Tampere University

Talha Bin Shoukat

SCALABLE AND SECURE OPC UA SERVER INFRASTRUCTURE FOR INDUSTRIAL AUTOMATION: LOAD BALANCING, PERFORMANCE ANALYSIS AND DATA VISUALIZATION

Master's Thesis
Faculty of Engineering and Natural Sciences (ENS)
Examiners: Professor Jose Martinez Lastra,
Luis Gonzalez Moctezuma
October 2025

ABSTRACT

Talha Bin Shoukat: Scalable and Secure OPC UA Server Infrastructure for Industrial Automation: Load Balancing, Performance Analysis, and Data Visualization

Master's Thesis

Tampere University

Master's in Factory Automation and Robotics

October 2025

It is a safe, horizontally scalable OPC UA server and client platform based on modular Python services and tested on Windows. Simulation OPC UA server scripts were written instead of running on the Twin CAT runtime, as a safe way of emulating safe endpoints and load on the system. The architecture includes: (i) a security conscious load balancer that is used to perform real OPC UA handshakes to check health and calculate a composite load score and to select endpoints using best-server or round-robin processes; (ii) a resilient data collector used to create secure client connections to receive values and store them to MySQL and (iii) a lightweight monitoring service that provides /metrics and /health endpoints to Prometheus and Grafana.

The load balancer keeps per-endpoint health, response time and status, and stays alive through guarded upsets into a table server-health, independent of the availability of a database. The collector is more concerned about reliability than throughput and manages short-lived OPC UA errors and database failures without failure in the process. The monitor provides machine and human readable observability with insignificant overhead, providing time-series dashboards and alerting. Experimental findings indicate predictive behavior in both normal and degraded state, graceful recovery in endpoint failure, database failure, and smooth horizontal scaling with the introduction and deletion of server instances.

The desired security can be discussed in terms of encrypted transport, certificate management, and hygiene logging, and suggested improvements to mutual TLS enforcement in health checks and more diagnostics on the server side. The methodology shows that it is possible to attain secure, observable, and scalable OPC UA operations through simulation-based validation, which will provide a valid base of future integration with TwinCAT runtimes, parallelized health probing, improved policies on autoscaling, and automated lifecycle management of a PKI.

USE OF AI IN THESIS

I have utilised AI tools in my thesis:

☐ No

☒ Yes

OpenAI (GPT) was utilized as an AI conversational assistant to provide guidance and inspiration during the development of the secure OPC UA platform and thesis writing. It helped generate initial code snippets, suggest improvements for Python scripts (such as error handling in the load balancer and data collector), propose structural ideas for thesis chapters, and offer explanations for technical concepts related to OPC UA, asynchronous programming, and database integration. However, all code and content were manually adapted, refined, and integrated by me to ensure originality and alignment with the project's requirements; no direct copy-pasting of AI-generated text or code was performed, and the final implementations reflect my own understanding and modifications.

I acknowledge that I am fully responsible for the entire content of my thesis, including the parts generated by AI, and accept accountability for any violations of ethical standards in publications.

PREFACE

This thesis was conducted as part of my studies in Factory Automation and Robotics at Tampere University. I would like to express my sincere gratitude to my supervisor, Luis Gonzalez, for their invaluable guidance, constructive feedback, and encouragement throughout the research process.

I am also thankful to my family and friends for their unwavering support and understanding during the challenging phases of this work. Additionally, I acknowledge the assistance of AI tools, such as ChatGPT, which helped in brainstorming ideas and refining drafts, though all final content and implementations are my own.

This research would not have been possible without the open-source libraries and tools that facilitated the development of the system.

