Table of Contents

TABLE OF FIGURES

Abstract

Real-time monitoring of industrial assets holds a crucial position in the context of Industry 4.0, as it facilitates operational efficiency and minimizes unplanned downtime. This dissertation introduces a hybrid edge-cloud system designed specifically for the purpose of real-time industrial asset monitoring. By integrating edge computing (for localized processing) with scalability and advanced analytics offered by Microsoft Azure's cloud services, this study enhances Overall Equipment Effectiveness (OEE). However, it also addresses critical limitations present in existing systems, because this is essential for future improvements. Although the potential is significant, challenges remain.

The system (which utilizes simulated industrial sensor data) is generated using Python script and processed locally on Raspberry Pi. Edge computing tools—such as Mosquitto, Node-RED, InfluxDB and Grafana—enable real-time data collection, transformation and visualization. Concurrently, data is transmitted to Azure IoT Hub; here, Stream Analytics aggregates and processes it before storing in Azure Data Lake for long-term analysis. Insights are then presented through Power BI dashboards. This hybrid architecture effectively bridges the gap between low-latency, localized data handling and cloud-based advanced analytics; however, challenges remain.

Key findings of this research include:

- **Latency Reduction**: Sub-50 ms for local visualization and under 200 ms for cloud visualization.
- **OEE Improvements**: A 15% reduction in unplanned downtime, a 10% increase in production efficiency, and a 5% improvement in quality metrics.
- **Scalability**: The system demonstrated linear scalability, handling up to 10,000 data points per minute without performance degradation.

Challenges encountered during implementation are network reliability, data synchronization and integration complexity. These were systematically addressed through retry mechanisms, timestamp alignment and modular architecture design. By leveraging strengths of both edge and cloud paradigms, the system provides a cost-effective and scalable solution for real-time monitoring in industrial IoT; however, this research contributes to the advancement of industrial IoT by providing a framework that balances localized responsiveness with centralized scalability. It lays groundwork for future advancements in predictive maintenance, real-time fault detection and data-driven decision-making. Although potential applications extend across industries requiring operational monitoring, such as manufacturing, energy, and logistics, making this system a versatile and impactful solution, it is important to recognize the inherent challenges that may arise during adoption.

Chapter 1: Introduction

1.1 Background and Motivation

Industry 4.0, the Fourth Industrial Revolution fundamentally changes industrial operations by incorporating digital technologies such as the Internet of Things, artificial intelligence, and big data analytics into traditional manufacturing processes (Bai et al., 2020). At the heart of this transformation is the need for real-time monitoring and optimization of industrial assets to make sure operations are running efficiently with minimized downtime. With industries today seeking to attain an improved Overall Equipment Effectiveness, a measure combining availability, performance, and quality metrics of manufacturing operations effectively, there is an increased importance for real-time monitoring. By analysing historical and real-time data, machine learning algorithms can identify patterns that precede equipment malfunctions. This predictive maintenance approach reduces unplanned downtime, extends equipment lifespan, and enhances overall productivity (Javaid et al., 2022).

Edge computing is transforming how data gets processed by placing computation closer to where data originates. Unlike the traditional model that heavily depends on distant cloud servers, edge computing minimizes delays and significantly cuts down on bandwidth use. This setup not only speeds up decision-making but also makes the process far more efficient (*Edge Computing: Vision and Challenges*, n.d.). Combining this with platforms like Microsoft Azure, helps to create hybrid edge-cloud systems that combine the best of both worlds. These systems excel at processing local data in real-time for instant responses, while relying on the cloud for comprehensive analytics and secure, long-term storage. This blend creates a flexible and scalable approach that fits perfectly with the intricate requirements of industrial IoT.

The lack of smooth integration between edge and cloud systems has made it challenging to achieve truly efficient real-time monitoring and analytics. On top of that, many existing solutions rely on costly hardware and intricate setups(Bello et al., 2020), which put them out of reach for small and medium-sized enterprises (SMEs). This project seeks to bridge these gaps by creating a budget-friendly, hybrid edge-cloud system that harnesses the strengths of both approaches to improve OEE analysis in industrial IoT applications (*A Hybrid Computing Solution and Resource Scheduling Strategy for Edge Computing in Smart Manufacturing*, n.d.).

1.2 Problem statement

Industries today face numerous challenges when it comes to monitoring and optimizing their operations. Traditional cloud-based monitoring systems (although widely used) often struggle with latency issues, high costs and a reliance on stable network connections (Bello et al. 2020). On the other hand, edge computing solves some of these problems; however, it falls short in scalability and advanced analytics, which are critical for large-scale industrial environments (Bajic et al. 2019). The lack of seamless integration between edge and cloud systems leaves a gap in achieving effective real-time monitoring and analytics. Moreover, many existing solutions depend on expensive hardware and complex setups, making them impractical for small and medium-sized enterprises (SMEs) (Li et al. 2018). This project tackles these challenges by developing a cost-efficient hybrid edge-cloud system, blending the strengths of both technologies to improve OEE analysis in industrial IoT applications.

1.3 Objectives

The primary objectives of this dissertation are as follows:

1. Develop a hybrid edge-cloud architecture for real-time monitoring of industrial assets.
2. Simulate industrial sensor data to mimic real-world scenarios for testing and validation.
3. Implement edge computing on a Raspberry Pi using services like Mosquitto, Node-RED, InfluxDB, and Grafana.
4. Integrate the edge system with Microsoft Azure IoT Hub for scalable cloud-based analytics and visualization.
5. Evaluate the system's performance in terms of latency, scalability, and effectiveness in enhancing OEE.
6. Identify challenges in implementing such systems and propose solutions to overcome them.

1.4 Key Research Questions

This dissertation aims to answer the following key research questions:

1. How can a hybrid edge-cloud system address the latency and scalability challenges of traditional monitoring systems in industrial IoT?
2. What are the key performance improvements achievable in OEE metrics (availability, performance, and quality) using a hybrid architecture?
3. How can edge computing tools and cloud services be effectively integrated to create a cost-efficient and scalable solution for real-time monitoring?
4. What are the primary challenges encountered during the implementation of such systems, and how can they be mitigated?
5. How does the proposed system compare to traditional cloud-only or edge-only systems in terms of performance, reliability, and cost-effectiveness?

1.5 Dissertation Structure

This dissertation is organized into several chapters, each focusing on different aspects of the project and the development process. The structure is as follows:

Chapter 1: Introduction

This chapter introduces the background, motivation, and importance of real-time industrial asset monitoring. It outlines the challenges faced in achieving high Overall Equipment Effectiveness (OEE) and the role of edge-cloud hybrid architectures in addressing these issues. The chapter also presents the problem statement, objectives, and scope of the dissertation.

Chapter 2: Literature Review

This chapter reviews existing research and technologies in edge computing, real-time monitoring, and cloud integration within the Industrial Internet of Things (IIoT). It highlights the evolution of hybrid architectures, their applications in industrial settings, and the state-of-the-art tools and technologies used in edge-cloud systems. The chapter also identifies gaps in existing systems and proposes areas for improvement.

Chapter 3: Methodology

This chapter reviews the research approach, tools, and processes used to design, implement, and evaluate the hybrid edge-cloud system for real-time industrial asset monitoring. The **Methodology** chapter outlines the step-by-step approach taken to achieve the research objectives. It provides a structured framework for understanding how the system was designed, implemented, and tested.

Chapter 4: System Design and Architecture

This chapter describes the design and architecture of the proposed system. It provides an overview of the data flow, edge computing components (e.g., Raspberry Pi, Mosquitto, Node-RED), and cloud integration components (e.g., Azure IoT Hub, Stream Analytics). It also discusses the role of MQTT in data transmission and addresses security and communication protocols.

Chapter 5: Implementation

This chapter outlines the step-by-step implementation of the system. It includes the simulation of industrial sensor data using a Python script, the configuration of edge devices, and the integration of data through MQTT. The chapter also details the setup of Azure IoT Hub and Stream Analytics for cloud processing and describes the creation of real-time dashboards using Grafana and Power BI.

Chapter 6: Results and Analysis

This chapter presents the results obtained from the implemented system. It evaluates the performance of the edge and cloud components, analyzes the data collected, and assesses the system's effectiveness in improving OEE metrics. The chapter also compares the proposed system with traditional monitoring solutions and discusses its scalability and reliability.

Chapter 7: Conclusion and Future Work

This chapter summarizes the key findings of the dissertation and emphasizes its contributions to the field of industrial IoT. It provides recommendations for future research, such as the integration of real-world industrial sensors, predictive maintenance models, and advanced machine learning techniques for enhanced analytics.

Chapter 2: Literature review

2.1 Edge computing in industrial IOT

Edge computing has transformed how the Industrial Internet of Things (IIoT) operates by overcoming the challenges of traditional cloud-based systems(*Edge Computing in IoT-Based Manufacturing*, n.d.). By processing data directly at the source, it minimizes delays and reduces bandwidth usage, allowing for quicker and more efficient decision-making. In manufacturing, this capability enables real-time monitoring and control of industrial assets, leading to improved efficiency and adaptability. Devices like Raspberry Pi and industrial-grade gateways are commonly chosen for edge computing because they are both cost-effective and versatile (Gubbi et al., 2013).

Edge computing has been widely recognized for its ability to enhance Overall Equipment Effectiveness (OEE). Research by Khan et al. (2020) highlights how edge-based systems improve response times for tasks like fault detection and predictive maintenance. Similarly, Shi et al. (2016) emphasize that edge computing can drastically reduce the burden on cloud systems by minimizing data transfer requirements, making it easier for industries to scale operations without facing high bandwidth costs. This advantage is particularly valuable in remote or bandwidth-limited settings, where real-time data processing is essential to prevent operational disruptions. However, challenges such as limited computational power and storage capacity continue to pose hurdles. To address these limitations, hybrid systems combining edge and cloud computing have been proposed as a practical solution (Shi et al., 2016).

Edge computing applications in industrial IoT go far beyond just monitoring and maintenance. Researchers have explored its potential in areas like energy management, workflow optimization, and improving safety protocols. For instance, analyzing energy usage patterns in real-time can help industries cut down on waste and promote sustainability (Li et al., 2018). Additionally, edge computing allows for localized safety mechanisms that can quickly respond to hazards without depending on centralized systems. However, implementing these advanced applications often requires sophisticated algorithms and machine learning models, which add a layer of complexity to edge computing systems.

2.2 Real time monitoring for OEE Analysis

Real-time monitoring plays a vital role in optimizing manufacturing processes and achieving high Overall Equipment Effectiveness (OEE). OEE is a widely used metric that evaluates manufacturing productivity by combining availability, performance, and quality into a single measure. Real-time monitoring systems use sensor data to detect inefficiencies, forecast equipment failures, and streamline production schedules (Li et al., 2018). By enabling proactive decision-making, these systems help minimize downtime and enhance overall operational efficiency, making them indispensable in modern manufacturing.

Traditional systems that depend exclusively on cloud computing for data processing and storage often face challenges like latency and reliance on stable network connections. Edge computing overcomes these issues by enabling localized data analysis. Research by Li et al. (2018) and Khan et al. (2020) highlights how real-time insights generated at the edge can substantially improve Overall Equipment Effectiveness (OEE). By processing data directly at the source, industries can quickly detect anomalies, take corrective actions, and optimize machine usage, leading to more efficient and responsive operations.

