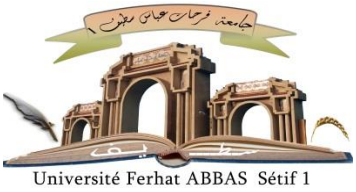
الجمهورية الجزائرية الديمقراطية الشعبية

République Algérienne Démocratique et Populaire

Ministère de L'Enseignement Supérieur et de la Recherche Scientifique



UNIVERSITÉ FERHAT ABBAS - SETIF1

FACULTÉ DE TECHNOLOGIE

THÈSE

Présentée au Département d'Electrotechnique

Pour l'obtention du diplôme de

MASTER

Domaine : Sciences et Technologie

Filière: Electrotechnique **Option: Commande électrique**

Par

M. Chemakh Rafik Seradj Eddine

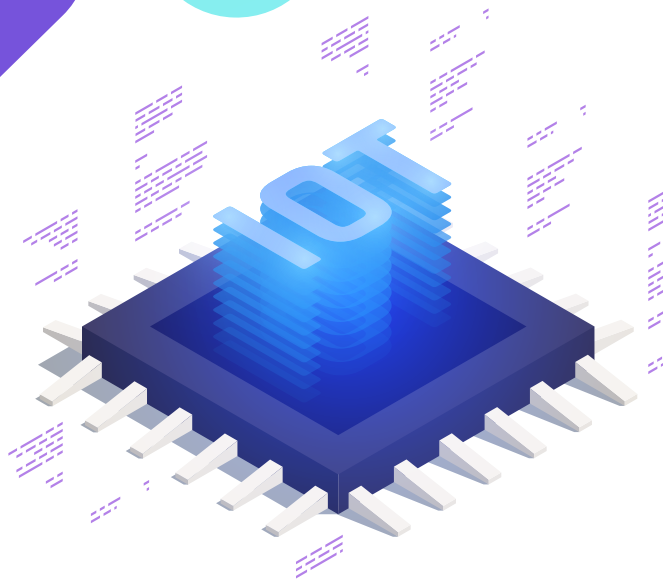
THÈME

Conception d'un système IoT: application à la collecte d'info d'un système électromécanique

Soutenue le 25/06/2024 devant le Jury:

Dr. Daili Yacine	**M.C.A**	**Univ. Ferhat Abbas Sétif 1**	**Président**
Pr. HARRAG Abdelghani	**Professeur**	**Univ. Ferhat Abbas Sétif 1**	**Directeur de Mémoire**
M. DAAS Abdelaziz	**M.A.A**	**Univ. Ferhat Abbas Sétif 1**	**Examinateur**

جامعة سطيف 1
UNIVERSITY OF SETIF 1



Ferhat Abbas University Setif 1

Master thesis

Concept of IoT model

application to the acquisition of data from an electromechanical system.

Created By:

Rafik Seradj Eddine Chemakh

2023/2024

DEDICATION & ACKNOWLEDGMENTS

JUN 23, 2024

First and foremost, to my beloved family, thank you for your unwavering love and support. I couldn't have reached this milestone without you.

To my fellow graduates of the class of 2024, thank you for the unforgettable memories and friendships we've shared over the past two years. You've made this journey so much brighter.

And to the incredible professors, mentors, and even those kind strangers who offered guidance and encouragement along the way, your contributions have shaped my path and made a lasting impact on my life.

Chemakh Rafik Seradj Eddine
2023/2024

Table of Contents

Chapter II: Setting up IoT tools for the Data Collection

Chapter III: Collecting Data and Monitoring a Box Sorting System

List of Figures

List of Scripts and Tables

List of Acronyms

- **IoT:** Internet of Things
- **MQTT:** Message Queuing Telemetry Transport
- **CoAP:** Constrained Application Protocol
- **HTTP:** Hypertext Transfer Protocol
- **AMQP:** Advanced Message Queuing Protocol
- **DDS:** Data Distribution Service
- **LPWAN:** Low-Power Wide Area Network
- **LoRaWAN:** Long Range Wide Area Network
- **PAN:** Personal Area Network
- **PLC:** Programmable Logic Controller
- **CNC:** Computer Numerical Control
- **RDBMS:** Relational Database Management System
- **SQL:** Structured Query Language
- **NoSQL:** Not Only SQL
- **TSDB:** Time-Series Database
- **AI:** Artificial Intelligence
- **DDoS:** Distributed Denial of Service
- **IDS:** Intrusion Detection System
- **GE:** General Electric
- **AMI:** Amazon Machine Image
- **SSH:** Secure Shell
- **TSDB:** Time Series Database
- **API:** Application Programming Interface
- **HMI:** Human-Machine Interface
- **SCADA:** Supervisory Control and Data Acquisition
- **OPC UA:** OPC Unified Architecture
- **OLE:** Object Linking and Embedding
- **CSV:** Comma-Separated Values

- **SSL:** Secure Sockets Layer
- **CA:** Certificate Authority
- **DNS:** Domain Name System
- **ACME.sh:** Automated Certificate Management Environment shell script
- **Socat:** SOcket CAT
- **HSC:** High-Speed Counter
- **SCL:** Structured Control Language
- **TON:** Timer ON Delay
- **IPv4:** Internet Protocol version 4
- **IPv6:** Internet Protocol version 6
- **TCP:** Transmission Control Protocol

General Introduction

The Internet of Things (IoT) is a rapidly growing field that connects everyday objects to the internet, enabling them to collect and exchange data. This technology has the potential to revolutionize various industries, including manufacturing, healthcare, and transportation. Electromechanical systems, which combine electrical and mechanical components, are essential in many IoT applications.

This work explores the integration of IoT with electromechanical systems, focusing on monitoring and controlling a box sorting process. By leveraging Factory I/O for simulation and TIA Portal for PLC programming, the thesis covers project creation, hardware configuration, scene setup, PLC simulation, input/output mapping, and ladder logic programming. It delves into data collection using OPC UA and Node-RED, storing data in InfluxDB, and visualizing it through Grafana dashboards, secured with SSL certificates and accessed via domain names for enhanced security. Additionally, feedback mechanisms are implemented to detect malfunctions and track lost boxes, enhancing system reliability. The thesis concludes by demonstrating the creation of a custom function block in TIA Portal to monitor machine runtime, providing valuable insights for maintenance and optimization. This research contributes to the growing field of IoT applications in industrial automation, showcasing the potential for improved efficiency, data-driven decision-making, and enhanced system performance in real-world scenarios.

- Chapter I provides a foundational understanding of IoT and electromechanical systems. It covers the basic concepts, history, and evolution of IoT, along with its various architectures. It also delves into the components of electromechanical systems, their applications, and how they integrate with IoT for data collection and analysis. Additionally, the chapter explores cloud computing, security and privacy concerns in IoT, and reviews existing IoT solutions with real-world case studies.

- Chapter II outlines the process of setting up the necessary tools for data collection in an IoT system. It covers the installation and configuration of software like Factory I/O for simulation, PLC SIM ADVANCE 6.0 for data acquisition, Node-RED for data processing,

and AWS EC2 with InfluxDB for data storage. It also explains how to integrate Grafana for real-time data visualization and monitoring.

- Chapter III focuses on the practical implementation of an IoT system for monitoring and controlling a box sorting process. It details the step-by-step process of creating the project in TIA Portal, configuring the PLC hardware, and setting up the Factory I/O scene with conveyors, sensors, actuators, and an electric switchboard. The chapter explains how to simulate a PLC device, create input and output mappings, and write the control program using ladder logic. It further explores data collection by activating the OPC UA server in the PLC and utilizing Node-RED to read tag values and send them to an InfluxDB database. The chapter also covers the creation of a Grafana dashboard for real-time monitoring and visualization of system performance. Additionally, it discusses feedback mechanisms for pivot arms and conveyor belts, enabling the detection of malfunctions and tracking of lost boxes. Finally, the chapter concludes by explaining how to create a custom function block in TIA Portal to track the runtime of machine components, providing valuable insights into their operational duration.

Chapter I

Foundations of IoT and Electromechanical Systems

I.1 Introduction to IoT

I.1.1 Definition and Overview

The Internet of Things (IoT) is a network of interconnected devices that can collect, share, and act on data. This network encompasses everything from smart home devices to industrial machines, all connected via the internet. The IoT's primary aim is to extend internet connectivity beyond standard devices like computers and smartphones to everyday objects, enabling them to communicate and interact with each other and the external environment [1].

I.1.2 History and Evolution

IoT's concept dates back to the early 1980s, when a Coke machine at Carnegie Mellon University was modified to report its inventory and whether newly loaded drinks were cold. Since then, the evolution of IoT has been driven by advancements in wireless communication, sensor technologies, and computing power, leading to widespread adoption in various sectors such as healthcare, manufacturing, and smart homes [1].

I.1.3 IoT Architectures

IoT architecture is typically divided into several layers, each serving a specific function:

- **Perception Layer (Sensors and Actuators):** The physical layer responsible for collecting data. Sensors collect data from the environment, such as temperature, humidity, light, motion, or sound. Actuators, on the other hand, perform actions based on the received data or commands, such as turning on lights, adjusting thermostats, or opening valves.
- **Network Layer (Connectivity)**: This layer enables the seamless transmission of data between devices and the cloud or other central systems. Various communication protocols are employed, including Wi-Fi, Bluetooth, Zigbee, LoRaWAN, and cellular networks, each with its own advantages and limitations in terms of range, power consumption, and data throughput:
    - → **MQTT (Message Queuing Telemetry Transport):** A lightweight, publish-subscribe protocol ideal for resource-constrained devices and unreliable networks. It is widely used in IoT applications for sending telemetry data and receiving commands. [2]

→ **CoAP (Constrained Application Protocol):** A specialized web transfer protocol designed for constrained devices and networks with limited bandwidth and processing capabilities. It is often used in resource-constrained IoT applications for simple interactions. [3]

→ **HTTP (Hypertext Transfer Protocol):** The foundation of data communication on the World Wide Web, HTTP can also be used in IoT systems for RESTful APIs and web-based device management. [4]

→ **WebSocket:** A protocol that provides full-duplex communication channels over a single TCP connection. It is useful for real-time data exchange and interactive IoT applications. [5]

→ **AMQP (Advanced Message Queuing Protocol):** An open standard application layer protocol for message-oriented middleware. It enables reliable message delivery and is often used in enterprise IoT deployments. [6]

→ **DDS (Data Distribution Service):** A data-centric publish-subscribe middleware protocol that supports real-time, scalable, reliable, and high-performance data exchange in distributed systems. It is commonly used in industrial IoT and mission-critical applications. [7]

→ **Bluetooth:** A wireless technology standard for exchanging data over short distances. It is often used in personal area networks (PANs) for connecting IoT devices like wearables and smart home gadgets. [8]

→ **Zigbee:** A low-power, low-data rate wireless mesh networking protocol designed for IoT applications like home automation, building automation, and industrial control systems. [9]

→ **LoRaWAN (Long Range Wide Area Network):** A low-power wide area network (LPWAN) protocol designed for long-range, low-bandwidth communications. It is well-suited for IoT applications that require battery-powered devices to transmit data over long distances. [10]

- **Middleware Layer (Data Processing and Storage)**: The vast amounts of data generated by IoT devices require efficient processing and storage to extract meaningful insights. This processing can occur at different levels:

→ **Edge Computing**: Processing data locally on the device or a nearby gateway minimizes latency and bandwidth usage, making it suitable for time-sensitive applications.

→ **Fog Computing**: An intermediary layer between the edge and the cloud, providing additional processing capabilities and reducing the load on the cloud.

→ **Cloud Computing**: Large-scale data storage and analysis are often performed in the cloud, leveraging powerful servers and advanced algorithms

- **Application Layer (User Interface)**: This component facilitates interaction between humans and the IoT system. It can manifest as mobile applications, web dashboards, voice assistants, or augmented reality interfaces [2].

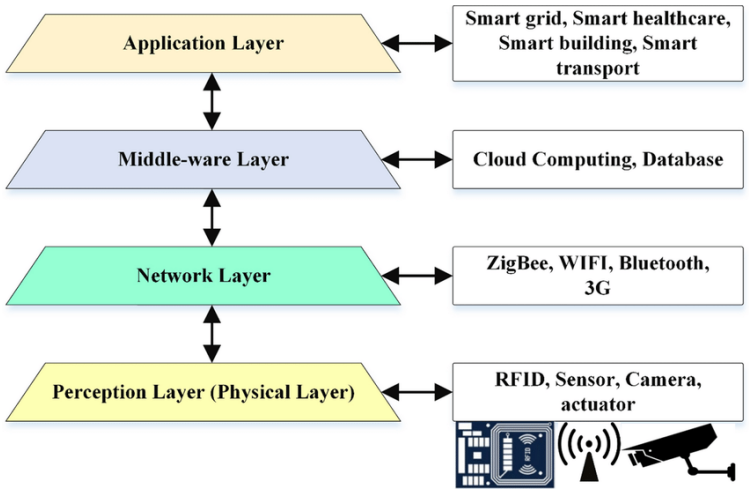


Figure I. 1 IoT Architecture Layers

I.2 Electromechanical Systems:

I.2.1 Definition and Components

Electromechanical systems are systems that combine electrical and mechanical processes to perform a specific function. They often involve the conversion of electrical energy into mechanical motion, or mechanical motion into electrical energy and often include:

- **Motors and Actuators:** These devices convert electrical signals into mechanical movement. Common types include electric motors, solenoids, and piezoelectric actuators.

Figure I. 2 Motors and Actuators

- **Sensors:** These devices convert mechanical motion or other physical quantities into electrical signals. Examples include position sensors, strain gauges, and temperature sensors.

Figure I. 3 Sensor

- **Control systems:** These systems process the electrical signals from sensors and control the operation of actuators to achieve desired performance. Control systems can range from simple on/off switches to using programmable logic controllers (PLCs)

Figure I. 4 Programmable Logic Controller

I.2.2 Applications:

Electromechanical systems are essential in numerous applications across different industries:

- **Manufacturing**: Automated production lines, CNC machines, and robotic arms.
- **Automotive**: Electric and hybrid vehicles, automated systems for safety and efficiency.
- **Robotics**: Industrial robots for assembly and manufacturing, as well as consumer robots for household tasks

Figure I. 5 Industrial Robot Arm

I.3 Data Collection and Analysis in IoT

I.3.1 Data Acquisition

Data acquisition involves collecting data from various sensors and devices. This process can include:

- **Real-Time Data Streaming**: Data is continuously collected by sensors and transmitted immediately to a central system (often cloud-based) for processing and analysis. This happens with very low latency, sometimes within milliseconds.

  Advantages:

  → **Immediate Insights:** Enables instant responses to events or changes in conditions.

  → **Time-Sensitive Applications:** Ideal for applications where rapid decision-making is crucial (e.g., anomaly detection, fraud prevention, real-time equipment monitoring).

→ **Continuous Monitoring:** Provides a constant stream of data, allowing for detailed trend analysis and pattern recognition over time.

Challenges:

→ **Bandwidth Requirements:** Can consume significant bandwidth, especially with large numbers of devices or high-frequency data collection.

→ **Cost:** Often more expensive due to increased data transfer and processing requirements.

→ **Complexity:** Requires robust infrastructure and sophisticated data processing capabilities to handle the continuous flow of data.

- **Periodic Data Collection**: Collecting data at regular intervals (e.g., hourly, daily, weekly) and sending it in batches.

Advantages:

→ **Reduced Bandwidth Usage:** Conserves bandwidth by transmitting data less frequently.

→ **Lower Cost:** Typically more cost-effective than real-time streaming due to reduced data transfer and processing needs.

→ **Simplicity:** Easier to implement and manage than real-time streaming, making it suitable for less complex applications.

Challenges:

→ **Delayed Insights:** Information is not available immediately, which may be problematic for time-sensitive applications.

→ **Limited Granularity:** The intervals between data collections may not capture all relevant events or trends.

I.3.2 Data Type:

Structured Data

- **Definition:** Data that fits neatly into a predefined format, typically tables with rows and columns. Each column represents a specific attribute (e.g., name, date, temperature), and each row is a record with values for those attributes.

- **Examples:**
  - → Sensor readings (timestamp, sensor ID, value)
  - → Financial transactions (transaction ID, amount, date)
  - → Customer information (name, address, email)

Unstructured Data

- **Definition:** Data that does not have a predefined format or organization. It's often text-heavy but can also include images, videos, audio files, social media posts, or sensor data streams.
- **Examples:**
  - → Email messages
  - → Social media posts
  - → Images and videos
  - → Sensor data streams (continuous, rapidly changing data)
  - → Log files

I.3.3 Data Storage Solutions

→ **Relational Databases (RDBMS):**

- **Purpose:** Ideal for structured data with well-defined relationships
- **Strengths:**
  - → Strong data consistency guarantees (ACID compliance)
  - → Mature technology with wide support and tooling
  - → Powerful querying capabilities with SQL
- **Weaknesses:**
  - → Can be less scalable for massive IoT data volumes
  - → Less flexible for handling unstructured or rapidly changing data
- **Examples:** MySQL, PostgreSQL, Microsoft SQL Server, Oracle Database

→ **NoSQL Databases:**

- **Purpose:** Well-suited for unstructured or semi-structured data that doesn't fit neatly into a relational model [11].
- **Strengths:**
  - → Highly scalable for large data volumes and high throughput
  - → Flexible schemas that can evolve with changing data requirements
  - → Distributed architecture for fault tolerance
- **Weaknesses:**
  - → May not provide the same level of strong consistency as RDBMS
  - → Can be more complex to query than SQL-based databases
- **Examples:** MongoDB, Cassandra, Couchbase, DynamoDB

→ **Cloud Storage:**

- **Purpose:** Offers scalable and flexible storage options for various data types, accessible from anywhere with an internet connection
- **Strengths:**
  - → Virtually unlimited scalability
  - → Managed services for ease of use and maintenance
  - → Pay-as-you-go pricing model
  - → Geo-redundancy and disaster recovery options
- **Weaknesses:**
  - → Can be more expensive than on-premises storage for large data volumes
  - → Data sovereignty and privacy concerns
  - → Potential latency for data access
- **Examples:** Amazon S3, Microsoft Azure Blob Storage, Google Cloud Storage

→ **Time-Series Databases (TSDB):**

- **Purpose:** Specifically designed for storing and analyzing time-stamped data, common in IoT applications
- **Strengths:**
  - → Optimized for high-speed data ingestion and retrieval
  - → Built-in support for time-based queries and aggregation
  - → Efficient storage for large volumes of time-series data
- **Weaknesses:**
  - → May not be suitable for general-purpose data storage
  - → Can be more complex to manage than traditional databases
- **Examples:** InfluxDB, TimescaleDB, Prometheus, Graphite

I.3.4 Data Analysis Techniques

Analyzing IoT data involves various techniques to extract meaningful insights:

- **Time-Series Analysis**: Involves analyzing data points collected or recorded at specific time intervals, useful for monitoring trends over time.
- **Predictive Maintenance**: Uses historical data and machine learning algorithms to predict equipment failures before they occur.
- **Machine Learning**: Employs algorithms to detect patterns, classify data, and make predictions based on the data collected [12].

I.4 Cloud Computing

I.4.1 Definition of Cloud Computing:

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [15].

I.4.2 Service Models: IaaS, PaaS, and SaaS

Cloud computing offers a variety of service models, each catering to different needs and levels of abstraction:

- **Infrastructure as a Service (IaaS):** IaaS providers offer fundamental computing resources like virtual machines, storage, and networking. Users have control over the operating system and software stack, providing flexibility but requiring more management overhead [15].
- **Platform as a Service (PaaS):** PaaS provides a platform for developing, testing, deploying, and managing applications. Users can focus on application development without worrying about infrastructure management, accelerating time-to-market [16].
- **Software as a Service (SaaS):** SaaS delivers software applications over the internet on a subscription basis. Users access the applications through a web browser or client, eliminating the need for local installation and maintenance [15].

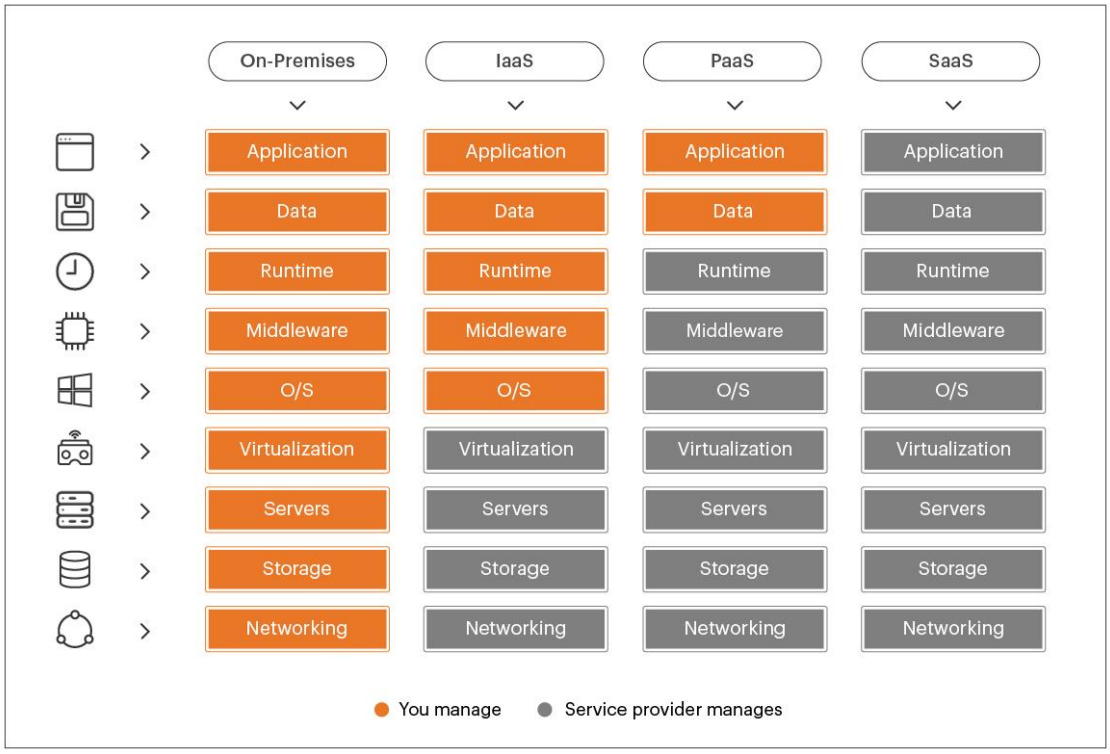


Figure I. 6 Cloud Computing Service Models

Figure I. 7 Cloud Computing Deployment Models

I.4.3 Deployment Models:

Cloud environments can be deployed in various models, each with distinct characteristics:

- **Public Cloud:** Owned and operated by a third-party provider, public clouds offer services to multiple customers over the public internet. They provide scalability, cost-efficiency, and accessibility but may raise concerns about security and control [15].
- **Private Cloud:** Operated solely for a single organization, private clouds offer enhanced control, security, and customization. However, they may require significant upfront investments and ongoing maintenance [15].
- **Hybrid Cloud:** Combining public and private clouds, hybrid clouds allow organizations to leverage the benefits of both models. This approach offers flexibility, enabling workloads to be shifted between environments based on specific requirements [15].
- **Community Cloud:** Shared by several organizations with common concerns (e.g., mission, security requirements, policy, and compliance considerations), community clouds offer a balance of cost-efficiency, security, and shared resources [15].

I.5 Security and Privacy in IoT

I.5.1 Security Challenges

IoT systems face numerous security challenges:

- **Data Breaches**: Unauthorized access to sensitive data can result in significant financial and reputational damage.

- **Cyber-Attacks**: IoT devices can be targets for various cyber-attacks, including Distributed Denial of Service (DDoS) attacks and malware.
- **Privacy Concerns**: Collecting and storing personal data raises privacy issues, requiring stringent measures to protect user data [17].

I.5.2 Security Measures

To address these challenges, several security measures are essential:

- **Encryption**: Protects data both in transit and at rest, ensuring that unauthorized parties cannot read the data.
- **Authentication**: Ensures that only authorized users and devices can access the system.
- **Regular Updates**: Keeping software and firmware up-to-date to protect against known vulnerabilities.
- **Intrusion Detection Systems (IDS)**: Monitor network traffic for suspicious activity and alert administrators to potential threats [18].

I.6 Review of Existing IoT Solutions

I.6.1 Existing Systems and Applications

Several IoT systems have been developed for monitoring and managing electromechanical systems:

- **Siemens MindSphere**: An industrial IoT as a service solution that connects products, plants, systems, and machines, enabling businesses to harness the wealth of data generated by the Internet of Things with advanced analytics [19].
- **GE Predix**: A platform used to monitor and optimize industrial machinery, resulting in improved efficiency and reduced downtime. It integrates with various industrial control systems and provides predictive analytics to anticipate maintenance needs [20].

I.6.2 Case Studies

Real-world case studies demonstrate the successful implementation of IoT in various industries:

- **Manufacturing**: Harley-Davidson uses IoT to monitor its manufacturing processes in real-time, leading to a 25% reduction in production schedule delays and a 40% improvement in the build-to-order cycle [21].
- **Automotive**: Tesla's connected cars constantly send data to the cloud for analysis, allowing the company to improve vehicle performance, predict maintenance needs, and enhance user experience through over-the-air updates [22].

I.7 Technological Developments

I.7.1 Recent Advances

Significant technological advancements have driven the evolution of IoT, particularly in areas such as sensor technology, connectivity, and data processing:

- **Sensor Technology**: Modern sensors are more accurate, consume less power, and can transmit data over longer distances.
- **Edge Computing**: Reduces latency by processing data closer to where it is generated, rather than sending it to a centralized cloud. This is particularly important for applications requiring real-time processing, such as autonomous vehicles and industrial automation.
- **Artificial Intelligence and Machine Learning**: These technologies enable more sophisticated data analysis, allowing for predictive maintenance, anomaly detection, and optimized control systems [23].

I.7.2 Innovative Techniques

Innovative techniques are continually being developed to enhance IoT applications:

- **Digital Twins**: Virtual replicas of physical devices or systems used for simulation and analysis. They provide a dynamic, real-time model of the physical counterpart, which can be used to predict performance, optimize operations, and improve maintenance [24].
- **Blockchain Technology**: Enhances security and transparency in IoT systems by providing a decentralized and immutable ledger for recording transactions. This is

particularly useful in supply chain management and other applications requiring robust security [25].

1.8 Integration with Electromechanical Systems

I.8.1 Integration Challenges

Integrating IoT with electromechanical systems presents several challenges:

- **Interoperability**: Ensuring different devices and systems can communicate effectively, often requiring standardization and compatibility protocols.
- **Scalability**: As the number of connected devices increases, managing the data and ensuring the system can handle the load becomes more complex.
- **Data Management**: Efficiently storing, processing, and analyzing the vast amounts of data generated by IoT devices, often in real-time [26].

I.8.2 Solutions and Approaches

Several approaches can address these challenges:

- **Standardization**: Adopting common standards and protocols, such as MQTT, CoAP, and OPC UA, ensures devices from different manufacturers can work together seamlessly.
- **Edge Computing**: Processing data closer to the source reduces latency and bandwidth usage, making the system more efficient and responsive.
- **Advanced Analytics**: Using machine learning and AI to analyze data more effectively, enabling predictive maintenance, anomaly detection, and optimization of system performance [27].

1.9 Performance Metrics

I.9.1 Evaluation Criteria

The performance of IoT systems can be evaluated using various criteria, including:

- **Latency**: The time taken for data to travel from the source to the destination. Lower latency is critical for real-time applications such as autonomous driving and industrial automation.
- **Reliability**: The system's ability to function correctly and provide accurate data consistently. This includes fault tolerance and robustness against failures.
- **Scalability**: The capacity to handle increasing numbers of devices and data volumes without performance degradation. This is crucial for large-scale deployments such as smart cities and industrial IoT [28].

I.9.2 Comparative Analysis

A comparative analysis of different IoT solutions can be performed based on these metrics. For example, a study comparing MQTT and CoAP for IoT communication found that MQTT is better suited for scenarios requiring reliable message delivery, while CoAP is more efficient in terms of resource usage. This kind of analysis helps in choosing the right protocol for specific applications [29].

Conclusion

This chapter provides a comprehensive introduction to the Internet of Things (IoT), covering its definition, history, evolution, and various architectures. It delves into the components of electromechanical systems, their applications, and how they integrate with IoT for data collection and analysis. The chapter also explores cloud computing, security and privacy concerns in IoT, and reviews existing IoT solutions with real-world case studies. Additionally, it discusses recent technological advancements and innovative techniques in IoT, emphasizing the integration challenges and solutions for electromechanical systems. Finally, the chapter outlines performance metrics for evaluating IoT systems and provides a comparative analysis of different solutions.

Chapter II

Setting up IoT tools for the Data Collection

Introduction

This chapter focuses on the practical implementation of IoT tools for data collection in the context of monitoring and controlling electromechanical systems. It provides a detailed overview of the system architecture, encompassing the perception layer (sensors and actuators), network layer (connectivity), middleware layer (data processing and storage), and application layer (user interface). The chapter outlines the specific tools and technologies used in each layer, including Factory I/O for simulation, PLC SIM ADVANCE 6.0 for data acquisition, Node-RED for data processing, and AWS EC2 with InfluxDB for data storage. It also explains the integration of Grafana for real-time data visualization and monitoring.

II.1 IoT System Architecture Overview

The architecture of the proposed IoT-based monitoring and control system, as shown in Figure II.1, is designed to facilitate seamless data acquisition, processing, storage, and visualization.

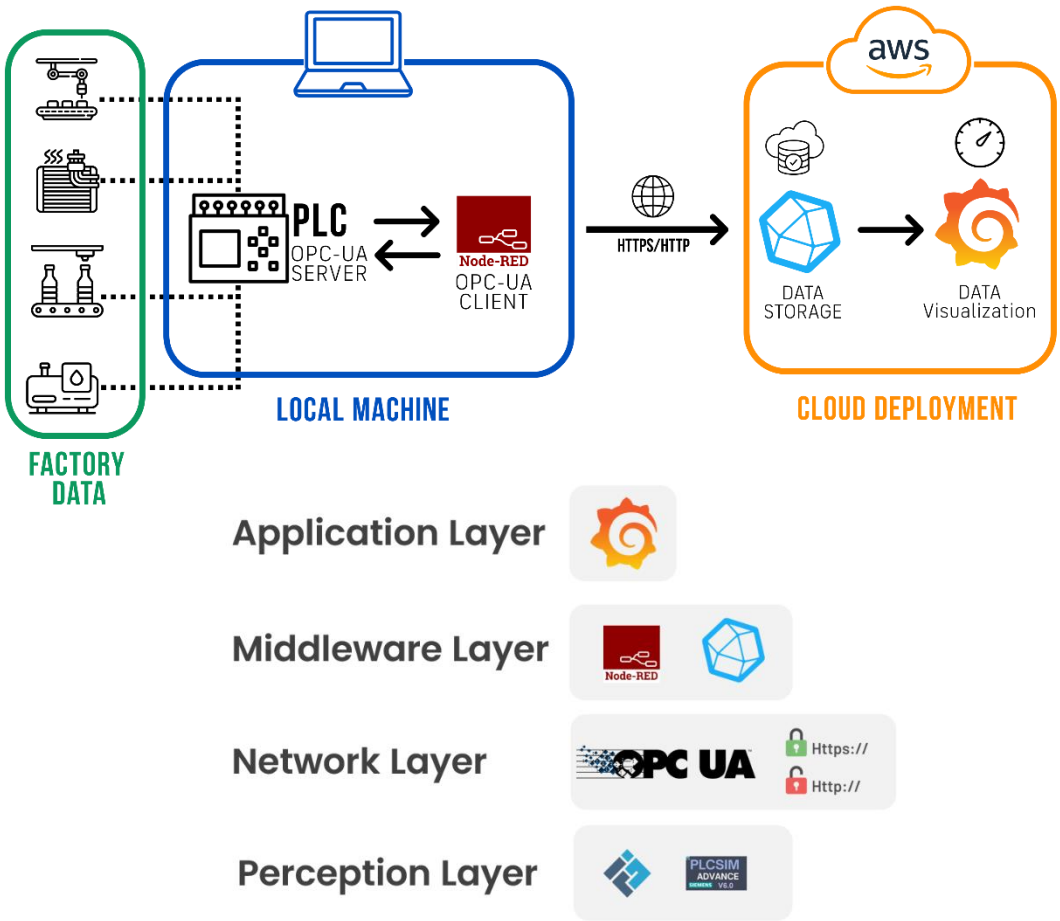


Figure II. 1 The architecture of the proposed IoT system

II.1.1 Perception Layer (Sensors and Actuators)

- **Factory Data:** Simulated factory data will be generated using Factory I/O, replicating real-world scenarios and conditions.
- **PLC (Programmable Logic Controller):** PLC SIM ADVANCE 6.0 will emulate the functionality of a real PLC, collecting data from the simulated sensors and machines in Factory I/O, performing initial processing, and executing control logic.

II.1.2 Network Layer (Connectivity)

- **OPC-UA Server:** This server, residing on the local machine, will act as the communication bridge between PLC SIM ADVANCE 6.0 and the higher layers. It will translate the PLC's proprietary data into the standardized OPC-UA format for seamless integration with Node-RED.
- **OPC-UA Client (Node-RED):** Node-RED's OPC-UA client component will establish a secure connection with the OPC-UA server to retrieve the simulated factory data in real time.
- **HTTPS/HTTP:** These protocols will be used for secure and efficient data transmission between the local machine running Node-RED and the cloud-based AWS environment.

II.1.3 Middleware Layer (Data Processing and Storage)

- **Node-RED:** Node-RED will act as the middleware, receiving the raw data from the OPC-UA client, performing further processing (filtering, aggregation, transformation), and preparing it for storage and visualization. It will also handle data validation, error management, and protocol translation as needed.
- **InfluxDB (AWS EC2):** The time-series database, InfluxDB, installed on an AWS EC2 instance, will be the central repository for storing the processed data from Node-RED. Its ability to handle time-stamped data makes it ideal for capturing and analyzing the temporal aspects of factory operations.

II.1.4 Application Layer (User Interface)

- **Grafana (AWS EC2):** Grafana, also hosted on the AWS EC2 instance, will be the visualization tool. It will connect to the InfluxDB database and provide interactive dashboards for real-time monitoring, historical analysis, and custom reporting of the factory data. Grafana's rich visualization capabilities will enable users to gain valuable

insights into the performance, trends, and anomalies of the simulated electromechanical systems.

II.2 AWS cloud platform

II.2.1 Definition and Overview

AWS stands for Amazon Web Services. It is the world's leading cloud computing platform, offering a vast array of services that businesses and individuals use to build, deploy, and manage applications and infrastructure over the internet. AWS provides on-demand access to computing power, storage, databases, networking, analytics, machine learning, and much more. These services are highly scalable and reliable, allowing users to adjust their resource consumption as needed.

- **Pay-As-You-Go Model:** Instead of purchasing and maintaining expensive hardware, AWS users pay only for the resources they actually use, making it a cost-effective solution for many.
- **Global Reach:** AWS has a massive global infrastructure with data centers in regions around the world. This ensures high availability and low latency for applications served to users worldwide.

II.2.2 History of AWS

- **Early Days (2002-2006):** Amazon initially developed its internal infrastructure to handle its growing e-commerce operations. They realized they could offer these capabilities as services to others.
- **Public Launch (2006):** AWS officially launched with its Simple Storage Service (S3) and Elastic Compute Cloud (EC2), laying the foundation for the vast platform it is today.
- **Rapid Expansion:** Over the years, AWS has continually added new services and features, expanding into areas like databases, artificial intelligence, Internet of Things (IoT), and more.
- **Industry Leader:** Today, AWS is the dominant player in the cloud computing market, serving millions of customers across various industries.

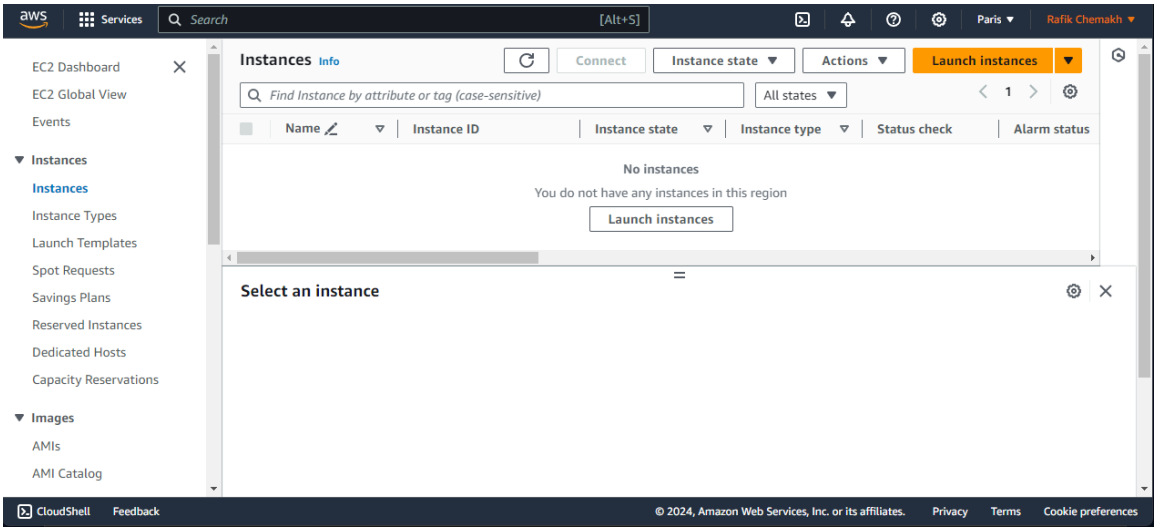


Figure II. 2 Amazon AWS EC2 Dashboard

II.3 InfluxDB

II.3.1 Definition and Overview

InfluxDB is an open-source time series database (TSDB) developed by InfluxData It is designed to collect, store, process and visualize time series data.