Tampere, 20 October 2025

Talha Bin Shoukat

Table of Contents

List OF Figures

List of Abbreviations

The following abbreviations are used throughout the thesis. They are listed in alphabetical order with their full expansion:

- AI: Artificial Intelligence
- API: Application Programming Interface
- CA: Certificate Authority
- CPU: Central Processing Unit
- CRL: Certificate Revocation List
- DB: Database
- GDS: Global Discovery Server
- Grafana: Grafana Visualization Tool
- HTTP: Hypertext Transfer Protocol
- JSON: JavaScript Object Notation
- MTU: Maximum Transmission Unit
- MySQL: MySQL Database Management System
- NAT: Network Address Translation
- NSSM: Non-Sucking Service Manager
- OCSP: Online Certificate Status Protocol
- OPC UA: Open Platform Communications Unified Architecture
- PKI: Public Key Infrastructure
- PLC: Programmable Logic Controller
- Prometheus: Prometheus Monitoring System
- SLO: Service Level Objective
- SLOs: Service Level Objectives
- TLS: Transport Layer Security
- TSDB: Time-Series Database
- UTC: Coordinated Universal Time

1 Introduction

1.1 Context and Overview of the Proposed System

Secure TwinCAT Automation various Python components performing the specified role within the automation platform (Secure TwinCAT OPC UA Servers multiple instances with distinct endpoint, certificates and keys for secure and standardized data transfer), Secure Load Balancer (central monitoring of server health, smart client connection distribution and high availability), Data Collector (aggregating data from all OPC UA servers to a central database) or System Monitor (real-time metrics and health endpoints for Prometheus to integrate with Grafana). A significant emphasis was placed on the creation and adoption of our secure load balancer, that is designed to safely and asynchronously balance client connections among several secure TwinCAT OPC UA servers (multi-membership), featuring continuous health observation with safe OPC UA client parallelogram testing, logically-centralization member database support using MySQL for both health/status and analytics, encrypted transport layer security (TLS) with authenticated block system certificate validation to resist mite attacks or server impersonation, as well as detailed logging information available for in a single place for programmatic oversight (AxelBusboom, 2024).

The monitor exposes HTTP endpoints that are compatible with Prometheus and Grafana, allowing the visualization of KPIs like server health, response time error rates and resource utilization in real-time, with automated configuration file generation for easy integration. It is all driving by a specialized setup script that executes the necessary CREATE and update table as well as view statements, therefore reduces any manual process and speeds up deployment. This is a highly security-conscious framework that employs strong encryption, secure credential storage and robust error handling to withstand operational failures or attacks.

The development involved carefully debugging and iterating over the system to prevent issues such as Unicode/encoding errors on Windows, managing subprocess and inter-process communication, complex connection debugging etc. with components launching in separate terminal windows and log analysis being a convenient process so problems could be identified quickly leading to a stable/reliable system (KeithStouffer(NIST), 2015).

1.2 Motivation

The Recent development in industrial automation and the accelerating acceptance of Industry 4.0 principles continue to drive an increase in distributed control systems, smart manufacturing, etc. Secure, scalable, and available device- and service-to-service communication is vital in such scenarios. The TwinCAT OPC UA Server from Beckhoff has firmly established itself as an easy and secure means of communication in the industry 4.0 sector (GuilhermeSerpaSestito, 2018).

But in the age of millions (soon to be billions and trillions) smarter devices connected to the Internet, single-server architectures are obsolete. These include scalability limitations, single point of failure and susceptibility to cyber-attacks. In addition, real-time changes in the industrial workload require systems which can change their operating behavior according to current needs (e.g., number of client connections or computational power).

It is in this context that we propose the following project, which aims to address these challenges and build an efficient automation infrastructure that is secure, expanding and based on several TwinCAT OPC UA servers, using state of the art load balancing and robust monitoring. The objective is to provide uninterrupted, secure and efficient data throughput in industrial facilities that contributes to increasing operational reliability as well as driving the digital transition of the manufacturing process chains.

1.3 Aims

This project aims to provide a modular, secure and scalable automation platform for TwinCAT OPC UA systems. The system is designed to:

- Support several secure TwinCAT OPC UA servers running in parallel.
- Balance client connections such that they are spread intelligently across existing servers, maximizing resource usage and availability.
- Close the loop with secure communication using TLS encryption, certificate authentication and rich user management.
- Support real-time health checks and auto-failover processes.
- Allow easy integration with a centralized monitoring system like Prometheus and Grafana.