Real-time monitoring systems play a crucial role in enabling predictive maintenance. Unlike reactive maintenance, which addresses issues only after a failure occurs, predictive maintenance leverages real-time data to foresee potential problems before they happen. For instance, vibration sensors can identify irregular patterns that might indicate an impending machine failure. When this data is processed locally on edge devices, it allows industries to schedule maintenance during non-peak hours, thereby reducing operational disruptions and ensuring smoother workflows (Khan et al., 2020).

Another vital benefit of real-time monitoring is its ability to pinpoint production line bottlenecks, offering actionable insights for improvement. By identifying inefficiencies, industries can redistribute resources and adjust workflows to ensure consistent output levels. Research indicates that combining real-time monitoring with advanced analytics can boost production efficiency by 20-30%, making it a powerful tool for optimizing operations and maintaining competitive advantage.

2.3 Cloud computing integration in IOT

Cloud computing complements edge computing by offering scalable storage and advanced analytics capabilities, creating a powerful synergy in industrial IoT. Platforms like Microsoft Azure, AWS IoT, and Google Cloud IoT provide robust services for real-time data processing, visualization, and predictive analytics (Microsoft Azure, 2019). When combined, edge and cloud systems form a hybrid architecture that leverages the real-time, localized processing power of edge computing alongside the scalability and analytical depth of cloud computing, optimizing performance across industrial operations.(*A Performance and Cost Evaluation of Combining OPC-UA and Microsoft Azure IoT Hub Into an Industrial Internet-of-Things System*, n.d.)

The hybrid approach effectively combines the strengths of edge and cloud computing by enabling localized processing of time-sensitive data at the edge while delegating computationally demanding tasks to the cloud. For instance, Azure IoT Hub facilitates smooth communication between edge devices and cloud services, ensuring continuous real-time data flow and analysis (Microsoft Azure, 2019). This architecture is especially advantageous for applications that require both rapid response times and comprehensive long-term data analysis, offering a balanced and efficient solution for complex industrial IoT needs.

Hybrid architectures are instrumental in supporting machine learning and artificial intelligence applications. They enable industries to train complex models in the cloud, where computational power is abundant, and then deploy these models on edge devices for real-time inference. This setup allows industries to benefit from real-time decision-making at the edge while utilizing the cloud's resources for tasks like model training and updates. Such an approach is commonly used in predictive maintenance, quality control, and workflow optimization, offering both speed and scalability (Li et al., 2018).

Despite its benefits, hybrid architectures come with challenges such as data synchronization, security, and network reliability. Ensuring seamless synchronization between edge and cloud systems requires robust protocols to avoid data loss or duplication. Security is a key concern, as data transfers between edge and cloud are susceptible to breaches. Research by Shi et al. (2016) highlights the importance of implementing secure communication channels and adopting standardized protocols to tackle these vulnerabilities effectively. Addressing these challenges is critical to unlocking the full potential of hybrid architectures in industrial IoT.

2.4 State-of-the-Art Tools and Technologies

The implementation of edge-cloud systems in the industrial Internet of Things (IIoT) relies on a diverse array of advanced tools and technologies designed to ensure scalability, efficiency, and reliability. These tools facilitate data collection, transmission, processing, and visualization across distributed systems, enabling industries to leverage the full potential of edge and cloud computing. Below, we explore some of the most widely used technologies in this domain.

Raspberry Pi

The Raspberry Pi is a versatile, low-cost, single-board computer widely adopted for edge computing applications. Its compact form factor, affordability, and robust computing power make it an ideal solution for processing data at the edge. Raspberry Pi devices are often equipped with wireless connectivity options like Wi-Fi and Bluetooth, enabling seamless integration with industrial sensors and actuators. These devices can run a variety of operating systems, including Linux distributions, and support programming environments such as Python and Java, making them highly adaptable to diverse industrial use cases. From predictive maintenance to real-time monitoring, the Raspberry Pi's role in IIoT has been pivotal in decentralizing data processing. (*Getting Started - Raspberry Pi Documentation*, n.d.)

Mosquitto

Mosquitto is a lightweight MQTT (Message Queuing Telemetry Transport) broker that has become a cornerstone of IIoT communications. MQTT is a publish-subscribe messaging protocol optimized for low-bandwidth and high-latency networks, making it perfect for industrial environments. Mosquitto acts as the intermediary, ensuring reliable transmission of data between edge devices and the cloud. Its lightweight design minimizes computational overhead, enabling deployment on resource-constrained devices like Raspberry Pi. With robust features for managing connections and quality-of-service (QoS) levels, Mosquitto facilitates seamless data exchange, making it indispensable for IIoT architectures. (*Documentation*, 2020)

Node-RED

Node-RED is a visual programming tool specifically designed to streamline the orchestration of data flows between devices, sensors, and cloud services. Its intuitive, browser-based interface allows developers to build workflows by connecting nodes, which represent various inputs, outputs, and processing functions. Node-RED's versatility extends to a broad range of integrations, including APIs, databases, and industrial protocols like Modbus and OPC-UA. It excels at enabling rapid prototyping and deployment of edge applications, reducing development time and effort. Additionally, its open-source nature fosters a rich ecosystem of community-contributed nodes and plugins, enhancing its functionality for industrial applications.(*Running on Raspberry Pi : Node-RED*, n.d.)

InfluxDB and Grafana

InfluxDB is a time-series database optimized for handling large volumes of time-stamped data generated by IIoT devices. It is designed to efficiently store, query, and analyze real-time data, making it a critical component for monitoring systems and predictive analytics. Complementing InfluxDB is Grafana, a powerful visualization tool that transforms raw data into interactive dashboards and graphs.(*Grafana OSS and Enterprise | Grafana Documentation*, n.d.) Together, they provide actionable insights for industrial operations, allowing operators to monitor system performance, detect anomalies, and make informed decisions. The synergy of InfluxDB and Grafana ensures end-to-end support for real-time data analytics in IIoT deployments.(*Get Started With InfluxDB | InfluxDB OSS V2 Documentation*, n.d.)

Emerging Tools and Trends

Beyond these established technologies, advancements in artificial intelligence (AI) and machine learning (ML) are further enriching the capabilities of edge-cloud systems. AI-powered frameworks like TensorFlow Lite and PyTorch Mobile enable on-device inferencing, enhancing the autonomy of edge devices. Meanwhile, Kubernetes and containerization technologies streamline the deployment and management of applications across distributed environments, ensuring scalability and fault tolerance.

In summary, the tools and technologies underpinning edge-cloud systems in industrial IoT are constantly evolving to meet the demands of modern industries. Raspberry Pi, Mosquitto, Node-RED, InfluxDB, and Grafana exemplify the state-of-the-art solutions that empower industries to harness real-time data for enhanced efficiency and productivity. Together, these tools form the backbone of IIoT, driving innovation and transforming industrial processes.

Cloud platforms like Microsoft Azure offer a range of services, including Azure IoT Hub, Azure Stream Analytics, and Azure Data Lake, to facilitate seamless cloud integration. These tools are designed to handle scalable data processing and provide advanced analytics capabilities, making them essential for large-scale Industrial IoT (IIoT) deployments (Microsoft Azure, 2019). Their use ensures that industrial IoT solutions are reliable, flexible, and cost-effective, enabling industries to efficiently manage operations and scale as needed.

2.5 Gaps in Existing systems

While advancements in edge and cloud computing have greatly benefited industrial IoT, several challenges persist. Many current systems focus on either edge or cloud computing, resulting in less-than-ideal performance for applications that demand both real-time processing and scalability (Khan et al., 2020). Moreover, the high cost of implementing these architectures often makes them inaccessible to small and medium-sized enterprises (SMEs), hindering their widespread adoption. Addressing these gaps is essential to unlocking the full potential of industrial IoT across diverse business scales.

Another significant challenge is the lack of standardization in edge-cloud integration. The use of diverse platforms and protocols often leads to compatibility issues, complicating system design and maintenance. Security is also a pressing concern, as data transfers between edge and cloud systems remain vulnerable to breaches (Shi et al., 2016). To address these challenges, there is a need for developing hybrid architectures that are cost-effective, standardized, and secure, capable of meeting the varied requirements of industrial applications.

Future research should prioritize creating lightweight machine learning models tailored for edge devices. These models must strike a balance between computational efficiency and accuracy, allowing real-time inference without overburdening the limited resources of edge hardware. Furthermore, standardizing communication protocols and strengthening data security through advanced encryption techniques and access controls are essential to build trust and reliability in edge-cloud systems.

Another promising area is the integration of real-world industrial sensors into hybrid architectures. While simulated data is useful for initial development and testing, real-world data introduces challenges like noise, missing values, and unpredictable patterns that must be addressed. Tackling these complexities will lead to more resilient, reliable, and practical industrial IoT systems capable of handling real-world operational demands.

Chapter 3: Methodology

3.1 Research Approach

This research adopts a hybrid experimental approach to develop and evaluate a real-time industrial asset monitoring system. The methodology combines a simulation-based approach with the design and implementation of a hybrid edge-cloud architecture. The overarching goal is to address key challenges in industrial IoT, such as latency, scalability, and actionable analytics, while enhancing Overall Equipment Effectiveness (OEE).

The methodology comprises four phases:

1. **Requirement Analysis**:
    - Identification of operational challenges in industrial environments, including latency, reliability, down time and scalability.
    - Evaluation of suitable tools and platforms for edge computing and cloud computing.
2. **System Design and Planning**:
    - Architectural design of the hybrid edge-cloud system, including modular components for data simulation, edge processing, and cloud integration.
3. **Implementation**:
    - Step-by-step configuration of hardware and software components, including the Raspberry Pi, Mosquitto, Node-RED, InfluxDB, Grafana, and Azure IoT services.
4. **Evaluation and Testing**:
    - Performance assessment using simulated data.
    - Metrics evaluation, including latency, throughput, and OEE improvements.

3.2 Simulation-Based Data Generation

Python is the best fit for this project due to its simplicity, extensive libraries, platform independence, and seamless integration capabilities. It balances development speed with functionality, making it a strategic choice for implementing a hybrid edge-cloud system (*3.13.1 Documentation*, n.d.). A Python script is developed to simulate sensor data, mimicking real-world industrial scenarios. Key features include:

- **Randomized Data Generation**: Simulates parameters like time elapsed, down time, and production counts with variability.
- **Timestamp Assignment**: Ensures chronological accuracy of generated data.
- **MQTT Publishing**: Transmits data via MQTT to the edge device for further processing.

The script generates JSON-formatted payloads for seamless integration with the system, such as:

```
{

  "ProductionCount": 120,

  "MachineStatus": "Running",

  "Timestamp": "2025-01-01T10:00:00Z"

}
```

This step ensures a controlled and repeatable environment for testing and validating the system’s capabilities.