II.3.2 Time Series Data

Time series data is a sequence of data points indexed in time order. Data points typically consist of successive measurements made from the same source and are used to track changes over time. Examples of time series data include**:**

- Industrial sensor data
- Server performance metrics
- Heartbeats per minute
- Electrical activity in the brain
- Rainfall measurements
- Stock prices

II.3.3 InfluxDB Data Organization

The InfluxDB data model organizes time series data into buckets and measurements. A bucket can contain multiple measurements. Measurements contain multiple tags and fields.

- **Bucket**: Named location where time series data is stored. A bucket can contain multiple *measurements*.
    - **Measurement**: Logical grouping for time series data. All *points* in a given measurement should have the same *tags*. A measurement contains multiple *tags* and *fields*.
        - **Tags**: Key-value pairs with values that differ, but do not change often. Tags are meant for storing metadata for each point–for example, something to identify the source of the data like host, location, station, etc.
        - **Fields**: Key-value pairs with values that change over time–for example: temperature, pressure, stock price, etc.
        - **Timestamp**: Timestamp associated with the data. When stored on disk and queried, all data is ordered by time.

II.3.4 Important Definitions

The following are important definitions to understand when using InfluxDB:

- **Point**: Single data record identified by its measurement, tag keys, tag values, field key, and timestamp.
- **Series**: A group of points with the same measurement, tag keys, and tag values.

_time	_measurement	city	country	_field	_value
2022-01-01T12:00:00Z	weather	London	UK	temperature	12.0
2022-02-01T12:00:00Z	weather	London	UK	temperature	12.1
2022-03-01T12:00:00Z	weather	London	UK	temperature	11.5
2022-04-01T12:00:00Z	weather	London	UK	temperature	5.9

Point

Series

Figure II. 3 Example InfluxDB query results

- **Query** : refers to a request for data from the database. It is a command written in a specific query language (either InfluxQL or Flux) that specifies what data you want to retrieve, how it should be filtered or aggregated, and how it should be returned.

- **InfluxQL:** This is the original query language used in InfluxDB 1.x. It's a SQL-like language tailored for time series data, with syntax and functions for filtering, aggregating, and transforming data points over time.

```
SELECT mean("temperature") FROM "sensor_data" WHERE time > now() - 1h GROUP BY time(5m)
```

This query shown in Figure II.4 calculates the mean temperature from the "sensor_data" measurement over the last hour, grouped by 5-minute intervals

- **Flux:** This is the newer query language introduced in InfluxDB 2.x. It's a more powerful and flexible functional scripting language designed for complex time series operations and data transformations. It also supports querying and processing data from multiple sources.

```
from(bucket: "my_bucket")
  |> range(start: -1h)
  |> filter(fn: (r) => r._measurement == "sensor_data")
  |> aggregateWindow(every: 5m, fn: mean, createEmpty: false)
  |> yield(name: "mean_temperature")
```

This Flux query achieves the same result as the InfluxQL query but uses a more functional and expressive syntax.

II.4 Setting up InfluxDB

To set up InfluxDB on AWS, we first need to choose and create an EC2 instance based on our workload requirements. This involves selecting an appropriate Amazon Machine Image (AMI), such as Amazon Linux or Ubuntu, and an instance type that can handle the anticipated data volume and processing needs. We then configure the instance's security groups to permit inbound traffic on port 8086, the default port for InfluxDB. Once the instance is launched, we note its public IP address or DNS name, which will be used to connect to and access the InfluxDB instance later

II.4.1 Key Pair Creation:

- **Purpose:** We need a key pair, which is a set of security credentials, to prove our identity when connecting to our EC2 instance. It consists of a public key, placed on the instance, and a private key, which we keep secure and use to authenticate our SSH connection. This mechanism provides a more secure alternative to traditional password-based authentication.

- **Process:** In the EC2 dashboard, we navigate to "Key Pairs" under "Network & Security," click "Create key pair," provide a name, and select the appropriate format (e.g., .ppk). We then download the private key file (.ppk) and store it securely, as it will be required for establishing an SSH connection.

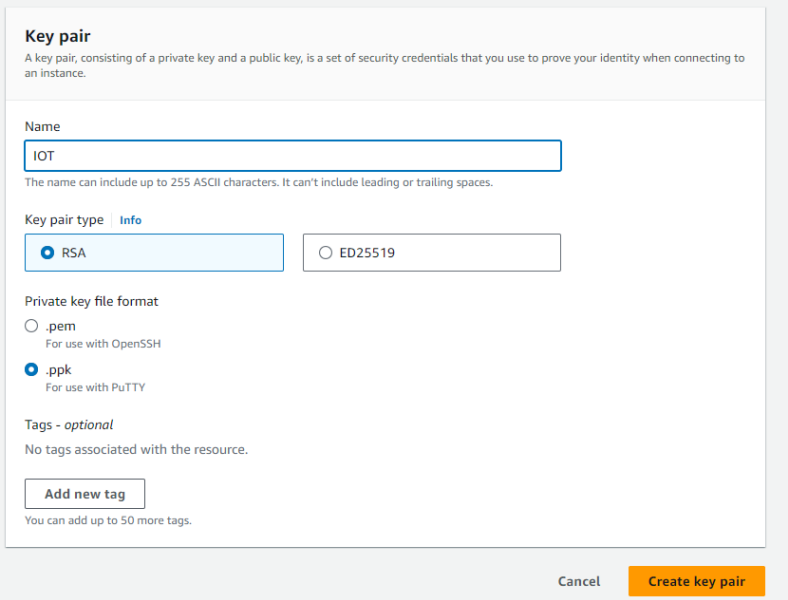


Figure II. 4 Creating a key pair

II.4.2 Instance Launch:

a) We navigate to EC2 in the AWS Management Console and choose “Europe (Paris) eu-west-3” as region for instance placement, given our location in Algeria, we have strategically chose this region for our EC2 instance deployment. This decision prioritizes minimizing latency and ensuring a responsive experience then we click on "Launch Instance."

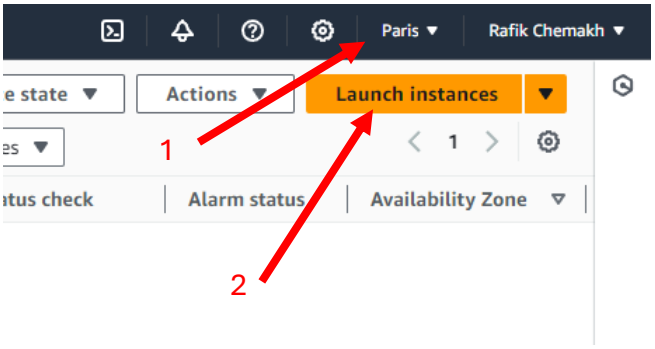


Figure II. 5 Choosing a region for the instance

b) We name our instance "InfluxDB".

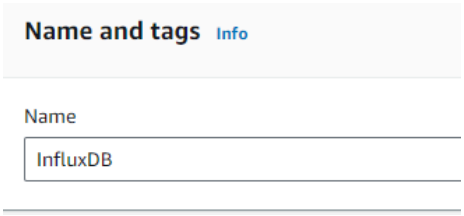


Figure II. 6 Naming the instance

c) We choose the Ubuntu 24.04 AMI as the operating system. AMI stands for Amazon Machine Image. It is a template that contains the software configuration (operating system, application server, and applications) required to launch an EC2 instance,

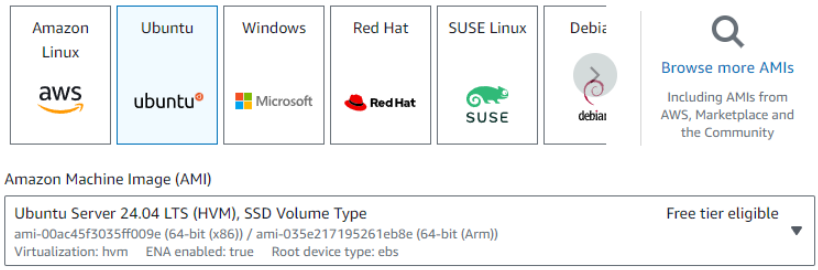


Figure II. 7 Choosing the operating system

d) We select the previously created key pair for authentication.

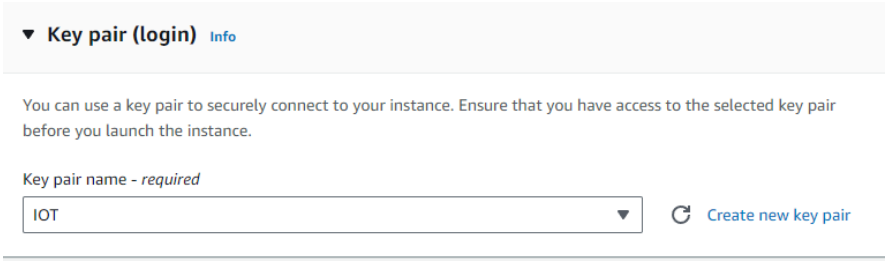


Figure II. 8 Selecting the key pair

e) We select the t3.medium instance type for resource allocation. we have selected the t3.medium instance type (2 vCPUs, 4GB RAM) for our InfluxDB deployment on AWS EC2. This configuration offers a suitable balance of performance and cost-effectiveness, ensuring sufficient resources for our projected processing needs while remaining financially prudent

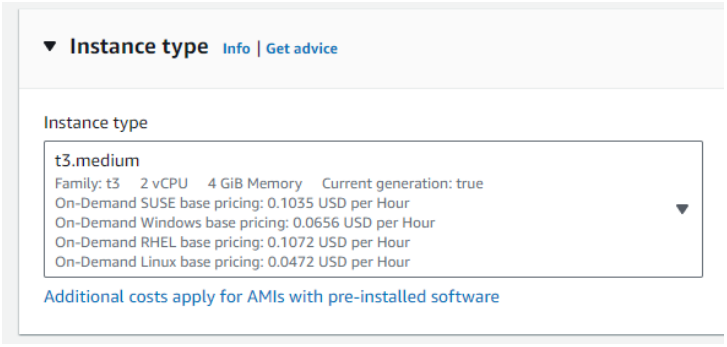


Figure II. 9 Selecting the instance type

f) In the " Configure storage" step, we configure a 30GB gp3 storage volume. gp3 refers to the latest generation of General Purpose SSD (Solid State Drive) volumes offered by Amazon Elastic Block Store (EBS). These volumes are designed to provide a balance of price and performance for a wide range of workloads, making them a suitable choice for our InfluxDB deployment.

Key advantages of gp3 volumes include:

- **Lower Cost**: gp3 volumes offer a 20% lower price per gigabyte compared to the previous generation gp2 volumes, providing cost savings for our storage needs.
- **Scalable Performance**: We can independently scale the IOPS (input/output operations per second) and throughput of the volume without needing to increase the storage capacity, ensuring optimal performance for our database.
- **Predictable Performance**: gp3 volumes offer a baseline performance of 3,000 IOPS and 125 MiB/s throughput, irrespective of the volume size, providing predictable performance for our application.

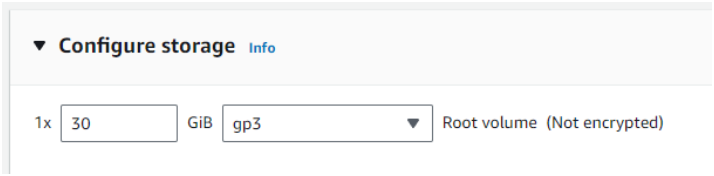


Figure II. 10 Configuring storage

g) We configure the security group of our AWS EC2 instance to allow inbound traffic on port 8086. To do this, we'll navigate to the "Network settings" section of the EC2 instance and click "Edit." Next, we'll add a new security group rule by clicking "Add security group rule." For the "Type," we'll select "Custom TCP," and for the "Port Range," we'll enter "8086." In the "Source" field, we'll choose "Anywhere," making our InfluxDB instance accessible from any IP address

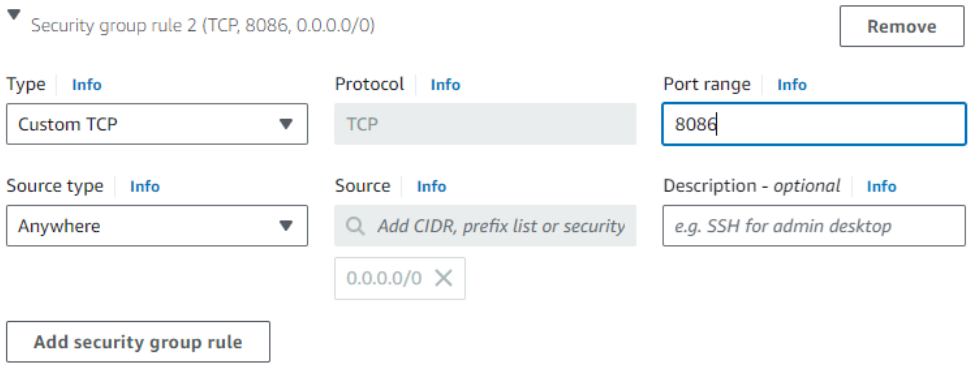


Figure II. 11 Configuring the security group (adding port 8086)

h) Once we finish configuring the instance we click on “launch instance”

II.5 Secure Shell (SSH) Connection:

To connect to the instance we created, we can directly simply use:

- **EC2 Instance Connect:** This feature simplifies SSH access by providing a browser-based SSH client within the AWS Management Console. It eliminates the need to manage SSH keys directly. It's integrated directly into the AWS Management Console. Here's how to use it:

→ **Locate instance:** In the AWS Management Console, we navigate to the EC2 dashboard and find the instance.

→ **Click "Connect":** We select the instance and click the "Connect" button at the top.

→ **Choose EC2 Instance Connect:** In the Connect to instance dialog, we select the "EC2 Instance Connect" tab.

→ **Connect:** We click the "Connect" button to initiate the connection. A browser-based SSH terminal will open directly within the console.

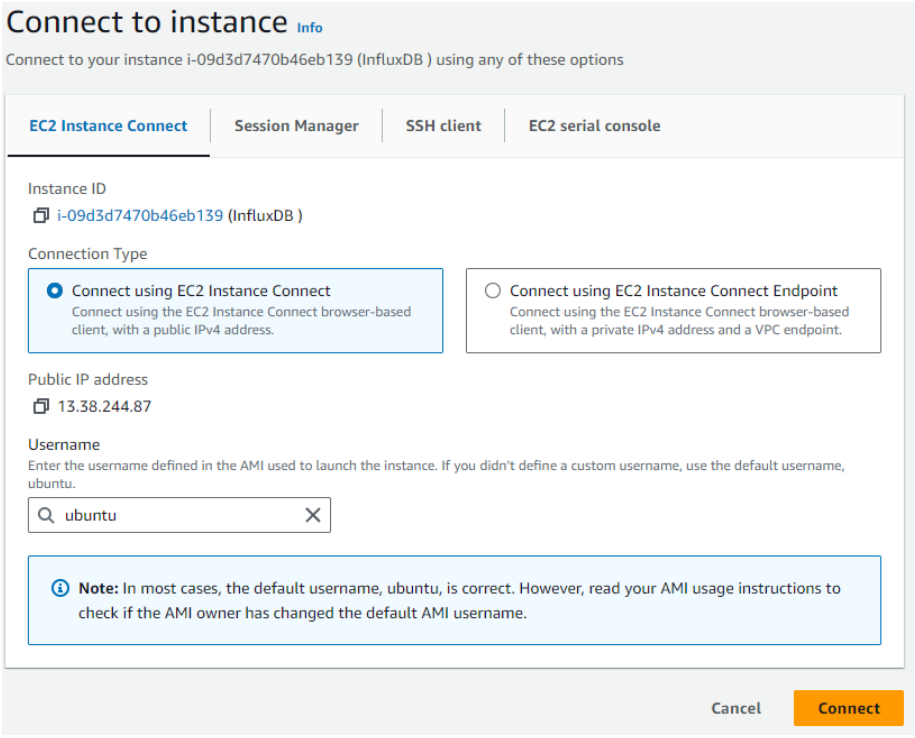


Figure II. 12 EC2 Instance Connect

- **SSH client:** Or by using PuTTY which is a popular SSH client for Windows that simplifies connecting to the AWS instance. Here's how we can use it:

1) **Enter instance details:**

   - In the "Host Name (or IP address)" field, we enter AWS instance's Public DNS name or Public IPv4 address.
   - We ensure the "Port" is set to 22 (the default SSH port).
   - We select "SSH" as the connection type.

2) **Load private key:**

- In the left-hand pane, we navigate to "Connection" -> "SSH" -> "Auth".
- We click the "Browse" button under "Private key file for authentication" and select (.ppk) private key file.

3) **Save session (optional):**

- We can save the session configuration for future use by entering a name in the "Saved Sessions" field and clicking "Save".

4) **Connect:** we click the "Open" button to initiate the SSH connection.

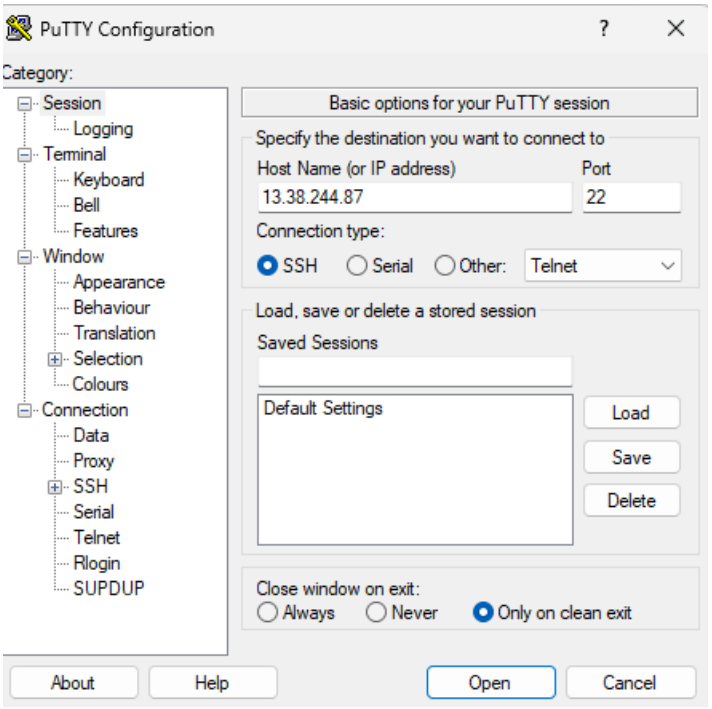


Figure II. 13 PuTTY Configuration

II.6 Installing InfluxDB:

Once connected to the EC2 instance, we'll initiate the InfluxDB installation process. The first step is to update the package lists on the instance to ensure we have access to the latest versions. This can be achieved by entering the following command in the terminal:

```
sudo apt update -y
```

With the package lists updated, we're ready to install InfluxDB OSS v2.7.6. We'll accomplish this using the following command

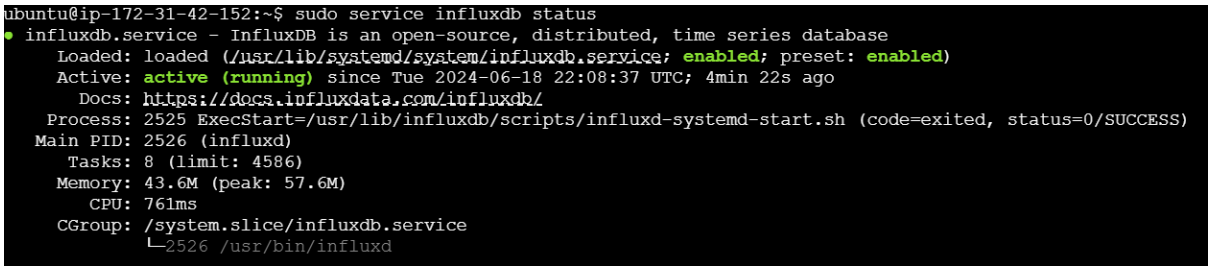
```
# influxdata-archive_compat.key GPG fingerprint:

#     9D53 9D90 D332 8DC7 D6C8 D3B9 D8FF 8E1F 7DF8 B07E

wget -q https://repos.influxdata.com/influxdata-archive_compat.key

echo '393e8779c89ac8d958f81f942f9ad7fb82a25e133faddaf92e15b16e6ac9ce4c influxdata-archive_compat.key' | sha256sum -c && cat influxdata-archive_compat.key | gpg --dearmor | sudo tee /etc/apt/trusted.gpg.d/influxdata-archive_compat.gpg > /dev/null

echo 'deb [signed-by=/etc/apt/trusted.gpg.d/influxdata-archive_compat.gpg] https://repos.influxdata.com/debian stable main' | sudo tee /etc/apt/sources.list.d/influxdata.list

sudo apt-get update && sudo apt-get install influxdb2
```

then we start the influxDB service : `sudo service influxdb start`

To verify that the service is running correctly, we enter the following command :

```
sudo service influxdb status
```

If successful, the output is the following:



II.6.1 Accessing InfluxDB:

We can access the InfluxDB web interface by typing our EC2 instance's public IP address followed by port 8086 into our web browser (http://13.38.244.87:8086). As this is the first-time access, we'll be greeted with a welcome screen. To proceed, we follow these steps:

1. Click **"Get Started."**
2. Set up initial user:
    - Enter a **Username** for the initial user.
    - Enter a **Password** and **Confirm Password** for the user.
    - Enter an initial **Organization Name.**
    - Enter an initial **Bucket Name.**
3. Click **"Continue."**

4. Copy the provided operator API token and store it securely.

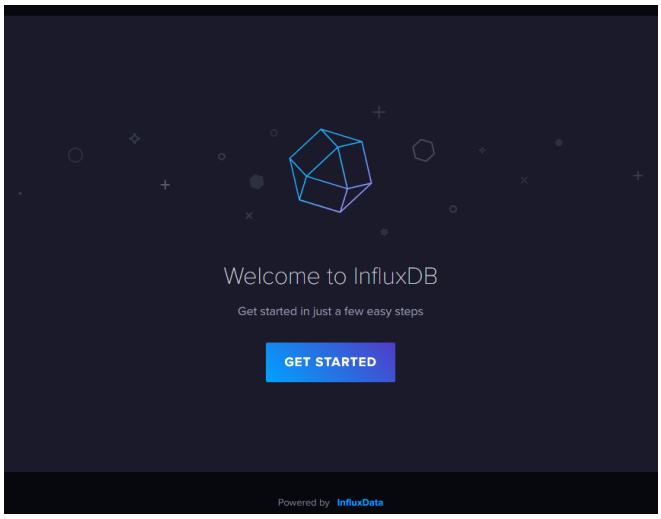


Figure II. 14 InfluxDB welcome screen

II.6.2 Configuring InfluxDB (Optional):

- InfluxDB configuration file is typically located at `/etc/influxdb/influxdb.conf`. we can modify settings like:
  - Storage directories
  - Authentication (if needed)
  - HTTP and HTTPS endpoints
  - Retention policies

II.7 Grafana

II.7.1 Definition and Overview

Grafana is an open-source platform for monitoring and observability. It allows you to visualize data from various sources, create interactive dashboards, and set up alerts. It is widely used for monitoring infrastructure, applications, and business metrics.

Key features of Grafana include:

- **Data visualization:** Grafana supports a wide range of visualization options, including graphs, charts, tables, and heatmaps.
- **Alerting:** You can set up alerts in Grafana to notify you when certain conditions are met.

- **Dashboards:** Grafana dashboards allow you to organize and visualize data in a way that is meaningful to you.
- **Plugins:** Grafana has a large ecosystem of plugins that extend its functionality

II.7.2 Setting up Grafana

To set up Grafana on AWS, we will follow a similar process as the InfluxDB installation. We'll create a new "t3.medium" EC2 instance, ensuring port 3000 is open for Grafana access. After establishing an SSH connection to the instance, we can proceed to paste the following commands into the terminal to initiate the Grafana installation process :

```
sudo apt-get install -y adduser libfontconfig1 musl
wget https://dl.grafana.com/oss/release/grafana_11.0.0_amd64.deb
sudo dpkg -i grafana_11.0.0_amd64.deb
```

Once the Grafana installation is complete, we can start the server and verify its status.To start the Grafana service, we'll execute the following commands in the terminal:

```
sudo systemctl daemon-reload

sudo systemctl start grafana-server
```

These commands first reload the system daemon configuration to recognize the Grafana service, then initiate the service itself.

To confirm that Grafana is running as expected, we'll use the following command:

```
sudo systemctl status grafana-server
```

This command will display the status of the Grafana service, and we should see an "active (running)" status message in the output.

To access the Grafana web interface, we can enter the public IP address of our EC2 instance followed by port 3000 into our web browser. Unlike InfluxDB, Grafana assigns a default username ("admin") and password ("admin"). Once we enter these credentials, we'll be prompted to change the password for security purposes

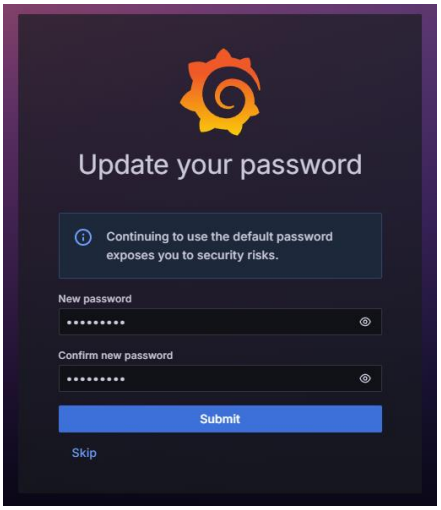


Figure II. 15 Grafana password creation

With the successful completion of these steps, we have now fully set up InfluxDB and Grafana on AWS. They are ready to receive, store, and visualize data from our Factory I/O simulation.

II.8 Nod-Red :

Node-RED is a open-source flow-based programming tool developed by IBM. It empowers developers to create complex event-driven applications with minimal coding, making it particularly well-suited for Internet of Things (IoT) projects. By connecting pre-built nodes in a visual editor, users can construct logic flows that interact with hardware devices, APIs, and online services, streamlining the development of automation and integration solutions.

II.8.1 Installation on Windows Systems:

Prerequisites:

→ **Node.js and npm:**

- Node-RED is built on Node.js, a JavaScript runtime. We can download the latest LTS version from the official website: https://nodejs.org/
- npm, a package manager bundled with Node.js, is required for installing Node-RED and its dependencies.

Once installed, we open a Command Prompt or PowerShell window with administrative privileges. and run the following command to ensure Node.js and npm are installed correctly

Using Powershell: `node --version; npm -version`

Using cmd: `node --version && npm --version`

We should receive output displaying the currently installed versions of Node.js and npm, then to Install Node-RED , we Execute the following command:

```
npm install -g --unsafe-perm node-red
```

II.9 Factory I/O

II.9.1 Definition and Overview

Factory I/O emerges as a leading platform for the exploration and advancement of industrial automation technologies. This sophisticated software empowers researchers, engineers, and educators to design, simulate, and optimize complex manufacturing processes within a realistic and risk-free virtual environment. Leveraging cutting-edge 3D visualization and seamless integration with programmable logic controllers (PLCs), Factory I/O offers a comprehensive toolkit for bridging the gap between theoretical concepts and practical implementation.

At its core, Factory I/O facilitates the creation of virtual factories that mirror the intricacies of real-world industrial settings. Users can readily assemble and configure a diverse range of components, including conveyors, sensors, actuators, robots, and more, drawn from an extensive library. This modular approach allows for the rapid prototyping and testing of various configurations, enabling users to iterate and refine their designs before committing to costly physical implementations.

II.9.2 Factory I/O simulators

Factory I/O seamlessly integrates with a variety of PLC simulators, providing a versatile platform to test automation designs and control logic. The choice of simulator often depends on specific research goals, familiarity with different PLC brands, and the desired level of complexity.

Here are some popular PLC simulators compatible with Factory I/O:

Built-in Simulators:

- **Control I/O:** This simulator is included with Factory I/O and is a great starting point for beginners. It features a user-friendly interface and is well-suited for learning basic ladder logic programming concepts.

External Simulators:

- **Siemens (PLCSIM, TIA Portal):** Factory I/O offers drivers for various Siemens PLCs, including S7-1200, S7-1500, and S7-300/400 series. Users can connect to PLCSIM (the official Siemens software simulator) or a real Siemens PLC using TIA Portal for more advanced simulations.
- **Allen-Bradley (RSLogix Emulate 500/5000):** If familiar with Allen-Bradley PLCs, users can leverage RSLogix Emulate to test ladder logic programs in Factory I/O.
- **CODESYS:** This open-source PLC development environment supports a wide range of hardware platforms. Factory I/O provides a CODESYS driver, allowing for simulation on various CODESYS-compatible devices.
- **Modbus TCP:** Factory I/O also supports Modbus TCP, which means it can potentially integrate with other PLC simulators or even physical PLCs that support Modbus communication.

II.10 TIA Portal

II.10.1 Definition and Overview

TIA Portal (Totally Integrated Automation Portal) is an engineering framework developed by Siemens for configuring, programming, and maintaining automation systems. TIA Portal integrates a range of Siemens automation solutions, including SIMATIC STEP 7 for PLCs, SIMATIC WinCC for Human-Machine Interface (HMI) and SCADA systems, and SINAMICS Startdrive for drive systems. It provides a unified platform that simplifies the engineering process by offering a common user interface and integrated tools for various automation tasks, leading to reduced engineering time and increased productivity [30].

II.10.2 Key features of TIA Portal :

- **Integrated Engineering Environment:** Combines multiple engineering tools into a single platform, ensuring seamless integration and communication between different components of the automation system.

- **Intuitive User Interface:** Designed to be user-friendly, with graphical editors and drag-and-drop functionality that make it easier to design and configure automation systems.
- **Comprehensive Diagnostics:** Offers built-in diagnostic tools to monitor system performance and quickly identify and resolve issues.
- **Scalability and Flexibility:** Suitable for projects of all sizes, from small machines to large-scale production facilities, and can be scaled according to the complexity of the project.
- **Security Features:** Provides robust security features to protect against unauthorized access and ensure the integrity of the automation system.

II.11 OPC UA

II.11.1 Definition and Overview

OPC UA (OPC Unified Architecture) is a machine-to-machine communication protocol for industrial automation developed by the OPC Foundation. It is designed to provide a secure, reliable, and platform-independent framework for data exchange between various industrial devices and systems. OPC UA enhances the capabilities of the older OPC standards (OLE for Process Control) by integrating them into a single, unified architecture [31].

II.11.2 Key features of OPC UA include

1. **Platform Independence**: OPC UA can be implemented on any platform, including embedded systems, servers, and cloud-based applications.
2. **Security**: Provides robust security features such as authentication, authorization, encryption, and data integrity to protect against unauthorized access and ensure the integrity of data exchange.
3. **Scalability**: Suitable for applications of all sizes, from small devices to large-scale industrial systems, allowing for scalability as the system grows.
4. **Information Modeling**: Allows for complex data structures and relationships to be modeled and communicated, supporting a wide range of industrial use cases.
5. **Interoperability**: Designed to ensure seamless interoperability between different vendors' devices and systems, facilitating integration in heterogeneous environments.

Conclusion

The chapter provides step-by-step instructions on setting up and configuring the necessary software tools, including Factory I/O for simulation, PLC SIM ADVANCE 6.0 for data acquisition, Node-RED for data processing, and AWS EC2 with InfluxDB for data storage. It also explains how to integrate Grafana for real-time data visualization and monitoring. By following this guide, we gain a practical understanding of how to build a functional IoT system for monitoring electromechanical systems in real-world scenarios.

Chapter III

Collecting Data and Monitoring a Box Sorting System

Introduction

This chapter focuses on the development of a comprehensive system for monitoring and controlling a box sorting process using Factory I/O for simulation and TIA Portal for PLC programming. It details the step-by-step process of creating the project in TIA Portal, configuring the PLC hardware, and setting up the Factory I/O scene with conveyors, sensors, actuators, and an electric switchboard. The chapter explains how to simulate a PLC device, create input and output mappings, and write the control program using ladder logic. It further explores data collection by activating the OPC UA server in the PLC and utilizing Node-RED to read tag values and send them to an InfluxDB database. The chapter also covers the creation of a Grafana dashboard for real-time monitoring and visualization of system performance. Additionally, it discusses feedback mechanisms for pivot arms and conveyor belts, enabling the detection of malfunctions and tracking of lost boxes. Finally, the chapter concludes by explaining how to create a custom function block in TIA Portal to track the runtime of machine components, providing valuable insights into their operational duration.

III.1 Creating the Project in TIA Portal

We begin by launching the TIA Portal software on our computer. Once TIA Portal is open, we proceed by clicking the "Create new project" button. We then choose a suitable location on our computer to save the project files and give our project a name. In this case, we will name it "Project_MEC19". After confirming the name and location, we click the "Create" button to finalize the creation of our project.

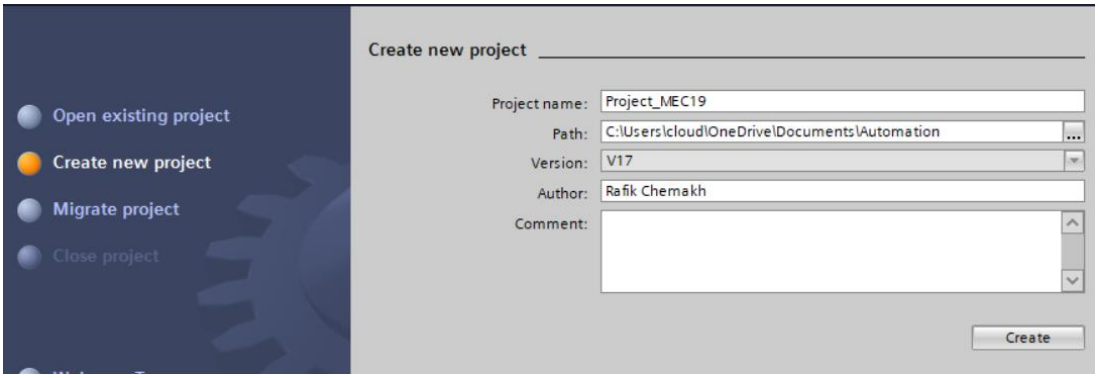


Figure III. 1 Creating a New Project in TIA Portal

III.1.1 Configuring PLC Hardware in TIA Portal

Since we're setting up our first program, let's click on "Configure a device" and then select "Add new device." For our project, we'll opt for the CPU 1515T-2 PN.

The Siemens S7-1500, CPU 1515T-2 PN is used as the central control unit for the box sorting system. This PLC was chosen for its versatile I/O configuration, processing power, and integrated Ethernet interface with support for the OPC UA communication protocol. The CPU 1515T-2 PN offers 750 KB of work memory for programs and 3 MB for data, sufficient for handling the control logic, data processing, and communication tasks of this application [47]

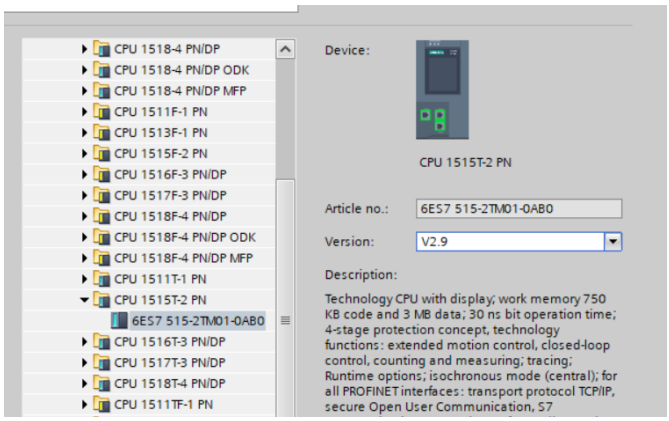


Figure III. 2 Configuring PLC Hardware

III.2 Setting up the Factory I/O Scene

We are setting up a system to sort three kinds of boxes: small (S), medium (M), and large (L). Our setup includes three conveyor belts: one is 6 meters long, another is 2 meters long, and the last one is a curved one. We're also using chute conveyors and two pivot arm sorters to precisely divert boxes onto designated paths based on their size.