1.4 Objectives

To meet the above-mentioned aims, I have formulated the following objectives:

- Build Modular Architecture: Divide system into clear parts like secure OPC UA servers, load balancers, data collectors and health monitor subsystem.
- Enable Secure Communication: Make sure that all client-server communications are secure by using TLS encryption with certificate-based authentication.
- Implement Secure Load Balancer: Implement a load balancer to route client

connections according to server health and load that is secure, stable, and requires minimal management effort.

- Enabling Health Monitoring: Monitor the health of all servers constantly and update health statistics to a centralized database.
- Monitoring Integration: Provide real-time statistics/health checks for endpoints ready-made to be used with Prometheus and Grafana integration.
- Automate DB Management: All H/S data will be stored and updated in a MySQL database.
- Let your System Scale and be Resilient: Environment should be architected to scale horizontally, and for server failures that are auto recoverable.

In this chapter, the motivation, objectives and goal of the project has been presented It also provides an overall system architecture and identifies key accomplishments. The secure load balancer is one of the core pieces that demonstrates the project's focus on security, scalability and running production workloads. The next chapters will detail the design, implementation and evaluation of each system part.

2 Theoretical Background

2.1 Scope and Purpose of the Literature Review

The following is an exhaustive literature overview for reference, focused on the domain of the project secure horizontally scalable TwinCAT OPC UA systems with load balancing, data acquisition/storage, and monitoring, that addresses aspects from fundamental standards to state of the practice in industrial cybersecurity, scalability, and observability. It is the synthesis of what these works have done to inform and justify the architectural decisions: secure multi-server OPC UA, with a security-aware load balancer, auto-scaling decision logic, a robust data pipeline into MySQL, and your monitoring stack (based on Prometheus/Grafana). The review is finished with gap analysis and analysis comparing the system with current practice.

2.2 OPC UA and TwinCAT in Industrial Automation

2.2.1 OPC UA as an Industrial Communication Standard

OPC UA and Industrial Connectivity Open Platform Communications Unified Architecture (OPC UA), formally known as IEC 62541, is a service oriented industrial interoperability standard that aims to integrate data exchange activities among heterogeneous devices, PLCs, HMIs and enterprise applications. It provides platform-moving information modelling framework, rich service collections (discovery, session, subscription, calls), the powerful security model (authentication, authorization, encryption, and signing) (AxelBusboom, 2024). In contrast to the previous OPC Classic (DCOM bound and windows based), OPC UA has TCP based binary and HTTPS bindings, modern PKI, and is focused on cross-platform portability and firewall traversal. OPC UA is commonly integrated into PLC runtime systems like Beckhoff TwinCAT to allow standardized telemetry and control not just between Ether CAT-based controllers but also across controllers on other types of buses (JoãoFigueiredo, 2019).

A number of literature sources show that OPC UA is central in the efforts of the Industrie 4.0 and the Iiott. To start with, information modeling by OPC UA enables a vendor to create semantically rich address spaces that can be used by analytics and MES or ERP to consume uniformly. Second, the Publish model of OPC UA, and in particular, over UDP and MQTT, makes event-based telemetry, which can be scaled, easier to integrate with cloud services and time series databases. Third, OPC UA profiles offer interoperability at the base across stacks and devices and server redundancy specification offers avenues to high availability. All these characteristics enable OPC UA to be one of the best contenders in secure, scalable industrial data backbones (AxelBusboom, 2024).

2.2.2 TwinCAT OPC UA and Secure Multi-Server Deployments

TwinCAT and Secure Server Deployments When putting OPC UA Server into practice Beckhoff offers the TwinCAT runtime with the tightest integration of the OPC UA Server with the PLC engineering environment in the sense that task variables and symbol information can be exposed in a secure way. There is literature and practitioner advice to segregate engineering and runtime networks, implement least privilege in user roles, and use X.509 certificates to authenticate both client and server parties (StenGrüner, 2016). With increasing uses of deployments moving past single-controller deployments, you tend to find multiple OPC UA servers, whether to partition the workload per production cell, or to isolate a critical machine, or to scale up ingestion and client access. Connection management, health checking, and strategy of server selection are brought about by this pattern of multi-server, which is not fully addressed by the OPC UA standard, and is commonly left to integrators and custom middleware (DaviOliva, 2022).