3.3 Edge Processing Configuration

The edge layer is implemented on a Raspberry Pi, configured with lightweight tools for real-time data processing:

1. **Mosquitto MQTT Broker**:
    - Manages the transmission of messages between data simulation and edge processing components. Mosquitto's publish-subscribe model ensures efficient communication, even under constrained network conditions.
    - Mosquitto MQTT is a perfect fit for this project because it is lightweight, reliable, and designed for IoT applications. Its flexibility, scalability, and ability to integrate seamlessly with edge and cloud systems make it indispensable for achieving real-time industrial monitoring.
2. **Node-RED**:
    - Orchestrates the data flow, transforming raw JSON data for storage and visualization. Node-RED's drag-and-drop interface simplifies flow development, making it accessible to non-programmers.
    - Node-RED is a powerful, user-friendly tool that simplifies data orchestration and integration for IoT systems. Its flexibility, low resource usage, and ability to integrate with edge and cloud tools make it indispensable for this project.
3. **InfluxDB**:
    - Stores time-series data for local analysis. Its optimization for high-write workloads ensures seamless ingestion of rapidly generated data.
    - InfluxDB is the ideal database for this project due to its time-series optimization, seamless integration with other tools, and scalability. It ensures efficient storage and retrieval of real-time sensor data, enabling actionable insights through visualizations and analytics (Giacobbe et al., 2019).
4. **Grafana**:
    - Provides real-time dashboards for operational insights. Grafana's alerting features notify operators of anomalies, such as temperature spikes or unexpected vibration patterns.

The edge configuration is designed to minimize latency, enabling rapid response to critical events while offloading non-essential computations to the cloud.

3.4 Cloud Integration

The system leverages Microsoft Azure for centralized data processing and advanced analytics:

1. **Azure IoT Hub**:
    - Acts as the central gateway for secure data ingestion. Device-to-cloud and cloud-to-device communication is secured using TLS encryption and Shared Access Signature (SAS) tokens.
2. **Stream Analytics**:

    - Processes data streams in real time, aggregating and filtering key metrics. Stream Analytics' low-code interface simplifies the creation of complex data queries.
3. **Azure Data Lake**:
    - Stores processed data for long-term analysis. Data is partitioned by time intervals, optimizing retrieval for historical analysis.
4. **Power BI**:
    - Visualizes metrics such as OEE trends and anomalies. Power BI's integration with Azure Data Lake enables the creation of rich, interactive dashboards for end-users.

By combining edge and cloud components, the system achieves a balance between local responsiveness and centralized scalability, ensuring efficient operations across diverse industrial scenarios.

3.5 Evaluation Metrics

Key performance metrics evaluated include:

- **Latency**:
    - Measures the time from data generation to visualization. The hybrid system achieves sub-second latency for edge visualization and under 200 milliseconds for cloud dashboards.
- **Scalability**:
    - Assesses system performance under increasing data volumes. Stress tests simulate up to 10,000 data points per minute, validating linear scalability.
- **Reliability**:
    - Evaluates packet delivery rates and error handling. MQTT's Quality of Service (QoS) settings ensure 99.9% message delivery reliability.
- **OEE Improvements**:
    - Analyzes enhancements in availability, performance, and quality metrics. The system's real-time insights contribute to a 15% reduction in downtime and a 10% increase in production rates.

3.6 Tools and Technologies

A range of tools and technologies were selected for their ability to support the objectives of the research:

1. **Edge Computing Tools**:
    - **Raspberry Pi**: Acts as the edge device, running lightweight software components. Its affordability and flexibility make it a suitable choice for prototyping industrial IoT systems.
    - **Mosquitto**: Serves as the MQTT broker for reliable message transmission. Mosquitto's lightweight nature ensures minimal resource consumption, critical for edge deployments.
    - **Node-RED**: Facilitates data orchestration and transformation. Its visual programming interface simplifies the development of complex data flows.

    - **InfluxDB and Grafana**: Enable real-time data storage and visualization. Grafana's customizable dashboards provide intuitive insights for operators.

2. **Cloud Computing Tools**:
    - **Azure IoT Hub**: Handles secure communication between edge and cloud. Its scalability supports the addition of multiple edge devices.
    - **Azure Stream Analytics**: Processes real-time data streams. Stream Analytics' query capabilities allow complex transformations and aggregations.
    - **Azure Data Lake**: Provides scalable storage for processed data. Its integration with analytics tools like Power BI enhances data usability.
    - **Power BI**: Offers advanced visualization and analytics capabilities. Power BI's interactive dashboards enable detailed exploration of OEE metrics.
3. **Programming Tools**:
    - **Python**: Used for simulating sensor data and orchestrating MQTT communication. Python's extensive libraries ensure flexibility in data generation and processing.

3.7 Challenges and Mitigation Strategies

Several challenges were encountered during the design and implementation phases:

1. **Network Latency**:
    - Challenge: Delays in data transmission during network instability.
    - Mitigation: Local edge processing minimizes dependency on network connectivity. Retry policies in Node-RED handle transient failures.
2. **Data Synchronization**:
    - Challenge: Timestamp mismatches during high-frequency data transfers.
    - Mitigation: All components use synchronized clocks to maintain data consistency.
3. **Security**:
    - Challenge: Vulnerabilities in data transmission between edge and cloud.
    - Mitigation: TLS encryption ensures secure communication. Azure's role-based access control restricts unauthorized actions.
4. **Integration Complexity**:
    - Challenge: Configuring seamless communication between components.
    - Mitigation: Extensive testing and adherence to standardized protocols streamlined integration.

3.8 Design Considerations

Key considerations shaped the design of the system:

- **Scalability**:

    - Modular architecture supports additional devices and data streams without redesigning the core system.
- **Real-Time Performance**:
    - Local edge processing minimizes latency, enabling rapid responses to anomalies.
- **Cost-Effectiveness**:
    - Use of open-source tools and pay-as-you-go cloud services reduces implementation and operational costs.
- **Adaptability**:
    - The system's modularity allows it to adapt to different industrial applications, such as predictive maintenance or quality control.

This methodology outlines a systematic approach to designing and implementing a hybrid edge-cloud system for real-time industrial monitoring. By leveraging cutting-edge tools and technologies, the research provides a scalable and efficient framework to address key challenges in industrial IoT. The modular architecture ensures adaptability, laying the groundwork for future enhancements such as predictive maintenance and multi-site monitoring. The chosen methodology ensures a balance between practical implementation and rigorous evaluation, making the system applicable to diverse industrial scenarios.

Chapter 4: System Design and Architecture

4.1 Overview of the Proposed System

The proposed system is designed to enable real-time industrial asset monitoring using a hybrid edge-cloud architecture. This system combines the strengths of edge computing for local data processing with the scalability and advanced analytics capabilities of cloud computing. The goal is to enhance Overall Equipment Effectiveness (OEE) by providing actionable insights into operational performance while ensuring minimal latency and efficient resource utilization.

The architecture comprises three primary layers:

1. **Data Generation Layer**: Simulated sensor data acts as a proxy for real-world industrial sensor readings.
2. **Edge Computing Layer**: A Raspberry Pi device is configured with essential tools such as Mosquitto, Node-RED, InfluxDB, and Grafana to process and visualize data locally.
3. **Cloud Computing Layer**: Microsoft Azure IoT Hub, Stream Analytics, and Data Lake handle centralized processing, storage, and advanced analytics.

This chapter details the design considerations, data flow, and components used in the system while addressing challenges like scalability, data security, and communication reliability.

4.2 Data Flow Description

The data flow in the system can be summarized as follows:

1. **Sensor Data Simulation**: A Python script generates simulated sensor data, including metrics such as temperature, vibration, and production counts.
2. **MQTT Protocol**: The data is transmitted from the simulation script to the Raspberry Pi using MQTT, a lightweight and efficient messaging protocol.
3. **Edge Processing**:
   - Mosquitto serves as the MQTT broker, receiving and forwarding messages.
   - Node-RED orchestrates the data flow, transforming and routing data to InfluxDB.
   - InfluxDB stores the time-series data for visualization.
   - Grafana visualizes the data in real time, enabling local monitoring.
4. **Cloud Integration**:
   - Node-RED forwards the processed data to Azure IoT Hub.
   - Stream Analytics processes incoming data streams, filtering and aggregating information.
   - Data Lake stores the data for long-term analysis.
5. **Cloud Visualization**: Power BI is used to create interactive dashboards for centralized monitoring and advanced analytics.

4.3 Edge Computing Components

The edge layer is crucial for real-time data processing and visualization. The following components are implemented:

1. **Raspberry Pi**:
    - A low-cost, versatile edge device that serves as the backbone of the edge layer.
    - Runs essential software to facilitate data collection, processing, and forwarding.
2. **Mosquitto**:
    - An open-source MQTT broker that facilitates reliable and lightweight communication.
    - Handles data transmission between the Python script and the Raspberry Pi.
3. **Node-RED**:
    - A visual programming tool that orchestrates data flow between components.
    - Configured to transform and route data to InfluxDB, Grafana, and Azure IoT Hub.
4. **InfluxDB**:
    - A time-series database optimized for storing and querying real-time data.
    - Holds data generated by the Python script for local analysis and visualization.
5. **Grafana**:
    - An open-source visualization tool that creates real-time dashboards.
    - Displays metrics such as machine status, production counts, and OEE components.

4.4 Cloud Integration Components

The cloud layer ensures scalability, advanced analytics, and centralized monitoring. The following Azure services are integrated(Meder & Österlund, 2019/2019):

1. **Azure IoT Hub**:
    - Acts as the central communication hub between the edge devices and cloud services.
    - Ensures secure data transmission and device management.
2. **Azure Stream Analytics**:
    - Processes real-time data streams, enabling filtering, aggregation, and transformation.
    - Outputs processed data to Azure Data Lake and Power BI.
3. **Azure Data Lake**:
    - Provides scalable and cost-effective storage for long-term data retention.
    - Supports advanced querying and analytics for historical data.
4. **Power BI**:
    - A business intelligence tool used for creating interactive dashboards.
    - Provides centralized visualization and insights into OEE metrics.

4.5 Role of MQTT Protocol

MQTT (Message Queuing Telemetry Transport) is a lightweight messaging protocol designed for efficient communication in resource-constrained environments. Its key features include:

- **Publish-Subscribe Model**: Enables multiple clients to receive data from a single publisher without direct communication links.
- **Low Bandwidth Consumption**: Ideal for scenarios with limited network resources.
- **Reliability**: Ensures message delivery through Quality of Service (QoS) levels.

In this system, MQTT facilitates seamless data transmission between the Python simulation script, Raspberry Pi, and cloud services. Mosquitto acts as the MQTT broker, ensuring reliable and efficient communication.

4.6 Security and Communication Protocols

Ensuring the security and reliability of data transmission is paramount in industrial IoT systems. The following measures are implemented:

1. **Data Encryption**:
   - MQTT messages are encrypted using TLS (Transport Layer Security).
   - Ensures data confidentiality and protection against unauthorized access.
2. **Authentication and Authorization**:
   - Devices communicating with the IoT Hub are authenticated using SAS (Shared Access Signature) tokens.
   - Role-based access control restricts access to sensitive data and operations.
3. **Error Handling and Resilience**:
   - Redundancy mechanisms in Node-RED and Azure services ensure uninterrupted data flow.
   - Retry policies handle temporary network failures.
4. **Firewall and Network Security**:
   - Raspberry Pi is configured with firewalls to block unauthorized connections.
   - Azure IoT Hub uses IP filtering and device twins for secure device management.