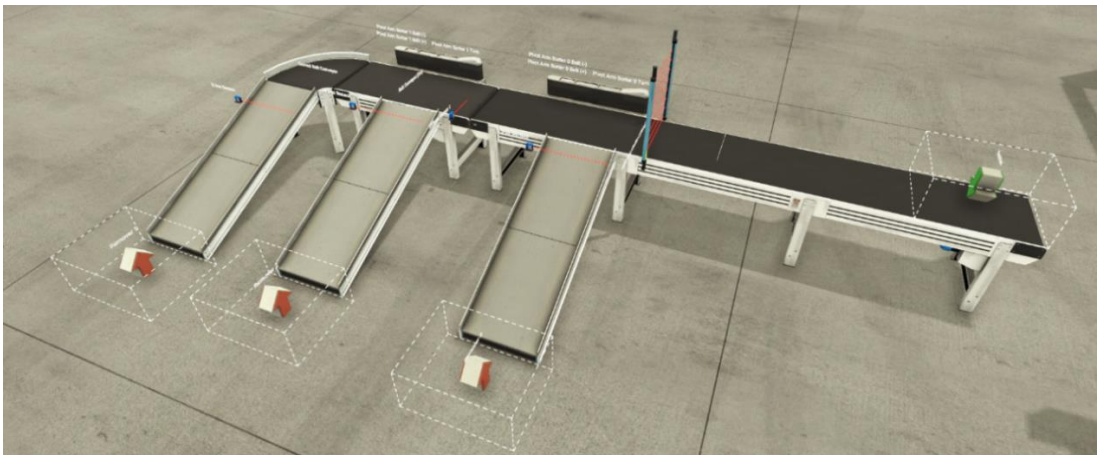


Figure III. 3 Factory I/O Scene

To manage the flow of boxes, we will implement an emitter for box generation at the beginning of the conveyor and removers stationed at the end of each chute conveyor to ensure box removal.

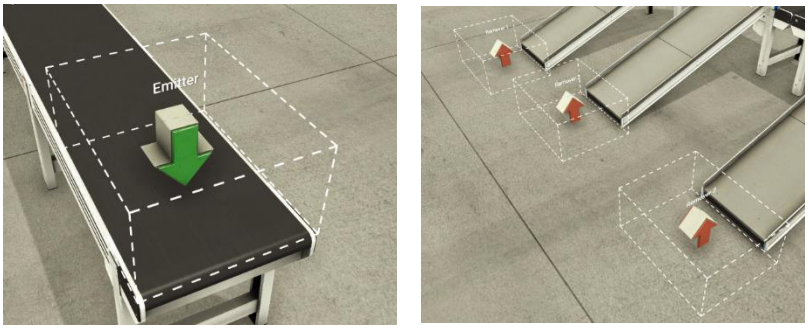


Figure III. 4 Factory I/O Scene Setup: emitter and removal

Since the heights of the L and (M, S) boxes are different, we'll use a sensor array to identify and direct L boxes using the first pivot arm sorter.

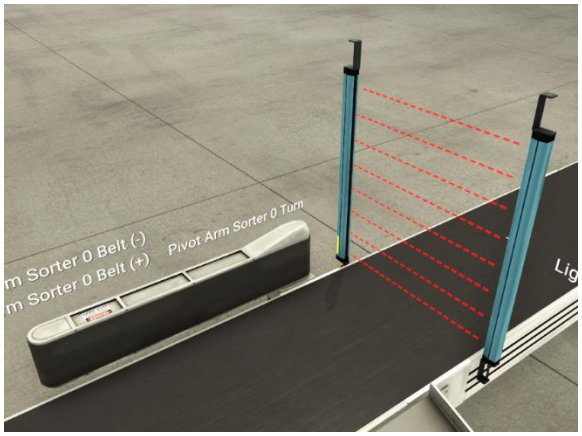


Figure III. 5 Sensor Array for L Box Detection

And because the S and M boxes have the same height but different widths, so to distinguish between S and M boxes we'll add another sensor close by to detect the M boxes. This will activate the second pivot arm sorter for M boxes.

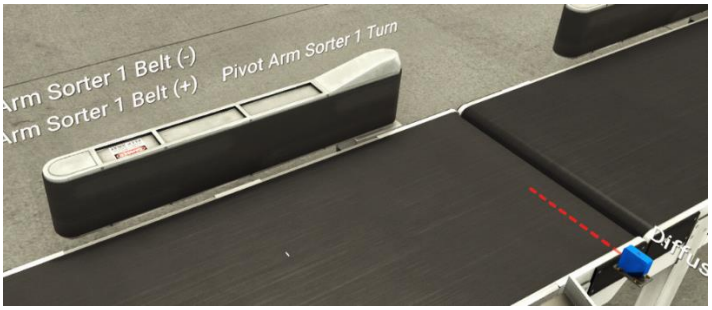


Figure III. 6 Sensor for M Box Detection

The S boxes will seamlessly continue along the conveyor's curve.

Figure III. 7 S Boxes Path

We'll put sensors at the start of each chute conveyor to check when a box arrives at its destination and to count the number of boxes. These sensors will help keep track of the boxes as they move through the system.

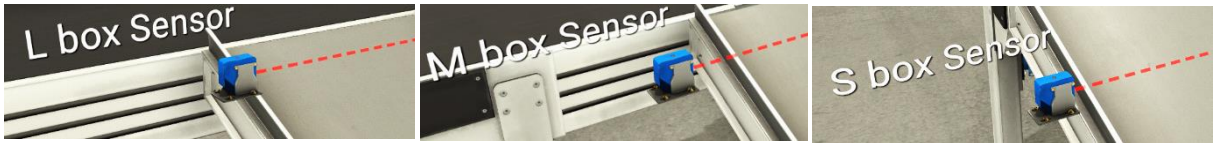


Figure III. 8 Sensors for count the number of boxes

In our electric switchboard setup, we'll have two buttons: one to start and one to stop the system. We'll also include a knob (potentiometer) to adjust the conveyor speed easily and a screen will show the speed settings. Additionally, we'll have three more screens to display how many boxes reach their destination. To reset box counts, we'll have buttons for each size and a master reset button to reset all counts at once.

Figure III. 9 Electric Switchboard

To ensure our conveyor system can operate at different speeds, we'll need to switch its control from digital to analog. To do this, we'll simply right-click on each conveyor, select 'configuration', and then choose the 'analog' option.

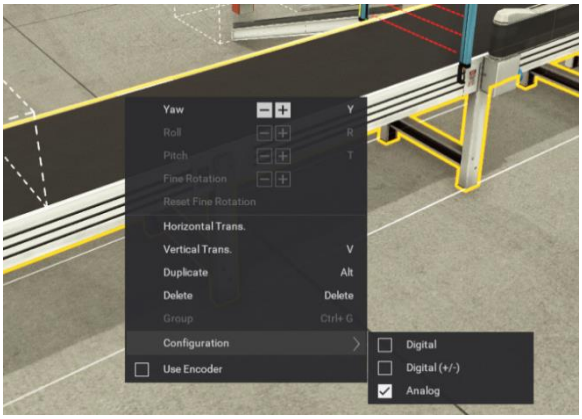


Figure III. 10 Conveyor Analog Configuration

III.3 Simulating a PLC Device

Now we have created the scene in factory it's time to control it with a PLC, To configure Factory I/O to work seamlessly with TIA Portal using PLC Sim Advanced, we'll first utilize the capabilities of PLCSIM Advanced, which provides a simulated environment mirroring real PLC hardware behavior

we open “PLCSIM Advanced V6.0” and in online access, we'll select TCP/IP Single Adapter and specify the instance name as "Project_IOT," with the IP address set to "192.168.0.19" and Subnet Mask as "255.255.255.0."

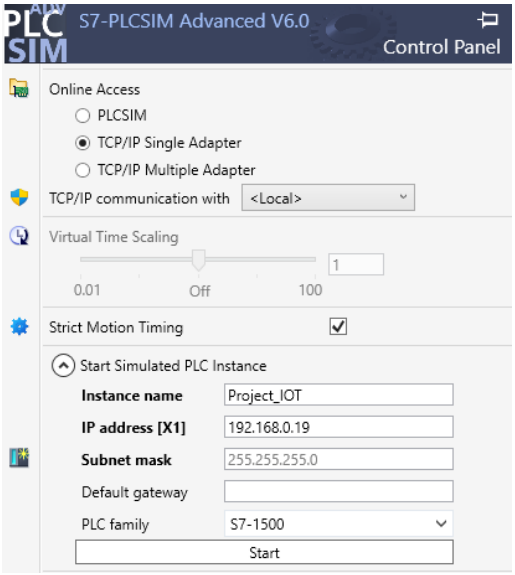


Figure III. 11 PLC Sim Advanced Configuration

Clicking on Start will initiate the process. Within a few seconds, the active instance status will be visible below.

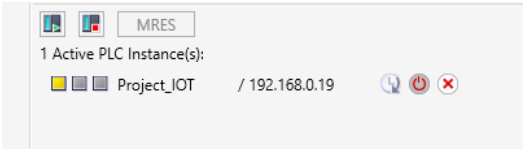


Figure III. 12 PLC status indication

Moving on to Factory I/O, we navigate to "File" and then "Drivers," or simply press "F4." From there, we select Siemens S7-1200/1500 from the driver drop-down list and click on CONFIGURATION to access the driver's Configuration Panel. We choose S7-1500 from the Model drop-down list and specify the IP address previously set in PLC Sim Advanced as "192.168.0.19." In the network adapter section, we select the PLCSIM virtual Ethernet adapter.

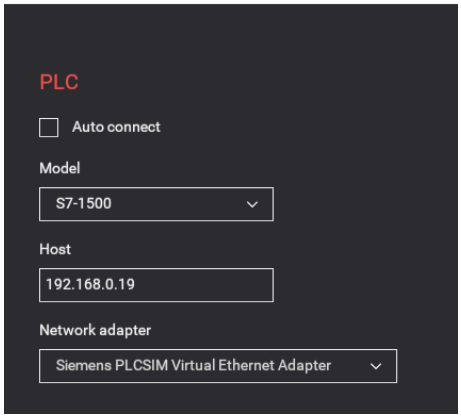


Figure III. 13 Factory I/O network Configuration

III.4 Creating Inputs and Outputs

For our input and output configurations, we'll designate 22 inputs and 20 outputs for digital signals with a offset 0, and 8 inputs and 14 outputs for analog signals with offset 100 .

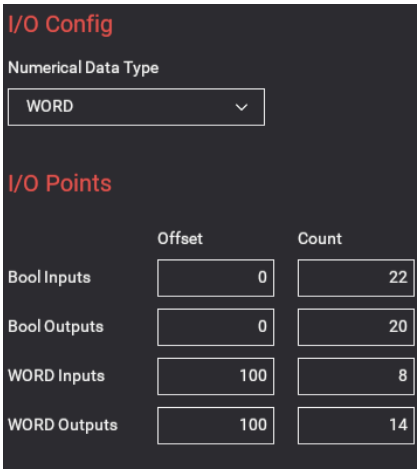


Figure III. 14 Factory I/O input and output configurations

When configuring inputs and outputs, it's crucial to ensure alignment between Factory I/O and TIA Portal addresses. We'll reference the provided table to make selections that match seamlessly across the software.

tag	Data Type	Data assignment	Adress
START	digital	Input	%I0.0
STOP	digital	Input	%I0.1
Reset ALL	digital	Input	%I0.2
Diffuse Sensor 1	digital	Input	%I0.3
S box Reset Button	digital	Input	%I0.4
M box Reset Button	digital	Input	%I0.5
L box Reset Button	digital	Input	%I0.6
L box Sensor	digital	Input	%I0.7
M box Sensor	digital	Input	%I1.0
S box Sensor	digital	Input	%I1.1
START LIGHT	digital	Output	%Q0.0
STOP LIGHT	digital	Output	%Q0.1
Reset ALL (Light)	digital	Output	%Q0.2
Emitter	digital	Output	%Q0.3
S box Reset Button (Light)	digital	Output	%Q0.4
M box Reset Button (Light)	digital	Output	%Q0.5
L box Reset Button (Light)	digital	Output	%Q0.6
Remover 1	digital	Output	%Q0.7
Remover 2	digital	Output	%Q1.0
Remover 3	digital	Output	%Q1.1
Pivot Arm Sorter 0 Turn	digital	Output	%Q1.2
Pivot Arm Sorter 0 Belt (+)	digital	Output	%Q1.3
Pivot Arm Sorter 1 Turn	digital	Output	%Q1.4
Pivot Arm Sorter 1 Belt (+)	digital	Output	%Q1.5
Conveyor Potentiometer	analog	Input	%IW100
Light Array Emitter 0 (Value)	analog	Input	%IW102
Belt Conveyor (6m)	analog	Output	%QW100
Belt Conveyor (2m)	analog	Output	%QW102
Curved Belt Conveyor	analog	Output	%QW104
Conveyor Input Voltage Screen	analog	Output	%QW106
S box Screen	analog	Output	%QW108
M box Screen	analog	Output	%QW110
L box Screen	analog	Output	%QW112

Table III. 1 Input and Output Tag Assignments

We Start mapping tags in factory i/o by dragging and dropping each one from “Sensors” and “Actuators” onto the intended port.

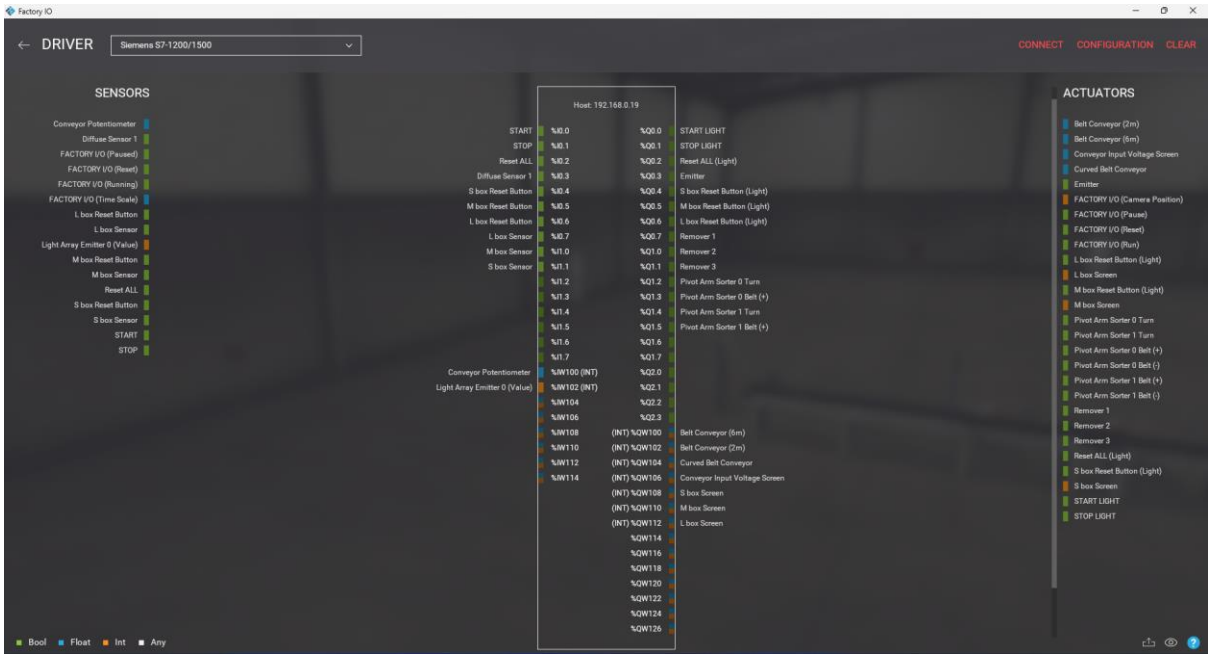


Figure III. 15 Tag Mapping in Factory

After that In TIA Portal, inside the project tree, we navigate to the “PLC Tags” folder. Then we click on “Add New Tag Table” and rename it “Sort Box Tags Table.” Next, we move on to creating the inputs and outputs with their associated address.

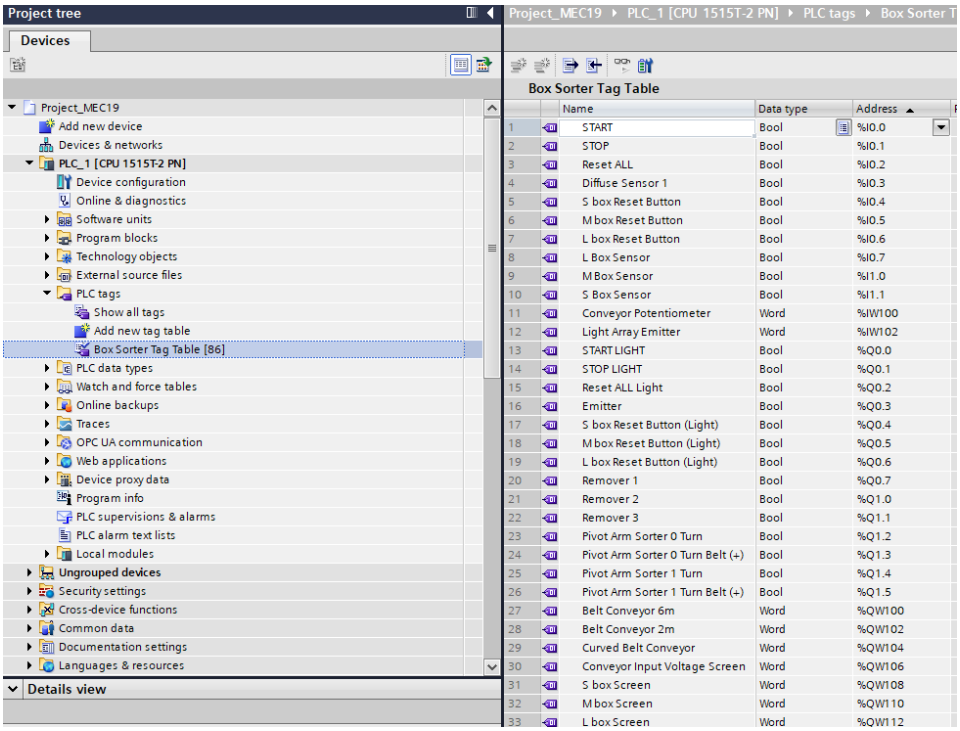


Figure III. 16 Tag Table Creation in TIA Portal

III.5 Writing the Program using Ladder Logic

In Project Tree we navigate to Program blocks and click on “Add new block” and pop up window appear, we will name it “ Box Sorter” and chose “Organization block”

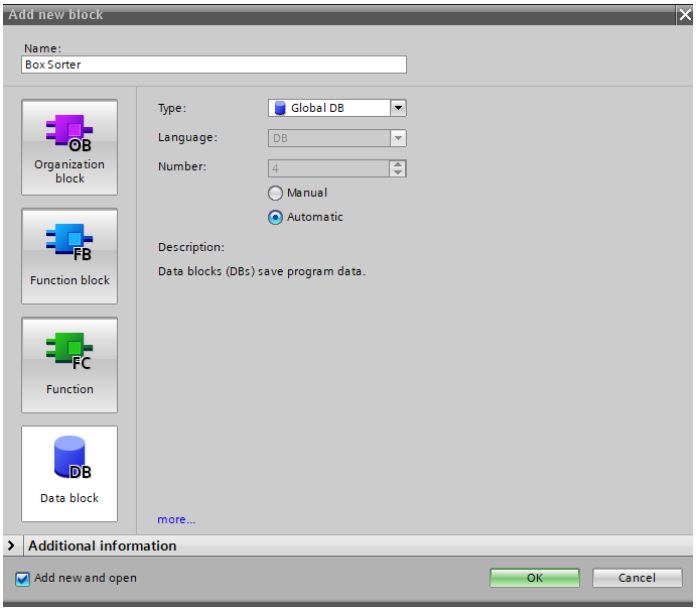


Figure III. 17 Adding an Organization Block in TIA Portal

III.5.1 Network 1

We click on insert network and give it name “START & STOP Emitter and Remover. “In this first network we will use “SET and RESET” block When we click on the start button, the “START LIGHT” turn on and the emitter start spawning boxes and Remover remove the boxes and by clicking on stop will stop the emitter and remover from functioning.

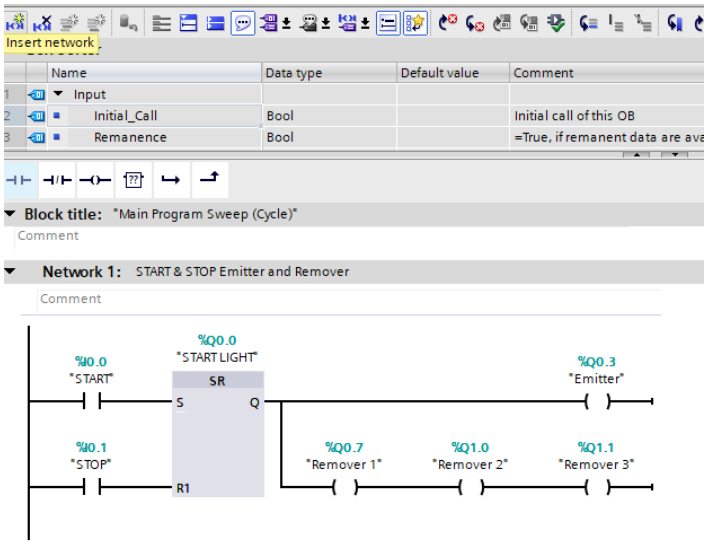


Figure III. 18 Ladder Logic Network 1: START & STOP Control

III.5.2 Network 2

For the second network, we'll incorporate a normally closed contact for the 'START LIGHT' and include an assignment instruction for the 'STOP LIGHT'. This configuration ensures that when the system is inactive, the stop light illuminates, effectively indicating the system's status.

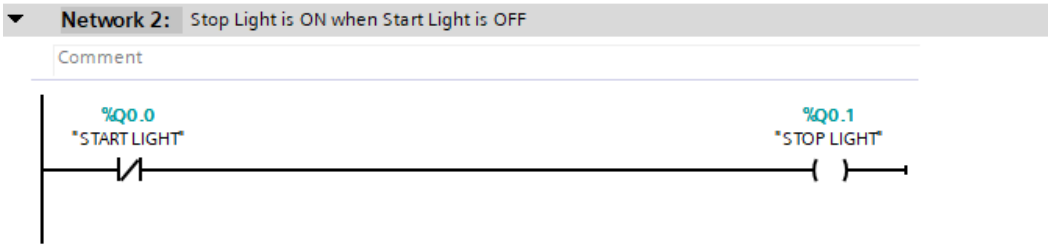


Figure III. 19 Ladder Logic Network 2: STOP Light

III.5.3 Network 3

We'll utilize the 'MOVE' function to transfer the value from the potentiometer to the screen. This allows us to verify the desired speed before starting the system. Then, we'll open a parallel branch and insert a normally open contact for the start light. Using another 'MOVE' function, we'll copy the potentiometer value to all three conveyor belts. This configuration ensures that whenever we click the start button, the conveyor belts begin moving.

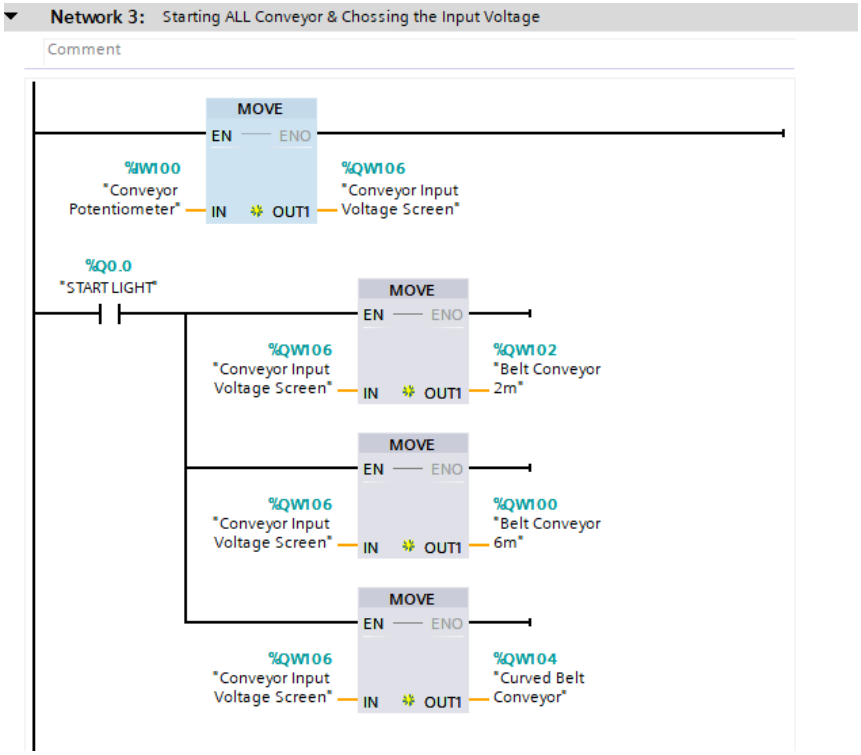


Figure III. 20 Ladder Logic Network 3: Conveyor Speed Control with Potentiometer

III.5.4 Network 4

For stopping the system we'll use the "MOVE" function to assign the value "0" to all three conveyor belts whenever we click on “STOP” button. This action effectively halts the system's operation.

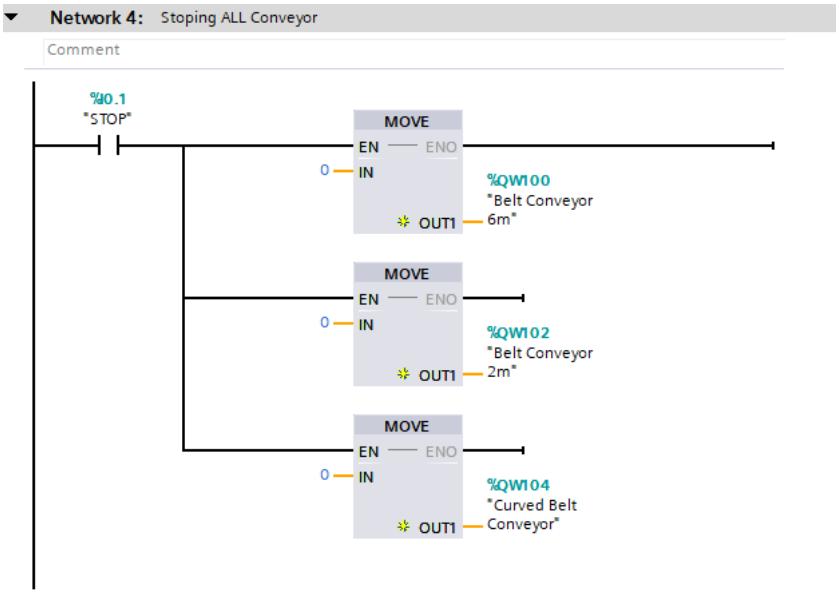


Figure III. 21 Ladder Logic Network 4: Stopping the Conveyor Belts

III.5.5 Network 5

In this network, the goal is to divert the "L" boxes. When a box "L" interrupts three light beams of the light array, it returns a numerical value of 224, so we'll employ a comparator operation, specifically "equal" (==). Upon detecting a value of 224 in the light array, it will set the outputs for "Pivot Arm" and "Pivot Arm belt," which will effectively divert the "L" box to its designated destination.

To ensure the Pivot Arm returns to its original position after diverting the box, we'll reset the outputs for "Pivot Arm" and "Pivot Arm belt" when the "L" box crosses the sensor at the start of the chute conveyor. This will ensure that the box successfully diverts to its intended destination before the Pivot Arm returns to its default position.

Moreover, we'll incorporate a safety measure to reset the Pivot Arm to its original position whenever the stop button is pressed. This feature provides reassurance that pressing the stop button will promptly return the Pivot Arm to its default position, safeguarding against any potential issues with its operation.

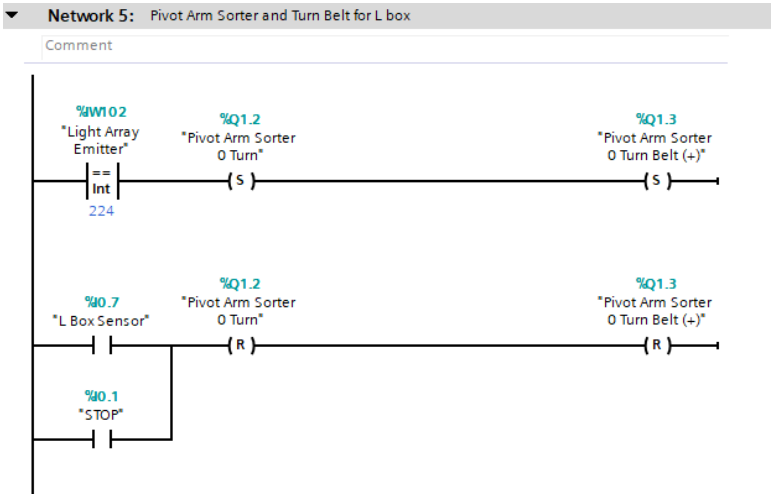


Figure III. 22 Ladder Logic Network 5: L Box Sorting Logic

III.5.6 Network 6

In network 6, the objective is to redirect boxes labeled "M". However, due to the similar height of "M" and "S" boxes, the light array returns the same value of 96, making it unsuitable for distinguishing between them. But since the boxes have different widths, we will utilize "Diffuse Sensor 1". This sensor, with its short distance capability, can specifically detect the "M" boxes as they cross its path.

So Following the methodology from “Network 5” we'll set the outputs for "Pivot Arm 1" and "Pivot Arm belt 1" when a box crosses "Diffuse Sensor 1” and reset these outputs when an "M" box crosses "Diffuse Sensor 1" at the beginning of the second chute conveyor or when the stop button is pressed.

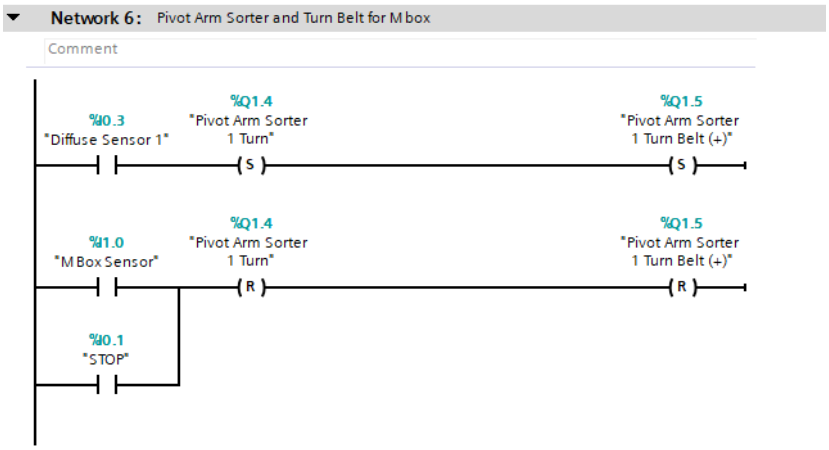


Figure III. 23 Ladder Logic Network 6: M Box Sorting Logic

For boxes labeled "S", we won't employ the Pivot Arm to divert them. Instead, these boxes will proceed along their path and transition to the last chute conveyor using curve belt.

Figure III. 24 S Boxes Path without Diversion

III.5.7 Network 7,8,9

In network 7 we will count the number of “M” boxes that reach it destination using Counter operation “CTU” which means "Count up" , this instruction to increment the value at output CV. When the signal state at the CU input changes from "0" to "1”,

The value at the CV output is reset to zero when the signal state at input R changes to "1". As long as the R input has signal state "1", the signal state at the CU input has no effect on the instruction.

We'll utilize sensors positioned at the start of each of the three chute conveyors to trigger the counter signal state at the "CU" input and to reset the count, we'll employ reset buttons corresponding to each box type. Alternatively, we can use the "Reset All" button, serving as a master button to reset the count for all box types simultaneously.

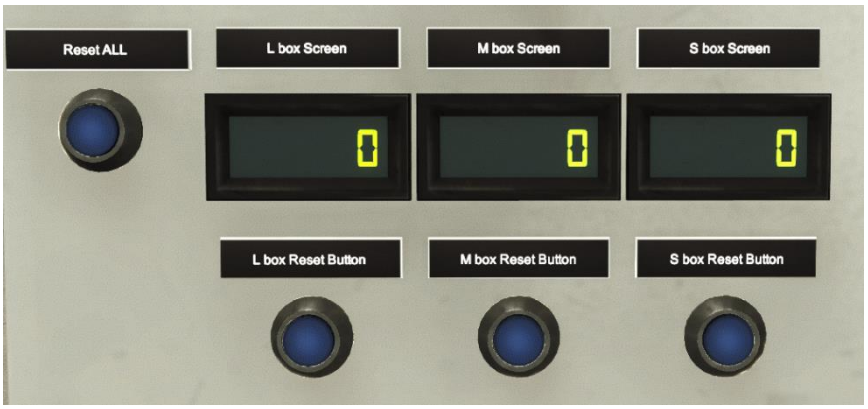


Figure III. 25 Reset Buttons

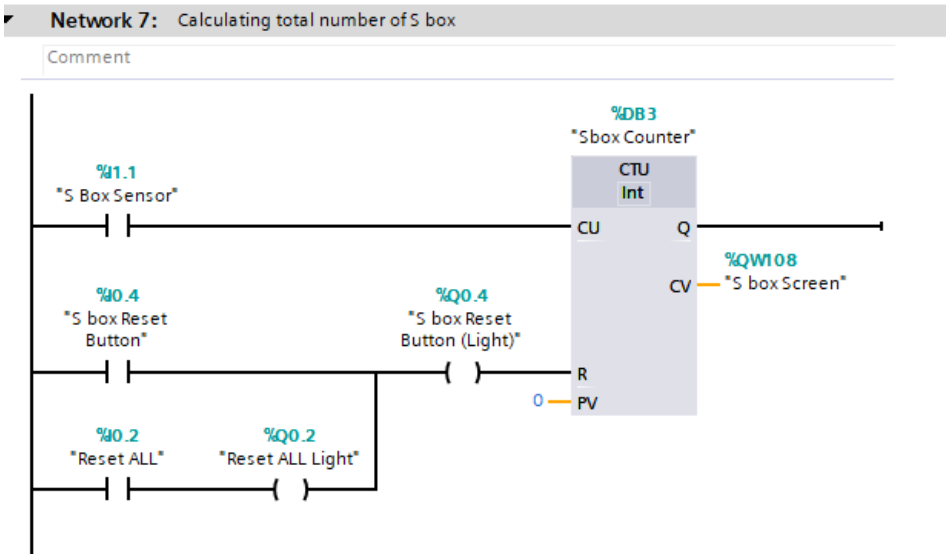


Figure III. 26 Counting S Boxes

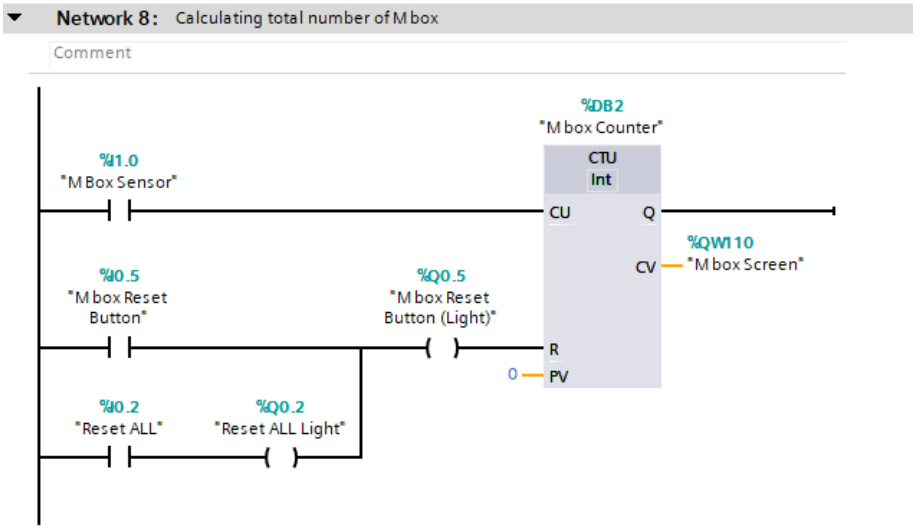


Figure III. 27 Counting M Boxes

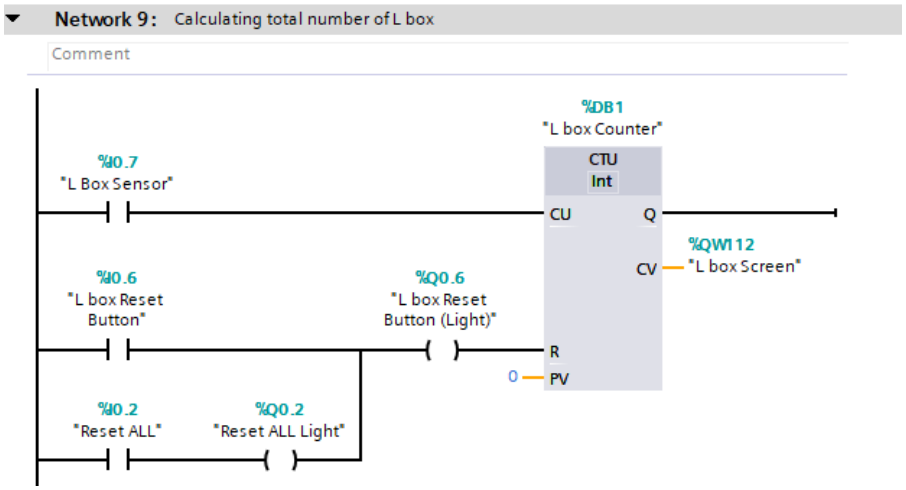


Figure III. 28 Counting L Boxes

III.5.8 Testing the Simulation

To begin simulating the demo scene project, we need to ensure that several settings are enabled to prevent errors during the download of the program to the PLC or while connecting to Factory i/o

First, in the project tree, right-click on the project name "Project_MEC19" we select "Properties". And we Navigate to the "Protection" tab and we check the option "Support simulation during block compilation".