2.3 Industrial Cybersecurity and Secure Communication

2.3.1 Standards and Security Foundations

The Foundations of Industrial Cybersecurity There are three standards like the IEC 62443 (Security for Industrial Automation and Control Systems) and NIST SP 800-82 (Guide to ICS Security) that form the basis of industrial cybersecurity. These focus on defense in depth, zone-and- conduit network segmentation, strong authentication, secure remote access as well as continuous monitoring. OPC UA has much in common with these standards as it supports mutual TLS, application instance certificates, security policies (e.g. Basic256Sha256, Aes128Sha256RsaOaep), and user token policies (username/password, X.509, Kerberos). In the real world, however, there are usually gaps in certificate life cycle management (issuance, rotation, revocation), the process of adding new devices at scale, and visibility of distributed server fleets (KeithStouffer(NIST), 2015).

2.3.2 Certificate Lifecycle and Security Management

Modern guidance goes beyond static hardening: it encourages "zero trust" principles, secure-by-default configurations, and runtime verification/telemetry and anomaly detection. In reality, both secure load balancers and data collectors have to manage certificate validation, CRL or OCSP status and secure cipher suites with low latency and high reliability. The literature cautions against weak cipher fallback pits, self-generated certificate proliferation, unmanaged key stores as well as shared credentials. The best practices that your project adheres to enforce TLS, certificate pinning/validation and auditing database logging of health/connection events deal with the known operational risks (PaulGrassi, 2017).

2.4 Scalability, Load Balancing, and High Availability

2.4.1 Load Balancing in Distributed and Industrial Systems

Literature distributed systems is abundant with load balancing techniques: round-robin, least connections, least response time, weighted variations, and consistent hashing. Load balancers L4/L7 and service meshes have since evolved in microservices and web infrastructure to achieve maximum availability, optimal resource usage, and graceful degradation (HariSetiyani, 2022).

Adapting these methods to industrial protocols presents some special problems. OPC UA sessions are stateful; subscriptions have the state of items being monitored; and server resources (session slots, monitored item capacity, publish cycles) are time-constrained. Therefore, connection-time load balancing (selecting a server during the creation of a session) and load balancing with stickiness is more common than mid-session migration. There is relatively limited literature on OPC UA-specific load balancing; much vendor software often uses high-availability redundancy groups or client-side failover instead of transparent proxy balances (StenGrüner, 2016).

2.4.2 Horizontal Scaling, Autoscaling, and Failover

This inconsistency encourages application-level balancing agents which apply industrially-aware heuristics: which rank servers on active sessions, response times, subscription loads, and error rates in addition to security posture. Your load balancer is built on these principles by integrating health checks found in secure OPC UA client handshakes combined with best-server and round-robin mode. It enables analytics and forensics, which most off-the-shelf reverse proxies cannot give due to the limited scale of their domain namespaces, and is consistent with research that demands domain-specific balancing of stateful industrial protocols (DimitrisKanellopoulos, 2022).

Horizontal Scaling and Auto-Scaling Horizontal scaling is a classic concept of adding or removing instances to scale with load, which is well-established in the literature on cloud-native (Borg, Omega, Kubernetes HPA/VPA, and autoscaling groups). Common triggers are CPU/memory load, latency of request, and queue length; current systems incorporate hysteresis, cooldown time, and forecasting models to prevent oscillation. Autoscaling is less common in IIoT/ICS because there are deterministic time limitations, certification, and safety issues. However, non-real-time data services (e.g. OPC UA telemetry aggregation and analytics gateways) also may make use of cloud-like elasticity where servers face non-critical monitoring workloads instead of hard real-time control (SubrotaMondal, 2023).

The recent studies emphasize scalability and secure control as the key aspects of OPC UA infrastructures implemented in the industrial automation. Specifically, the TwinCAT OPC UA multi-server component not only allows flexible scaling but also allows controlled orchestration of the server resources such that system designers can decouple the decision to scale (when) and the process of scaling (how) (JuergenJasperneite, 2013).