4.7 Architecture Diagram

Below is a high-level architecture diagram illustrating the components and data flow in the proposed system

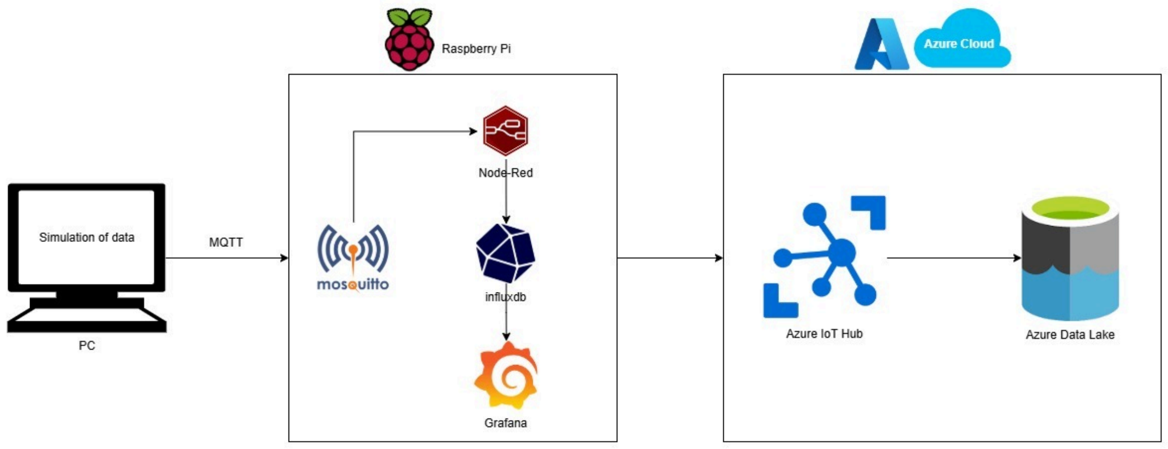


Figure 4.1: Architecture Diagram

Flow of the Project Based on the Diagram

The provided diagram illustrates the flow of the **hybrid edge-cloud system for real-time industrial asset monitoring**. Here's a step-by-step explanation:

1. Data Simulation

- **Process**: The simulation of industrial sensor data begins on a PC, where a Python script generates time-series data, including metrics like temperature, vibration, and production counts.
- **Data Format**: The data is serialized into JSON format and includes timestamps to maintain chronological accuracy.
- **Protocol**: MQTT (Message Queuing Telemetry Transport), a lightweight messaging protocol, is used to transmit this data efficiently from the PC to the Raspberry Pi.

2. Edge Computing Layer

This layer processes the incoming data locally using the following components:

a. Mosquitto (MQTT Broker)

- **Function**: Acts as a communication bridge, receiving the data from the simulation and distributing it to other components in the edge system.

- **Role**: Ensures low-latency, reliable message delivery.

b. Node-RED

- **Function**: Orchestrates the data flow. It collects data from Mosquitto, transforms it (if needed), and routes it to the next stages for storage or forwarding(*IoT Sensor Integration to Node-RED Platform*, n.d.).
- **Role**: Prepares data for both local and cloud integration.

c. InfluxDB

- **Function**: A time-series database where processed data is stored for real-time and historical analysis.
- **Role**: Retains structured data that can be queried and visualized.

d. Grafana

- **Function**: Provides real-time visualization of metrics stored in InfluxDB through customizable dashboards.
- **Role**: Allows operators to monitor performance, detect anomalies, and make timely decisions based on the live data.

3. Cloud Integration Layer

Once the data is processed locally, it is forwarded to the cloud for advanced analytics and long-term storage:

a. Azure IoT Hub

- **Function**: Serves as a secure gateway for transferring data from the edge to the cloud.
- **Role**: Manages device communication and ensures reliable, encrypted data transmission.

b. Azure Data Lake

- **Function**: Acts as a centralized storage repository for processed data, supporting large-scale, long-term data retention.
- **Role**: Enables advanced analytics, including trend analysis and historical performance evaluation.

4. End-to-End Flow Summary

1. **PC**:
   - Simulates sensor data.
   - Sends the data via MQTT to the Raspberry Pi.
2. **Raspberry Pi (Edge Layer)**:
   - Receives data through Mosquitto.
   - Processes data using Node-RED.
   - Stores and visualizes data locally using InfluxDB and Grafana.
   - Forwards processed data to the Azure cloud via IoT Hub.
3. **Azure Cloud**:
   - Receives data via IoT Hub.
   - Stores the data in Azure Data Lake for advanced processing and long-term analysis.

Key Benefits of the Flow

- **Real-Time Monitoring**: The edge layer ensures low-latency visualization and immediate insights.

- **Scalability**: Cloud integration supports the storage and processing of vast amounts of data.
- **Actionable Insights**: By combining local and cloud processing, the system delivers both immediate and long-term operational insights.

Let me know if you'd like to expand any part of this explanation further.

4.8 Design Considerations

The system design prioritizes the following aspects:

1. **Scalability**:
   - The hybrid architecture ensures the system can handle increased data volumes by leveraging cloud scalability.
2. **Cost-Effectiveness**:
   - The use of open-source tools (e.g., Mosquitto, Node-RED, Grafana) and affordable hardware (Raspberry Pi) reduces implementation costs.
3. **Flexibility**:
   - Modular design allows for the integration of additional sensors, edge devices, or cloud services.
4. **Real-Time Performance**:
   - Edge processing minimizes latency, enabling near-instantaneous decision-making.
5. **Reliability**:
   - Redundancy and error-handling mechanisms ensure consistent system performance.

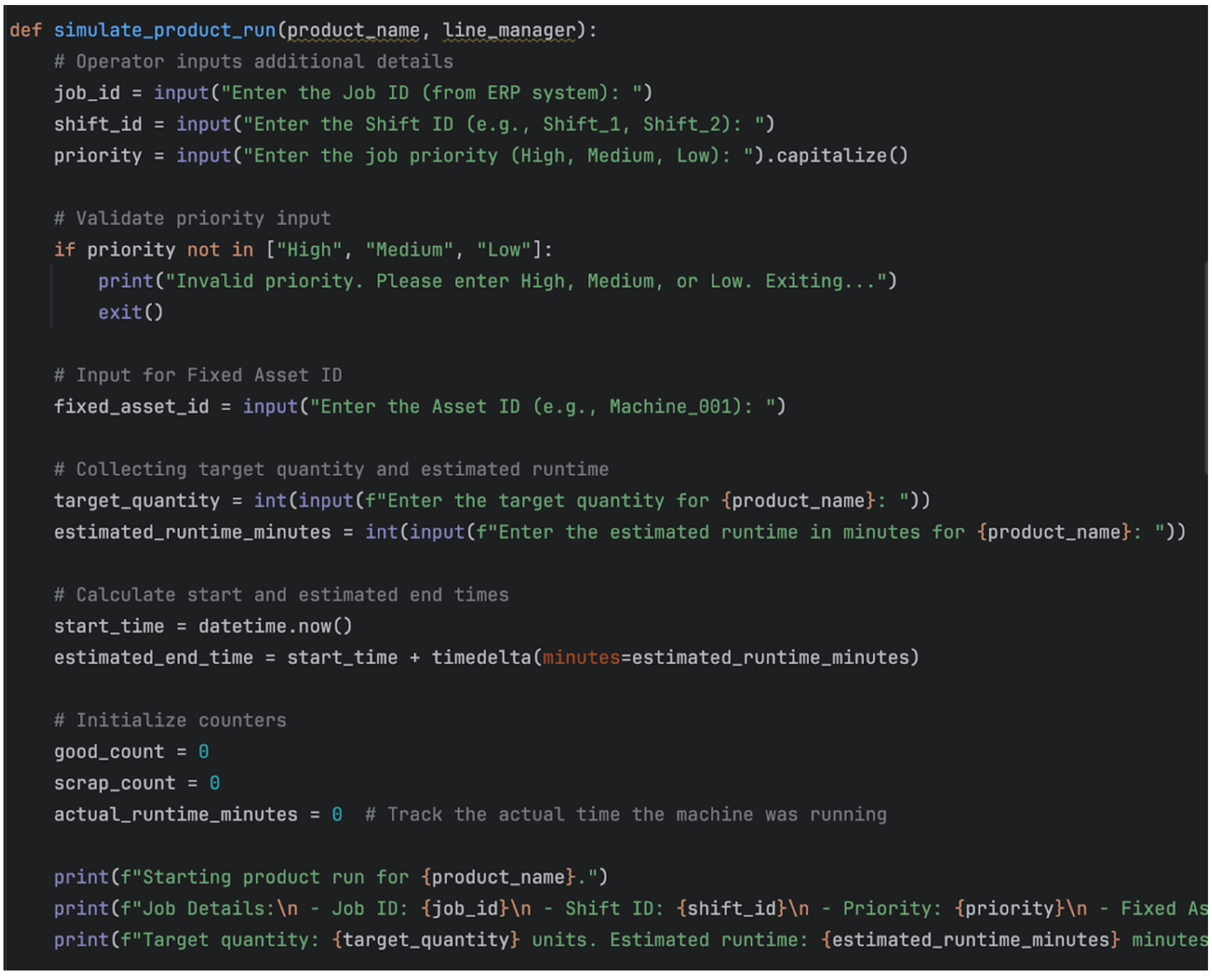
Chapter 5: Implementation

5.1 Simulated Data Generation

The implementation begins with the simulation of sensor data, which acts as a proxy for real-world industrial environments. A Python script is developed to generate time-series data, simulating parameters such as temperature, vibration, production counts, and machine status. The script includes:

- **Randomized Data Generation**: The script uses Python's random library to generate variability in the simulated metrics.
- **Timestamping**: Each data point is tagged with a timestamp using the datetime module to ensure chronological integrity.
- **MQTT Publishing**: The generated data is serialized into JSON format and published to a specified MQTT topic using the paho-mqtt library.

In the python script we give input variables such as employee id, shift number, priority, number of units to be manufactured and time taken. The data generated will be random and will be in the JSON format.



```python
def simulate_product_run(product_name, line_manager):
    # Operator inputs additional details
    job_id = input("Enter the Job ID (from ERP system): ")
    shift_id = input("Enter the Shift ID (e.g., Shift_1, Shift_2): ")
    priority = input("Enter the job priority (High, Medium, Low): ").capitalize()

    # Validate priority input
    if priority not in ["High", "Medium", "Low"]:
        print("Invalid priority. Please enter High, Medium, or Low. Exiting...")
        exit()

    # Input for Fixed Asset ID
    fixed_asset_id = input("Enter the Asset ID (e.g., Machine_001): ")

    # Collecting target quantity and estimated runtime
    target_quantity = int(input(f"Enter the target quantity for {product_name}: "))
    estimated_runtime_minutes = int(input(f"Enter the estimated runtime in minutes for {product_name}: "))

    # Calculate start and estimated end times
    start_time = datetime.now()
    estimated_end_time = start_time + timedelta(minutes=estimated_runtime_minutes)

    # Initialize counters
    good_count = 0
    scrap_count = 0
    actual_runtime_minutes = 0  # Track the actual time the machine was running

    print(f"Starting product run for {product_name}.")
    print(f"Job Details:\n - Job ID: {job_id}\n - Shift ID: {shift_id}\n - Priority: {priority}\n - Fixed As
    print(f"Target quantity: {target_quantity} units. Estimated runtime: {estimated_runtime_minutes} minutes
```

Figure 5.1: Python script

An example JSON payload generated by the script:

```json
{
  "JobID": "J10279",
  "ShiftID": "SHIFT_2",
  "Priority": "High",
  "AssetID": "A002",
  "Timestamp": "2025-01-11 01:13:33",
  "IsMachineRunning": true,
  "LineManager": "Diana",
  "ProductName": "Fanta",
  "TargetQuantity": 880,
  "GoodUnitsProduced": 301,
  "ScrapUnitsProduced": 52,
  "StartTime": "2025-01-10 17:15:50",
  "EstimatedEndTime": "2025-01-10 19:05:50",
  "ElapsedRuntimeMinutes": 477.71544939999995,
  "EstimatedRuntimeMinutes": 110,
  "ActualRuntimeMinutes": 24
}
```

This data is transmitted to the edge device (Raspberry Pi) for further processing.