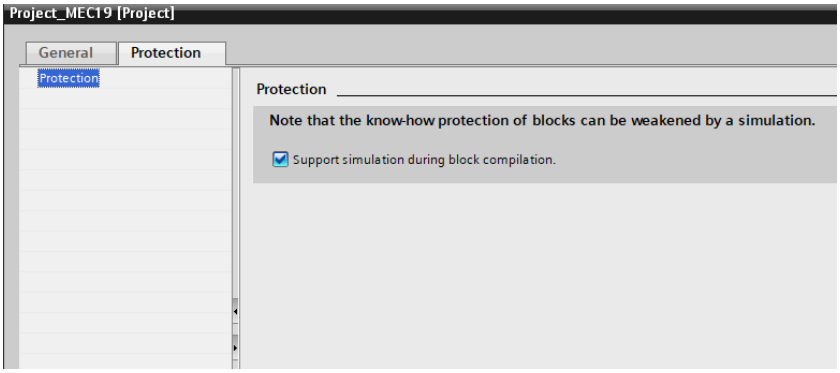


Figure III. 29 Enabling Simulation Support and Communication Access

Next, right-click on the PLC name "PLC_1 [CPU 1515T-2 PN]" then we go to "Properties". And we navigate to "Protection and Security" and we enable the option "Permit access with PUT/GET communication from remote partner".

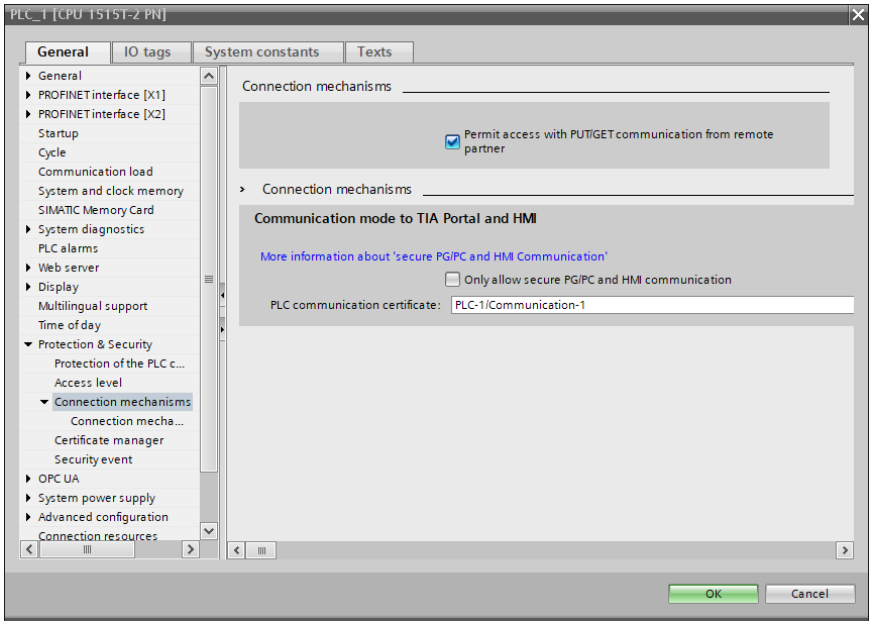


Figure III. 30 Enabling Permit access

Then, we go to the PROFINET interface [X1] and under Internet Protocol version 4 (IPv4), we enter the IP address (192.168.0.19) and Subnet mask (255.255.255.0) that was specified earlier in PLCSim Advanced when we created the PLC instance (PROJECT_IOT). And we Click "OK" to save the changes.

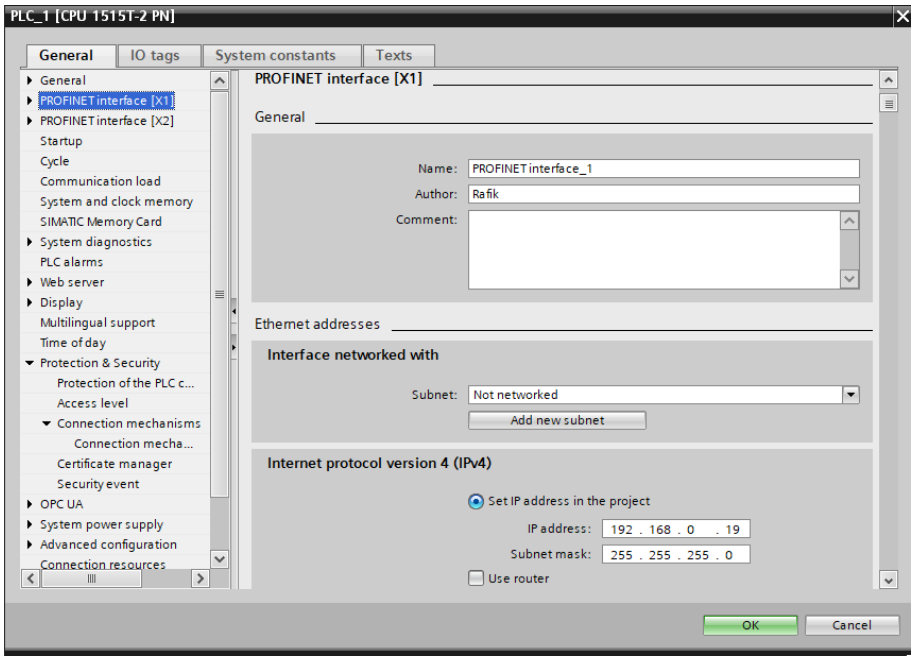


Figure III. 31 Configuring PROFINET Interface with IP Address and Subnet Mask

With these settings enabled, we can now proceed to load the program we created in TIA Portal to the PLC instance and start simulating the demo scene project.

III.5.9 Loading the Program onto the PLC Instance

After compiling the ladder program, we proceed by clicking on "Download to device". A window then appears, prompting us to select the appropriate network interface. Under "PG/PC interface", we specifically choose "Siemens PLCSIM Virtual Ethernet Adapter". Next, we initiate a scan by clicking on "Start Search". Once the scan is complete, two accessible devices will be listed. We opt for the first one, which corresponds to the IP address specified in PLCSIM Advanced: "192.168.0.19". and we click on “Load.”

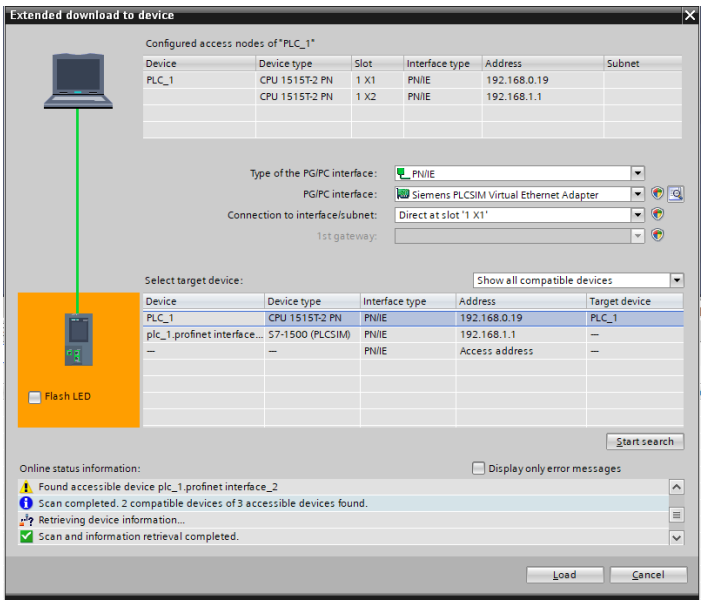


Figure III. 32 Downloading Program to PLC Instance: Selecting Network Interface

then we click again on “Load”

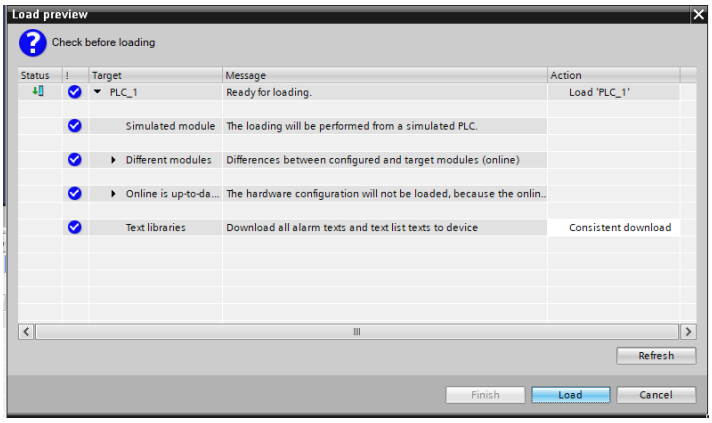


Figure III. 33 Downloading Program to PLC Instance: Initiating Load

Then we chose “Start module” and click on “Finish”

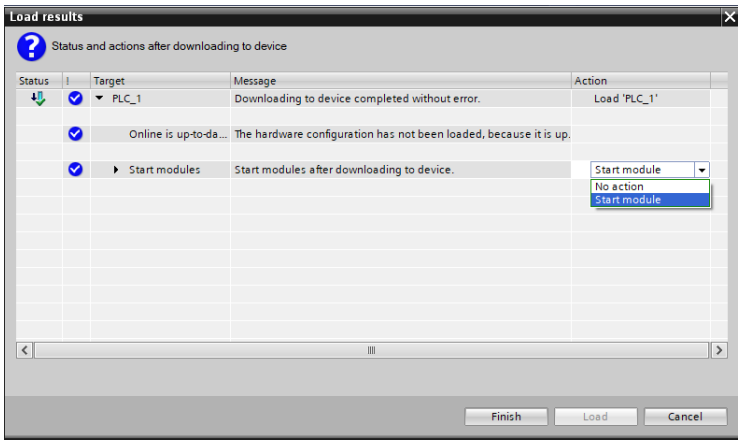


Figure III. 34 Downloading Program to PLC Instance: Starting Module

In PLCSIM Advanced, the light changes from orange to green, indicating that the program has been successfully loaded into the PLC instance

Figure III. 35 Program Loaded Successfully in PLCSIM Advanced

Now, we go to Factory IO and in the DRIVER option, we click on connect. A green check mark appears, indicating that Factory IO has successfully connected to the PLC.

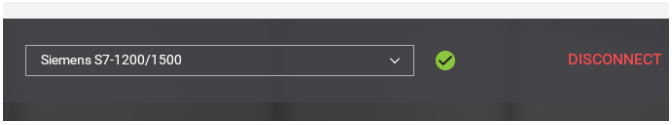


Figure III. 36 Factory I/O Connected to PLC

We return to the scene in Factory IO and click on "Run" mode.

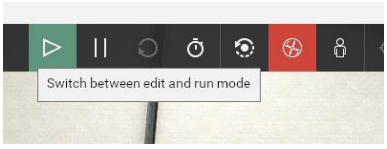


Figure III. 37 Running the Simulation in Factory I/O

Now, we'll begin testing System 01. In TIA Portal, we enable monitoring.

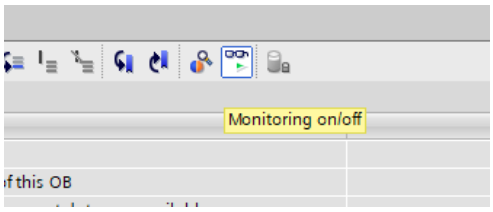


Figure III. 38 Enabling Monitoring in TIA Portal

Upon observation in Network 2, we notice that the STOP light is on. Confirming in Factory IO's electric switchboard, we see that the STOP light is indeed illuminated red, affirming proper functionality.

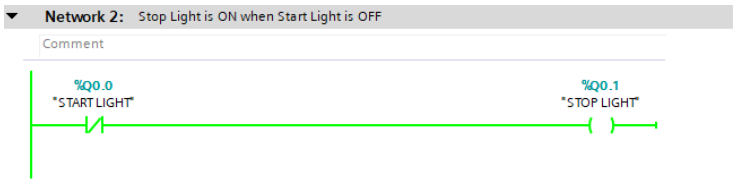


Figure III. 39 Stop Light Status in TIA Portal and Factory I/O

Next, as we rotate the potentiometer knob, we observe the selected value displaying on the screen. Let's select a random value for the input voltage speed, such as 3.14 volts. Now, when we click on the START button, the START button light turns green, and the STOP light turns off and The boxes begin spawning ,confirming that our first and second and third networks are functioning correctly.

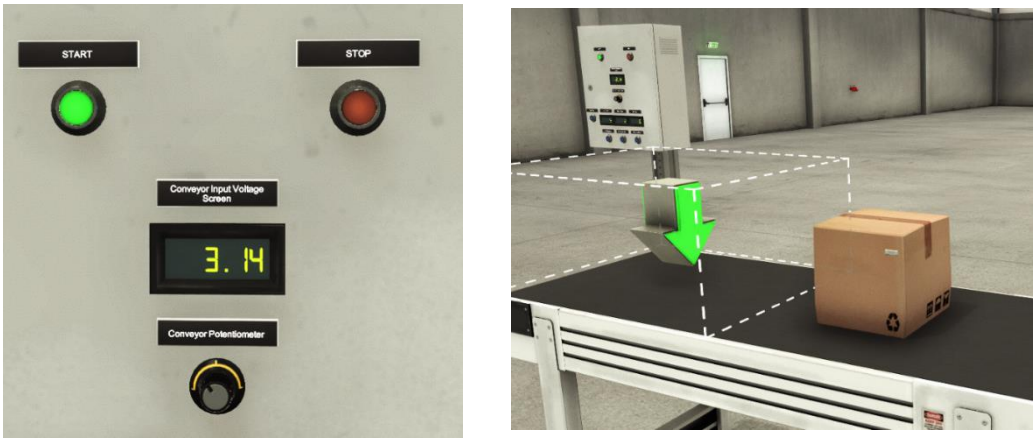


Figure III. 40 START Light Status in Factory I/O

Now, let's check if the sensors and pivot arm are working correctly with the corresponding boxes. We'll start with the "L" box. To ensure that only "L" boxes are spawned, we switch to edit mode in Factory IO. Then, we right-click on the Emitter and navigate to Part to Emit. Here, we deselect "S" and "M" and keep only "L". This ensures that only "L" boxes will be spawned. Now, we can switch back to run mode, and "L" boxes start spawning.

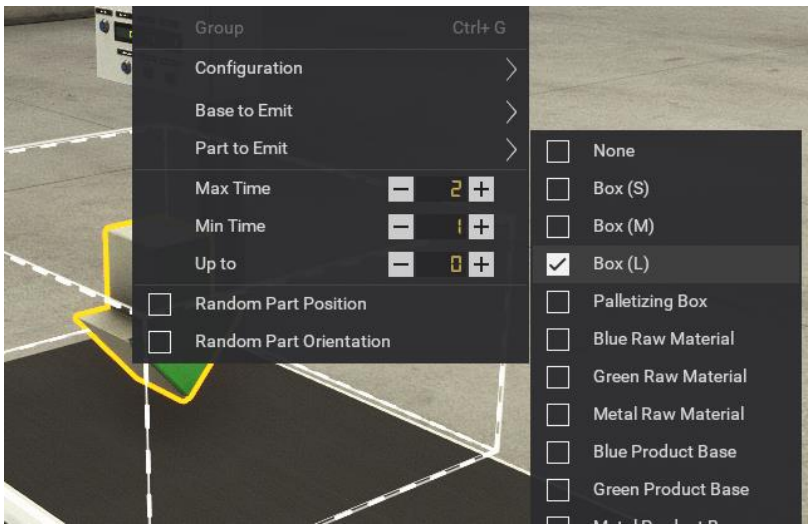


Figure III. 41 Emit L box only

Next, once the box interrupts the light array beam, we notice that the pivot arm and pivot arm belt get activated. Once the box crosses the sensor at the beginning of the chute conveyor, the pivot arm returns to its original position, and the pivot arm belt turns off. Additionally, in the electric switchboard, we observe that one box in the category "L" has been counted. This confirms that both Network 5, responsible for detecting and diverting "L" boxes and Network 9 which is responsible for counting the number of "L" boxes that reach their final destination, are working correctly.

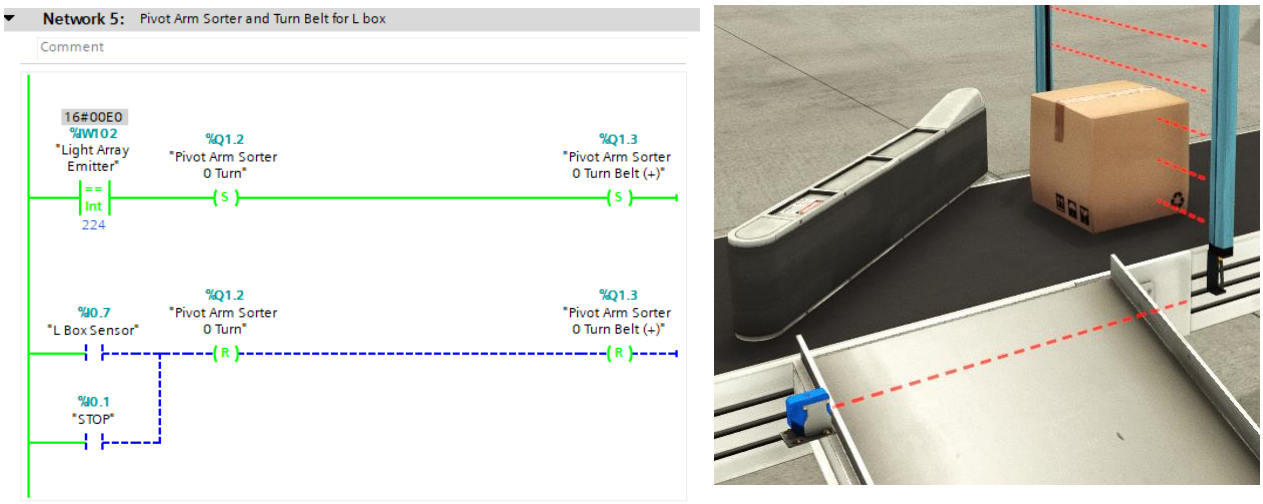


Figure III. 42 pivot arm and pivot arm belt status

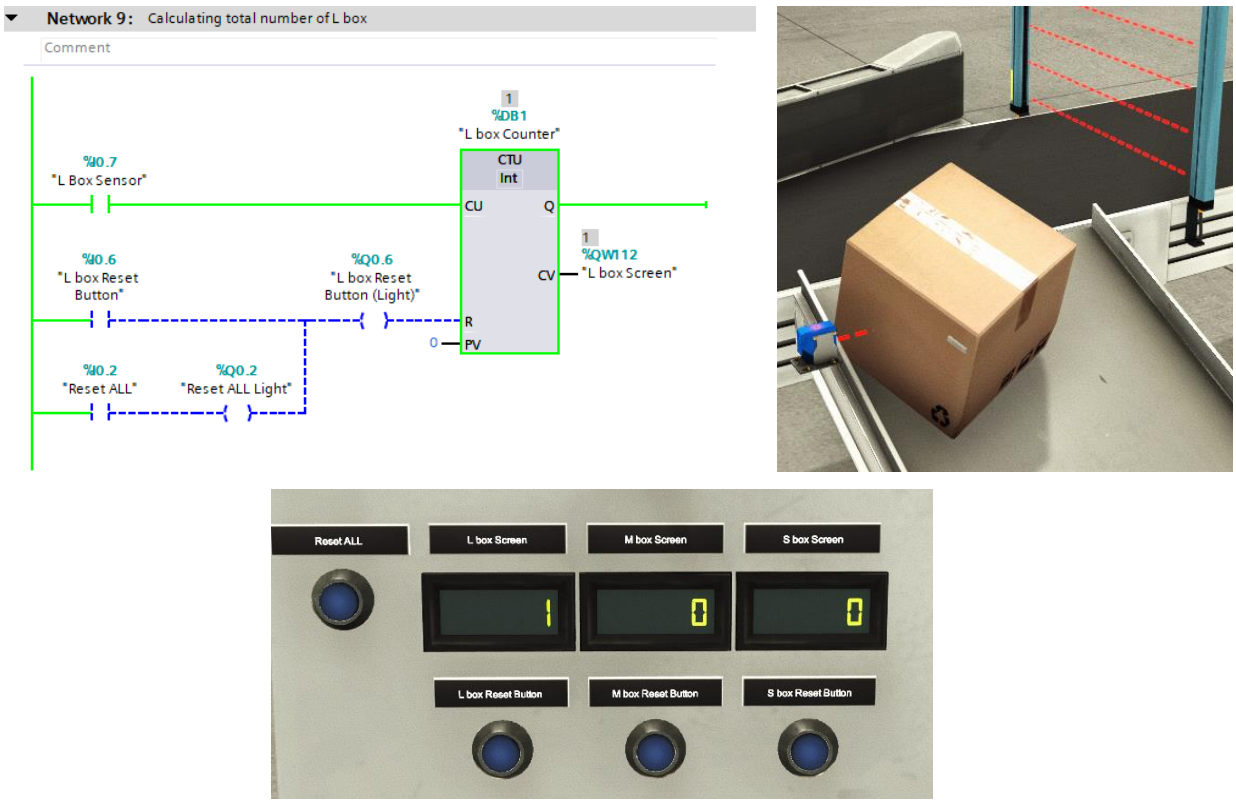


Figure III. 43 L Box Sorting and Counting: Pivot Arm Activation

Now, we will repeat the same process for box "M". First, we ensure that only box "M" is spawned, and we observe the system's behavior. As expected, everything works fine. The second pivot arm and pivot arm belt get activated as soon as the "M" box crosses the sensor. Once it crosses the sensor at the second chute conveyor, the pivot arm and pivot arm belt get deactivated, and the counter counts one box,so the network 6 and network 8 is working perfectly

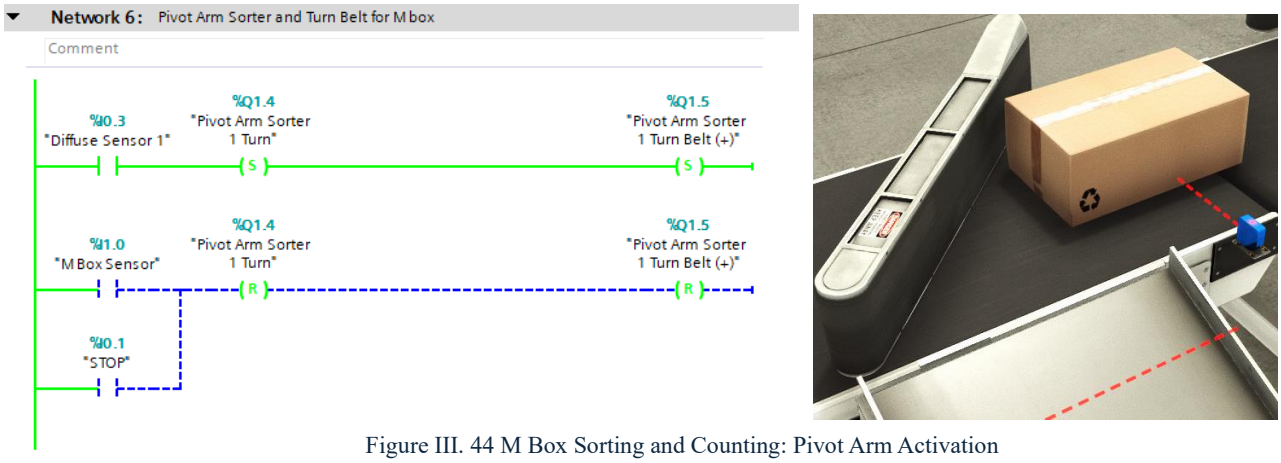


Figure III. 44 M Box Sorting and Counting: Pivot Arm Activation

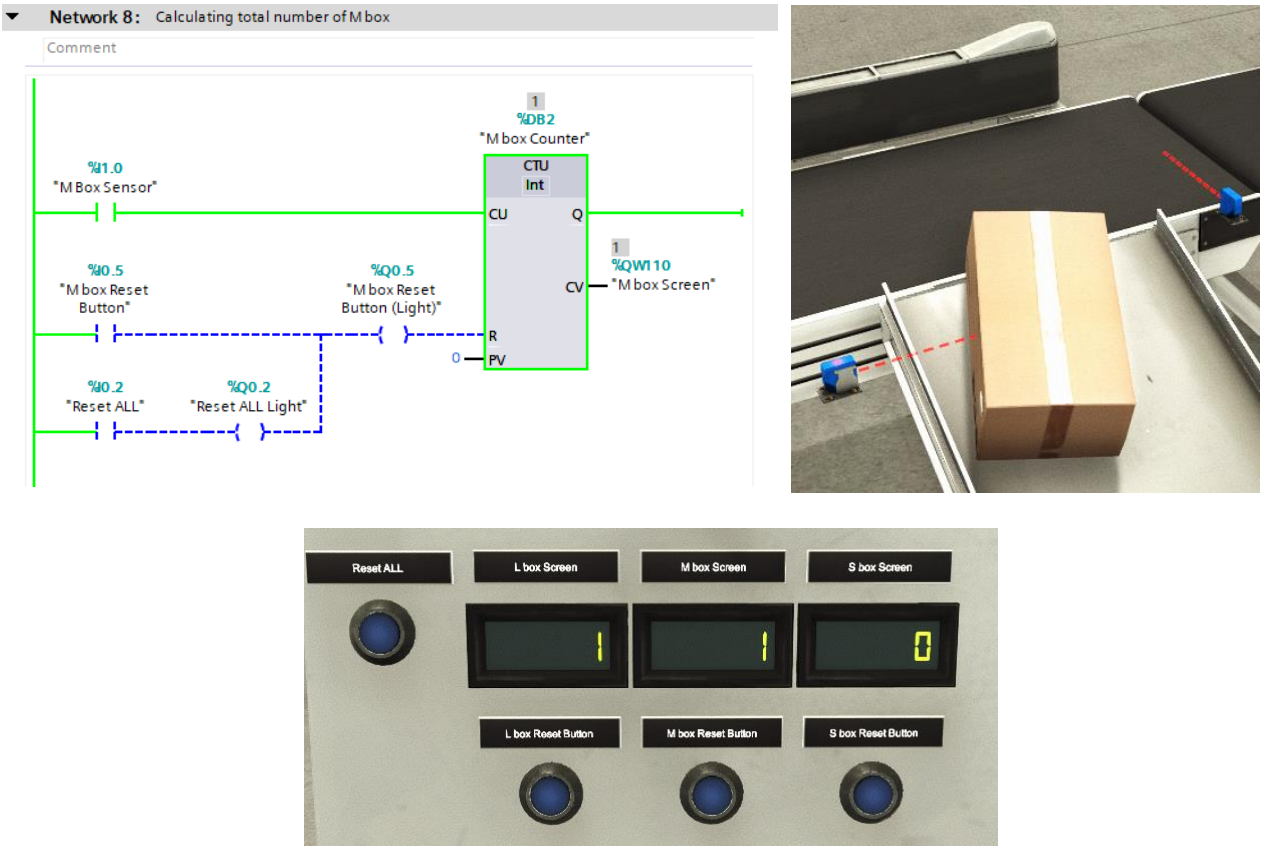


Figure III. 45 M Box Sorting and Counting: Pivot Arm Reset and Box Count

Next, we repeat the same procedure for "S" boxes. Upon observation, we notice that the "S" boxes continue their path to the last chute conveyor. Once they cross the sensor, the counter on the screen increments by 1, confirming the successful of the network 7

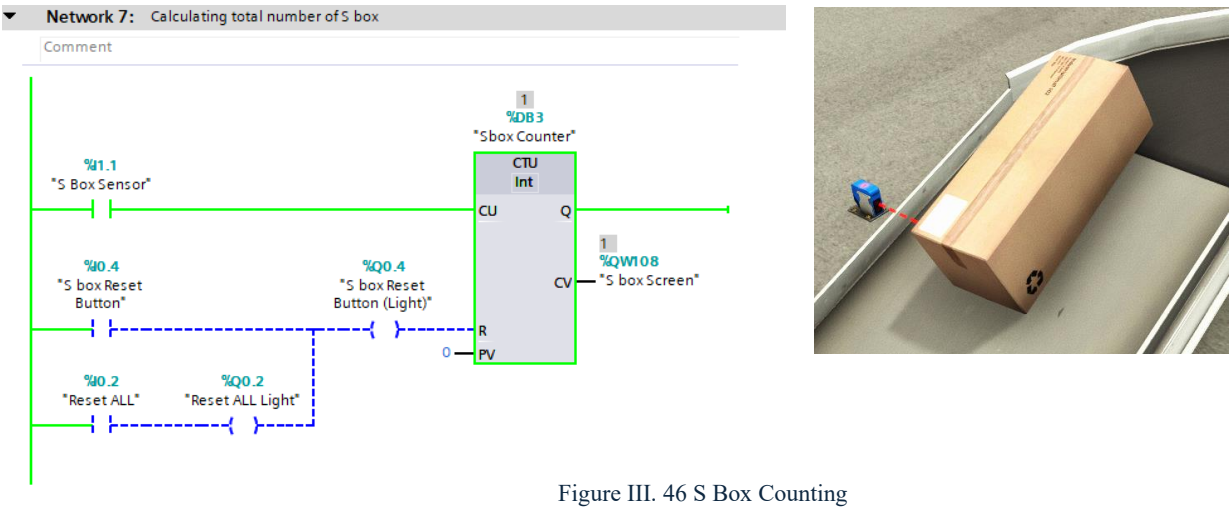


Figure III. 46 S Box Counting

III.6 Data Collection

III.6.1 Activating OPC UA Server

It is now time to start collecting data from both systems we created using Node-RED. However, before we do that, we need to enable the OPC-UA server in the PLC and test that it is working correctly.

First, we head to TIA Portal and click on "Offline" from the top menu. From the project tree, we click on the PLC name "PLC_1 [CPU 1515T-2 PN]" and choose "Properties." In the General tab menu, we look for the OPC UA section and click on it to open the settings.

In the OPC UA menu, we look for the Server option and click to select it. This will open the OPC UA Server options. Here, we activate the checkbox “Activate OPC UA Server” to enable the OPC Server. Under Server Addresses, we take note of the link and IP address with the port “192.168.0.19:4840” of the OPC Server. This information will be needed in the Client Settings to connect to the Server.

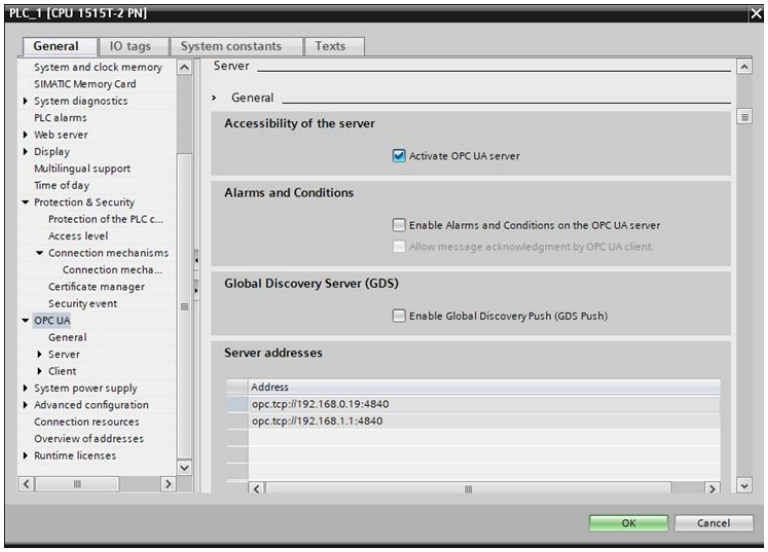


Figure III. 47 Activating OPC UA Server in TIA Portal

III.6.2 Testing OPC UA Server

Now, we are going to test the OPC UA server. There are many OPC UA client softwares available, and for this test, we will be using “UaExpert”. UaExpert is designed as a general-purpose test client that supports various OPC UA features.

To begin, we open the UaExpert software. From the top menu, we click on the plus icon to add a new server. Once clicked, a window pops up, and we select "Custom Discovery". Next, we insert the IP address "192.168.0.19" followed by the port "4840", and then click OK.

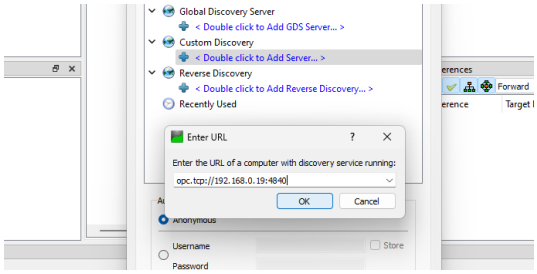


Figure III. 48 Adding a New Server in UaExpert

the server information will be displayed. In this tab, we choose the authentication methods. For this test, we proceed with "None", indicating that no authentication is required for accessing the server.

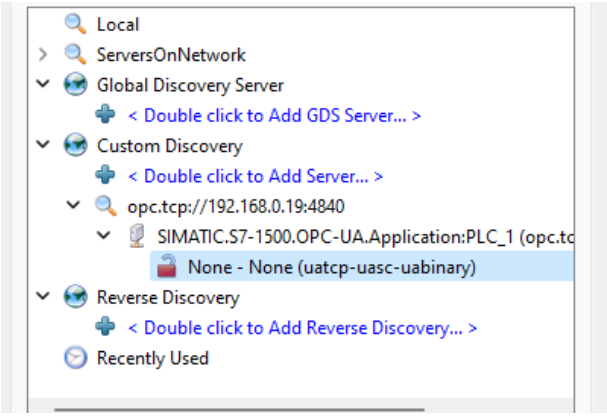


Figure III. 49 Server Configuration in UaExpert: Authentication Method

From the "Project" field on the left side of the software interface, we right-click on the server and choose "Connect". Once the connection is completed, we can see the address information in the "Address Space" field. This confirms that the connection to the OPC UA server has been established successfully, and we are now ready to access the server data

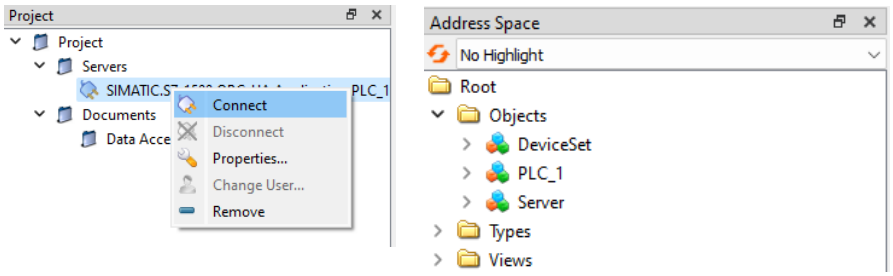


Figure III. 50 Server Information in UaExpert

Next, from the "Address Space" field, we navigate to the "PLC_1" folder, and then we scroll through and open the "INPUT" and "OUTPUT" folders. Here, we find all the tags we created in TIA Portal. By dragging a tag to the "Data Access View" field, we can examine its details and modify its value if it's a writable address.