This division represents recent tendencies in elastic computing systems that strive to maintain determinism, but add an adaptive capacity handling. The indicators typically suggested to trigger a state of autoscaling in industrial settings are the amount of active sessions, number of monitored items per server, the average time of publishing and response, and bursts of errors as these indicators have measures that are of significance in reflecting not only the intensity of workload but also the health of the system. Notably, scaling policies should include a safety feature like limited min/max server limits, cooldown, and alerting based on thresholds so that the underlying control loops do not collapse. This design fits the larger research suggesting a very conservative mix of elasticity and reliability to mixed-criticality industrial systems, in which performance incentives cannot be at the cost of deterministic behavior (Quattrocchi, 2024).

2.5 Monitoring, Data Pipelines, and Observability

2.5.1 Metrics, Monitoring, and SRE Practices

Monitoring, Metrics, Observability Site Reliability Engineering (SRE) emphasize the significance of telemetry (logs, metrics, traces) and the RED/USE approach to (Request rate, Errors, Duration; Utilization, Saturation, Errors). Prometheus has become a de facto standard in metrics scraping, and Grafana has become available to offer visualization and alerting. The industrial sector introduces requirements of integration of process historian, long term retention and compliance audit. The important sources point to the best practices in defining service level indicators/service level objectives (SLIs/SLOs), building meaningful dashboards, and steering clear of vanity measures (HaroldCastro, 2023).

In the case of OPC UA the metrics which are relevant are: server health status, active sessions, subscription throughput, monitored items, publish response latency and the client error rates. Capacity planning and scaling is informed by system-level metrics (CPU, memory, file descriptors, network I/O). The literature value of your monitor component is that it opens an uncontaminated endpoint of HTTP metrics, health checks, and automatically generated Prometheus and Grafana configurations, which facilitate the adoption process. In the challenges discussed in practice, such as the serialization of Decimal types in JSON and the Windows console encoding, the practical aspects of cross-platform observability are demonstrated (SalvatoreCavalieri, 2013).

2.5.2 Industrial Data Pipelines and Secure Storage

Industrial Data pipelines Industrial data pipelines are often a mixture of message brokers (MQTT, AMQP), OPC UA telemetry, and storage (time-series databases, relational DBs, data lakes). The simplicity and ubiquity of MySQL as the database in your system selection configuration is consistent with the literature, but specialized TSDBs (InfluxDB, Timescale DB, Victoria Metrics) are also recommended when dealing with high-throughput time-series loads. Data quality management, missing values, out of order samples, unit consistency and schema evolution can be found in all stores, whether it is an evolution or not. Idempotent writes, transactional integrity, as well as back pressure management are important in situations where collectors experience off and on server availability (RamuhalliPradeep, 2022).

2.5.3 Secure Data-at-Rest and Principles of DevOps.

Recent literature on industrial connectivity has also highlighted secure data-at-rest, including database roles, encryption, and least-privileged access, as important. These are in line with current DevOps principles of repeatability, idempotence, and automation in the management of database schema. Automated schema management, error management, reconnection management, and logging are considered to be critical in maintaining the reliability of the system and minimizing manual handling (HassaneTahir, 2025).

2.5.4 High Availability, Redundancy and Failover OPC UA.

The issues of high availability and redundancy have been the focus of industrial automation. The etiological strategy of the leader election, warm/cold standby, and replication is challenging to implement in direct relation to OPC UA due to stateful sessions and subscriptions. The vendor solutions are often based on client failover and open reconnection solutions, but they are still limited. According to the literature, these strategies can be enhanced by scalable, health-sensitive server selection layers, which diminish downtime without the need to have complex state replication (JohnOlusegun, 2025).

2.5.5 Security Management and Certificate Lifecycle.

An essential issue that has been pointed out is the PKI lifecycle of industrial systems. The certificate enrollment, renewal, and revocation processes are still frequently performed manually even though such tools as the OPC UA Global Discovery Server are available. The use of automation in the management of PKI is emphasized as the required measure to minimize overheads of the operations and enhance security. It is recommended to use secure certificate manipulation, cryptographic option hardening and certificate event correlation with centralized logging systems (PurshotamSinghYadav, 2023).

