2": 20.497004949089806,
    "methane": 1.1515705346487894,
    "dust": 73.55341925986316,
    "temp_hum": { "temp": 17.958559265331747, "hum": 48.05138085617887 },
    "ground_vib": 0.49057599310378974
  },
  "status": "normal"
}
```


- **Parent Nodes**: Data acquisition devices aggregate sensor data, supporting wired (Modbus/RS485) and wireless (LoRa, WiFi, BLE, Zigbee) protocols.

- **Gateways**: LoRaWAN gateways for long-range, WiFi/Ethernet for high-bandwidth, and cellular (4G/5G) for remote sites. Base64-encoded JSON ensures compatibility.
- **Edge Compute**: Raspberry Pi or NVIDIA Jetson running AWS IoT Greengrass:
    - Local MQTT broker (Mosquitto) for edge communication.
    - Offline buffering in SQLite or file-based queues (as in synthetic_generator.py intermittent mode).
    - DLQ via Greengrass SQS integration for failed messages.
    - Local Lambda functions for preprocessing (e.g. filtering invalid data).
    - ML models (e.g. SageMaker Edge Manager) for anomaly detection (e.g. vibration spikes).
    - Local IoT rules for immediate alerts (e.g. methane >5ppm).
- **Cloud Ingestion**: AWS IoT Core handles device authentication (X.509), MQTT publishing (QoS 1), and rules-based routing to Kinesis, SNS, or Lambda.
- **Processing & Storage**:
    - Kinesis Data Streams (On-Demand) for scalable ingestion.
    - AWS IoT Analytics for data aggregation and insights.
    - Amazon Timestream for time-series storage (e.g. 5-second telemetry).
    - Amazon S3 for raw data and analytics outputs, with Athena for querying.
    - AWS Lambda for stream processing and rule evaluation.
- **AI/ML**: AWS SageMaker for training anomaly detection models (e.g. high methane or vibration) and Lambda for inference.
- **Monitoring & Alerting**:
    - CloudWatch for metrics (e.g. device uptime) and logs.
    - SNS for notifications (email, SMS, push).
    - Grafana dashboards for real-time visualization (e.g. ground vibration trends).
- **Applications**:
    - Custom web/mobile apps for operator interfaces.
    - AWS QuickSight for analytics dashboards.