For instance, we drag the input tags "START" and "STOP" buttons, along with the output tag "Belt Conveyor 2m", into the "Data Access View" field. Then, we initiate the simulation of the scene in Factory IO. Upon doing so, we observe that the values of the "START" and "STOP" buttons toggle to true each time they are clicked. Similarly, upon selecting the speed for the conveyor and starting it, we can monitor the value of the belt conveyor speed.This verifies the proper functioning of our OPC UA server

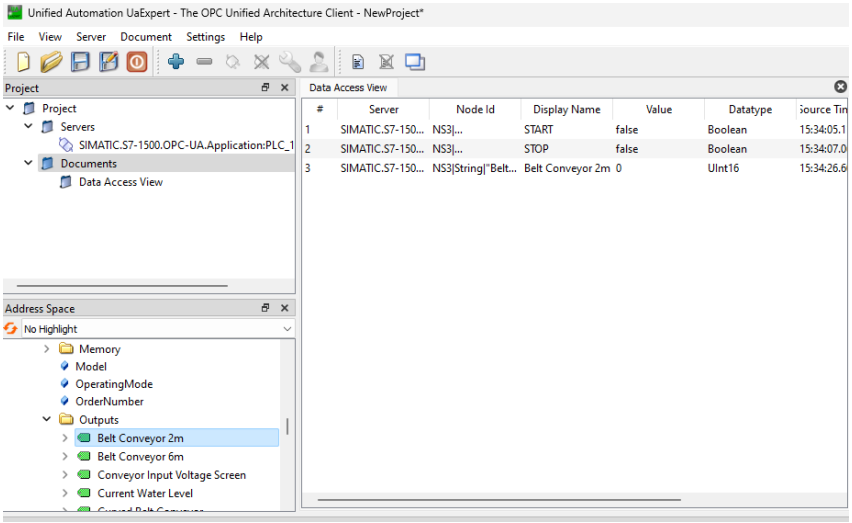


Figure III. 51 Testing OPC UA Server with UaExpert

III.7 Node Red

III.7.1 Adding the Necessary Palette

Now, let's transition to Node-RED. We open our browser and navigate to the local address "127.0.0.1" and port "1880" to access the Node-RED User Interface.

Our first step in Node-RED is to incorporate the necessary libraries. We open "Settings" and then "Palette". In the "Install" tab, we search for and install "node-red-contrib-opcua" and "node-red-contrib-influxdb", These additions are necessary for data collection in Node-RED.

Upon returning to the nodes tab, we can see that the two libraries we installed have been successfully added.

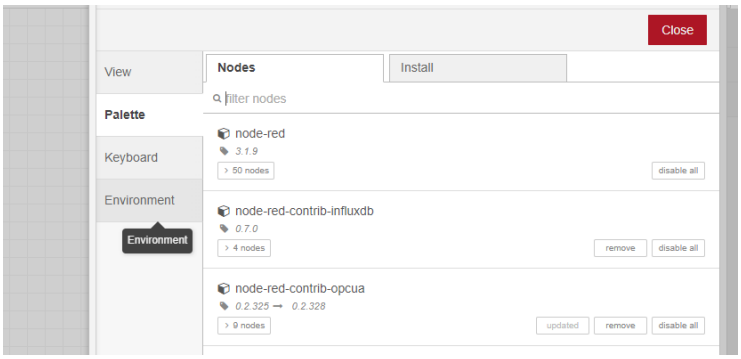


Figure III. 52 Adding Node-RED Libraries

III.7.2 Reading Values and Creating Flows

We will now begin the process of data collection. Start by dragging the "inject", "opc ua item", "opc ua client", and "debug" nodes into the layout, then connect them together.

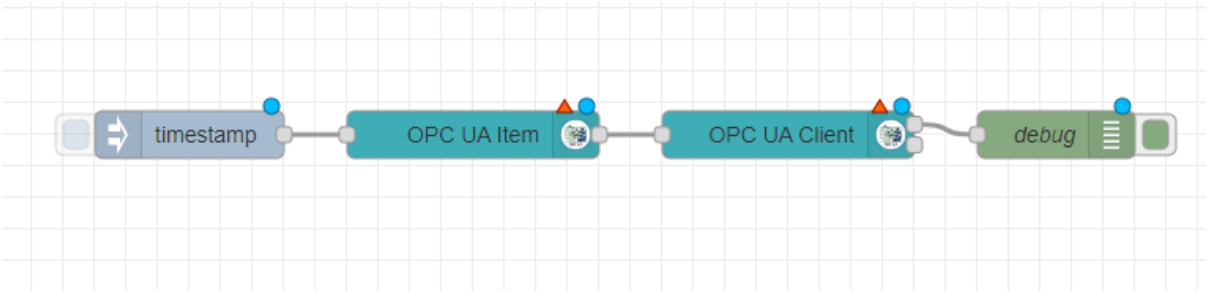
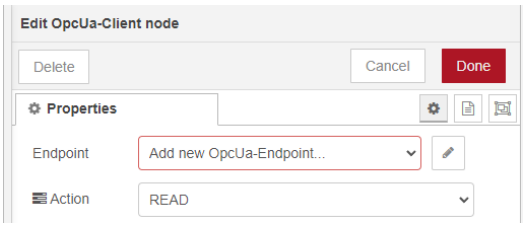


Figure III. 53 Node-RED Flow

Next, double-click on the "opc ua client" node. Edit the Endpoint and input our server address endpoint along with the port: opc.tcp://192.168.0.19:4840. Change the action to "Read" since we want to read the value of tags.

Since we chose 'No security' security policy in TIA Portal while activating the OPC UA server, we will leave "Use credentials" and "User certificate" unchecked in the Node-RED "opc ua client" node settings. This ensures that no additional security credentials or certificates are required for the connection



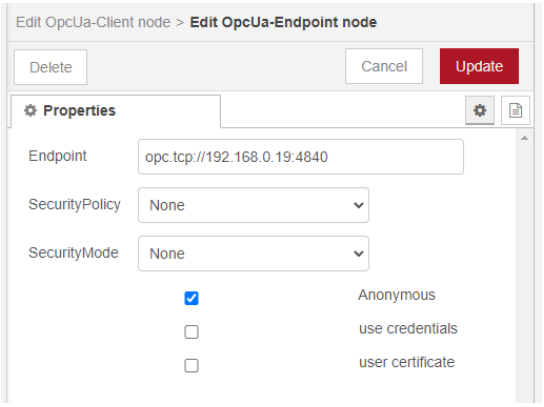


Figure III. 54 OPC UA Client Node Configuration: Endpoint and Action

Then, we move to the "opc ua item" node to select which tag we need to read its value. We'll start with the input "Conveyor Potentiometer".In the item column, we need to fill in the namespace index ("ns") and the String type NodeId ("s"). To get these values, we can refer back to "UaExpert". In the Attributes tab, under NodeId, we can see the values for the namespace index ("ns") and the String type NodeId ("s"). Copy these values and paste them into the item column in Node-RED

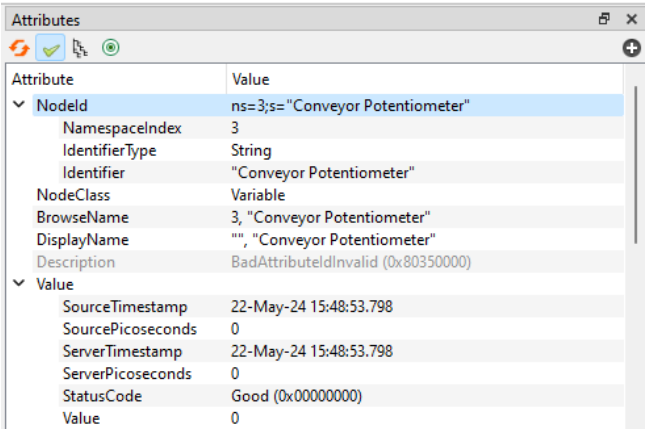


Figure III. 55 OPC UA Item Node Configuration: Namespace and NodeId

If we scroll down in the Attributes tab, under "Datatype," we see that the tag is of type "word." Therefore, we will choose "UInt16" (unsigned integer with 16 bits) for the type of the tag . In the "name" field, we can choose any label we want, so we will name it "Conveyor Potentiometer".

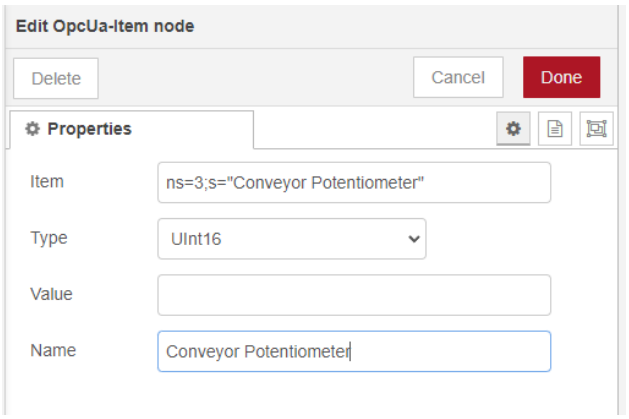


Figure III. 56 OPC UA Item Node Configuration: Data Type and Name

Once we finish, we click on "Done" and then on the "Deploy" button. Under the “opc ua client” node, it should report “session active,” indicating that the server is configured correctly

Figure III. 57 OPC UA Client session status

Next, in "UaExpert", we drag the input tag "Conveyor Potentiometer" into the "Data Access View" field. Then, we go to Factory IO and choose a random value from the switchboard.

Returning to Node-RED, we click on the inject node. In the debug window, we can notice that the value matches the same value displayed in “UaExpert.” This confirms that our configuration is working correctly.

Figure III. 58 Testing Value Reading in Node-RED: Debug Output

we will repeat the same processe for the other tags that we want to collect data about them . so in the end will have these tags accorrding to this table.

tag	Data Type	Data assignment
START	Boolean	Input
STOP	Boolean	Input
Reset ALL	Boolean	Input
S box Reset Button	Boolean	Input
M box Reset Button	Boolean	Input
L box Reset Button	Boolean	Input
Pivot Arm Sorter 0 Turn	Boolean	Output
Pivot Arm Sorter 0 Belt (+)	Boolean	Output
Pivot Arm Sorter 1 Turn	Boolean	Output
Pivot Arm Sorter 1 Belt (+)	Boolean	Output
Conveyor Potentiometer	UInt16	Input
Belt Conveyor (6m)	UInt16	Output
Belt Conveyor (2m)	UInt16	Output
Curved Belt Conveyor	UInt16	Output
S box Screen	UInt16	Output
M box Screen	UInt16	Output
L box Screen	UInt16	Output

Table III. 2 Tags for Data Collection

III.7.3 Grouping Flows into Categories

Once we've finished creating all the necessary flows, we'll group them into four categories to keep our workspace organized. To do this, we'll select the nodes we want to group, then right-click and choose the “Groups” option, followed by “Group selection”.

Groups can be customized with a border, background color, and optional label. To edit a group's properties, we can double-click on it or press Enter when the workspace has focus and the group is selected.

We'll create four groups: "Box Counter", "Buttons", "Pivot Arm", and "Belt Conveyor". Each group will have a distinct label and fill color to differentiate them effectively

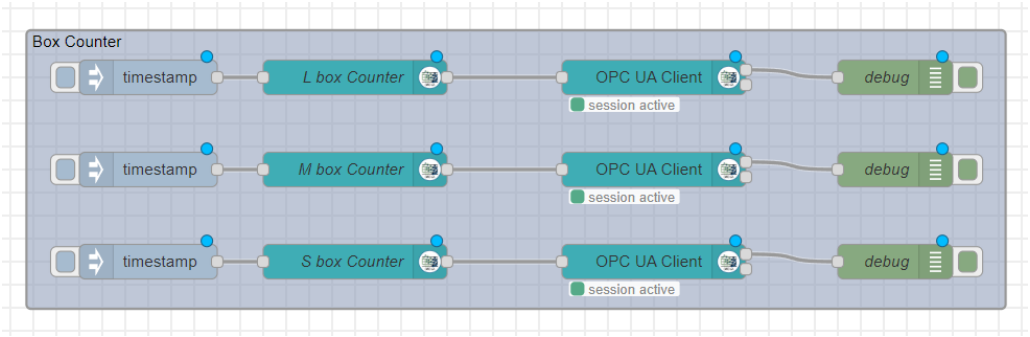


Figure III. 59 Grouping Flows in Node-RED: Box Counter Group

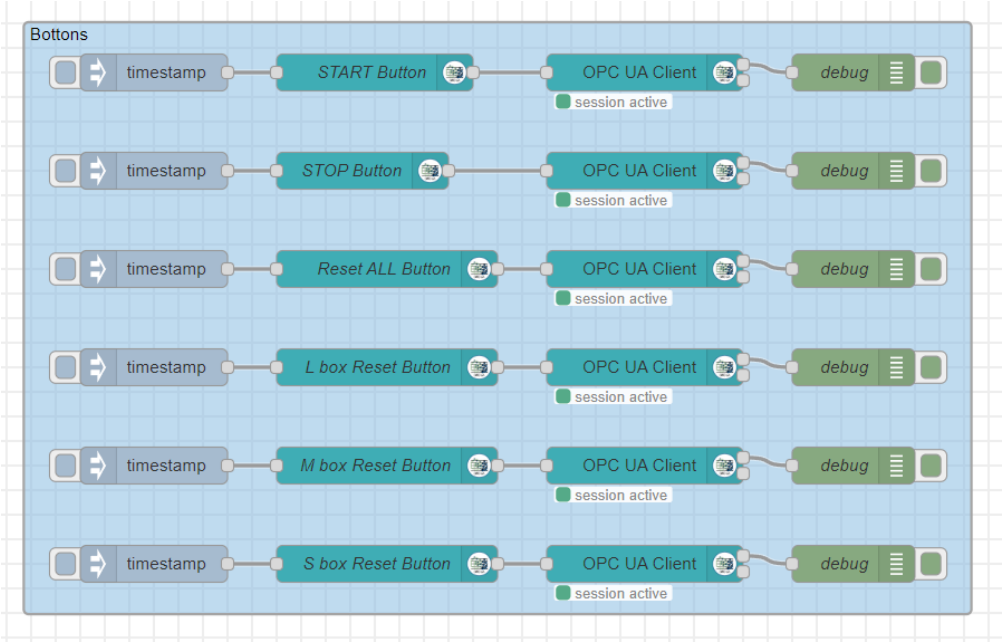


Figure III. 60 Grouping Flows in Node-RED: Buttons Group

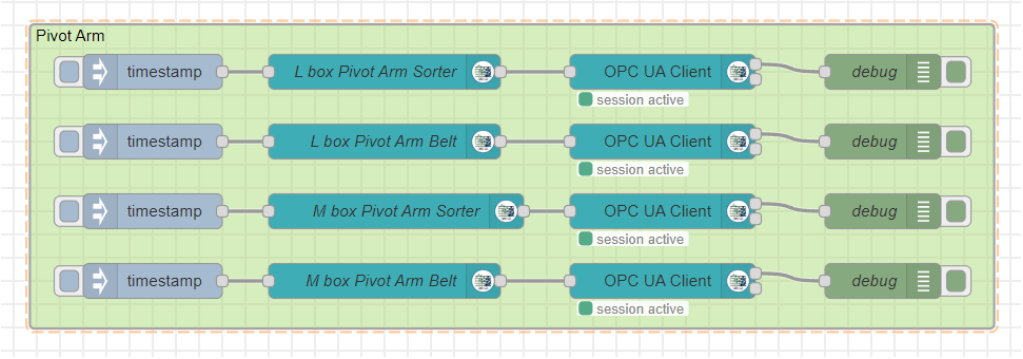


Figure III. 61 Grouping Flows in Node-RED: Pivot Arm Group

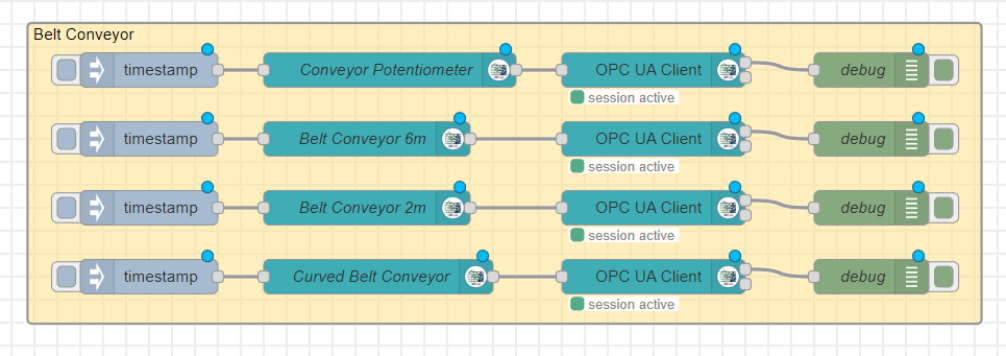


Figure III. 62 Grouping Flows in Node-RED: Belt Conveyor Group

III.8 Converting Values

III.8.1 Converting Boolean Values

We will create a script function that converts a boolean value into either 0 or 1. We will use this script for the pivot arm. First, we drag a "function" node into the flow. Then, we double-click the function node to edit it , we name it “Boolean Convert” and enter the following script.

```
if (typeof msg.payload === 'boolean') {
    msg.payload = msg.payload ? 1 : 0;
} else {
    node.error("Input is not a boolean", msg);
    return null;
}
return msg;
```

Script III. 1 Boolean Conversion Script

The script checks if “msg.payload” is a boolean.If it is a boolean, it converts true to “1” and false to “0”. And If “msg.payload” is not a boolean, it logs an error and stops processing the message by returning null.

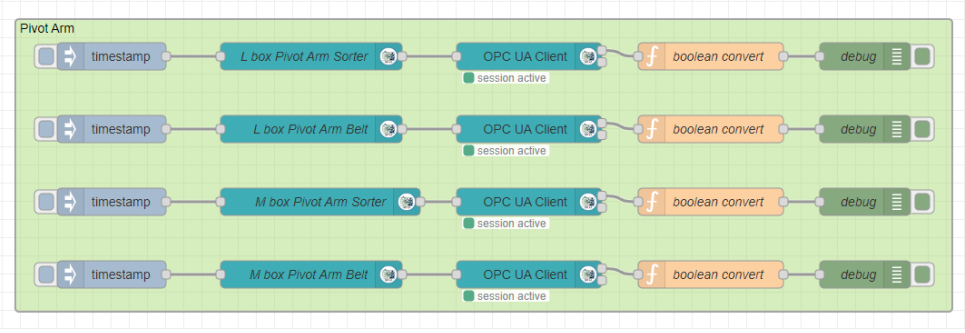


Figure III. 63 Boolean Script Placement in pivot Arm Group

We will also creat the same script for the button groups, but we will modify it to exclude the false value, so the script only returns the true value. So we will drag "function" node into the flow and paste the following script , and once we finish we name it “Boolean Convert (true only)”

```
if (typeof msg.payload === 'boolean') {
    if (msg.payload === true) {
        msg.payload = 1;
        return msg;
    } else {
        return null;
    }
} else {
    node.error("Input is not a boolean", msg);
    return null;
}
```

Script III. 2 Boolean Conversion Script (true only)

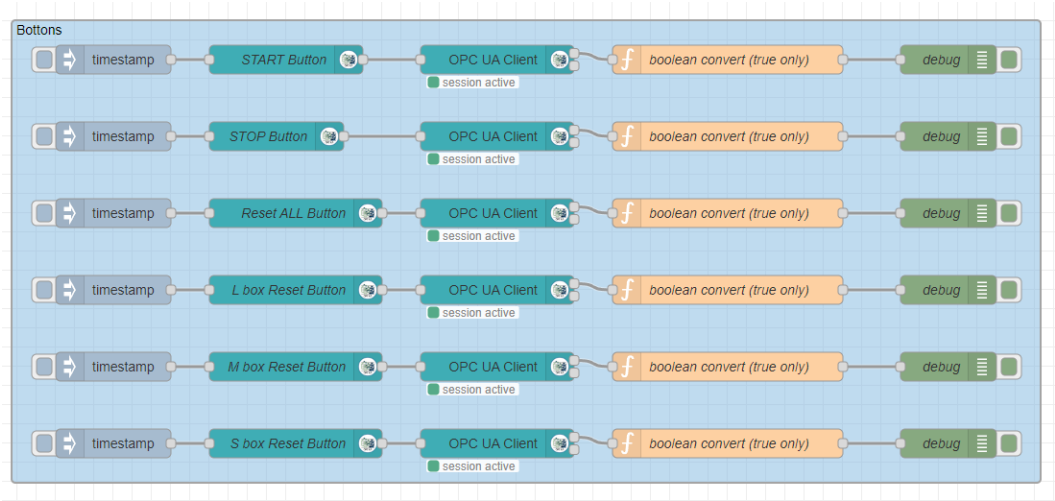


Figure III. 64 Boolean Script Placement in Buttons Group

III.8.2 Converting UInt16 Values

We will be creating a script to convert the value set by the Conveyor Potentiometer to control the speed of the belt conveyor. According to the Factory I/O documentation, the maximum speed of the conveyor, when using an analog value ranging from 0 to 10 volts, is 3 meters per second. Using this information, we will create a script that performs a linear scaling transformation.

First, we will drag a "function" node into the "belt conveyor" flow. we will then double-click the function node to open the editor and enter the following script:

```
// Read the input value
var inputMaxValue = msg.payload;
// Define the conversion factors
var maxValue = 27648;
var maxSpeed = 3;
var maxVoltage = 10.0;
// Perform the conversion
var speed = ((inputMaxValue / maxValue) * maxSpeed).toFixed(2);
var voltage = ((inputMaxValue / maxValue) * maxVoltage).toFixed(2);
// Output the converted values with the timestamp
msg.payload = {
    speed: parseFloat(speed),
    voltage: parseFloat(voltage)
};
return msg;
```

Script III. 3 UInt16 Conversion Script

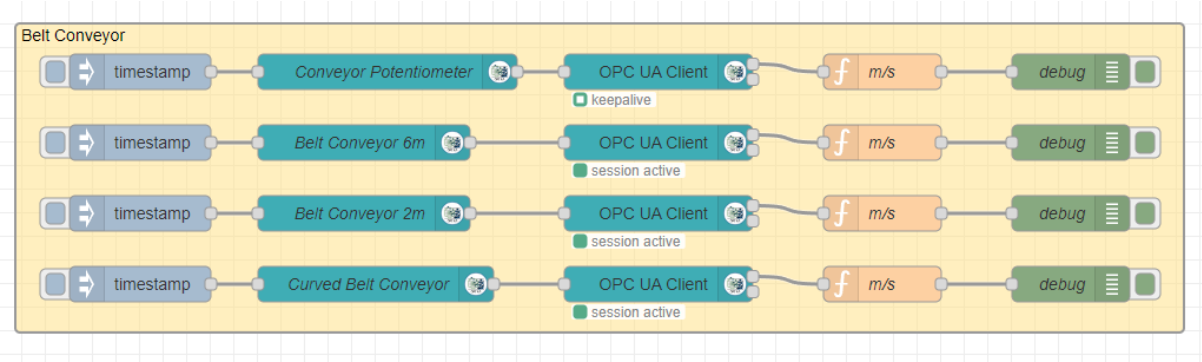


Figure III. 65 UInt16 Conversion Script in Belt Conveyor Group

III.9 Sending Data to InfluxDB

We will begin sending the data retrieved from the OPC UA server to the InfluxDB database. Using the palette, we'll drag and drop the 'influxdb out' node, then proceed to configure the server settings.

III.9.1 InfluxDB Server Configuration

1. Double-click on the “influxdb out” node to open its properties.
2. Next to the "Server" dropdown, click on the pencil icon to configure a new server.
3. In the "Version" dropdown, we select "2.0".
4. In the "URL" field, we enter the IP address and port number of InfluxDB OSS instance
5. In the "Token" field, we paste the API token that we generated when setting up InfluxDB OSS.
6. Finally, we Click "Add" to save the server configuration.

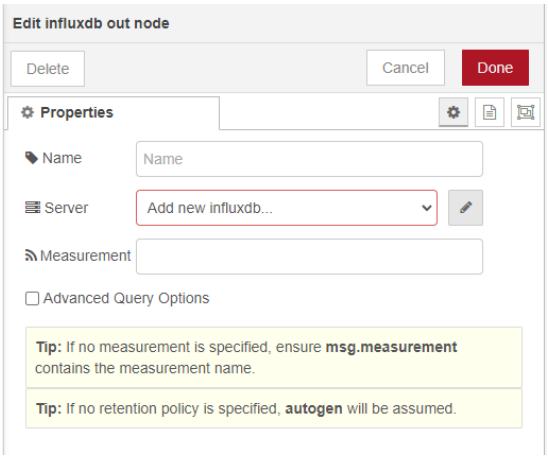


Figure III. 66 InfluxDB Server Configuration in Node-RED

now everytime we drag a new "influxdb out" node into the workspace, the server settings will already be configured, so we won't need to re-enter the server parameters.

III.9.2 Creating InfluxDB Buckets

Afterward, we navigate to the InfluxDB server and log in using the server credentials Then, from the left-side panel, we go to “Buckets,” and click on “CREATE BUCKET.” We will create four buckets named "Box Counter," "Buttons," "Pivot Arm," and "Belt Conveyor," setting “Delete Data” to "never."

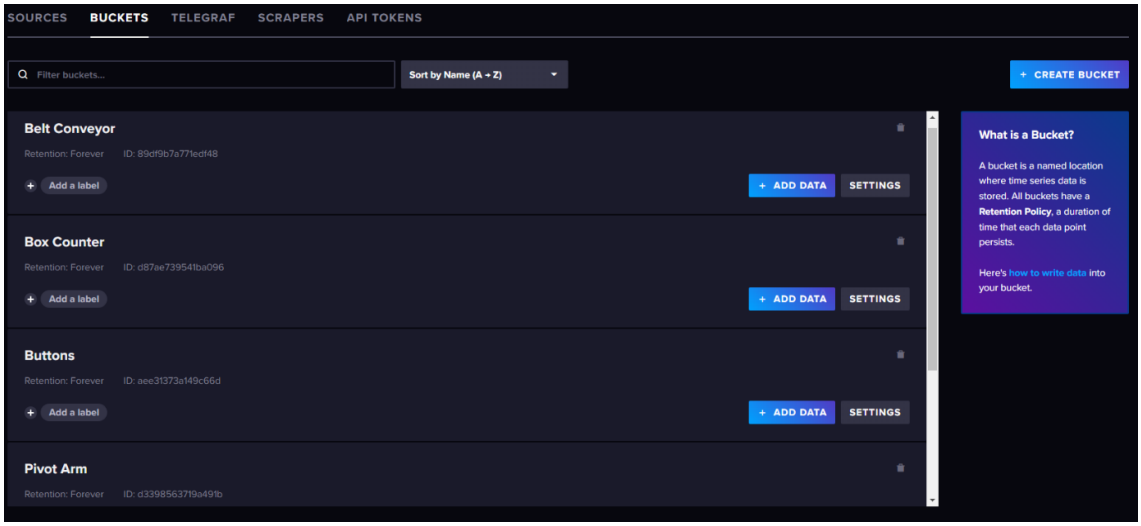


Figure III. 67 Creating InfluxDB Buckets

Next, in the Node-RED workspace, we will create four copies of the "influxdb out" node. For each node, we will enter the corresponding bucket name in the "bucket" label option and the organization name in the "organization" option, and we must ensure they match exactly as they are case-sensitive. Finally, we will specify the measurement name for each flow where the data will be written. According to the table below, we will have a total of 17 measurements, resulting in 17 nodes

BUCKET	**measurement**
Belt Conveyor	Conveyor Potentiometer
	Belt Conveyor 6m
	Belt Conveyor 2m
	Curved Belt Conveyor
Box Counter	L box Counter
	M box Counter
	S box Counter
Bottons	START Button
	STOP Button
	Reset ALL Button
	L box Reset Button
	M box Reset Button
	S box Reset Button
Pivot Arm	L box Pivot Arm Sorter
	L box Pivot Arm Belt
	M box Pivot Arm Sorter
	M box Pivot Arm Belt

Table III. 3 InfluxDB Buckets and Measurements

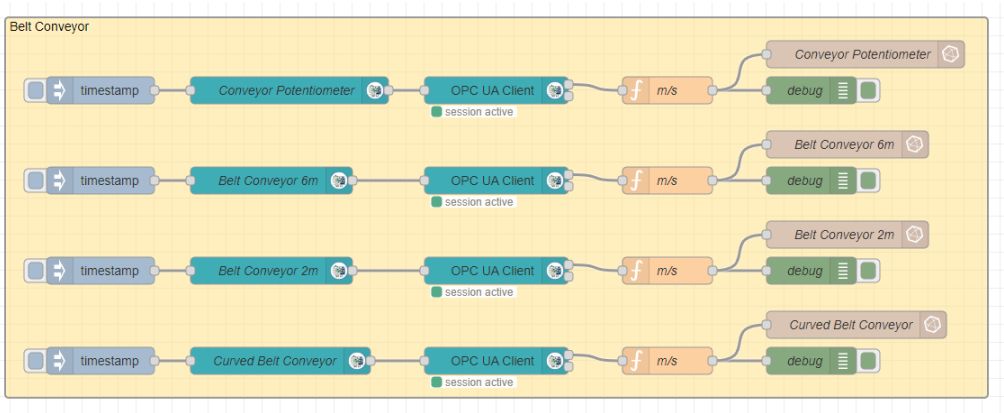


Figure III. 68 Creating InfluxDB Buckets

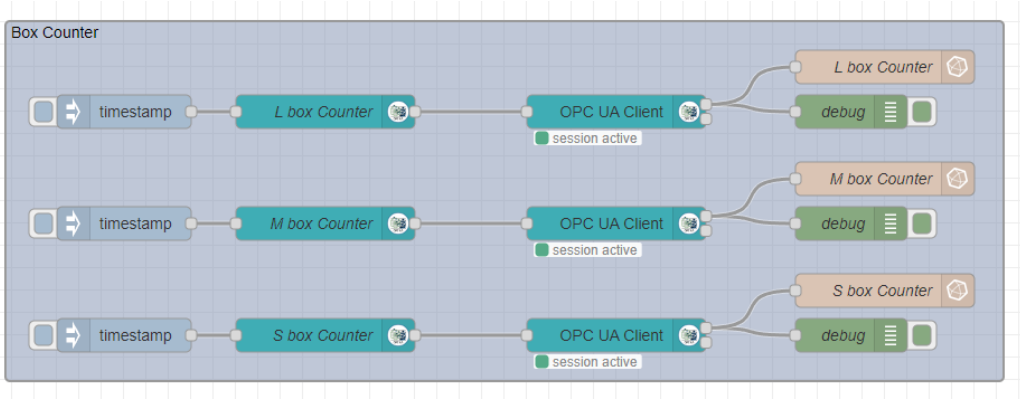


Figure III. 69 InfluxDB Output Nodes: Box Counter

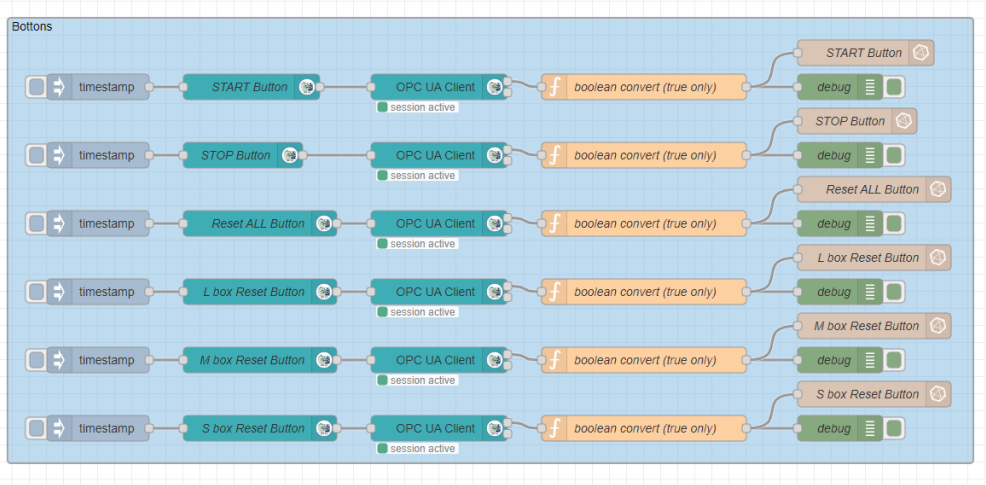


Figure III. 70 InfluxDB Output Nodes: Buttons

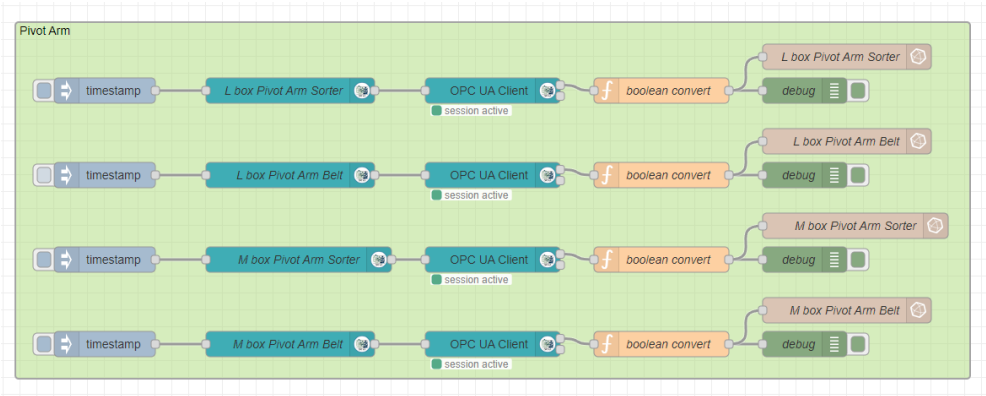


Figure III. 71 InfluxDB Output Nodes: Pivot Arm

III.9.3 Deploy and Test

Now that we have finished preparing all the flows to read and send data, the next step is to deploy and test to ensure all measurements are appropriately registered correctly in InfluxDB.

To begin, we deploy the flow by clicking the Deploy button located at the upper right side. Then, we navigate to the Factory IO scene and from the Switchboard. We choose a random

value using the Potentiometer, for example, selecting "1.15". Next, we head to the corresponding flow that reads the value from the Potentiometer and click the inject node button to trigger the flow. In the debug panel, we observe a message showing the speed as 0.35 m/s and the voltage as 1.15

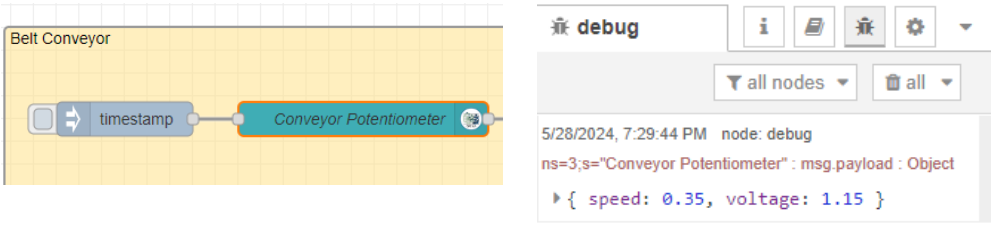


Figure III. 72 InfluxDB Output Nodes: Belt Conveyor

Now, if we navigate to the InfluxDB server and select the bucket named 'Belt Conveyor,' we notice a new measurement labeled 'Conveyor Potentiometer.' Upon selecting this measurement, we find two field data entries: 'speed' and 'voltage.' We have the option to view the speed and voltage data independently or combined, depending on our specific needs. By selecting one or both of the fields and toggling the 'View Raw Data' slider, and specifying the time range (by default, it's set to 1 hour, meaning we will only receive data from the last hour), we can then click on the submit button. After that, we observe a table showing up under Data Explorer with the measurement names 'Conveyor Potentiometer' and a values of 0.35 for speed and 1.15 for voltage."

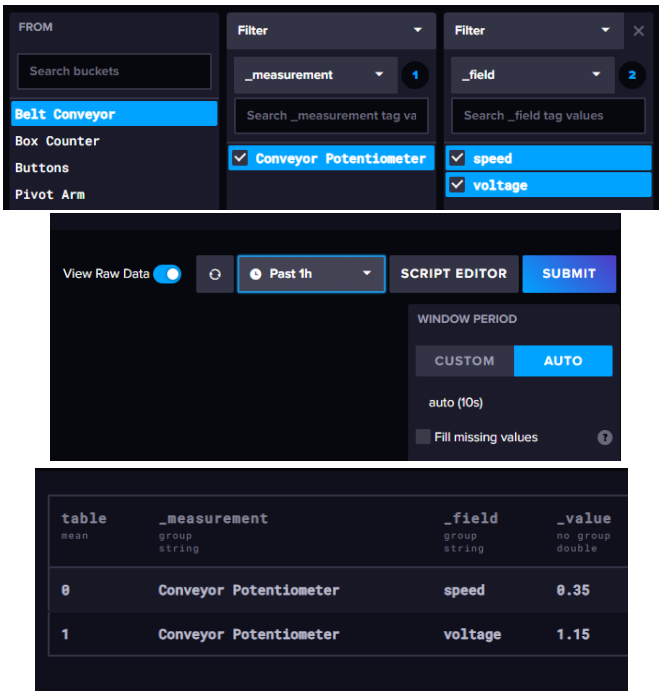


Figure III. 73 Testing Data in InfluxDB: Conveyor Potentiometer Measurement

Moving forward, we repeat the same steps for the remaining set of 'Belt Conveyor' flows, that includes all three belt conveyors. We expect getting a value of 0 for both voltage and speed since we haven't yet clicked the "START" button, leaving the conveyors in a stationary state. Upon triggering 'inject' for the three flows, we confirm the expected 0 value in the debug panel. Moving to InfluxDB and within the 'Belt Conveyor' bucket, we see the appearing of three new measurements: 'Belt Conveyor 6m,' 'Belt Conveyor 2m,' and 'Curved Belt Conveyor.' Afterwards, we proceed to verify their values by selecting each one and clicking on submite boutten, and in the Data Explorer Table, we can see that the measurement has the correct value.