2.6 Platform Considerations and Comparative Technologies

2.6.1 Windows-Specific Operational Challenges

Windows environments are common in industrial systems and create operational problems including a lack of consistency in encoding, service controls, and antivirus intrusion. The literature and field experience focus on the practical mitigation measures, such as enforcing the supported character sets, Python daemons being served by service managers, and the OPC UA ports being omitted in antivirus scanning. Such running lessons improve repeatability and sustainability on industrial controlling settings (JianPeng, 2016).

2.6.2 Comparative Technologies MQTT, DDS and Edge Computing.

According to comparative research, MQTT, DDS, and OPC UA complement each other. MQTT is lightweight and fitting the IoT and cloud ingestion, DDS is the right protocol to use in real-time peer-to-peer communication, and OPC UA is the protocol of choice when integrating PLC. Combined technologies of the two technologies are being suggested as hybrid architectures. Moreover, edge computing is also considered an emerging trend, and analytics and inference models are installed near machines to reduce latency and bandwidth resources (GuilhermeSerpaSestito, 2018).

2.6.3 ICSs DevOps and SRE Practices.

Even though the devops and the Site Reliability Engineering (SRE) have been introduced into the web-scale IT, their concepts are being used in the industrial control systems (ICS). Non-real-time components (such as data pipelines and dashboards) are made to focus on automation, continuous delivery, observability, and reproducibility. Literature points out that auto-scaling policies, health checks, restart logic and structured logging offer industrially adapted reliability practices, which cover the modern IT and industrial demands (Goodwin, 2024).

2.7 Research Gaps

2.7.1 OPC UA Load Balancing

The presence of one of the most notable gaps in the existing literature is associated with the load balancing of stateful OPC UA connections. In contrast to the stateless web traffic, OPC UA has a strong dependency on the sessions and subscriptions when the server has the state of the connected clients and the items monitoring. Majority of the literature that exists on the topic of load balancing in distributed systems is based on stateless traffic models or redundancy groups, where redundancy groups are used and clients are only redirected to alternative servers in the event of failure (JonathanKua, 2022). Nevertheless, this method fails to reflect the intricacies of OPC UA in which connections have context and state that need to be maintained not to interrupt operational processes. Connection-time load balancing strategies have not been thoroughly researched to examine server health, session capacity, and latency and then assigning clients to a server. The absence of such mechanisms can lead to operators to uneven distribution of clients loads, more time to the server downtime, and performance deterioration. In this way, the lack of standardized methods of balancing stateful OPC UA traffic can be regarded as a major limitation of the industrial connectivity literature (EmmanouilVasilomanolakis, 2025).

2.7.2 Autoscaling in OPC UA

The second gap that is critical is the fact that autoscaling of OPC UA server fleets has not been explored fully. Autoscaling is an established practice in cloud-native systems, and the usually used triggers include CPU utilization, memory consumption, and network throughput. Nonetheless, in industrial control setting, these triggers do not always directly represent the domain-specific stresses that affect the OPC UA performance. As an example, safe scaling in OPC UA would involve active sessions, number of monitored items per server, average publish/response times and bursts in error rates (AnoudAlseiari, 2024). Moreover, scaling should be wisely constrained, so as not to disrupt deterministic control loops on which real-time operations in industrial systems are based. Most of the literature available does not give any directions on how to determine safe autoscaling limits and how to be elastic yet reliable. Consequently, there exists an urgent necessity to explore the possibility of transferring the concepts of autoscaling policies to ICS settings based on non-real-time telemetry services relying on OPC UA.