    - AWS IoT TwinMaker for Digital Twin visualization of mining site assets.

3. High-Level Architecture (attached as Image and Draw.io XML)

The architecture follows a layered approach, integrating edge devices, gateways, cloud ingestion, processing, storage, and monitoring components, as depicted in the provided diagram.

3.1 Architecture Diagram

The high-level architecture diagram illustrates the data flow from edge devices to cloud services across multiple mining sites (Site 1, Site 2, etc.):

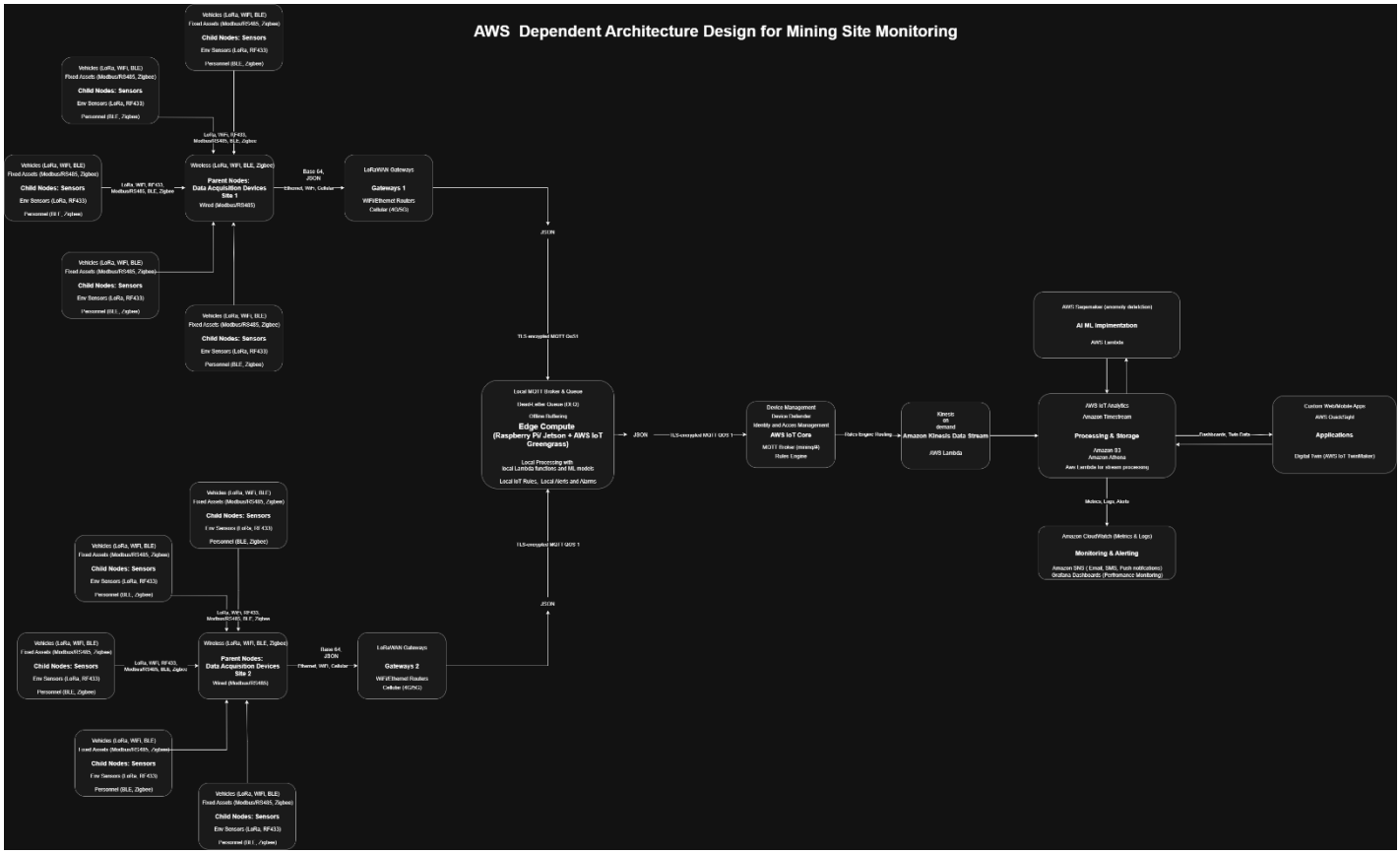


Diagram Description:

- **Child Nodes (Sensors)**: Deployed across multiple sites, sensors (vehicles, fixed assets, environmental, personnel) use LoRa, WiFi, RF433, Modbus/RS485, BLE, and Zigbee for connectivity. Each sensor includes an RTC clock for timestamped payloads.
- **Parent Nodes (Data Acquisition Devices)**: Aggregate data from child nodes via wired (Modbus/RS485) or wireless (LoRa, WiFi, BLE, Zigbee) connections. Each site has dedicated parent nodes.
- **Gateways**: LoRaWAN gateways, WiFi/Ethernet routers, and cellular (4G/5G) gateways connect parent nodes to edge compute, supporting Base64-encoded JSON payloads.
- **Edge Compute (Raspberry Pi/Jetson with AWS IoT Greengrass)**: Hosts local MQTT brokers, offline buffering, dead-letter queues (DLQs), local Lambda functions, ML models, and IoT rules for processing and alerting.
- **AWS IoT Core**: Centralized cloud ingestion with device management, MQTT broker (topics: mining/#), rules engine, Device Defender, and IAM for secure communication.

- **Processing & Storage**: Includes Amazon Kinesis Data Streams, AWS IoT Analytics, Amazon Timestream (time-series DB), Amazon S3 (cold storage), Amazon Athena (querying), and AWS Lambda (stream processing).
- **AI/ML Implementation**: AWS SageMaker for anomaly detection and AWS Lambda for ML inference.
- **Monitoring & Alerting**: Amazon CloudWatch for metrics/logs, Amazon SNS for notifications (email, SMS, push), and Grafana for dashboards.
- **Applications**: Custom web/mobile apps, AWS QuickSight for analytics, and AWS IoT TwinMaker for Digital Twin visualization.