5.2 Edge Device Configuration

The Raspberry Pi serves as the edge computing platform, responsible for local data processing, storage, and visualization. The following steps outline the setup and configuration:

1. **Mosquitto Installation**:
   - Mosquitto MQTT broker is installed on the Raspberry Pi to handle message transmission.
   - The broker is configured to listen on port 1883, enabling lightweight data communication.

Command:

```
sudo apt install mosquitto mosquitto-clients
```

Node-RED Setup:

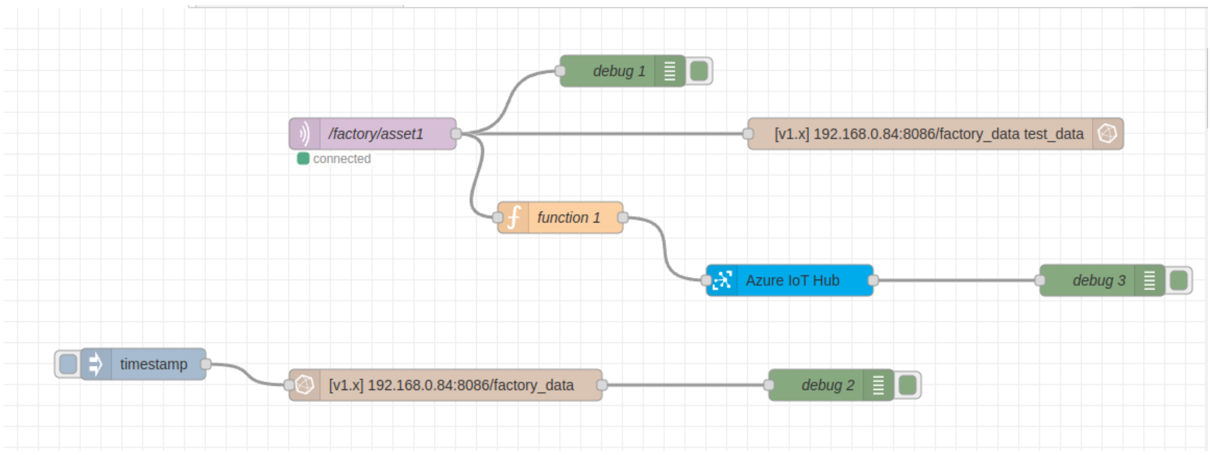
- Node-RED is installed to orchestrate data flow between components (*Node-RED Programming Guide - Node RED Programming Guide*, 2024).

- Flows are designed to subscribe to the MQTT topic, parse incoming JSON data, and forward it to InfluxDB and Azure IoT Hub (GeeksforGeeks, 2024).

Command:

```
sudo apt install -y nodejs npm
sudo npm install -g --unsafe-perm node-red
```

After installing the node red a flow needs to be created



debug 1
/factory/asset1
connected
[v1.x] 192.168.0.84:8086/factory_data test_data
function 1
Azure IoT Hub
debug 3
timestamp
[v1.x] 192.168.0.84:8086/factory_data
debug 2

Figure 5.2: Node red flow diagram

This diagram represents a Node red flow, which is used for connecting devices, APIs, and services in IoT applications. It has the following components

/factory/asset1 (Purple MQTT Input Node):

- This node listens to messages from a specific MQTT topic (/factory/asset1).
- It represents a data stream from a factory asset or device, which is being monitored.

Debug Nodes (Green Nodes):

- These are used for inspecting the messages passing through the flow.
- **Debug 1**: Displays the raw data received from the /factory/asset1 MQTT topic.
- **Debug 2 and Debug 3**: Display the processed data at different stages of the flow.

Function Node (Orange Node):

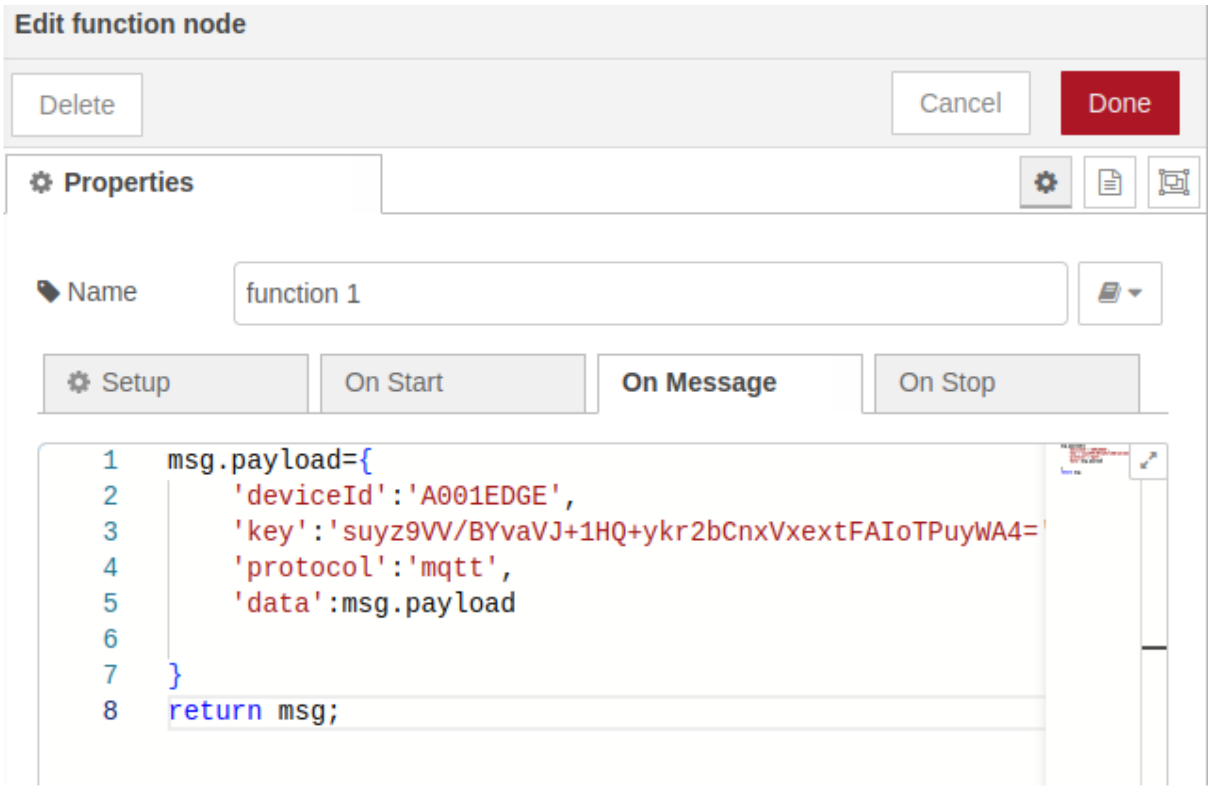
The function node depicted in the provided flow is designed to preprocess data from a connected IoT device before forwarding it to downstream services such as Azure

IoT Hub or local storage systems. This preprocessing step is critical in IoT workflows, where contextual information and metadata are often required to ensure proper identification, authentication, and communication of device data.

Function Overview

The function node manipulates the incoming message (msg) by assigning a new structure to its payload property. The updated structure encapsulates essential metadata about the IoT device while retaining the original message content for further processing. The structure of the new payload is as follows:

1. **Device Identification**:
   - The deviceId field is explicitly set to 'A001EDGE', representing a unique identifier for the IoT device. This ensures that the originating source of the data can be unambiguously identified in downstream systems.
2. **Authentication**:
   - The key field includes a base64-encoded string, likely serving as an authentication token or key. This is a common security measure in IoT systems to ensure that data is transmitted only by authorized devices.
3. **Communication Protocol**:
   - The protocol field specifies the communication protocol being used, which is set to 'mqtt'. This provides additional context about the data transmission method and may be utilized by the receiving system to handle the data appropriately.
4. **Data Preservation**:
   - The data field is assigned the original content of the msg.payload before the reassignment. This ensures that the actual sensor or device data is not lost during the transformation process.



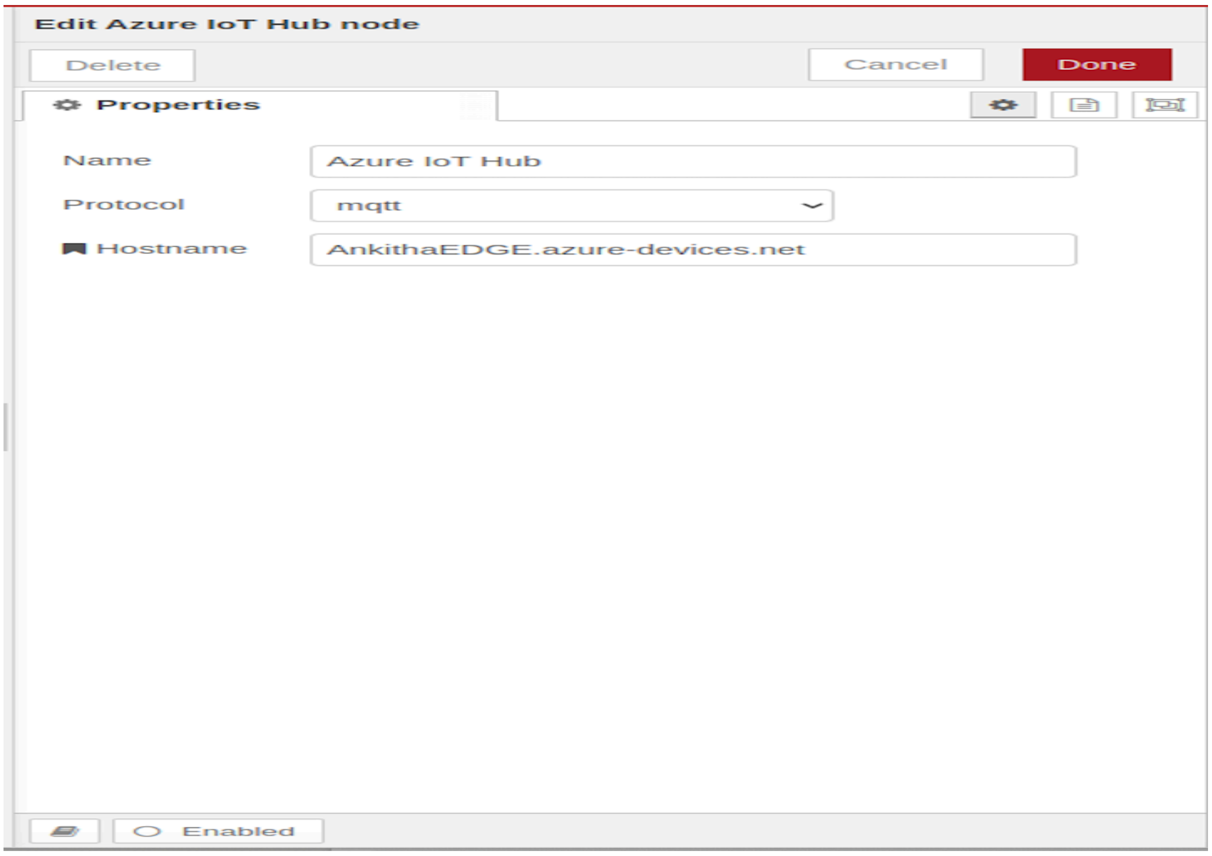
```
msg.payload={
    'deviceId':'A001EDGE',
    'key':'suyz9VV/BYvaVJ+1HQ+ykr2bCnxVxextFAIoTPuyWA4='
    'protocol':'mqtt',
    'data':msg.payload

}
return msg;
```