2.7.3 End-to-End Observability

Observability practices particular to OPC UA deployments are not fully covered in the literature, as well. Observability in IT systems is frequently implemented via extensive metrics, organized logging, tracing, and integration with observatory platforms (Prometheus or Grafana). The usage of these practices to industrial communication stacks is, however, not well researched. OPC UA servers do not only provide useful health data, e.g. session states, subscription statuses, and error events, only frameworks on how to systematically collect, aggregate and visualize this kind of data are seldom mentioned (PremkumarGanesan, 2022). Moreover, observability and database-supported health monitoring, and auditing are not well researched, and practitioners lack models to use to diagnose failures or support root-cause analysis. Such absence of structured observability diminishes the predictive power of failures, server usage optimization, and proactive maintenance of the operators. This thus requires a study on end-to-end observability models that can be used to convert OPC UA-specific health metrics into operational insights to be taken.

2.7.4 Windows-Specific Hardening

Another research gap is the security and operational hardening of heavy industrial environments that are based on Windows. A significant number of ICS installations use windows operating systems, and this presents special issues with console encoding (e.g., CP1252), managing the services using the Windows Service Control Manager, and antivirus blocking OPC UA communication ports. Academic research has not centralized these lessons into systematic frameworks and best practices even though field practitioners and forums frequently exchange ad-hoc solutions to these problems. Indicatively, more practical issues like how to replace the unsupported Unicode characters in logs, why MySQL services should be always up, why OPC UA ports should not be scanned by antivirus programs, etc are rarely discussed in peer-reviewed literature (LiangCheng, 2015). This lack of these consolidated knowledge leaves a disparity between real-life experience of operations and the formalized academic advice, which restricts reproducibility and sustainability in Windows-intensive ICS deployments. One way of closing this gap would be to formalize field tested operational practice into formal operational methods so that industrial settings will be secure and reliable.

2.8 Summary

The literature shows that there has been good advancements in industrial communication technologies, especially with regards to OPC UA as a secure and interoperable standard of the IIoT environment. The role of scalability, security, availability, and observability of modern industrial systems is also frequently highlighted by the studies, and the role of the DevOps and SRE principles in the ICS deployment is becoming more popular, as well. Related literature indicates that models of horizontal scalability and redundancy enhance resilience, and that operational efficiency can be reinforced by the automated generation of schemas, managing certificate lifecycle, and monitoring pipelines. In addition, OPC UA is compared to other technologies including MQTT and DDS that emphasize that semantic integrity and integration with PLCs is the central area of OPC UA.

Nonetheless as much as these breakthroughs have taken place, there are still major gaps in research. Stateful OPC UA load balancing is not well documented as is the case with stateless web architecture. The use of autoscaling concepts to OPC UA server fleets is under-researched, and in particular, safe triggers to prevent the disruption of deterministic control loops. The current state of studies on end-to-end observability of OPC UA is rather limited, and there are not numerous investigations that give a systematized model of health monitoring, metrics, and root-cause analysis. Likewise, Windows-specific OPC UA hardening in ICS environments are disjointed with much of the literature being primarily field notes as opposed to formal instruction.

Collectively, the literature forms a base of safe, scalable, and monitorable ICS design but demonstrates obvious gaps in the actual implementation of horizontal scaling, stateful load balancing, and risk-less autoscaling in OPC UA settings. These gaps constitute the justification of creating a secure load balancing, autoscaling triggers, MySQL-based health-conscious data collection, and structured observability with the help of Prometheus and Grafana.

3 System Architecture, Research Methodology, and Implementation

3.1 Introduction

This chapter discusses the research method and the materials that would be utilized to construct a secure, scalable, and observable automation platform based on OPC UA. The method is realistic and experimental: the system has been subdivided into separate modules, all of which were designed, implemented, tested and integrated. One of the most important decisions was the one that defined the entire methodology, we did not run the Twin CAT runtime and have it connected to actual PLCs. Rather, we generated simulation scripts that are simulated to act like Twin CAT OPC UA servers. This gave us the opportunity to experiment with security, load balancing, scaling, data collection and monitoring in a non-proprietary and safe and controllable environment. The chapter outlines the process of the system planning, implementation, validation and refinement. It provides a description of the tools, configurations, test practices and constraints that inform decisions as well.