Data Flow:

1. Sensors → Parent Nodes (LoRa, WiFi, RF433, Modbus/RS485, BLE, Zigbee).
2. Parent Nodes → Gateways (Ethernet, WiFi, Cellular).
3. Gateways → Edge Compute (TLS-encrypted MQTT QoS 1, JSON payloads).
4. Edge Compute → AWS IoT Core (TLS-encrypted MQTT QoS 1).
5. AWS IoT Core → Processing & Storage (via Kinesis and Rules Engine).
6. Processing & Storage → AI/ML, Monitoring, and Applications.

4. System Requirements

4.1 Scalability

- **Hundreds of Vehicles, Thousands of Sensors**:
    - AWS IoT Core supports millions of devices and messages, with auto-scaling for MQTT connections.
    - Amazon Kinesis Data Streams (On-Demand mode) scales dynamically for high-throughput data ingestion (e.g. thousands of messages/second).
    - Edge Compute uses AWS IoT Greengrass groups to manage multiple sites, with load-balanced Raspberry Pi/Jetson devices per site.
    - Gateways are deployed redundantly (LoRaWAN, WiFi, Cellular) to handle increased device counts.
    - Amazon Timestream scales for high-frequency time-series data (e.g. 5-second intervals for thousands of sensors).

4.2 Reliability

- **Offline Buffering**: Edge Compute (Greengrass) buffers data locally during connectivity loss using persistent storage (e.g. SQLite or file-based queues), as implemented in synthetic_generator.py for intermittent mode.
- **Retries**: MQTT QoS 1 ensures at-least-once delivery, with Greengrass retrying failed messages to AWS IoT Core.
- **Dead-Letter Queues (DLQs)**: Greengrass and AWS IoT Core route undeliverable messages to DLQs (Amazon SQS) for manual processing or debugging, as shown in the edge compute layer.
- **Fault Tolerance**: Multiple gateways per site (LoRaWAN, WiFi, Cellular) ensure redundancy. Greengrass supports failover across edge devices.

4.3 Security

- **Device Authentication**: AWS IoT Core uses X.509 certificates for device authentication, provisioned via AWS IoT Device Management.
- **Encryption**: TLS 1.2/1.3 encrypts all MQTT communications (edge to cloud). Data payloads use Base64-encoded JSON for compatibility.
- **Access Control**: AWS IAM policies restrict device actions (e.g. iot:Connect, iot:Publish to mining/{device_id} topics). AWS IoT Device Defender monitors for unauthorized access.
- **Key/Cert Rotation**: AWS IoT Device Management automates certificate rotation every 90 days, integrated with AWS KMS for secure key storage.

4.4 Observability

- **Metrics**: Amazon CloudWatch collects device metrics (e.g. message rates, connection errors) and Greengrass runtime metrics (e.g. CPU, memory).
- **Logs**: Greengrass logs (stored locally and synced to CloudWatch) capture edge processing events. AWS IoT Core logs MQTT interactions.
- **Dashboards**: Grafana, integrated with CloudWatch and Timestream, provides real-time dashboards for metrics like methane levels, vehicle speeds, and fault rates.
- **Alerts**: AWS IoT Rules Engine triggers Amazon SNS for email/SMS/push notifications on critical events (e.g. methane >5ppm, fall detection).