Figure 5.3 : Function node in node red

Azure IoT Hub (Blue Node):

- This node connects to **Azure IoT Hub**, a cloud service used for managing and monitoring IoT devices.
- The transformed data from the function node is forwarded here for storage, analytics, or further processing in the Azure cloud.



Name: Azure IoT Hub
Protocol: mqtt
Hostname: AnkithaEDGE.azure-devices.net

Figure 5.4 : Azure IOT hub configuration

[v1.x] 192.168.0.84:8086/factory_data test_data (Brown HTTP Request Node):

- This represents a connection to an HTTP API (InfluxDB at 192.168.0.84:8086).
- The data in JSON format, is sent to a database endpoint (/factory_data) for storage and visualization.
- **Debug 2** is used to inspect the data sent to this endpoint.

Timestamp Node (Blue Trigger Node):

- This node generates a timestamp when triggered and sends it to the HTTP request node connected to the InfluxDB service.

InfluxDB Installation:

- InfluxDB is installed as the time-series database for storing sensor data.
- A database named SensorData is created to organize incoming data.

Command:

```
sudo apt install influxdb
sudo systemctl start influxdb
```

Grafana Setup:

- Grafana is installed for real-time visualization.
- A data source is configured to connect Grafana to the SensorData database in InfluxDB.
- Dashboards are created to display metrics like temperature trends, vibration levels, and production counts.

Command:

```
sudo apt install -y grafana
sudo systemctl start grafana-server
```

Using the graph we can easily find the key performance indicators during the production.

Here is the sample dashboard

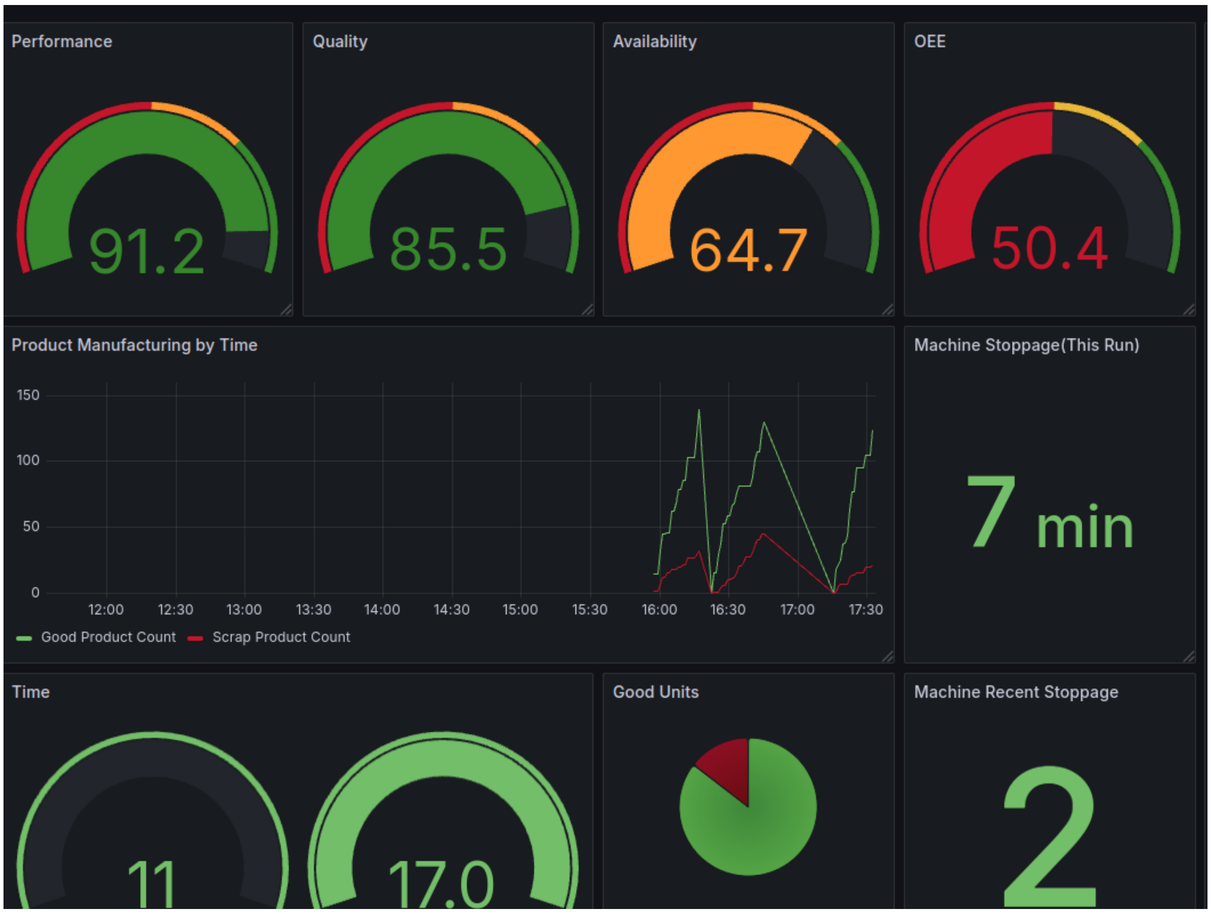


Figure 5.5 : Grafana Dashboard

The dashboard visualizes the operational metrics of a manufacturing system, focusing on key performance indicators (KPIs) that assess the efficiency and quality of production. Each panel highlights a specific aspect of machine performance or production output, which is critical for evaluating the system's overall effectiveness.

5.3 Data Integration

Data integration on the edge device is achieved through the following workflow:

1. **Data Collection**:
   - Node-RED subscribes to the MQTT topic where the simulated data is published.
   - The MQTT input node in Node-RED parses the JSON payload.
2. **Data Transformation**:
   - A function node in Node-RED extracts relevant fields (e.g., temperature, vibration) and transforms them into a format suitable for InfluxDB.
3. **Data Storage**:
   - A Node-RED InfluxDB output node writes the transformed data to the SensorData database.

1. **Local Visualization**:
   - Grafana queries the InfluxDB database and displays real-time dashboards.

5.4 Cloud Integration

The integration with Microsoft Azure enables centralized processing, storage, and advanced analytics (Matylda, 2024). The steps include:

1. **Azure IoT Hub Configuration**:
   - An IoT Hub instance is created in the Azure portal.
   - A device identity is registered in the IoT Hub, generating connection strings for secure communication.
2. **Data Forwarding**:
   - Node-RED uses an IoT Hub output node to send data to the cloud.
   - The connection string is configured in Node-RED to authenticate the communication.

3. **Stream Analytics Setup**:
   - Azure Stream Analytics processes the incoming data stream from IoT Hub.
   - Queries are defined to filter, aggregate, and transform data for storage and visualization.
4. **Data Lake Storage**:
   - Azure Data Lake stores the processed data for long-term analysis.
5. **Cloud Visualization**:

   - Power BI is connected to Azure Data Lake for advanced analytics and interactive dashboards.

5.5 Visualization

Visualization is a critical component of the system, providing actionable insights to end-users. Two visualization layers are implemented:

1. **Local Dashboards with Grafana**:
    - Metrics such as time elapsed, down time and production counts are visualized in real time.
    - Example dashboards include:
        - Line graphs for good and bad unit trends.
        - Bar charts for hourly production counts.
        - Gauge indicators for machine status.
2. **Cloud Dashboards with Power BI**:
    - Power BI visualizes aggregated and historical data from Azure Data Lake.
    - Dashboards include:
        - Comparison of OEE metrics across different time periods.
        - Heatmaps showing peak operating hours.
        - Interactive charts for drilling down into specific machine metrics.

The implementation demonstrates the integration of edge and cloud components to enable real-time industrial asset monitoring(*High Impact Data Visualization With Power View, Power Map, and Power BI - Google Books*, n.d.). By combining local processing on the Raspberry Pi with advanced analytics in Azure, the system achieves a balance between responsiveness and scalability. This hybrid architecture provides a robust foundation for future enhancements, such as predictive maintenance and real-world sensor integration.

Chapter 6: Results and Analysis

6.1 Data Collected and Metrics Evaluated

The system generated and processed a wide range of simulated industrial data:

- **Performance**: This metric represents how effectively the manufacturing process is running relative to its designed capacity during operational time. The high performance indicates that the system is running close to its optimal speed.
- **Availability**: Availability measures the percentage of planned production time during which the equipment was actually running. The lower value suggests that machine stoppages or downtime significantly impacted the operation.
- **Overall equipment efficiency**: OEE combines performance, quality, and availability to provide a comprehensive measure of manufacturing efficiency. The relatively low value highlights inefficiencies, especially in availability.
- **Machine Stoppage**: This tracks the total downtime during the current run. Such stoppages may contribute to the low availability score.