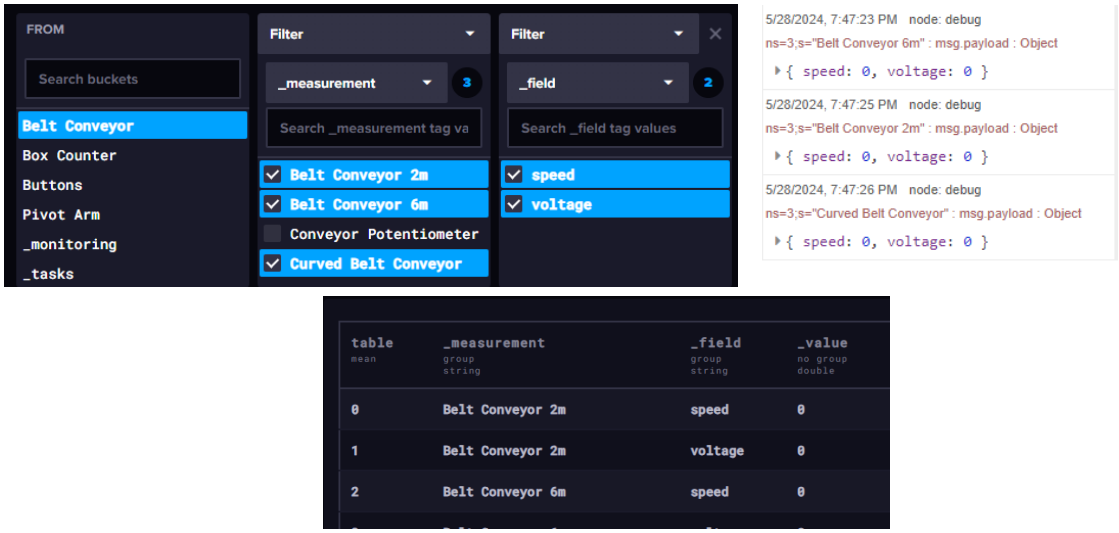


Figure III. 74 Testing Data in InfluxDB: Belt Conveyor Measurements

Now we click on the 'START' button in Factory IO, the conveyor begins to roll. When we return to Node-RED and inject the node once more, we see new values in the debug panel: speed = 0.35 and voltage = 1.15. After returning to InfluxDB and selecting all three measurements, we click submit again. In the table, we notice the new values for speed and voltage alongside the old values. This indicates that our setup for InfluxDB is functioning correctly without any issues.

III.10 Automating Data Sending

To automate the data sending process without manually clicking inject every time, we can edit the properties of the inject node. Inside the inject node, we change the repeat option from "none" to "interval." Since we're simulating, we will choose to send data every 1 second. This

frequency can be adjusted depending on the requirements or the desired precision of the application, such as adjusting it to 500ms or 200ms

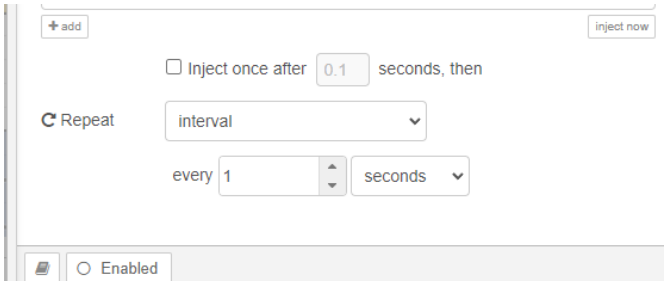


Figure III. 75 Automating Data Sending in Node-RED: Inject Node Configuration

III.10.1 Optimizing Automated Data Sending

Currently, we have configured the data to be sent every second, which results in sending the same value repeatedly. This causes a lot of redundant data events. In some cases, we want to avoid sending the same value multiple times. For example, if the potentiometer value is "1.15" and it is recorded every second, we need to modify the flow so that the value is only sent when it changes,and we can achieve this in two ways:

→ By changing the action in the OPC UA client node from "Read" to "Subscribe". The "Read" action polls the value every X seconds, while the "Subscribe" action provides updates only when a change occurs.

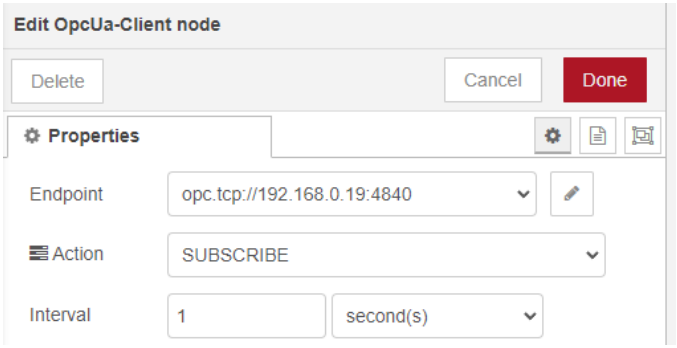


Figure III. 76 Optimizing Data Sending in Node-RED: Subscribe Action

→ By creating a function script and inserting it into the flow. This approach is useful when the OPC UA Server doesn't support the action “"Subscribe" and only the "Read" action is available.

```
// Initialize the previous value variable as a string representation of the
object
context.prevValue = context.prevValue === undefined ? null :
context.prevValue;

// Convert the current payload object to a JSON string
let currentPayloadString = JSON.stringify(msg.payload);

// Check if the incoming value as a string is different from the previous
value string
if (context.prevValue === null || currentPayloadString !== context.prevValue)
{
    // Update the previous value with the current payload string
    context.prevValue = currentPayloadString;
    // Pass the message to the next node only if the value has changed or if
it's the first time
    return msg;
}
```

Script III. 4 Optimizing Data Sending

This code initializes a context variable prevValue to store the previous payload value as a JSON string, defaulting to null if it hasn't been set before. It then converts the current msg.payload object to a JSON string (currentPayloadString). If this is the first run (i.e., prevValue is null) or if the current payload string differs from the previous one, it updates prevValue with the current payload string and returns the msg object, allowing it to be passed to the next node. This ensures that only messages with changed payloads are forwarded, effectively filtering out duplicates or unchanged messages

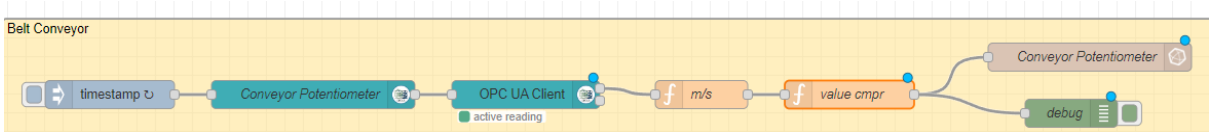


Figure III. 77 Optimizing Data Sending in Node-RED with script

For this project, we will change the OPC UA client action from "Read" to "Subscribe" for all the groups in the flow. After making these changes, we will click on deploy. With that, we have successfully completed the process of saving data into the InfluxDB database

III.11 Grafana

III.11.1 Adding a Data Source

Now we have finished the process of collecting data from the system we created in Factory IO and saved it into the database. Next, we will move on to Grafana for monitoring and visualizing the data in real-time, as well as querying the data.

We begin by opening an internet browser and typing in the server IP address where Grafana is installed followed by the port number “3000” . Then, we log in with the username and password we created

Before we can start creating our first dashboard, we must first add a data source to Grafana. To do this, we click on the "open menu" icon located at the top left corner. From there, we navigate to "data source" and click on "add data source." and from the list of Time series databases, we choose "InfluxDB."

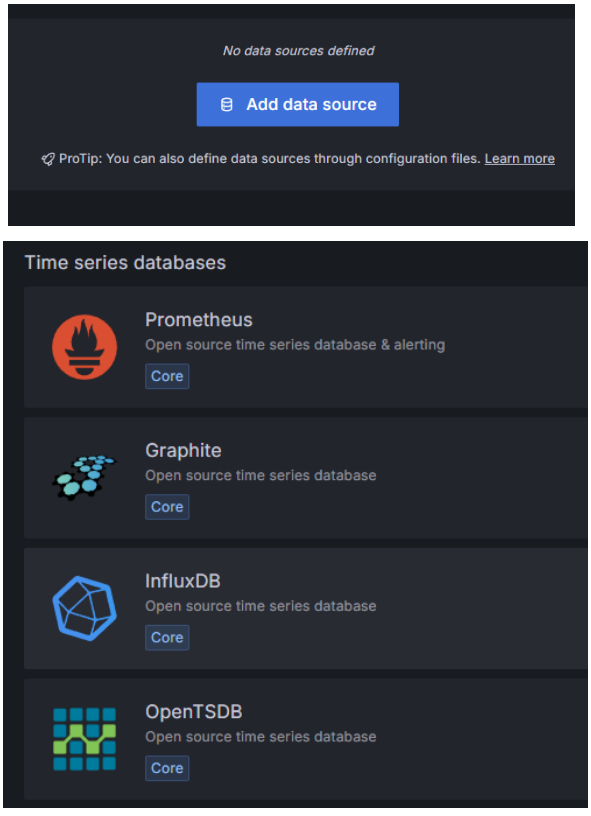


Figure III. 78 Adding a Data Source in Grafana

In the Query Language drop-down menu, we select “Flux”. Then we configure the InfluxDB connection. Under HTTP, in the URL label, we enter the IP address and port of the InfluxDB server instance.

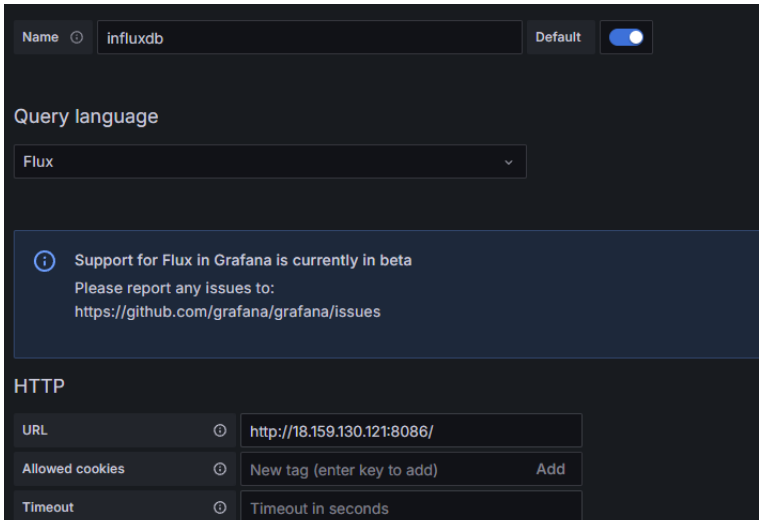


Figure III. 79 InfluxDB Connection Configuration in Grafana: URL Settings

we scroll down to the InfluxDB details section. Here, we insert the Organization name and the API Token. Once completed, we click on “Save & test.” If everything was configured correctly, we should see a green check mark indicating that the datasource is working, along with the number of buckets that were found.

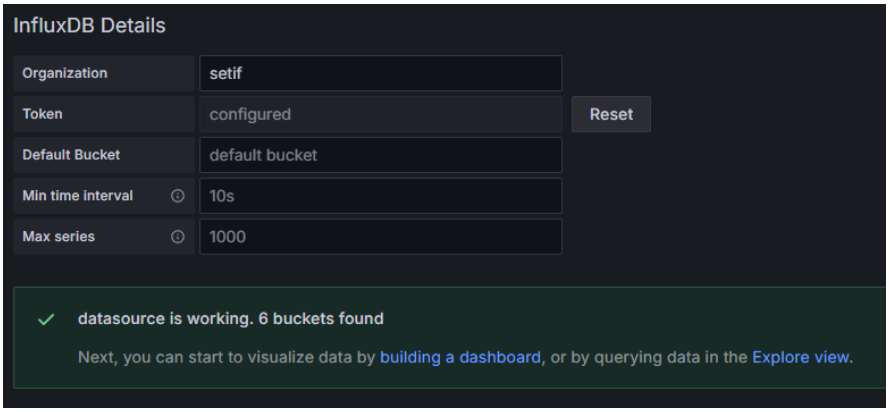


Figure III. 80 InfluxDB Connection Configuration in Grafana: API and Organization

III.11.2 Creating a Dashboard

To visualize our system in real-time, we can create dashboards. Dashboards allow us to present data in a visual format. To create a dashboard, first click 'Dashboards' in the left-side menu, then click 'New' and select 'New Dashboard.' On the empty dashboard, click '+ Add Visualization.'

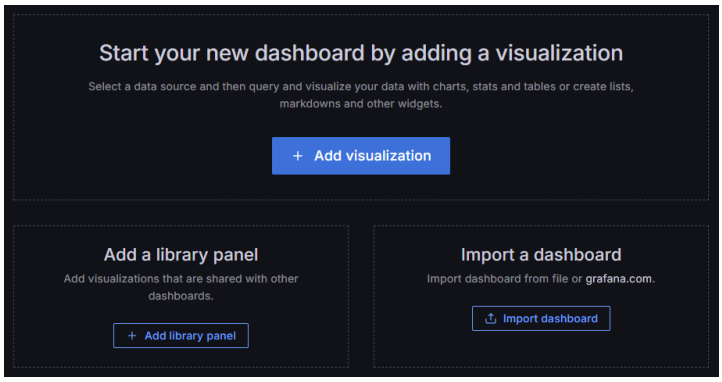


Figure III. 81 Creating a Dashboard in Grafana

In the dialog box that opens, we select the existing data source we created earlier. This action opens the Edit panel view.

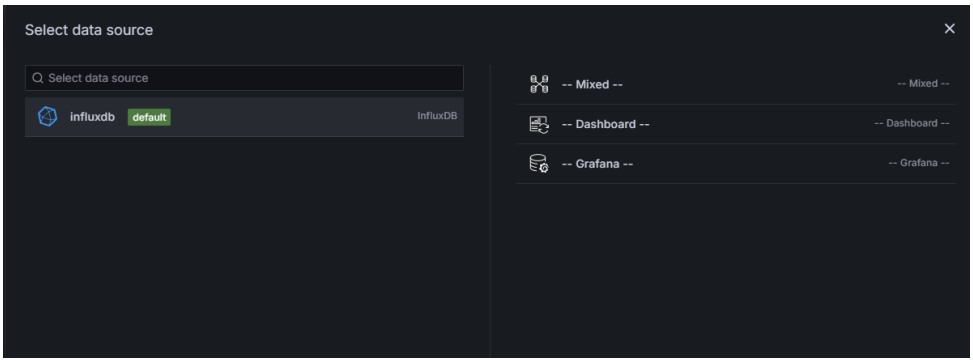


Figure III. 82 Creating a Dashboard in Grafana: Adding data source

III.11.3 Creating the First Panel

In Edit panel view, To visualize one of the measurements, we need to write an InfluxQL query in the query language window

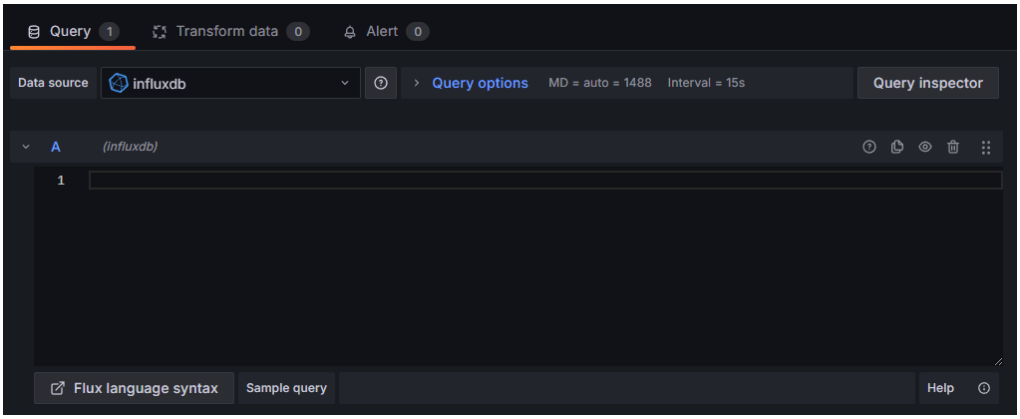


Figure III. 83 Creating a Dashboard in Grafana: Adding Visualization

We will start with 'Conveyor Potentiometer.' We can obtain the query directly by selecting the measurement we want to visualize, then change the aggregation function from 'mean' to 'last,' as 'last' returns the field value with the most recent timestamp, while 'mean' returns the

average of non-null values. Next, we click on the 'Script Editor' button, which automatically generates a query based on our selection. We copy the generated query and paste it into the query language window.

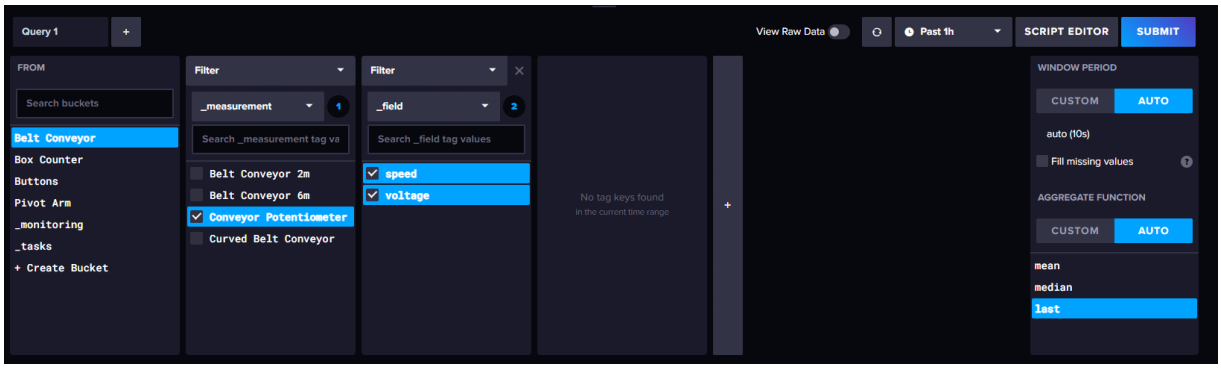


Figure III. 84 influxQL Query for Generating

```
from(bucket: "Belt Conveyor")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_measurement"] == "Conveyor Potentiometer")
  |> filter(fn: (r) => r["_field"] == "speed" or r["_field"] == "voltage")
  |> aggregateWindow(every: v.windowPeriod, fn: last, createEmpty: false)
  |> yield(name: "last")
```

Script III. 5 influxQL Query Language for Visualization

In the visualization list, we can select a visualization type. we have a list of different visualization options available. Let's change the current type from "Time Series" (which shows data points over time) to "Stat" (which displays a single value).

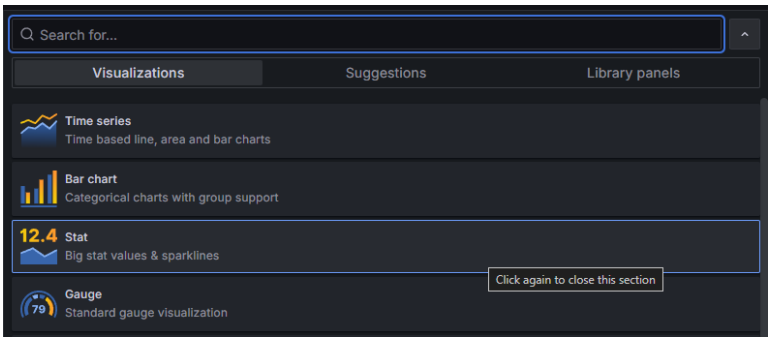


Figure III. 85 Visualization Options in Grafana: Stat Panel

Let's customize the panel to display the information clearly. First, we'll change the title at the top to "Conveyor Potentiometer" to identify the data being shown.

Next, we need to adjust the display style. Scroll down to the "Stat Styles" section. Since we're using the "Stat" visualization, which displays a single value, we can remove any

unnecessary graphs by changing the "Graph mode" to "None." Finally, for better readability, let's change the text alignment from "Auto" to "Center" to ensure the value is displayed in the middle of the panel.

Next, we Scroll down to the "Thresholds" section and We'll delete the default threshold (usually set to 80). This prevents the value from changing color based on an arbitrary limit, keeping it consistently visible

To specify the unit of a value, navigate to 'Standard Options' and enter the desired unit in the 'Unit' field. Since we need to set different units for speed and voltage, using the 'Unit' field directly would apply the same unit to both.

we instead we select “Overrides” and click on “Add Field Override” We then select “Fields with Name” and choose the field we want to edit, in this case “speed”, We click “Add Override Property”, and we can add as many override properties as needed. we will choose “Standard Options > Unit”, then we scroll down to “Velocity” and choose “m/s”, or we type “m/s” directly. If “m/s” is available, it indicates the unit already exists, if not, we can click on “custom unit”, which will add a new unit if it doesn't already exist.

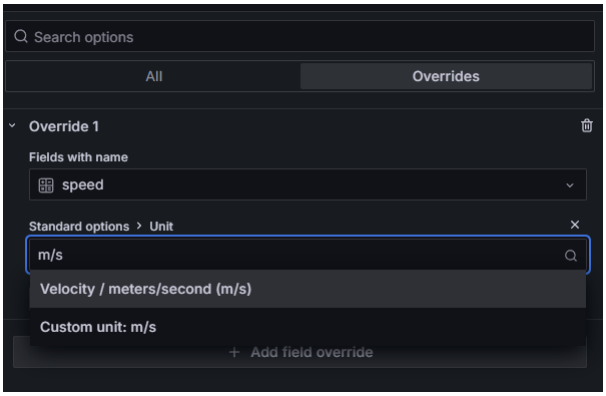


Figure III. 86 Panel Customization in Grafana

To change the color, we add another override property by selecting “Standard Options > Color Scheme” and choose any color we want.

And if we want to change the name, we can do that by adding the override property “Standard Options > Display Name.”

Once we finish editing the parameters of the field 'speed,' we repeat the same process for 'voltage

After finalizing the parameters, we click on 'Apply.' This creates our first panel displaying the value of the potentiometer. We can resize the panel and position it wherever we like

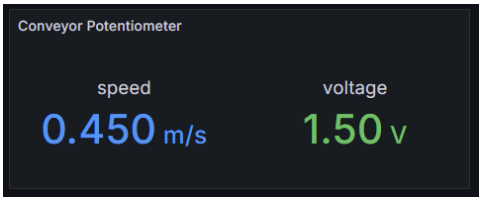


Figure III. 87 Conveyor Potentiometer Panel

To complete the dashboard, we incorporate additional panels for comprehensive monitoring. We will add a Bar Gauge to display the speed of each of the three conveyors, the control buttons are combined into a single table featuring a drop-down menu. This allows users to select the desired button and view the historical click times for each button

Additionally, we will add a panel to display the number of boxes that have successfully reached their destination, Furthermore Another panel provides real-time status updates for the pivot arm output for both L and M boxes. To enhance visual clarity, we utilize Value Mappings within the Override Property to map the value '1' to 'ON' displayed in green, and the value '0' to 'OFF' displayed in red."

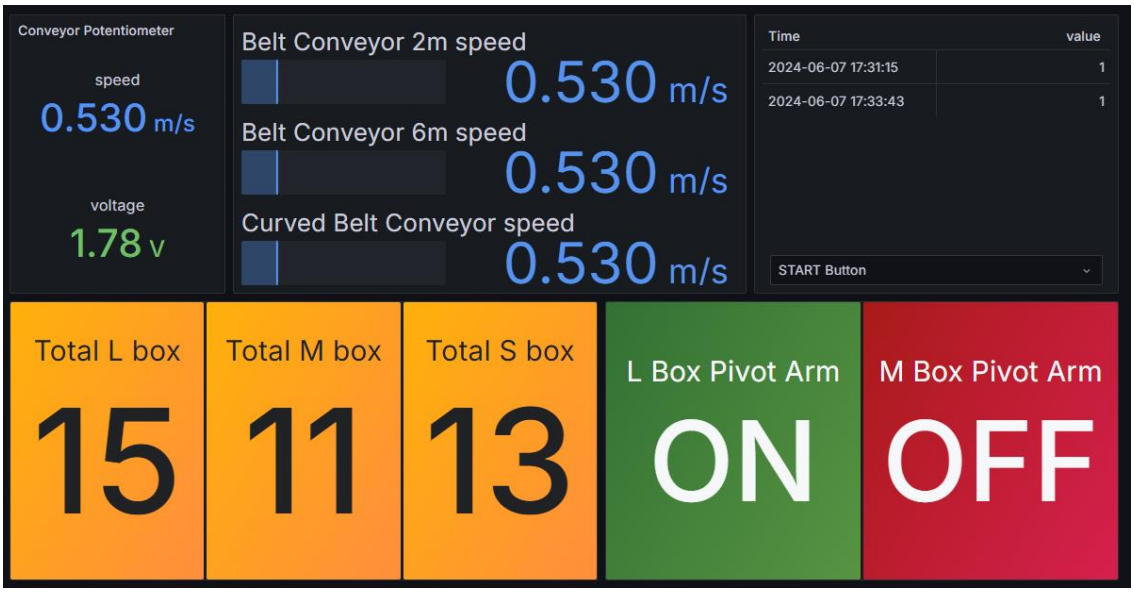


Figure III. 88 initial dashboard of the system

And with this we have successfully created a comprehensive dashboard that effectively represents our input and output values, this initial dashboard serves as a foundation for further customization and expansion as needed

III.12 Creating Feedback for the Simulation

To refine the simulation, we will introduce additional sensors into the scene to obtain feedback from the pivot arm and maintain continuous tracking of the boxes. By incorporating

pivot arm failure scenarios, we will generate alert messages that pinpoint the exact location where a box is diverted from its intended destination due to that malfunction. Moreover, we will provide clear status messages regarding the operational condition of the pivot arm, indicating whether it's functioning correctly or experiencing a failure.

III.12.1 Adding Sensors for Feedback

Within the Factory I/O scene, we'll integrate two capacitive sensors: one at the end of each pivot arm. These sensors will provide valuable feedback on the pivot arms' operation. Later, we'll develop a script in node-red to compare this feedback with the pivot arms' output signals, allowing us to determine their operational status accurately.

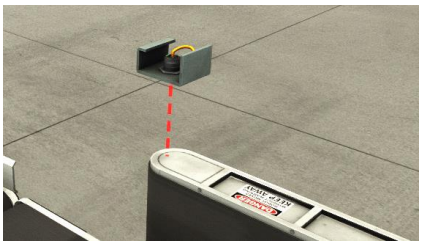


Figure III. 89 Feedback Sensors: Capacitive Sensors on Pivot Arms

Next, we'll position a light array sensor next to the existing diffuse sensor responsible for sorting M boxes. This addition enables comprehensive tracking of L boxes. If an L box deviates from its intended path, we can track whether it's headed towards the M box slot or the S box slot. Similarly, we can monitor M boxes for potential diversions to the S box slot and S boxes for diversions to the M box slot.

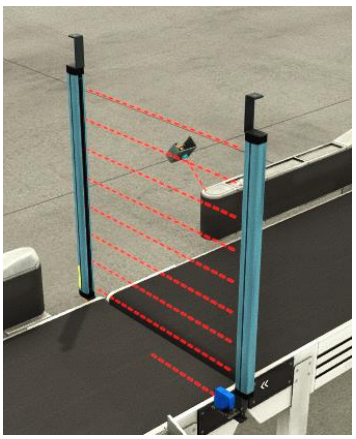


Figure III. 90 Light Array Sensor for L Box Tracking

Finally, we'll install an additional diffuse sensor next to the first light array sensor, which is responsible for sorting L boxes. This new sensor will help us detect if an S or M box mistakenly enters the L box slot

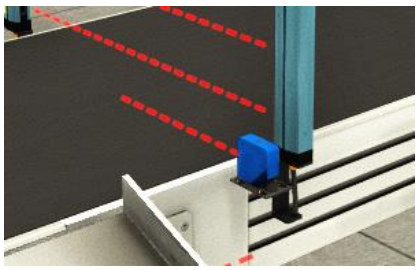


Figure III. 91 Diffuse Sensor For box width feedback

III.12.2 Enabling the Incremental Encoder

To obtain feedback from the conveyor belt, we can utilize Factory I/O's built-in incremental encoder functionality. This encoder can be enabled by right-clicking on the conveyor belt part and selecting "Use Encoder" from the context menu. The encoder generates two wave forms, A and B, which are 90 degrees out of phase (in quadrature) and provide information about the conveyor's movement. With a resolution of 0.0225 meters, the encoder allows us to precisely track the position and speed of the conveyor belt

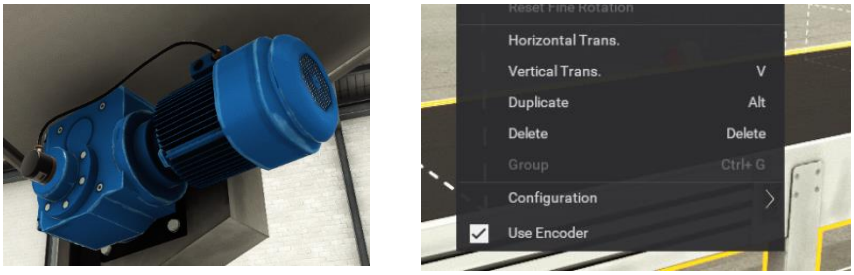


Figure III. 92 Incremental Encoder

Next, in the TIA Portal project, we create a new tag table and name it 'Feedback Tag Table.' We then add our new tag

	Name	Data type	Address
1	L Pivot Arm Sensor OFF	Bool	%I1.2
2	M Pivot Arm Sensor OFF	Bool	%I1.3
3	Diffuse Sensor 2	Bool	%I1.4
4	Encoder Signal A (6m)	Bool	%I1.5
5	Encoder Signal A (2m)	Bool	%I1.6
6	Encoder Signal A (Curved)	Bool	%I1.7
7	Encoder Signal B (6m)	Bool	%I2.0
8	Encoder Signal B (2m)	Bool	%I2.1
9	Encoder Signal B (Curved)	Bool	%I2.2
10	Light Array Emitter 2	Int	%IW104

Figure III. 93 Feedback Table Tag

III.12.3 Creating a Feedback Flow for the Pivot Arm

To maintain an organized workspace in Node-RED, we'll create a new tab flow named "Feedback" by clicking the plus icon in the top right corner. Within this flow, we'll develop

logic to generate alert messages when the pivot arms behave incorrectly. This involves comparing the actual position of each pivot arm, derived from sensor feedback, with its desired position, as indicated by the signal output value.

We'll begin by reading the two values from the OPC UA server, just as we did previously, only this time we will use action “Read” instead of “Subscribe”

Next, we'll invert the value of the sensor feedback to ensure it aligns with the signal output value for comparison, then we'll utilize a function script to convert Boolean values to 0 or 1, after that, we'll combine both values into an array

```
var inputBoolean = msg.payload;
// Check if the input is a boolean
if (typeof inputBoolean === 'boolean') {
// Invert the boolean value
msg.payload = !inputBoolean;
} else {
// If the input is not a boolean, return an error message
msg.payload = "Error: Input is not a boolean";
}
// Return the modified message object
return msg;
```

Script III. 6 Boolean Conversion and Invert Script

Now, we introduce another function script designed to detect any discrepancies between the values in the array. If a mismatch is found, the script will output an appropriate message.

```
var inputArray = msg.payload;
// Check if the input is an array with exactly two elements
if (Array.isArray(inputArray) && inputArray.length === 2) {
    // Compare the values of the array
    if (inputArray[0] === 0 && inputArray[1] === 0) {
        msg.payload = "Correct";
    } else if (inputArray[0] === 1 && inputArray[1] === 1) {
        msg.payload = "Correct";
    } else if (inputArray[0] === 0 && inputArray[1] === 1) {
        msg.payload = "invalide,L Pivot Arm is ON where it is supposed to
be OFF";
    } else if (inputArray[0] === 1 && inputArray[1] === 0) {
        msg.payload = "invalide,L Pivot Arm is OFF where it is supposed
to be ON";
    } else {
        msg.payload = "Error: Unexpected input values"; // Optional,
handle unexpected cases
    }
} else {
    msg.payload = "Error: Input is not a valid array of two elements";
}
// Return the modified message object
return msg;
```

Script III. 7 Pivot Arm Feedback Script

The primary goal of this script is to validate and interpret the two-element array:

→ [0, 0] or [1, 1]: The script sets msg.payload to "Correct."

→ [0, 1]: The script sets msg.payload to "Invalid, L Pivot Arm is ON where it is supposed to be OFF."

→ [1, 0]: The script sets msg.payload to "Invalid, L Pivot Arm is OFF where it is supposed to be ON."

Subsequently, we'll transmit this message to InfluxDB, storing it in a new bucket named "Pivot Arm Monitoring" under the measurement name "L Pivot Arm Monitoring." and We'll repeat this approach for the M pivot arm also

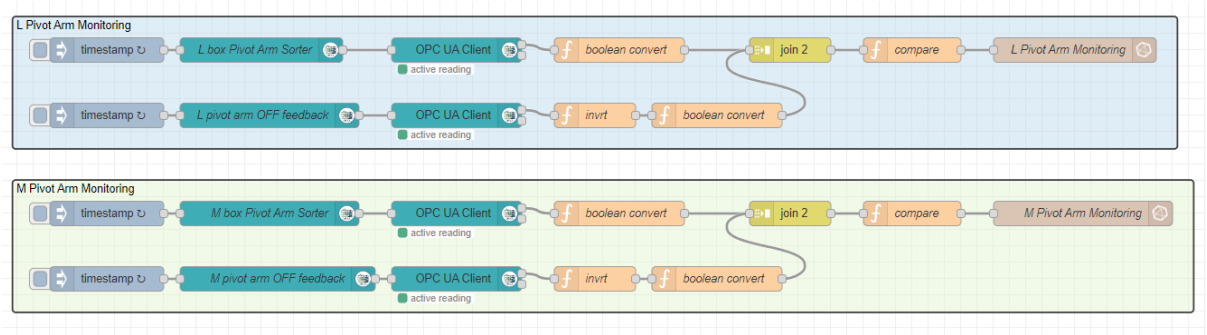


Figure III. 94 Feedback Flow for Pivot Arm in Node-RED

III.12.4 Tracking Lost L Boxes

We'll leverage the feedback from the L and M pivot arms to track lost boxes that fail to reach their designated destinations. Let's start with the L box scenario.

In the event that an L box misses its target, it has two possible destinations: the M slot or the S slot. To discern its path, we'll employ the second light array sensor installed near the M pivot arm, in conjunction with diffuse sensor 1 (responsible for M box sorting) and the M pivot arm's feedback sensor. We'll combine the values from these three sensors into a 3-element array.

After joining the values, we'll introduce a script designed to filter incoming arrays. This script will only allow an array to pass through if it differs from the previously seen array. This is essential because we're using "read" instead of "subscribe."

```
// Initialize the previous value variable
context.prevArray = context.prevArray || [];

// Check if the incoming array is different from the previous array
if (
    context.prevArray.length !== 3 ||
    JSON.stringify(msg.payload) !== JSON.stringify(context.prevArray)
) {
    // Update the previous array
    context.prevArray = msg.payload.slice(0, 3);
    // Pass the message to the next node only if the array has changed
or if it's the first time
    return msg;
}
```

Script III. 8 Array Filtering Script

When an L box crosses the light array sensor, it returns a value of [224, 1, 0]. The first element of our array holds the value from the light array sensor, the second element holds the feedback value from the M pivot arm, and the third element holds the value of diffuse sensor 1. As the L box passes diffuse sensor 1, the array becomes [224, 1, 1]. This array is always present, but the next array will have one of two possible values

[224, 0, 1]: The second element becomes 0, representing the feedback from the pivot arm, indicating that it has sorted the box. This signifies that the L box has gone to the M slot.

Figure III. 95 Tracking Lost L Boxes: L Box in M Slot

[0, 1, 1]: The L box has crossed the entire light array sensor and diffuse sensor 1 is on, but the pivot arm feedback remains 1, indicating that it hasn't sorted. This means the L box is heading to the S slot.

Figure III. 96 Tracking Lost L Boxes: L Box in S Slot

Based on this logic, we'll create a script that processes incoming arrays in sequence. It will react to specific array patterns and generate corresponding outputs based on its internal state

```
// Define the state variables
context.state = context.state || 0;

// Retrieve the input array from msg.payload
var inputArray = msg.payload;

// State machine logic
if (context.state === 0 && inputArray.length === 3 && inputArray[0] === 224 &&
inputArray[1] === 1 && inputArray[2] === 1) {
    // Transition to state 1 when the array is [224,1,1]
    context.state = 1;
    return null; // Wait for the next input
} else if (context.state === 1) {
    // Handle the next input based on the state
    if (inputArray.length === 3 && inputArray[0] === 224 && inputArray[1] === 0
&& inputArray[2] === 1) {
        // Set the output for [224,0,1]
        msg.payload = "L box in M slot";
        context.state = 0; // Reset state
        return msg;
    } else if (inputArray.length === 3 && inputArray[0] === 0 && inputArray[1]
=== 1 && inputArray[2] === 1) {
        // Set the output for [0,1,1]
        msg.payload = "L box in S slot";
        context.state = 0; // Reset state
        return msg;
    } else {
        // If the input doesn't match expected patterns, reset state
        context.state = 0;
        return null;
    }
} else {
    // Reset state if input doesn't match expected patterns
    context.state = 0;
    return null;
}
```

Script III. 9 L Box Tracking Script

State 0

- The script awaits the input array [224, 1, 1].
- Upon receiving this array, it transitions to State 1 and returns null to await the next input.
- Any other input leaves the state at 0, and the script returns null, discarding the input.