4.5 Data Storage & Analytics

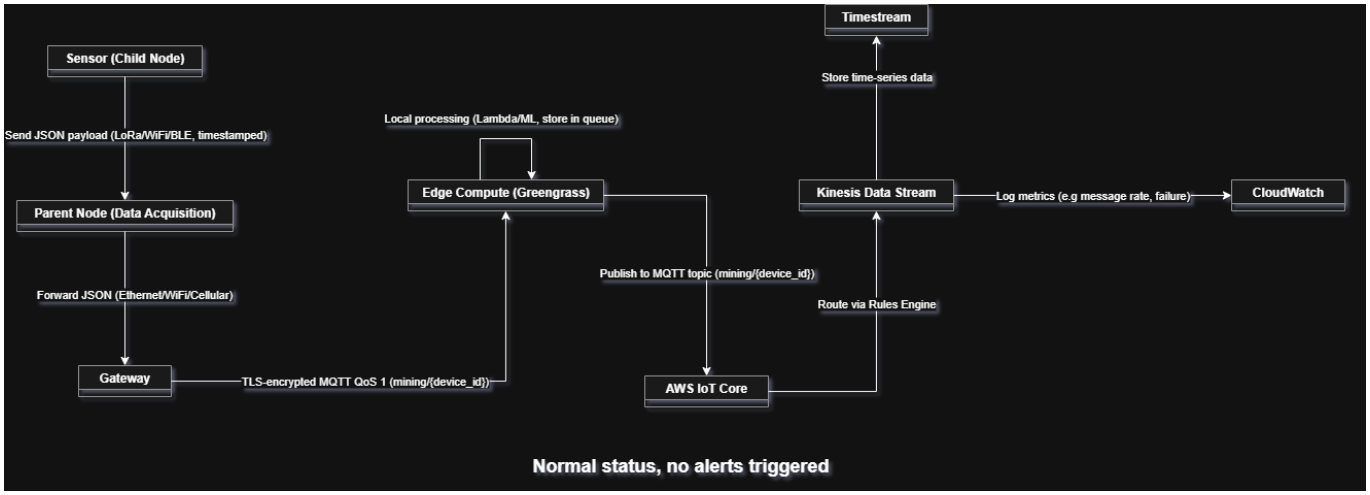
- **Time-Series Database**: Amazon Timestream stores high-frequency telemetry (e.g. 5-second intervals) with optimized querying for time-based analytics.
- **Cold Storage**: Amazon S3 stores raw and processed data for long-term retention, with lifecycle policies to transition to Glacier for cost savings.
- **Rule Evaluation**: AWS IoT Rules Engine evaluates conditions (e.g. engine_temp > 100) and routes data to Kinesis, Lambda, or SNS.
- **Analytics**: AWS IoT Analytics processes data for insights (e.g. fleet performance trends). Amazon Athena queries S3 for ad-hoc analysis. AWS SageMaker detects anomalies (e.g. unusual vibration patterns).

5. Sequence Diagrams

The following sequence diagrams illustrate key operational scenarios: normal operation, fault events, and connectivity loss. These align with the synthetic_generator.py script's modes (normal, fault, intermittent).

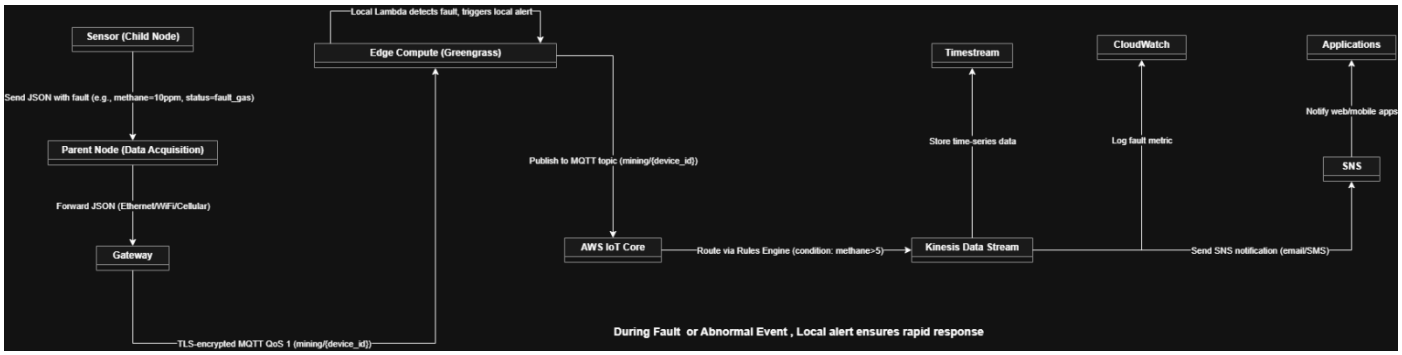
5.1 Normal Operation (draw.io XML attached)

Description: A sensor generates and sends telemetry data to the cloud under normal *conditions.*



5.2 Fault Event (draw.io XML attached)

Description: A sensor detects a fault (e.g. methane >5ppm), triggering an alert.



5.3 Connectivity Loss (draw.io XML attached)

Description: Edge compute buffers data during a network outage and syncs upon reconnection, as in intermittent mode.