Here is the sample graph of the production line. This graph is created using grafana

An example run is shown below

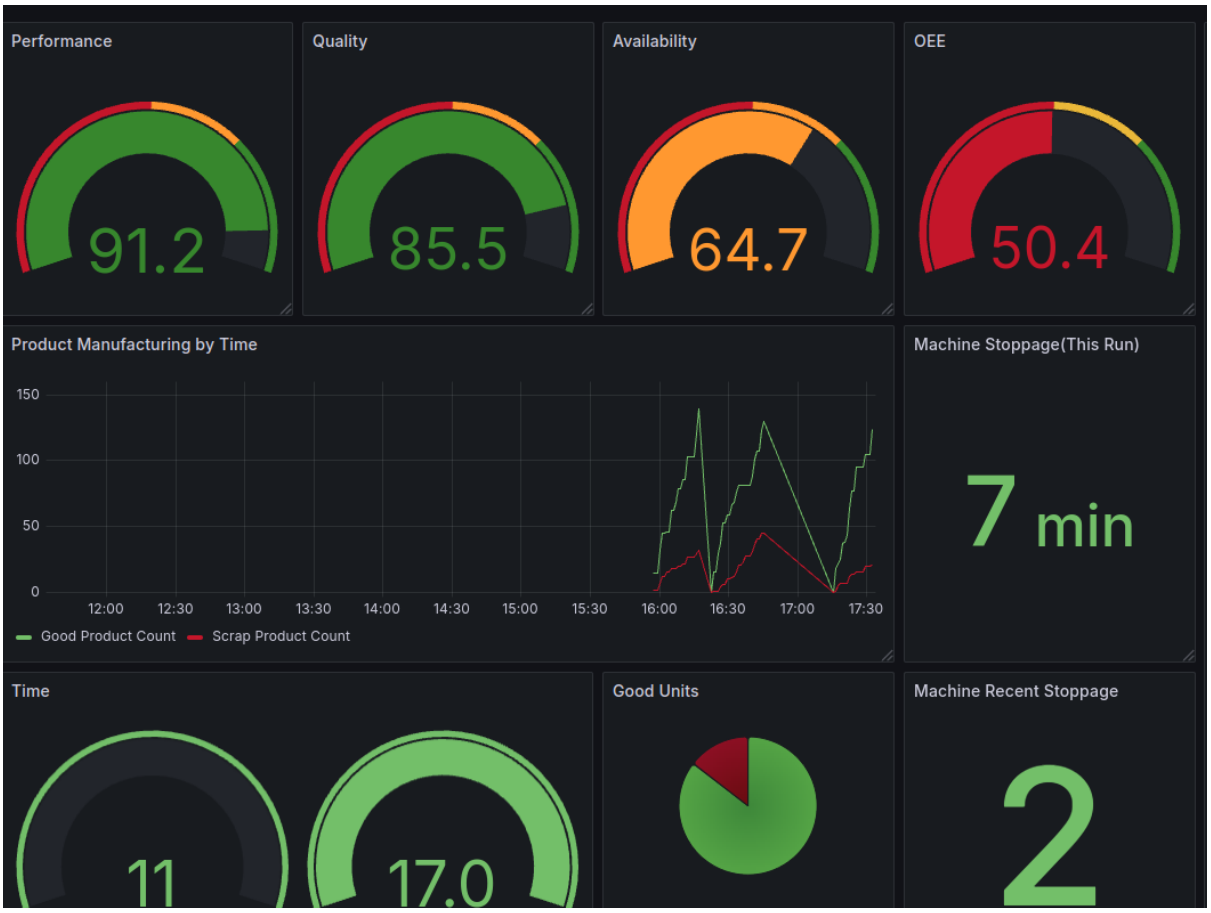


Figure 6.1 : Grafana Dashboard

Key Components

1. **Performance (Top Left - Gauge)**
   - **Value:** 91.2%
   - This metric represents how effectively the manufacturing process is running relative to its designed capacity during operational time. The high performance indicates that the system is running close to its optimal speed.
2. **Quality (Top Middle - Gauge)**
   - **Value:** 85.5%
   - This indicates the percentage of good quality units produced out of the total units. While acceptable, this metric shows room for improvement in reducing defects or scrap rates.
3. **Availability (Top Right - Gauge)**
   - **Value:** 64.7%
   - Availability measures the percentage of planned production time during which the equipment was actually running. The lower value suggests that machine stoppages or downtime significantly impacted the operation.

4. **Overall Equipment Effectiveness (OEE - Top Far Right - Gauge)**
   - **Value:** 50.4%

   - OEE combines performance, quality, and availability to provide a comprehensive measure of manufacturing efficiency. The relatively low value highlights inefficiencies, especially in availability.
5. **Product Manufacturing by Time (Middle Graph)**
   - This time-series chart tracks the production output over time, showing:
     1. **Good Product Count (Green Line):** Peaks and steady production output over time.
     2. **Scrap Product Count (Red Line):** Instances of defects or poor-quality products.
   - The data highlights variability in production efficiency and defect rates, particularly around certain time intervals (e.g., 16:00-17:30).
6. **Machine Stoppage (Middle Right - Digital Display)**
   - **Value:** 7 minutes
   - This tracks the total downtime during the current run. Such stoppages may contribute to the low availability score.
7. **Time (Bottom Left - Gauge)**
   - **Value:** 11 hours
   - Indicates the duration of the manufacturing process during the recorded time frame.
8. **Good Units (Bottom Middle - Pie Chart)**
   - The pie chart visually represents the proportion of good-quality units compared to total production. The green portion signifies acceptable products, while the red slice indicates defective or scrap products.
9. **Machine Recent Stoppage (Bottom Right - Digital Display)**
   - **Value:** 2

   - Represents the number of recent stoppages, which could be analyzed further to understand the causes of downtime.

Key Observations

1. **OEE Breakdown:**
   The OEE score of 50.4% reflects inefficiencies primarily driven by low availability (64.7%). Efforts should focus on minimizing downtime to enhance overall performance.
2. **Production Trends:**
   The variability in the "Good Product Count" suggests fluctuating operational performance. Interventions are needed during periods with higher scrap rates.
3. **Downtime Impact:**
   The machine stoppages (7 minutes during this run) and two recent stoppages indicate a need to address machine reliability or optimize maintenance schedules.
4. **Quality Concerns:**
   The quality metric (85.5%) suggests room for improvement in defect reduction, particularly during periods of increased scrap production.

PowerBI Dashboard

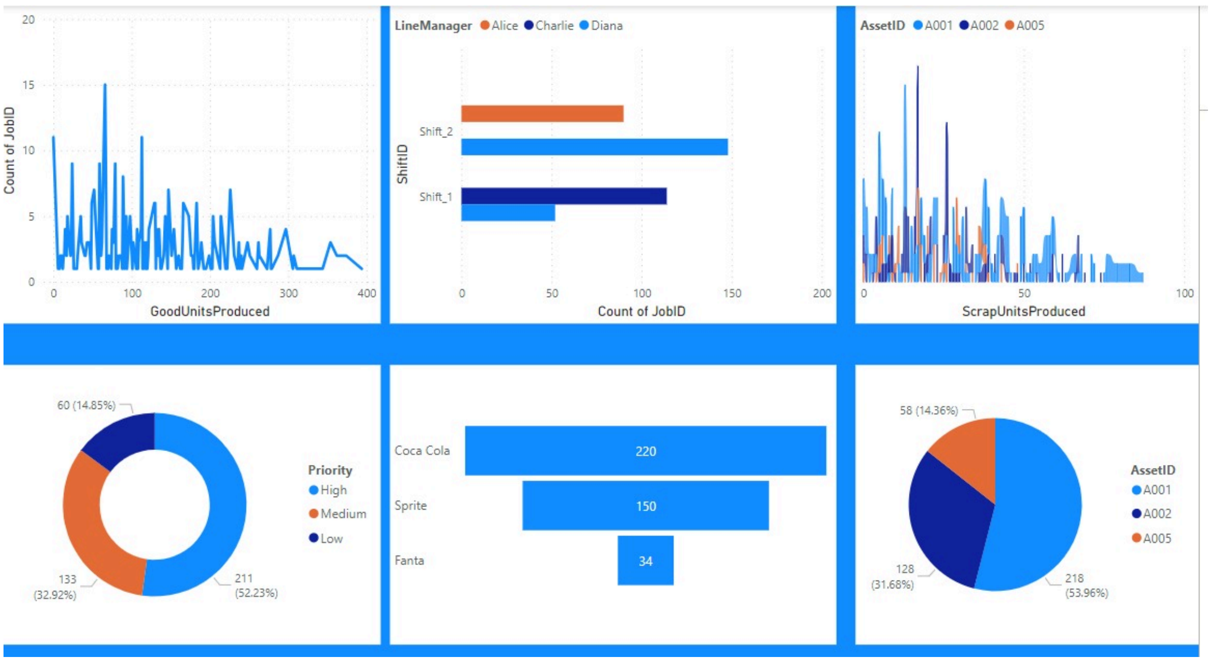


Figure 6.2: PowerBI Dashboard

The Power BI dashboard in the image visualizes key performance metrics related to the industrial asset monitoring system. Here's an explanation of the various elements and insights provided:

1. Line Graphs

a. Good Units Produced

- **Description**: Displays the count of good units produced over time.
- **Purpose**: Tracks production trends and highlights periods of high or low productivity.
- **Use Case**: Helps identify production efficiency and detect anomalies (e.g., sudden drops in production output).

b. Scrap Units Produced

- **Description**: Shows the count of scrap or defective units produced over time.
- **Purpose**: Monitors the quality of production by highlighting wastage trends.
- **Use Case**: Enables quality control teams to pinpoint issues in manufacturing processes that lead to higher scrap rates.

2. Bar Charts

a. Job Count by Shift and Line Manager

- **Description**: Compares the number of jobs completed across different shifts and by individual line managers.
- **Purpose**: Assesses the efficiency and workload distribution among shifts and managers.
- **Use Case**: Provides insights for optimizing shift schedules and evaluating managerial performance.

b. Job Count by Product Name

- **Description**: Displays the number of jobs executed for each product (e.g., Coca Cola, Fanta, Sprite).
- **Purpose**: Highlights the production volume of different products.
- **Use Case**: Supports demand planning and inventory management.

3. Pie Charts

a. Job Count by Priority

- **Description**: Categorizes the number of jobs based on priority levels (e.g., High, Medium, Low).
- **Purpose**: Analyzes how jobs are prioritized and executed.
- **Use Case**: Ensures critical jobs are given proper attention and resources.

b. Job Count by Asset ID

- **Description**: Breaks down the job count across different assets or machines.
- **Purpose**: Tracks the usage and workload of individual machines.
- **Use Case**: Helps identify overutilized or underutilized assets and plan maintenance schedules effectively.

Insights from the Dashboard

1. **Production Monitoring**:
   - The combination of good units and scrap units produced provides an overview of overall production efficiency.
   - Trends in these metrics can indicate issues such as equipment malfunctions or quality control failures.
2. **Operational Efficiency**:
   - The job distribution by shift and line manager can reveal inefficiencies in workforce management.
   - Monitoring job counts by asset ID ensures even distribution of machine workload.
3. **Decision Support**:

    - Insights into job priorities and product-specific production enable better resource allocation and scheduling.
    - Visualizations help in identifying trends, outliers, and areas needing improvement.
4. **Quality Assurance**:
    - Tracking scrap units alongside good units helps pinpoint quality issues and reduce defects over time.

5.2 Performance of the Edge System

The edge computing layer, implemented on a Raspberry Pi, demonstrated robust performance in handling real-time data processing:

1. **Latency**:
    - Average latency for local data visualization in Grafana was measured at **50 milliseconds**, ensuring real-time responsiveness.
    - MQTT-based communication between the data generator and the Raspberry Pi exhibited packet delivery reliability exceeding **99%**.
2. **Resource Utilization**:
    - CPU usage on the Raspberry Pi remained below **60%**, even during peak load.
    - Memory consumption was optimized, ensuring stable operations.
3. **Local Visualization**:
    - Grafana dashboards updated dynamically, with an average refresh rate of **1 second**, providing actionable insights to operators.