State 1:

- The script checks for the arrays [224, 0, 1] and [0, 1, 1].
- If [224, 0, 1] is received, it sets (msg.payload) to "L box in M slot," resets the state to 0, and returns the modified msg object.
- If [0, 1, 1] is received, it sets (msg.payload) to "L box in S slot," resets the state to 0, and returns the modified msg object.
- Any other input resets the state to 0, and the script returns null.

Next, since the output message from the script can contain two different messages, we'll utilize a switch node to split them. This will allow us to route each message to its corresponding measurement in InfluxDB.

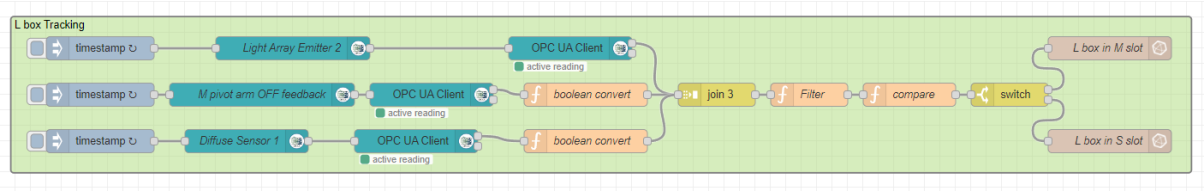


Figure III. 97 Switch Node for L Box Messages in Node-RED

III.12.5 Tracking Lost M and S Boxes

Next, to track M and S boxes, we'll adopt a similar approach with the L box. However, since both M and S boxes can end up in the L slot, we'll utilize the first light array sensor and pivot arm feedback. As M and S boxes have the same height, the light array sensor alone is insufficient for differentiation. Therefore, we'll incorporate the second diffuse sensor to detect M boxes due to their distinct width. By combining these sensor values into an array, we can create a script to determine the box type.

For an M or S box to reach the L slot, the pivot arm must be in the ON position. Consequently, we'll monitor for the array to become [192, 0, 0]. If the subsequent array is [192, 0, 1], the second diffuse sensor has been triggered by the M box, indicating that it's in the L slot. However, if the array changes to [0, 0, 0], it means the box has passed through the light array sensor without activating the second diffuse sensor, implying that an S box is in the L slot.

In the scenario where an M or S box reaches the second light array sensor, we'll wait for the array to become [192, 1, 1]. This indicates that a box has crossed the second light array sensor and the first diffuse sensor while the pivot arm is OFF. If the following array is [0, 1, 1], it means that the pivot arm hasn't sorted it, suggesting that the M box is in the S slot. Conversely,

if the array is [192, 0, 0] and becomes [0, 0, 0], it implies that a box has crossed the second light array sensor without activating the diffuse sensor, and the pivot arm has sorted, indicating that an S box is in the M slot.

By implementing this logic, we've comprehensively addressed all six possible scenarios for the misplaced boxes

```
// Define the state variables
context.state = context.state || 0;

// Retrieve the input array from msg.payload
var inputArray = msg.payload;

// State machine logic
if (context.state === 0 && inputArray.length === 3 && inputArray[0] === 192
&& inputArray[1] === 0 && inputArray[2] === 0) {
    // Transition to state 1 when the array is [192,0,0]
    context.state = 1;
    return null; // Wait for the next input
} else if (context.state === 1) {
    // Handle the next input based on the state
    if (inputArray.length === 3 && inputArray[0] === 192 && inputArray[1] ===
0 && inputArray[2] === 1) {
        // Set the output for [192,0,1]
        msg.payload = "M box in L slot";
        context.state = 0; // Reset state
        return msg;
    } else if (inputArray.length === 3 && inputArray[0] === 0 &&
inputArray[1] === 0 && inputArray[2] === 0) {
        // Set the output for [0,0,0]
        msg.payload = "S box in L slot";
        context.state = 0; // Reset state
        return msg;
    } else {
        // If the input doesn't match expected patterns, reset state
        context.state = 0;
        return null;
    }
} else {
    // Reset state if input doesn't match expected patterns
    context.state = 0;
    return null;
}
```

Script III. 10 M or S Box in L Slot Script

```
// Define the state variables
context.state = context.state || 0;
context.firstDetection = context.firstDetection || false; // Flag to track if
[192,1,1] was detected first
var inputArray = msg.payload;
// State machine logic
if (context.state === 0 && inputArray.length === 3 && inputArray[0] === 192 &&
inputArray[1] === 1 && inputArray[2] === 1) {
    // Transition to state 1 when the array is [192,1,1]
    context.state = 1;
    context.firstDetection = true;
    return null; // Wait for the next input
} else if (context.state === 1 && context.firstDetection) {
    // Handle the next input based on the state
    if (inputArray.length === 3 && inputArray[0] === 0 && inputArray[1] === 1 &&
inputArray[2] === 1) {
        msg.payload = "M box in S slot";
        context.state = 0; // Reset state
        context.firstDetection = false; // Stop further processing
        return msg;
    } else {
        // If the input doesn't match the expected pattern [192,1,1], reset state
        context.state = 0;
        context.firstDetection = false;
        return null;
    }
} else if (context.state === 0 && inputArray.length === 3 && inputArray[0] === 192 &&
inputArray[1] === 0 && inputArray[2] === 0) {
    // Transition to state 2 when the array is directly [192,0,0]
    context.state = 2;
    return null; // Wait for the next input
} else if (context.state === 2) {
    // Handle the next input when in state 2
    if (inputArray.length === 3 && inputArray[0] === 0 && inputArray[1] === 0 &&
inputArray[2] === 0) {
        msg.payload = "S box in M slot";
        context.state = 0;  // Reset state
        return msg;
    } else {
        // If the input doesn't match the expected pattern [0,0,0], reset state
        context.state = 0;
        return null;
    }
} else {
    // Reset state if input doesn't match expected patterns
    context.state = 0;
    return null;
}
```

Script III. 11 M Box in S Slot or S Box in M Slot Script

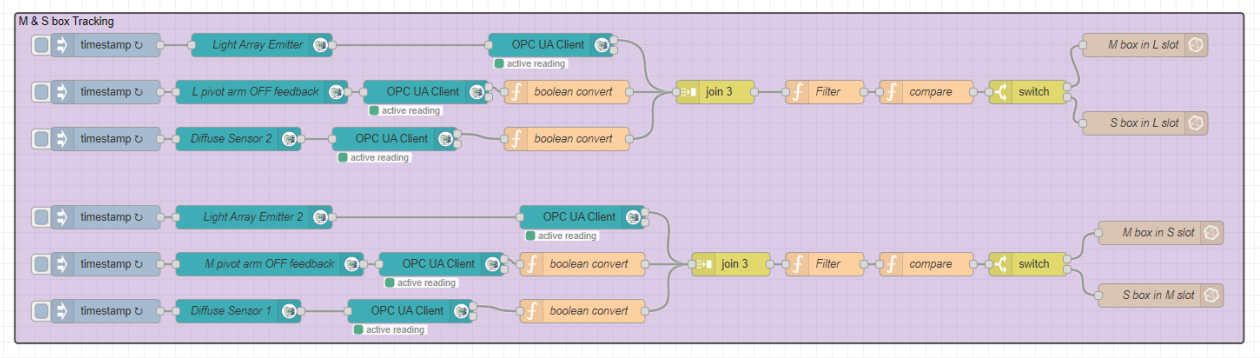


Figure III. 98 Lost M and S Boxes Flows

III.12.6 Using the Incremental Encoder

To integrate the incremental encoder signal from the Factory I/O conveyor belt into our control system, we'll leverage a custom function block within TIA Portal. Ideally, we would use a High-Speed Counter (HSC) block for pulse counting, position tracking, and speed measurement. However, due to limitations in the simulator, which requires a physical PLC to set up the HSC block, we'll create a custom block to read the encoder signal instead.

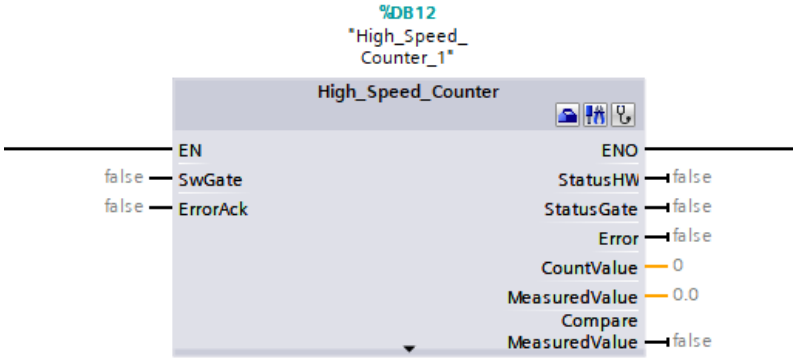


Figure III. 99 HSC bloc

Within TIA Portal, we'll click on "Add new block" and choose "Function Block," selecting "SCL" as the programming language. We'll then define our variables based on the encoder's specifications, outlined in the Factory I/O documentation: quadrature encoding with two bits (A and B), a resolution of 0.0225 meters per count

→ Inputs:

- Signal_A, Signal_B: These are Boolean (TRUE/FALSE) signals from the encoder. The pattern of changes in these signals indicates the direction and amount of movement.

→ Outputs:

- Direction: A Boolean indicating the direction of movement.
- PulsesPerSecond: The number of encoder pulses detected in the last second (a measure of speed).
- Velocity: The calculated velocity in meters per second (m/s).
- PulseCount: The total number of pulses counted since the last reset.

→ Variables and Constants:

- LastSignalA, LastSignalB: These variables store the previous state of the input signals. This is used to detect changes (edges) in the signals.
- TON_1s: This is a timer function block (TON stands for "Timer ON Delay"). It's used to trigger calculations every second.
- EncoderResolution: A constant defining the relationship between encoder pulses and the physical distance moved (meters per pulse)

Encoder Counter

	Name	Data type	Default value	Retain
1	Input			
2	Signal_A	Bool	false	Non-ret...
3	Signal_B	Bool	false	Non-retain
4	Output			
5	Direction	Bool	false	Non-retain
6	PulsesPerSecond	DInt	0	Non-retain
7	Velocity	Real	0.0	Non-retain
8	PulseCount	DInt	0	Non-retain
9	InOut			
10	<Add new>			
11	Static			
12	LastSignalA	Bool	false	Non-retain
13	LastSignalB	Bool	false	Non-retain
14	TON_1s	TON_TIME		Non-retain
15	Temp			
16	<Add new>			
17	Constant			
18	EncoderResolution	Real	0.0225	

Figure III. 100 Function Block Variables and Constants for Incremental Encoder

Next, we'll enter the SCL script into the function block's editor window, taking into account the encoder's characteristics and desired functionalities. Once complete, we'll compile the script and address any errors.

```
// Timer Configuration

#TON_1s(IN := TRUE,
        PT := T#1S); // Start the timer with a 1-second preset time
// Main Program Cycle
// Detect Rising/Falling Edges
IF #Signal_A AND NOT #LastSignalA THEN // Rising edge on Signal A
    // Determine Direction
    #Direction := #Signal_B;
    #PulseCount := #PulseCount + 1; // Count pulse

ELSIF NOT #Signal_A AND #LastSignalA THEN // Falling edge on Signal A
    // Determine Direction
    #Direction := NOT #Signal_B;
    #PulseCount := #PulseCount + 1; // Count pulse

ELSIF #Signal_B AND NOT #LastSignalB THEN // Rising edge on Signal B
    // Determine Direction
    #Direction := NOT #Signal_A;
    #PulseCount := #PulseCount + 1; // Count pulse

ELSIF NOT #Signal_B AND #LastSignalB THEN // Falling edge on Signal B
    // Determine Direction
    #Direction := #Signal_A;
    #PulseCount := #PulseCount + 1; // Count pulse
END_IF;

// Update Last States for the Next Cycle
#LastSignalA := #Signal_A;
#LastSignalB := #Signal_B;

// Calculate Velocity and PulsesPerSecond every second
IF #TON_1s.Q THEN
    #Velocity := DINT_TO_REAL(#PulseCount) * #EncoderResolution; // Pulses
* meters/pulse
    #PulsesPerSecond := #PulseCount;  // Directly assign the pulse count
    #PulseCount := 0; // Reset the counter for the next interval
    #TON_1s(IN := FALSE,
            PT := T#1S);
    #TON_1s(IN := TRUE,
            PT := T#1S);
END_IF;
```

Script III. 12 SCL Script for Incremental Encoder

The code uses IF...ELSIF statements to check for changes (rising or falling edges) in the Signal_A and Signal_B inputs. The pattern of these changes determines the direction of movement (Direction) ,Each detected edge increments the PulseCount variable.

The TON_1s timer triggers every second. Inside the timer's block, the code calculates:

- Velocity: Multiplies the PulseCount by the EncoderResolution to convert pulses into a physical distance unit.
- PulsesPerSecond: This is simply the current value of PulseCount.
- PulseCount is reset to zero to start counting for the next second.

Next ,We'll create a new main block named "Conveyor Feedback" and insert our custom function block into a network labeled "Belt Conveyor 6m Feedback." The "Signal_A" and "Signal_B" inputs of the function block will be connected to the corresponding encoder signals from the 6m conveyor belt.

After compiling and downloading the project to the device, we'll enable monitoring and return to Factory I/O. By setting the potentiometer to 1.5 volts, corresponding to 0.45 m/s, we should observe values of 20 to 21 in the "PulsesPerSecond" output and a velocity range of 0.45 to 0.47 in TIA Portal. This confirms that our code is functioning as intended.

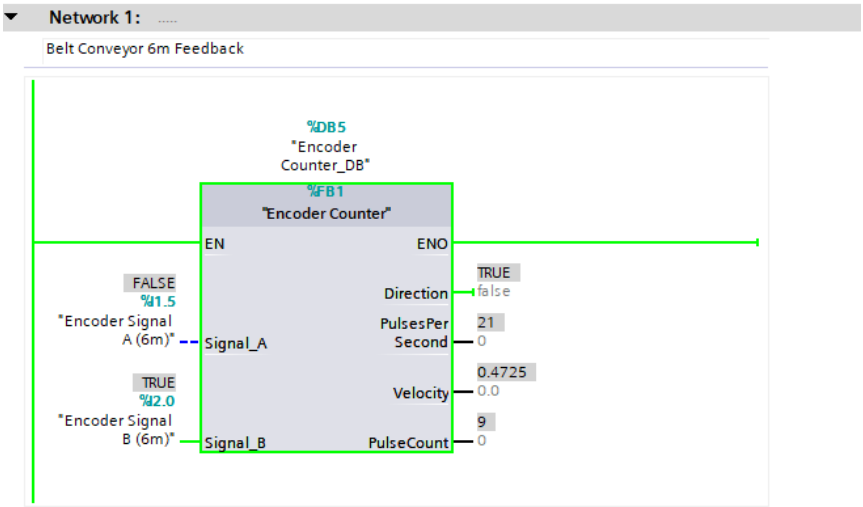


Figure III. 101 Conveyor Feedback Block in TIA Portal

The slight fluctuation in pulse count is due to the nature of the software and the communication speed between Factory I/O and PLCSIM. It's important to note that this workaround bypasses the use of an HSC block and that the maximum detectable pulse rate is limited to 31 pulses per second, even at higher speeds.

This custom function block provides a viable solution for reading and interpreting the encoder signals from the Factory I/O conveyor belt within the simulation environment, allowing us to track its speed, and direction.

For the remaining conveyor belts, we will replicate the same procedure. We'll create two additional networks within the TIA Portal project and insert our custom function block into each. The encoder signals for each respective conveyor belt will be connected to the Signal_A and Signal_B inputs of the function block

III.12.7 Creating a Feedback Flow for Conveyor Belts

Moving to the Node-RED workspace, we'll create three flows dedicated to monitoring the conveyor belt behavior

Flow 1 & 2: Conveyor Status and Speed Comparison:

The first two flows read speed feedback and output signal of conveyor belt, next we'll join these values and create two scripts:

- Conveyor Status Script

```
var inputArray = msg.payload;
// Initialize an empty string to store the message
var message = "";
// Check if the input is an array with exactly two elements
if (Array.isArray(inputArray) && inputArray.length === 2) {
    // Compare the values of the array
    if (inputArray[0] === 0 && inputArray[1] !== 0) {
        message = "The conveyor is malfunctioning";
    } else if (inputArray[0] === 0 && inputArray[1] === 0) {
        message = "The conveyor is stopped manually.";
    } else {
        message = "The conveyor is running normally"; // Optional: handle
other conditions
    }
} else {
    message = "Error: Input is not a valid array of two elements";
}
// Set the message to msg.payload
msg.payload = message;
return msg;
```

Script III. 13 Conveyor Status Script

This script will analyze the data and generate a message indicating the conveyor's status:

→ "The conveyor is malfunctioning"

→ "The conveyor is stopped manually"

→ "The conveyor is running normally"

➢ Speed Comparison Script:

```
var inputArray = msg.payload;
// Initialize an empty string to store the message
var message = "";
// Define a fixed tolerance
var tolerance = 0.025;
// Check if the input is an array with exactly two elements
if (Array.isArray(inputArray) && inputArray.length === 2) {
    // Extract the values from the array
    var speed1 = inputArray[0];
    var speed2 = inputArray[1];
    // Check if the speeds are valid numbers
    if (typeof speed1 === 'number' && typeof speed2 === 'number') {
        // Calculate the absolute difference
        var difference = Math.abs(speed1 - speed2);

        // Compare the speeds with tolerance
        if (difference <= tolerance) {
            message = "correct";
        } else if (speed1 > speed2) {
            var percentageHigh = ((speed1 - speed2) / speed2) * 100;
            message = "The speed of the conveyor is high by " +
percentageHigh.toFixed(2) + "%";
        } else if (speed1 < speed2) {
            var percentageLow = ((speed2 - speed1) / speed1) * 100;
            message = "The speed of the conveyor is low by " +
percentageLow.toFixed(2) + "%";
        }
    } else {
        message = "Error: Speeds are not valid numbers";
    }
} else {
    message = "Error: Input is not a valid array of two elements";
}
// Set the message to msg.payload
msg.payload = message;

return msg;
```

Script III. 14 Speed Comparison Script

This script will calculate the difference between the actual speed (from feedback) and the setpoint speed (from the output signal). It will then generate a message indicating whether the speeds are within a specified tolerance or, if not, the percentage by which one speed is higher or lower than the other.

The third flow will directly read the conveyor's direction signal.

→ If the signal is 1, it indicates the conveyor is moving in the correct direction.

→ If the signal is 0, it signifies that the conveyor is rolling in the opposite direction, which is incorrect.

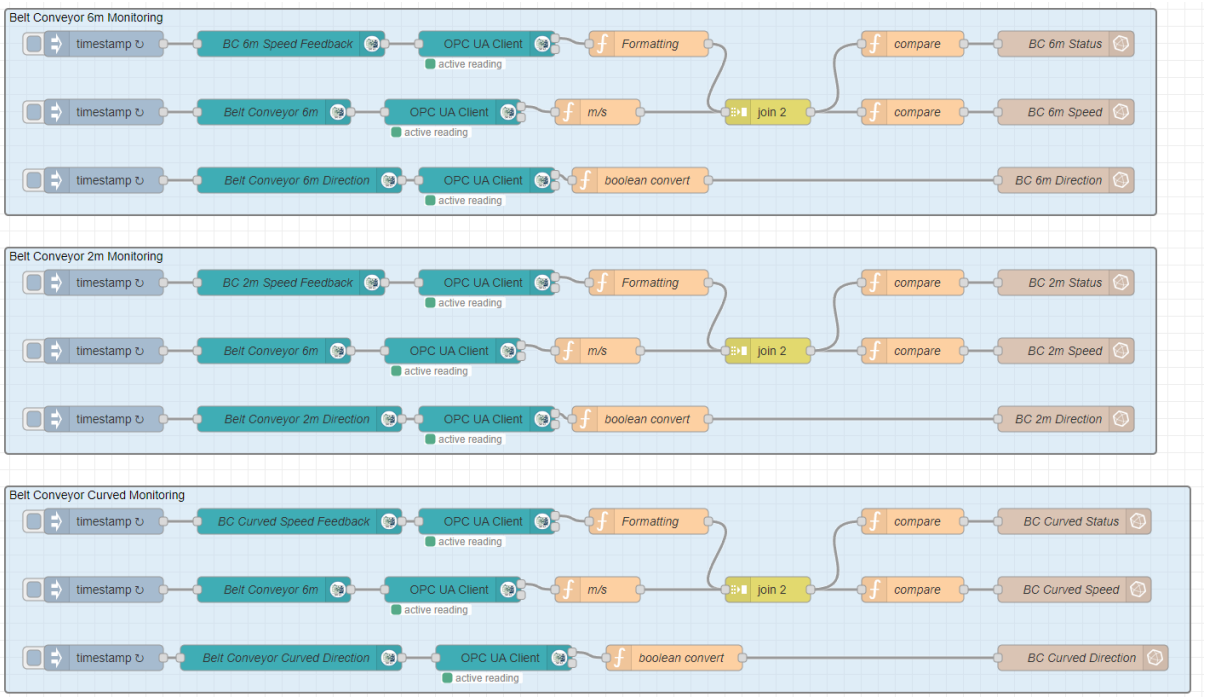


Figure III. 102 Conveyor Status and Speed Comparison Flows in Node-RED

III.13 Tracking Machine Runtime

In TIA Portal, we'll develop a specialized function block to monitor the operational duration of our machines. This block, named "Run Time Calculator," will be programmed using the SCL language

This PLC code implements a timer triggered by a rising edge (transition from off to on) of an input signal. It meticulously tracks the duration for which the signal remains on, calculating the elapsed time in seconds, minutes, hours, and days.

The code begins by detecting when the input signal activates and ensuring it doesn't repeatedly trigger the timer for a continuous signal. A separate reset input allows for manual timer reset.

To ensure precise timekeeping in the "Run Time Calculator" function block, a 1Hz clock signal is essential. This signal can be easily enabled within TIA Portal by navigating to the PLC name, right-clicking, and selecting "Properties." Within the "System and clock memory" section, simply activate the checkbox labeled "Enable the use of clock memory byte." This will generate a 1Hz clock signal that the function block can utilize to accurately increment the seconds counter and track the machine's runtime

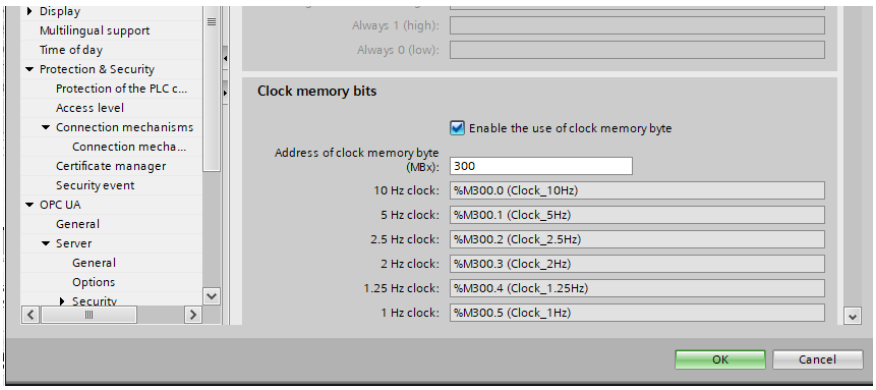


Figure III. 103 Run Time Calculator Function Block in TIA Portal

The 'LastSecondTick' flag ensures the counter advances only once per second, even if the input signal stays on.

As the seconds accumulate, the code systematically converts them into minutes, hours, and days through sequential checks and resets, maintaining a continuous and accurate representation of the elapsed time.

	Name	Data type	Default value	Retain
1	Input			
2	InputSignal	Bool	false	Non-ret...
3	ResetInput	Bool	false	Non-retain
4	Output			
5	Seconds	DInt	0	Non-retain
6	Minutes	DInt	0	Non-retain
7	Hours	DInt	0	Non-retain
8	Days	DInt	0	Non-retain
9	InOut			
10	<Add new>			
11	Static			
12	PreviousSignalState	Bool	false	Non-retain
13	ElapsedTime	UDInt	0	Non-retain
14	LastSecondTick	Bool	false	Non-retain
15	Temp			
16	<Add new>			
17	Constant			
18	<Add new>			

Figure III. 104 Run Time Calculator Function Block Variables and Constants

```
(* Check for rising edge of the input signal *)
IF #InputSignal AND NOT #PreviousSignalState THEN
    #PreviousSignalState := TRUE; // Mark that input is now active
END_IF;

(* Update the previous state after checking for rising edge *)
#PreviousSignalState := #InputSignal;

(* Reset logic *)
IF #ResetInput THEN
    #Seconds := 0;
    #Minutes := 0;
    #Hours := 0;
    #Days := 0;
    #LastSecondTick := FALSE; (* Reset the last second tick flag *)
END_IF;

(* Increment seconds when input is active and it's a new second tick *)
IF #InputSignal AND "Clock_1Hz" AND NOT #LastSecondTick THEN
    #Seconds := #Seconds + 1;
    #LastSecondTick := TRUE; // Mark that we've counted this second
ELSIF NOT "Clock_1Hz" THEN  // Reset the last second tick flag when the
clock pulse is low
    #LastSecondTick := FALSE;
END_IF;

(* Calculate minutes, hours, and days *)
IF #Seconds >= 60 THEN
    #Minutes := #Minutes + 1;
    #Seconds := 0;
END_IF;

IF #Minutes >= 60 THEN
    #Hours := #Hours + 1;
    #Minutes := 0;
END_IF;

IF #Hours >= 24 THEN
    #Days := #Days + 1;
    #Hours := 0;
END_IF;
```

Script III. 15 SCL Script For Run Time Calculator

Next, we'll incorporate a new block named "Run Time" into our TIA Portal project. This block will house instances of our custom "Run Time Calculator" function block to monitor the operational duration of various components: the two pivot arms and the three conveyor belts.

For the pivot arms, we'll utilize their respective feedback sensors as triggers for the timing calculations. Additionally, we'll include a tag named "Factory IO running," which is automatically generated by Factory I/O upon scene creation. This tag acts as a safeguard, preventing time accumulation when the simulation is paused or stopped. Since sensors and parts deactivate when the simulation stops, this tag ensures that the timers remain inactive during those periods.

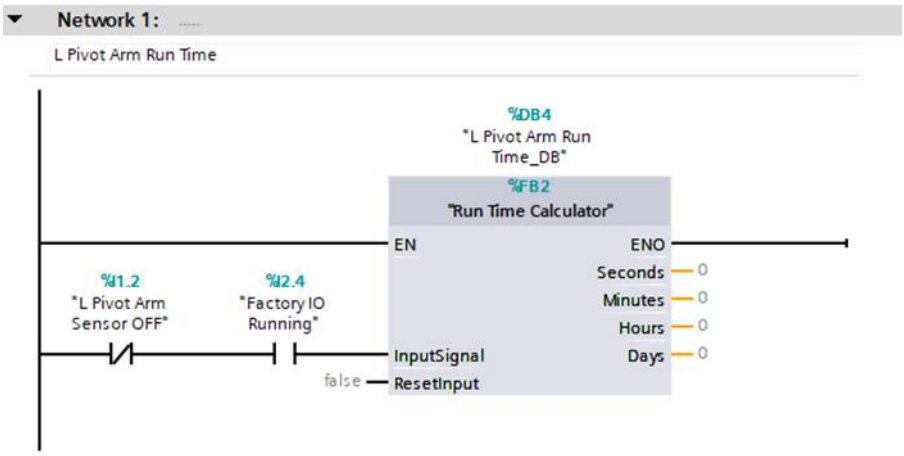


Figure III. 105 Pivot Arm Run Time Network

For the conveyor belts, we'll introduce a memory tag that becomes true when the speed calculated from the incremental encoder exceeds zero. This memory tag will be connected to the input signal of the "Run Time Calculator" block. Consequently, when the memory tag is true, signifying that the conveyor belt is in motion, the timer will start calculating the runtime. This approach ensures that the runtime calculation for each conveyor belt only commences when it's actively operating

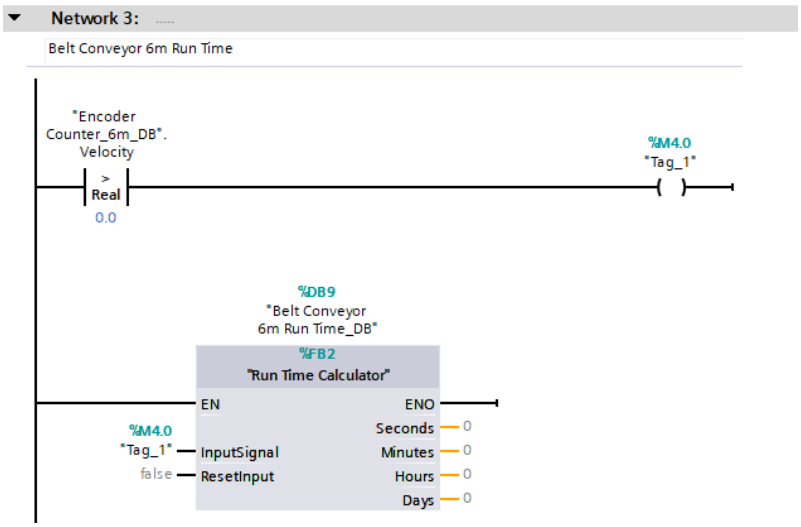


Figure III. 106 Belt Conveyor Run Time Network

As an alternative approach to reading multiple values from the OPC UA server, instead of using multiple OPC UA client nodes for each value (seconds, minutes, hours, and days), we can leverage a single client node to retrieve all four values simultaneously.

This approach not only simplifies the workflow but also optimizes resource utilization, particularly for PLC CPUs with limited session capacity.

our optimized workflow involves using a single Inject node to trigger the flow at regular intervals. Four Function nodes will be employed to set the appropriate OPC UA Node ID and label for each value (seconds, minutes, hours, and days). These configurations will be passed to a single OPC UA Client node, which will then read all four values in a single request. A Switch node will subsequently route the retrieved values based on their labels, ensuring they are directed to the correct destinations within the Node-RED flow

By implementing this optimized workflow, we significantly reduce the number of sessions required from 20 to 5.

```
msg.topic = "ns=3;s=\"Belt Conveyor 6m Run Time_DB\".\"Seconds\"";

msg.label = "Seconds";
return msg;
```

```
msg.topic = "ns=3;s=\"Belt Conveyor 6m Run Time_DB\".\"Minutes\"";
msg.label = "Minutes";
return msg;
```

```
msg.topic = "ns=3;s=\"Belt Conveyor 6m Run Time_DB\".\"Hours\"";
msg.label = "Hours";
return msg;
```

```
msg.topic = "ns=3;s=\"Belt Conveyor 6m Run Time_DB\".\"Days\"";
msg.label = "Days";
return msg;
```

Script III. 16 Optimized OPC UA Client Node Script

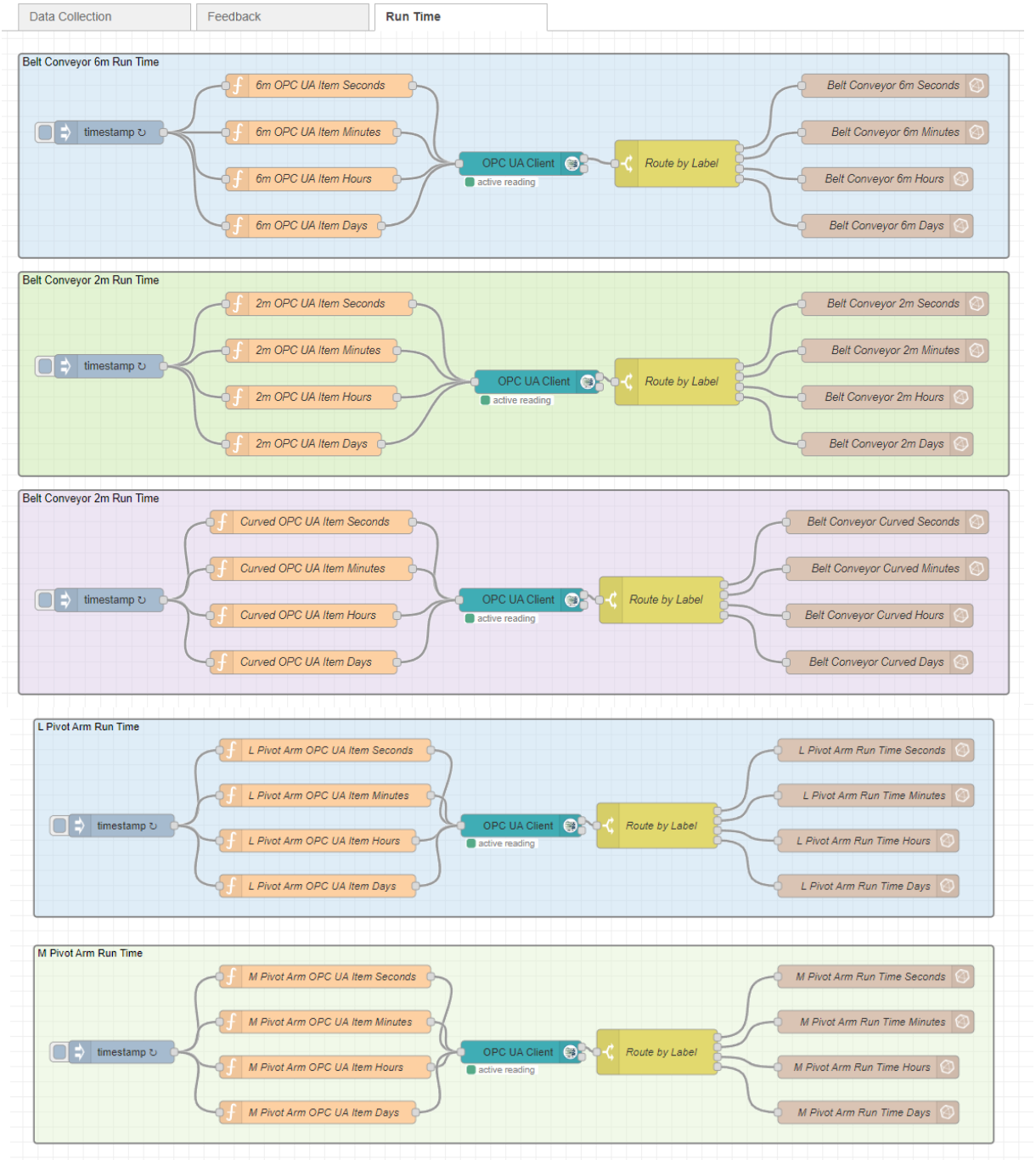


Figure III. 107 Run time Optimized OPC UA Client Node

While this optimization could be applied to our existing flows, we've chosen to maintain the current structure due to the project completion. Currently, we're utilizing 44 out of 48 sessions, providing sufficient headroom for potential future expansions. However, if the need arises to incorporate additional OPC UA values and the session limit becomes a concern, we can readily adopt this optimized workflow to ensure seamless data collection while minimizing resource consumption.

III.14 Completing the Dashboard

Now that we have gathered sufficient data to effectively monitor our system, we will proceed to enhance our Grafana dashboard by adding new panels. Specifically, we will incorporate tables to track the behavior of both the conveyor belts and pivot arms, as well as to maintain a log of any lost boxes, with the integration of these new panels, we'll have successfully completed the creation of our comprehensive dashboard

Figure III. 108 Completed Grafana Dashboard

III.15 Creating Local Backups

To minimize the risk of data loss in scenarios where internet connectivity is unavailable, we'll establish a local backup mechanism for the collected data. This strategy involves regularly creating copies of the data stored in the InfluxDB database and storing these backups on a local storage device.

By implementing this approach, we ensure that critical factory data is not solely reliant on cloud storage. In the event of a network outage or disruption in cloud services, we can access the local backups to retrieve and analyze past data, maintaining operational continuity and minimizing potential downtime. Furthermore, local backups offer an additional layer of redundancy, safeguarding against data loss due to unforeseen circumstances, such as accidental deletion or data corruption in the cloud.

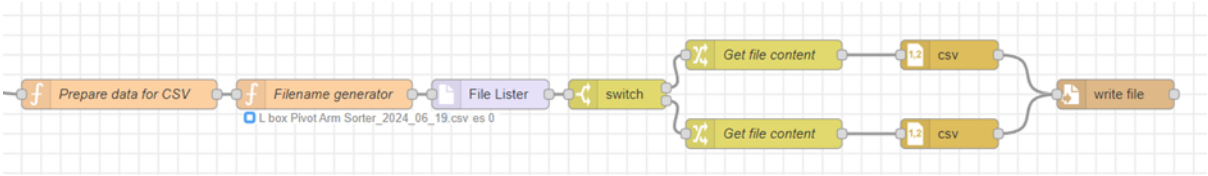


Figure III. 109 Local Backup Flow

we will add flow that operates on the principle of automated daily archiving into CSV files. It begins by receiving data from the flow we created. This data is then processed by a function node to add a timestamp for accurate tracking and organization.

```
var now = new Date();
// Format the timestamp as desired (e.g., "YYYY-MM-DD HH:MM:SS")
var timestamp = now.getFullYear() + "-" +
                ("0" + (now.getMonth() + 1)).slice(-2) + "-" +
                ("0" + now.getDate()).slice(-2) + " " +
                ("0" + now.getHours()).slice(-2) + ":" +
                ("0" + now.getMinutes()).slice(-2) + ":" +
                ("0" + now.getSeconds()).slice(-2);
msg.payload = {
    "timestamp": timestamp,
    "Pivot Arm": msg.payload,
};
return msg;
```

Script III. 17 Function Node to Add Timestamp

Next, a second function node dynamically generates a unique filename for the backup file based on the current date, ensuring each day's data is stored separately. To avoid overwriting previous backups, the flow checks if a file with the same date already exists in the designated backup folder. If found, new data is appended to this existing file. However, if no file exists for that day, a new CSV file is created.

The "Switch" node acts as a decision point, directing the data flow based on whether or not the file exists. In either case, the data is transformed into CSV format using dedicated nodes and ultimately written or appended to the appropriate file.

```
var now = new Date();
var yyyy = now.getFullYear();
var mm = ("0" + (now.getMonth() + 1)).slice(-2); // Ensure double digits for
month
var dd  = ("0" + now.getDate()).slice(-2); // Ensure double digits for day
// Generate the filename pattern with underscores
msg.fname = "L box Pivot Arm Sorter_" + yyyy + "_" + mm + "_" + dd + ".csv";
// Full filename with path for the file node later
msg.filename = "D:/iot/Pivo Arm/" + msg.fname;
// Save the current payload into a different place on the msg object
msg.filecontent = msg.payload;
// Pass the file name search pattern to fs node to tell if the file exists or not
msg.payload = {"pattern": msg.fname};
node.status({fill:"blue", shape:"ring", text: msg.fname});
return msg;
```

Script III. 18 Function Node to Generate Filename

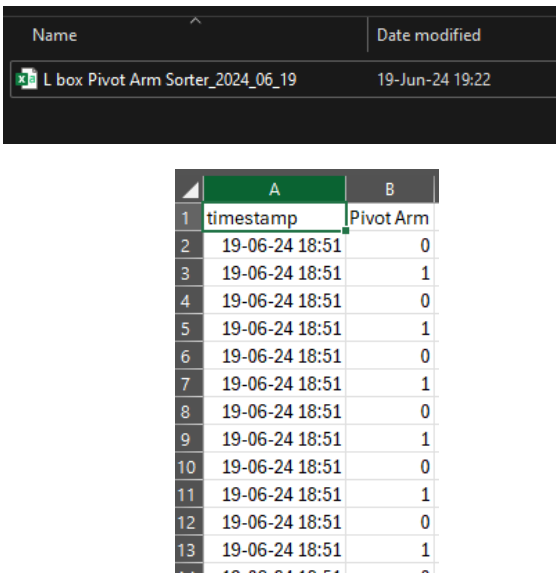


Figure III. 110 Local Backup .csv file

III.16 Downloading Data from Grafana

To retrieve the collected data within a specific timeframe, we'll navigate to the Grafana dashboard and select the desired time range from the top menu. Next, we'll click on the panel title and choose "Inspect." In the "Data" tab, we can view the query results for that specific time range. Finally, we'll click "Download" to obtain a CSV file containing our filtered data.

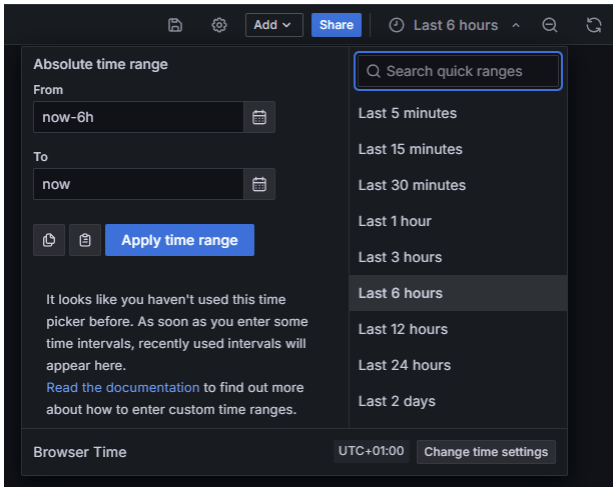


Figure III. 111 Downloading Data from Grafana: Time Range

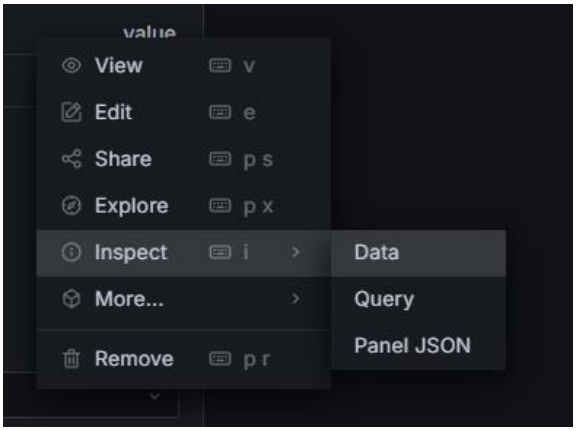


Figure III. 112 Downloading Data from Grafana: Data Tab

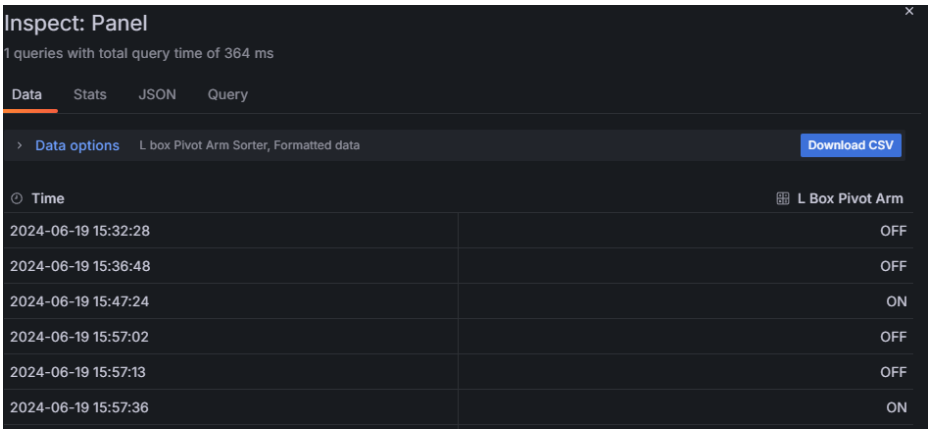


Figure III. 113 Downloading Data from Grafana: Download CSV

III.16.1 Adding users to Grafana

We have the ability to add users and assign roles in Grafana, enabling us to manage access and permissions effectively. This ensures that our monitoring environment is collaborative, secure, and tailored to the needs of different team members.

There are four types of organization roles in Grafana:

- **Grafana Admin:** Manage organizations, users, and view server-wide settings.
- **Organization Administrator:** Manage data sources, teams, and users within an organization.
- **Editor:** Create and edit dashboards.
- **Viewer:** View dashboards.

To add a new user, we follow this step:

a) In the sidebar, we click the Server Admin (shield) icon.

b) In the Users tab, we click new user**.**

c) In Name, we enter the name of the user.

d) In E-mail, we enter the email of the user.

e) In Username**,** we enter the username that the user will use to log in.

f) In Password**,** we enter a password. (users can change their password once they log in).

g) We click Create user to create the user account.

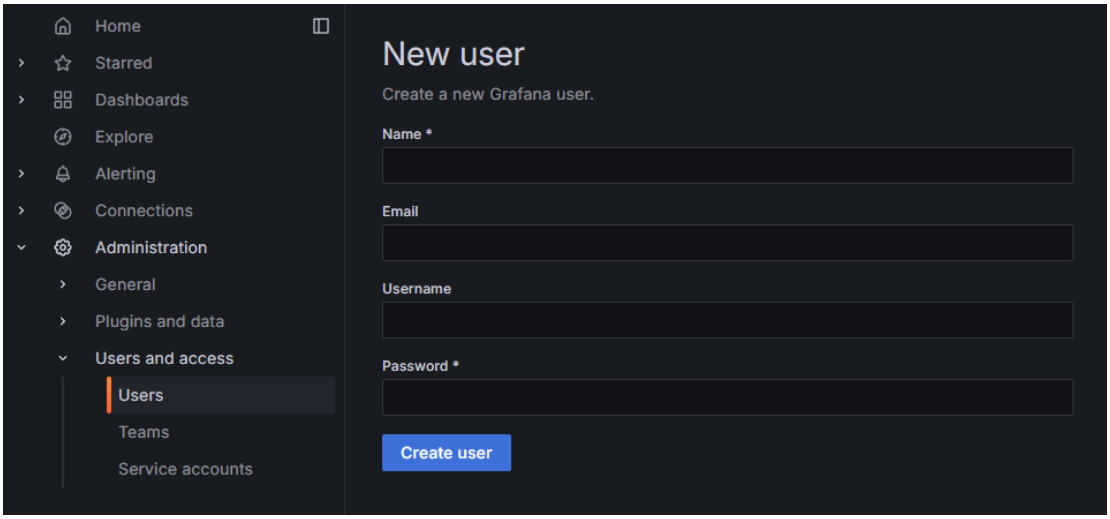


Figure III. 114 Adding Users in Grafana

III.17 Securing Grafana and InfluxDB with SSL Certificates

Securing our web applications (InfluxDB and Grafana) is crucial, one effective way to do this is by using SSL certificates to encrypt traffic between our server and client browsers. we will obtaining free SSL certificates from Let's Encrypt and configure Nginx reverse proxy to enhance the security of our web interfaces, and we'll also integrate domain name "ft-mec19.me" to point to the server of our application

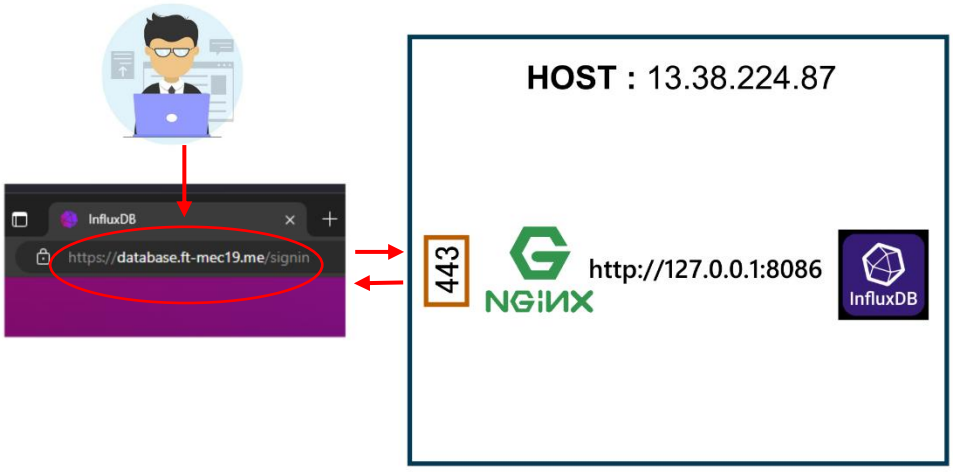


Figure III. 115 Securing Grafana and InfluxDB with SSL Certificates

III.17.1 Understanding the Core Components

- **SSL Certificates:** SSL stand for (Secure Sockets Layer) is an encryption-based Internet security protocol. It was first developed by Netscape in 1995 for the purpose of ensuring privacy, authenticate a website's identity and establish an encrypted connection, ensuring the confidentiality and protection of sensitive data transmitted between the server and browser.

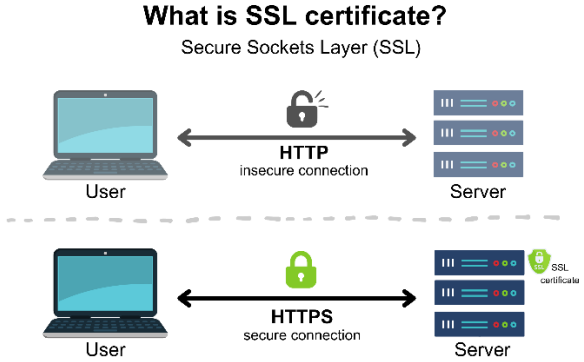


Figure III. 116 SSL Certificate Explanation

- **Let's Encrypt:** A non-profit certificate authority that offers free SSL certificates, making encryption accessible and promoting a safer web environment.
- **Nginx:** A high-performance web server and reverse proxy renowned for its speed, efficiency, and flexibility. As a reverse proxy, it sits in front of the web applications, handling client requests and directing them to the appropriate backend servers.
- **Domain Names:** Human-readable addresses (e.g., yourdomain.com) that point to the IP addresses of web applications, making them easier to remember and access.
- **ACME.sh:** (Automated Certificate Management Environment shell script) is a versatile command-line tool designed to simplify the process of obtaining, managing, and renewing SSL certificates from Let's Encrypt
- **Socat:** (SOcket CAT) is a versatile networking utility that can establish two-way communication channels between different types of addresses and protocols. In the context of ACME.sh, socat plays a crucial role in the standalone mode of certificate issuance.
    - **Standalone Mode**: In this mode, ACME.sh temporarily starts a small web server to handle the Let's Encrypt domain verification challenge. socat acts as a bridge, forwarding requests from the Let's Encrypt server to the temporary web server and vice versa

III.17.2 Pointing Domain Name to server's IP address

Since we are having 2 web applications, we will create 2 subdomains with our main domain name **ft-mec19.me,** Subdomains are essentially extensions the main domain name They allow us to organize and differentiate different sections and services within our website. For instance, we will be having:

- **www.ft-mec19.me** (the main website)
- **database.ft-mec19.me** (the InfluxDB interface)
- **dashboard.ft-mec19.me** (the Grafana interface)

Each subdomain can point to a different IP address or even a different server altogether.

III.17.2.1 Why Using Subdomains

Using subdomains like database.ft-mec19.me and dashboard.ft-mec19.me for InfluxDB and Grafana instances respectively brings the following benefits:

- **Organization:** It helps separate different web applications, making management easier.
- **Clarity:** Visitors can easily understand what each subdomain is for based on its name.
- **Flexibility:** If we ever decide to move InfluxDB or Grafana to a different server, we can simply update the DNS records for the respective subdomains without changing the main domain.

III.17.2.2 Creating and Pointing Subdomain

With our domain registered in Namecheap, we'll access the DNS settings within our Namecheap account. There, we'll add two new DNS A records:

- **A record for InfluxDB:** This record will point the subdomain (database.ft-mec19.me) directly to the public IP address of our AWS EC2 instance hosting InfluxDB.
- **A record for Grafana:** This record will point another subdomain (dashboard.ft-mec19.me) to the public IP address of our AWS EC2 instance hosting Grafana.

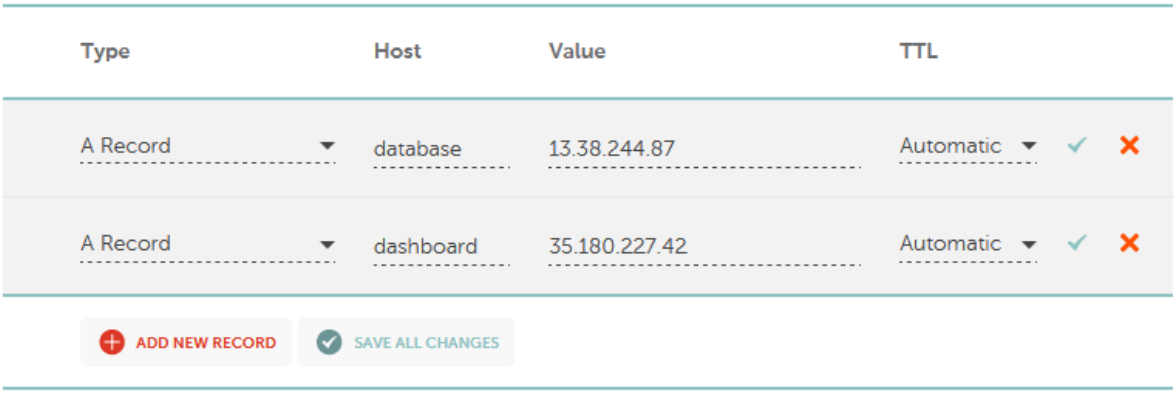


Figure III. 117 Adding DNS A Records in Namecheap

III.17.3 Nginx Workflow

After setting up our DNS records, we'll SSH into the server and proceed with the following steps

Step 1: Install Nginx

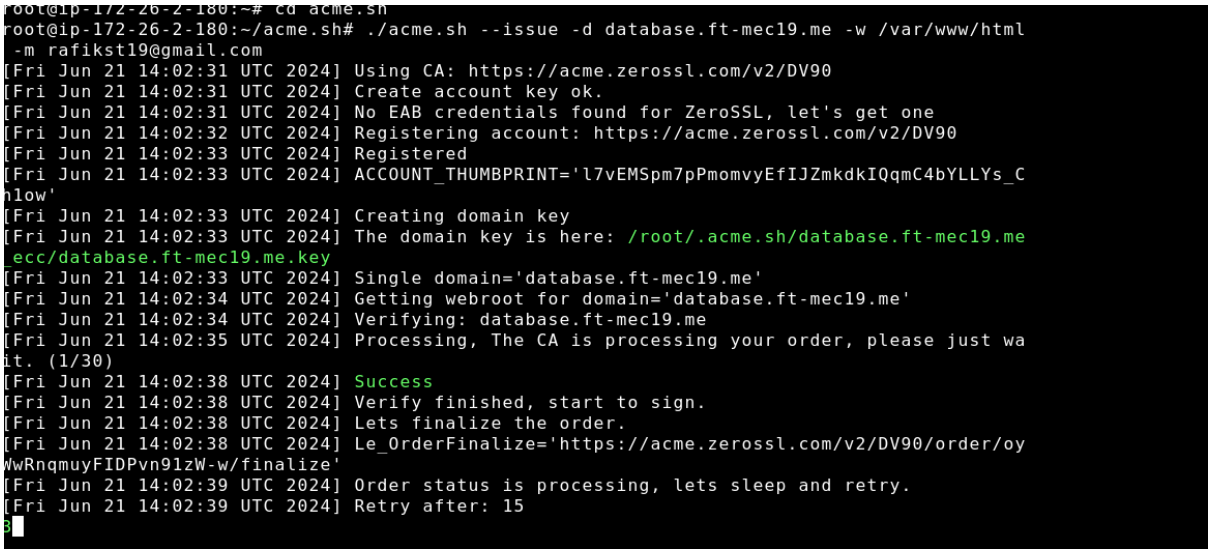
```
sudo apt-get update

sudo apt-get install -y nginx
```

Step 2: Install ACME.sh and socat

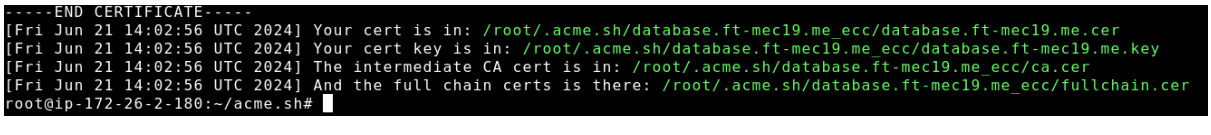
```
git clone https://github.com/acmesh-official/acme.sh.git

cd acme.sh

apt-get install socat -y
```

Step 3: Generate SSL Certificate

```
./acme.sh --issue -d database.ft-mec19.me -w /var/www/html -m rafikst19@gmail.com –server letsencrypt
```



ACME.sh generates four files to ensure compatibility with a wide range of web servers and clients. Some servers or clients might require different combinations of these files, and Nginx Usually needs the full chain certificate (fullchain.cer) and the private key (database.ft-mec19.me.key).



1. **Certificate (/root/.acme.sh/database.ft-mec19.me_ecc/database.ft-mec19.me.cer)**
    - This is our website's primary SSL certificate. It contains the public key that our server uses to prove its identity to web browsers.

2. **Full Chain Certificate (/root/.acme.sh/database.ft-mec19.me_ecc/fullchain.cer)**
    - This file combines our website's certificate with the intermediate CA certificate. It's a convenient way to provide all the necessary information for proper SSL/TLS authentication.

Step 4 : Configure Nginx

Create separate configuration file for the subdomain in /etc/nginx/sites-available/ directory. using vim editor

```
cd /etc/nginx/sites-available

vim database.ft-mec19.me
```

and inside the created file we add the following commands

```
server {
   listen 80;        # Listen for HTTP traffic
   server_name database.ft-mec19.me;
   return 301 https://$host$request_uri;  # Redirect to HTTPS
}
server {
   listen            443 ssl;
   server_name       database.ft-mec19.me;
   ssl_certificate     /root/.acme.sh/database.ft-mec19.me_ecc/fullchain.cer;
   ssl_certificate_key /root/.acme.sh/database.ft-mec19.me_ecc/database.ft-mec19.me.key;
   ssl_protocols       TLSv1 TLSv1.1 TLSv1.2;
   ssl_ciphers       HIGH:!aNULL:!MD5;
location / {
   proxy_pass http://127.0.0.1:8086;
}
}
```

Step 5 : links to Nginx

```
cd /etc/nginx/sites-enabled/
```

- This command changes the current working directory within the terminal to the folder /etc/nginx/sites-enabled/. This directory is crucial as it holds symbolic links to Nginx configuration files that are currently active and loaded by the web server.

```
ln -s /etc/nginx/sites-available/database.ft-mec19.me /etc/nginx/sites-enabled/
```

- This command creates a symbolic link (essentially a shortcut) within the sites-enabled directory. This link points to the configuration file named database.ft-mec19.me located in the sites-available directory; by creating this link, we're essentially telling Nginx, "use this configuration file when handling web requests."

```
nginx -t
```

This command is a critical safety check. It tells Nginx to parse the entire configuration (including the newly linked file) and verify that it's free of syntax errors or inconsistencies. If any errors are found, they'll be displayed in the terminal, allowing us to fix them before restarting Nginx

```
root@ip-172-26-2-180:/etc/nginx/sites-enabled# nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
root@ip-172-26-2-180:/etc/nginx/sites-enabled#
```

Step 6 : Restarting Nginx web server

```
systemctl reload nginx
```

With these steps, we've completed the SSL setup for InfluxDB and pointed it to our subdomain,we'll apply the same process, but in the Nginx configuration, we'll add Proxy Headers under proxy_pass to ensure the backend service receives accurate information,this configuration will prevent potential "origin not allowed" errors by explicitly allowing requests from the Grafana subdomain

```
location / {
        proxy_pass http://127.0.0.1:3000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;  # Pass HTTPS to the backend if
needed
    }
}
```

Once we complete these configuration steps, we can test our setup. By typing "database.ft-mec19.me" or "dashboard.ft-mec19.me" into our web browser, we should see a padlock icon in the address bar, indicating that our connection is secure.

If we further inspect the certificate details, we'll notice that it was issued by Let's Encrypt and has an expiration date of 90 days from the creation date. This is because Let's Encrypt provides free SSL certificates with a 90-day validity period.

The advantage of using Let's Encrypt is that we can easily renew the certificate whenever we need to, ensuring continuous security for our InfluxDB and Grafana services. And we can automate the renewal process to ensure that our certificates remain up-to-date without requiring manual intervention

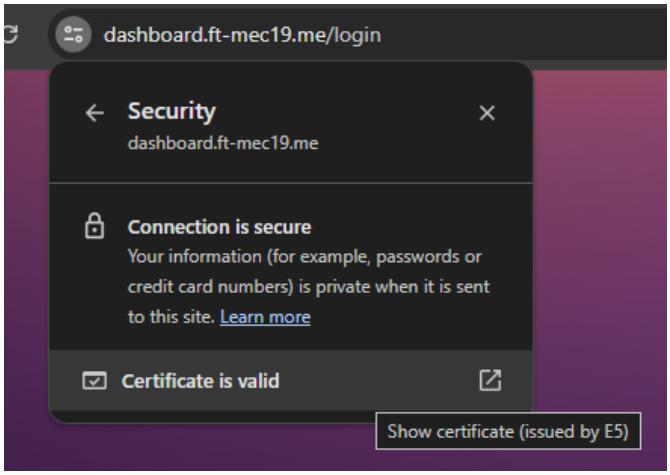


Figure III. 118 Secure Connection for database.ft-mec19.me

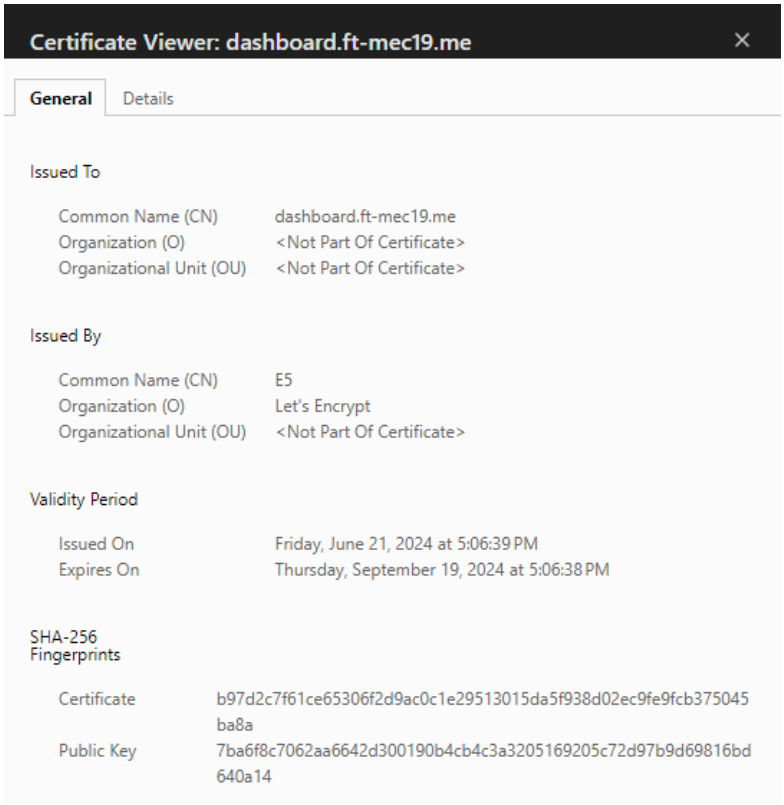


Figure III. 119 SSL certificate details

Conclusion

this chapter has successfully demonstrated the implementation of a comprehensive system for monitoring and controlling a box sorting process. By integrating Factory I/O, TIA Portal, Node-RED, InfluxDB, and Grafana, we have achieved real-time data collection, analysis, and visualization of the system's performance. The implementation of feedback mechanisms for pivot arms and conveyor belts has enhanced the system's reliability by enabling the detection of malfunctions and tracking of lost boxes. The creation of a custom function block in TIA Portal to monitor machine runtime has provided valuable insights for maintenance and optimization. Additionally, securing the web interfaces with SSL certificates and setting up domain names has ensured the confidentiality and integrity of data transmission. This chapter has showcased the practical application of IoT in industrial automation, highlighting its potential for improving efficiency, data-driven decision-making, and overall system performance.

General Conclusion and Future Perspectives

The research presented in this thesis demonstrates a commendable effort to integrate the Internet of Things (IoT) with electromechanical systems for the purpose of monitoring and controlling a box sorting process. The approach is comprehensive, covering various aspects from project creation and hardware configuration to data collection, analysis, and visualization. The use of Factory I/O for simulation and TIA Portal for PLC programming provides a realistic and practical environment for testing and validating the system.

The integration of OPC UA, Node-RED, InfluxDB, and Grafana showcases a well-rounded understanding of IoT technologies and their application in industrial automation. The implementation of feedback mechanisms and the custom function block for runtime tracking further demonstrate the researcher's ability to address real-world challenges and optimize system performance.

However, there are areas where the research could be strengthened. The discussion of security measures, while mentioning SSL certificates and domain names, could be more in-depth, addressing potential vulnerabilities and mitigation strategies in greater detail. Additionally, the thesis could benefit from a more extensive literature review, comparing and contrasting the chosen approach with alternative solutions in the field.

Overall, this research provides a valuable contribution to the field of IoT and industrial automation. It demonstrates a practical and effective approach to monitoring and controlling a box sorting process, highlighting the potential of IoT technologies to improve efficiency and decision-making in industrial settings. With further refinement and expansion, this research could serve as a foundation for future work in this rapidly evolving field.

References

[1]. Ashton K. That 'Internet of Things' Thing. RFID Journal. 2009.

[2]. Wikipedia contributors. (2023, May 14). *MQTT*. In Wikipedia, The Free Encyclopedia. Retrieved from https://en.wikipedia.org/wiki/MQTT

[3]. AWS. (n.d.). *What is MQTT?*. Retrieved from https://aws.amazon.com/what-is/mqtt/

[4]. EMQX. (2023, March 3). *What Is the MQTT Protocol: A Beginner's Guide*. Retrieved from https://www.emqx.com/en/blog/the-easiest-guide-to-getting-started-with-mqtt

[5]. All About Circuits. (n.d.). *Internet of Things (IoT) Communication Protocols—IoT Data Protocols*. Retrieved from https://www.allaboutcircuits.com/technical-

[6]. Research AIMultiple. (2023). *Top 9 IoT Communication Protocols & Their Features in 2024*. Retrieved from https://research.aimultiple.com/iot-communication-protocol/

[7]. Barbara. (n.d.). *IoT communication protocols you should know*. Retrieved from https://www.barbara.tech/blog/iot-communication-protocols-you-should-know-about

[8]. Microsoft Azure. (n.d.). *IoT Technologies and Protocols*. Retrieved from https://azure.microsoft.com/en-us/solutions/iot/iot-technology-protocols

[9]. Zigbee Alliance. (n.d.). *Zigbee for Developers*. Retrieved from https://zigbeealliance.org/

[10]. LoRa Alliance. (n.d.). What is LoRaWAN. Retrieved from https://lora-alliance.org/

[11]. Stonebraker M. SQL databases v. NoSQL databases. Commun ACM. 2010;53(4):10-11.

[12]. Zhang Y, Yang L, Chen J. RFID and IoT: Architectures, protocols, and applications. John Wiley & Sons; 2016.

[13]. Sicari S, Rizzardi A, Grieco LA, Coen-Porisini A. Security, privacy and trust in Internet of Things: The road ahead. Comput Netw. 2015;76:146-164.

[14]. Roman R, Najera P, Lopez J. Securing the Internet of Things. Computer. 2011;44(9):51-58.

[15]. Mell, P., & Grance, T. (2011). The NIST Definition of Cloud Computing. National Institute of Standards and Technology (NIST), Special Publication 800-145. 2.

[16]. Dastjerdi, A. V., & Buyya, R. (2016). Fog computing: Helping the internet of things realize its potential. Computer, 49(8), 112-116. 3.

[17]. Sicari S, Rizzardi A, Grieco LA, Coen-Porisini A. Security, privacy and trust in Internet of Things: The road ahead. Comput Netw. 2015;76:146-164.

[18]. Roman R, Najera P, Lopez J. Securing the Internet of Things. Computer. 2011;44(9):51-58.

[19]. Siemens. MindSphere - Industrial IoT as a Service Solution. Siemens; 2021. Available from: https://siemens.mindsphere.io/en.

[20]. General Electric. Predix Platform: The Industrial IoT Platform for Digital Industrial Transformation. General Electric; 2021. Available from: https://www.ge.com/digital/predix.

[21]. Harley-Davidson Case Study. In: General Electric [Internet]. 2019. Available from: https://www.ge.com/digital/blog/harley-davidson-modernize-its-manufacturing-operations-predix

[22]. Tesla, Inc. Annual Report. 2020. Available from: https://ir.tesla.com/financial-information/sec-filings

[23]. Satyanarayanan M. The emergence of edge computing. Computer. 2017;50(1):30-39.

[24]. Tao F, Zhang M. Digital Twin Shop-Floor: A New Shop-Floor Paradigm Towards Smart Manufacturing. IEEE Access. 2017;5:20418-20427.

[25]. Christidis K, Devetsikiotis M. Blockchains and smart contracts for the Internet of Things. IEEE Access. 2016;4:2292-2303.

[26]. Hsu CL, Lin JCC. An empirical examination of consumer adoption of Internet of Things services: Network externalities and concern for information privacy perspectives. Comput Hum Behav. 2016;62:516-527.

[27]. Shi W, Cao J, Zhang Q, Li Y, Xu L. Edge computing: Vision and challenges. IEEE Internet Things J. 2016;3(5):637-646.

[28]. Rao S, Govardhan A. A study on the IoT protocols. In: 2018 International Conference on Inventive Research in Computing Applications (ICIRCA). IEEE; 2018. p. 905-910.

[29]. Thangavel D, Ma X, Valera A, Tan H, Tan C. Performance evaluation of MQTT and CoAP via a common middleware. In: 2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP). IEEE; 2014. p. 1-6.

[30]. Siemens. Totally Integrated Automation Portal. [Online]. Available: https://www.siemens.com/global/en/products/automation/industry-software/automation-software/tia-portal.html

[31]. OPC Foundation. OPC Unified Architecture [Internet]. Available from: https://opcfoundation.org/about/opc-technologies/opc-ua/ [31].

[32]. Factory I/O Documentation. (n.d.). Retrieved from https://docs.factoryio.com/manual/

[33]. Siemens S7-1500, CPU 1515T-2 PN. (n.d.). Retrieved from https://cache.industry.siemens.com/dl/files/167/81162167/att_92038/v1/s71500_cpu_1515_2_pn_manual_en-US_en-US.pdf

[34]. Nginx documentation Available from: https://nginx.org/en/docs/

[35]. ACME.sh documentation Available from : https://github.com/acmesh-official/acme.sh

ملخص :

يقدم هذا العمل دراسة شاملة عن جمع البيانات من النظم الكهربائية الميكانيكية في الميدان الصناعي مع تأمينها وعرضها. العملية مفصلة من البداية إلى النهاية، من جمع البيانات الى تخزينها بأمان في قاعدة بيانات سحابية مع نسخة احتياطية محلية. كما ان تامين خوادم تطبيقات الويب تم عن طريق بروتوكول (طبقة مآخذ التوصيل الآمنة) وجعلها متاحة عن طريق نطاق وضع خصيصا لهته العملية

بفضل إطار العمل لـ Node-RED وقدرات جافا سكربت، منحنا القدرة على التلاعب الواسع بالبيانات وإنشاء السيناريوهات لمختلف الحالات والعوائق التي قد تصادف النظام الكهربائي الميكانيكي، ومن خلال المفاهيم النظرية والمحاكات التي قمنا بها. هذا العمل يوفر مورداً قيماً للمهنيين في الميدان، حيث يبين كيفية بناء بنية أساسية متينة وآمنة لجمع البيانات وعرضها . وتقديم نظام جاهز للإنتاج ينطبق على أي نوع من التطبيقات الصناعية العالمية الحقيقية المتنوعة

كلمات مفتاحية: طبقة مآخذ التوصيل الآمنة

Résumé:

Ce travail fournit une étude complète sur la collecte, la sécurisation et l'affichage des données provenant de systèmes électromécaniques dans un domaine industriel. Le processus est détaillé du début à la fin, de la collecte de données à son stockage sécurisé dans une base de données cloud avec une sauvegarde locale. Les serveurs d'applications Web sont sécurisés à l'aide du protocole SSL et rendus accessibles par l'intermédiaire d'un domaine dédié.

En utilisant le framework Node-RED et les capacités JavaScript, nous permettons une manipulation de données et la création de scénarios étendus pour diverses situations et obstacles que le système électromécanique peut rencontrer, et grâce aux concepts théoriques et aux simulations que nous avons réalisés, Le projet offre une ressource précieuse pour les professionnels du terrain, démontrant comment construire une infrastructure robuste et sécurisée pour la collecte et la présentation des données, et fournissant un système prêt à la production applicable à n'importe quel type d'application industrielle du monde réel.

Summary:

This work provides a comprehensive study on collecting, securing, and displaying data from electromechanical systems in an industrial field. The process is detailed from start to finish, from data collection to its secure storage in a cloud database with a local backup. Web application servers are secured using SSL protocol and made accessible through a dedicated domain.

By utilizing the Node-RED framework and JavaScript capabilities, we enable extensive data manipulation and scenario creation for various situations and obstacles that the electromechanical system may encounter, and through the theoretical concepts and simulations that we've done, The project offers a valuable resource for field professionals, demonstrating how to build a robust and secure infrastructure for data collection and presentation, and delivering a production-ready system applicable to any type of real-world industrial application.