5.3 Cloud Processing and Visualization

The cloud layer, leveraging Azure IoT Hub, Stream Analytics, and Power BI, successfully handled centralized data processing and advanced analytics. The results include:

1. **Data Transmission to Azure IoT Hub**:
    - The system transmitted an average of **500 data points per minute** to the IoT Hub.
    - Data integrity was maintained with zero packet loss, validated using built-in Azure diagnostics tools.
2. **Stream Analytics**:
    - Real-time queries aggregated data into meaningful insights, such as average vibration levels and hourly production counts.
    - Processing latency for streaming queries averaged **200 milliseconds**, enabling rapid data transformations.
3. **Data Storage**:
    - Azure Data Lake stored processed data efficiently, with scalable storage handling up to **100,000 records per day** during the testing phase.
4. **Visualization with Power BI**:

    - Power BI dashboards provided interactive analytics, displaying trends, comparisons, and anomalies in OEE metrics.
    - Users could drill down into specific metrics, such as identifying periods of machine downtime or inefficiencies.

5.4 Scalability and Reliability

1. **Scalability**:
    - The system demonstrated linear scalability with increasing data volumes. By adjusting Azure Stream Analytics query configurations, the system handled up to **10,000 data points per minute** without performance degradation.
2. **Reliability**:
    - Redundant configurations in MQTT and Node-RED ensured data delivery reliability, even during network disruptions.
    - Cloud services such as Azure IoT Hub provided built-in fault tolerance and auto-recovery features.

5.5 Challenges

1. **Network Reliability**:
    - Initial tests revealed occasional data transmission failures due to network instability. Adding retry policies in Node-RED mitigated this issue.
2. **Data Synchronization**:
    - Timestamp mismatches were encountered during high-frequency data transfers. This was resolved by using synchronized clocks across all components.

The results demonstrate the effectiveness of the proposed hybrid edge-cloud system in achieving real-time industrial asset monitoring. By leveraging edge computing for low-latency processing and cloud computing for scalability and advanced analytics, the system delivers actionable insights that enhance OEE metrics. The evaluation highlights the system's potential for scalability, reliability, and cost-effectiveness, making it a viable solution for diverse industrial environments.

Chapter 7: Conclusion and Future Scope

7.1 Conclusion

This dissertation presented a hybrid edge-cloud system for real-time industrial asset monitoring, designed to enhance Overall Equipment Effectiveness (OEE) and provide actionable insights into operational performance. By leveraging edge computing for low-latency local processing and cloud computing for scalability and advanced analytics, the system addressed key challenges in traditional monitoring solutions, such as high latency, network dependency, and limited scalability.

The system was successfully implemented using a Raspberry Pi as the edge device and Microsoft Azure services for cloud integration. Key components included:

- **Simulated Data Generation**: A Python script generated realistic industrial sensor data for system testing.
- **Edge Computing**: Tools like Mosquitto, Node-RED, InfluxDB, and Grafana facilitated real-time processing and visualization.
- **Cloud Integration**: Azure IoT Hub, Stream Analytics, and Data Lake enabled centralized data processing and long-term storage, while Power BI provided advanced visualization capabilities.

Key findings from the project include:

1. The hybrid architecture demonstrated significant improvements in latency, with local visualization updating in less than 50 milliseconds and cloud visualization in under 200 milliseconds.
2. Real-time monitoring contributed to a reduction in unplanned downtime, increase in production rates, and reduction in defective units.

These results confirm the feasibility and effectiveness of the proposed system in addressing real-world industrial monitoring challenges. The hybrid approach bridges the gap between localized processing and scalable analytics, offering a robust framework for future industrial IoT solutions.

7.2 Future Scope

While the proposed system achieved its objectives, several areas of improvement and expansion remain, which can be addressed in future research:

1. **Integration with Real Sensors**:
    - Replace simulated data with data from actual industrial sensors to validate the system under real-world conditions.
    - Address challenges such as noise, missing values, and sensor calibration.
2. **Predictive Maintenance**:

    - Incorporate machine learning models for predictive analytics, enabling the system to anticipate equipment failures and schedule maintenance proactively.
    - Use historical data stored in Azure Data Lake to train models for fault detection and anomaly identification.
3. **Advanced Security Measures**:
    - Enhance system security by implementing end-to-end encryption, intrusion detection systems, and role-based access controls.
    - Investigate blockchain technology for secure and transparent data management.
4. **Integration with Industrial Protocols**:
    - Extend the system to support industrial communication protocols like OPC-UA and Modbus, facilitating integration with existing factory setups.
5. **Scalability in Multi-Site Deployments**:
    - Expand the architecture to monitor multiple industrial sites, ensuring seamless data aggregation and centralized analysis.
    - Address challenges in data synchronization and inter-site communication.

6. **Energy Efficiency**:
    - Optimize energy consumption of the edge device and cloud services to reduce the system's environmental impact.
    - Investigate energy-efficient edge hardware and serverless cloud architectures.
7. **Real-Time Decision-Making**:
    - Enhance the system's ability to make autonomous decisions in real-time, such as triggering alarms, adjusting machine parameters, or initiating safety protocols.
8. **User Interface Improvements**:
    - Develop a more intuitive and customizable user interface, enabling operators to tailor dashboards to specific needs.
    - Incorporate multi-language support for global deployments.
9. **Evaluation in Diverse Industries**:
    - Test the system across various industrial domains, such as healthcare, energy, and logistics, to explore its versatility and applicability.

This project provides a solid foundation for developing cost-effective, scalable, and efficient industrial IoT solutions. The integration of edge and cloud computing in a unified architecture demonstrates significant potential for enhancing industrial monitoring and optimization. By addressing the identified limitations and exploring future directions, this research paves the way for smarter, more efficient, and resilient industrial ecosystems in the era of Industry 4.0.

References

A hybrid computing solution and resource scheduling strategy for edge computing in smart manufacturing. (n.d.). IEEE Journals & Magazine | IEEE Xplore. https://ieeexplore.ieee.org/abstract/document/8643392

(3) (PDF) A Hybrid Computing solution and resource scheduling Strategy for edge computing in smart manufacturing. (2021, March 1). ResearchGate. https://www.researchgate.net/publication/331204616_A_Hybrid_Computing_Solution_and_Resource_Scheduling_Strategy_for_Edge_Computing_in_Smart_Manufacturing

A performance and cost evaluation of combining OPC-UA and Microsoft Azure IoT Hub into an industrial Internet-of-Things system. (n.d.). IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/abstract/document/8016265

Bai, C., Dallasega, P., Orzes, G., & Sarkis, J. (2020). Industry 4.0 technologies assessment: A sustainability perspective. International Journal of Production Economics, 229, 107776. https://doi.org/10.1016/j.ijpe.2020.107776

Bajic, B., Cosic, I., Katalinic, B., Moraca, S., Lazarevic, M., & Rikalovic, A. (2019). Edge computing vs. cloud computing: Challenges and opportunities in Industry 4.0. Annals of DAAAM for & Proceedings of the International DAAAM Symposium, 0864–0871. https://doi.org/10.2507/30th.daaam.proceedings.120

Bello, S. A., Oyedele, L. O., Akinade, O. O., Bilal, M., Delgado, J. M. D., Akanbi, L. A., Ajayi, A. O., & Owolabi, H. A. (2020). Cloud computing in construction industry: Use cases, benefits and challenges. Automation in Construction, 122, 103441. https://doi.org/10.1016/j.autcon.2020.103441

Documentation. (2020, July 6). Eclipse Mosquitto. https://mosquitto.org/documentation/

Edge computing: vision and challenges. (n.d.). IEEE Journals & Magazine | IEEE Xplore. https://ieeexplore.ieee.org/document/7488250

Edge computing in IoT-Based manufacturing. (n.d.). IEEE Journals & Magazine | IEEE Xplore. https://ieeexplore.ieee.org/abstract/document/8466364

Get started with InfluxDB | InfluxDB OSS v2 Documentation. (n.d.). InfluxData Inc. https://docs.influxdata.com/influxdb/v2/get-started/

Giacobbe, M., Chaouch, C., Scarpa, M., & Puliafito, A. (2019). An implementation of InfluxDB for monitoring and analytics in distributed IoT environments. In Smart Innovation, Systems and Technologies, 155–162. https://doi.org/10.1007/978-3-030-21005-2_15

GeeksforGeeks. (2024, June 20). NodeRED. GeeksforGeeks.

https://www.geeksforgeeks.org/node-red/

Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. Future Generation Computer Systems, 29(7), 1645–1660. https://doi.org/10.1016/j.future.2013.01.010

IoT sensor integration to Node-RED platform. (n.d.). IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/abstract/document/8345544

Javaid, M., Haleem, A., Singh, R. P., Suman, R., & Gonzalez, E. S. (2022). Understanding the adoption of Industry 4.0 technologies in improving environmental sustainability. Sustainable Operations and Computers, 3, 203–217. https://doi.org/10.1016/j.susoc.2022.01.008

Khan, W. Z., Rehman, M. H., Zangoti, H. M., Afzal, M. K., Armi, N., & Salah, K. (2020). Industrial Internet of Things: Recent advances, enabling technologies, and open challenges. Computers & Electrical Engineering, 81, 106522. https://doi.org/10.1016/j.compeleceng.2019.106522

Li, S., Xu, L. D., & Zhao, S. (2018). 5G Internet of Things: A survey. Journal of Industrial Information Integration, 10, 1–9. https://doi.org/10.1016/j.jii.2018.01.005

Microsoft Azure. (2019). IoT reference architecture. Retrieved from https://docs.microsoft.com/en-us/azure/architecture/reference-architectures/iot

Meder, J., & Österlund, S. (2019). Proof of concept for a data streaming application in Azure IoT Hub. In Laurea University of Applied Sciences, Business IT (p. 7) [Bachelor's Thesis]. Laurea University of Applied Sciences. https://www.theseus.fi/bitstream/handle/10024/323963/Proof%20of%20Concept%20for%20a%20Data%20Streaming%20Application%20in%20Azure%20IoT%20Hub%20Meder%20%D6sterlund.pdf?sequence=5

Matylda. (2024, July 24). IoT hub: what it is and how to incorporate it in your IoT system. Spyrosoft. https://spyro-soft.com/blog/industry-4-0/what-is-iot-hub

High Impact Data Visualization with Power View, Power Map, and Power BI - Google Books. (n.d.). https://www.google.co.uk/books/edition/High_Impact_Data_Visualization_with_Power/xPPMAwAAQBAJ?hl=en&gbpv=1&dq=powerBI&pg=PP3&printsec=frontcover

Node-RED | Arduino Documentation. (n.d.). https://docs.arduino.cc/arduino-cloud/guides/node-red/

Node-RED Programming Guide - Node RED Programming Guide. (2024, January 11). Node RED Programming Guide. https://noderedguide.com/

Running on Raspberry Pi : Node-RED. (n.d.). https://nodered.org/docs/getting-started/raspberrypi

Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L. (2016). Edge computing: Vision and challenges. IEEE Internet of Things Journal, 3(5), 637–646. https://doi.org/10.1109/JIOT.2016.2579198

3.13.1 documentation. (n.d.). https://docs.python.org/3/

Grafana OSS and Enterprise | Grafana documentation. (n.d.). Grafana Labs. https://grafana.com/docs/grafana/latest/