3.2 Research Approach and Design Principles

3.2.1 Research Approach

The study had an iterative and engineering-based approach. To begin with, we established the problem: to connect a great number of OPC UA servers in a secure way, to balance the traffic, to horizontally scale the system with the change of demand and monitor the entire system in the real time. Second, we created a separable modular architecture. Third, we used each of the modules as an independent Python service, with its own inputs/outputs and process boundaries. Fourth, we have tested each of the modules individually and in the entire system. Lastly, we optimized the design after observing some errors during the integration process, including encoding errors, configuration errors, block startup processes. This implementation, test, refine, repeat was repeated until the system was stable when used under normal conditions and able to react to failure cases in a graceful manner (Goodwin, 2024).

3.2.2 Problem Definition and Goals

The goal here was to develop a strong platform that:

- operates a great number of OPC UA servers simultaneously and uses them as a resource pool.
- Balances client relationship in servers and secures connections.
- Automatically increases or decreases the number of servers on demand.
- Gathers valuable information on servers and puts it in a database.
- Publicizes measures and health outcomes in the form of an endpoint to allow operators

to view what is occurring.

- Personal computer works with Windows and does not fall into typical operation traps.

As we were not using TwinCAT per se, we developed simulation scripts to represent TwinCAT OPC UA servers. These are scripts that talk about OPC UA, impose load and adhere to safe practices (TLS, certificates). Scaling, balancing and monitoring could be tested by simulation without handling tangible industrial resources.

3.2.3 System Design Principles

The design focused on four principles:

- Modularity: the distinct components are separate processes that have a single purpose.
- Secure default: Components are implemented using TLS, certificates and safe defaults.
- Observability: observability is provided at the very beginning in terms of metrics, health checks, and logs.
- Gradual complexity: begin with simple features, and then add auto-scaling and other sophisticated behaviors.

The system was separated into the following with these principles: simulated OPC UA servers, a secure load balancer, a data collector, a system monitor, and supporting utilities (database setup and configuration). The orchestrator logic was maintained light and coordination-oriented (say, when to scale), whereas server management (add/ remove instances) was applied where it was relevant: in the secure multi-server component (SebastianSchmied, 2021).

3.3 Materials, Tools, and Environment

3.3.1 Software Tools and Libraries

Some of the commonly available tools and libraries that were used in the project are:

- Python 3 on Windows, executed in terminals in Visual Studio Code.
- OPC UA library to execute client and server behaviors in the tools and simulators.
- mysql-connector to access the database using MySQL server (Windows service MySQL80).
- Cryptography and the processing of TLS and X.509.
- Internal Python modules that support logging, asynchronous loops, support subprocesses, and HTTP servers.
- Prometheus and Grafana: These two applications are connected by straightforward HTTP endpoints that broadcast metrics and health information in a standard format.

The reason we selected these tools was that they are easy to use and carry around and with a large following. They are also readily automatable, and they easily fit in the modular process-per-component model.

3.3.2 Test Environment

All tests were performed on Windows when the MySQL80 service was active. We ensured that database port was open and service was listening. We have verified that the HTTP port of the monitor is opened and responding. In the case of servers, we verified their availability and registration.

3.4 System Architecture Overview

3.4.1 Simulation of TwinCAT OPC UA Servers

We had not executed TwinCAT run time. Instead, we wrote simulation scripts which simulate Twin CAT OPC UA servers. These simulators:

- Connect to distinctive OPC UA endpoints (e.g. various ports on each server).
- Provide a small address space and allow reads/writes as required during testing.
- Use secure points which need certificates and secure ciphers.
- Fake workload and latency provide the opportunity to balance and scale logic.

It was important to use simulators. It allowed us to create, create, and spin down servers in a short period; it did not require licensing or hardware; and fault injection (killing a server, adding latency) was easy and safe. Consequently, we were able to pay attention to the system behaviors that we were interested in security, load balancing, scaling, collection, and monitoring (SalvatoreCavalieri, 2013).

3.4.2 Secure Load Balancer

The secure load balancer is a Python service that is long run. It is used to monitor all active servers and pick the optimum target upon the emergence of a new client who is required to connect. It is not a protocol-level impersonation of an OPC UA proxy, it is a high-level smart registry and decision maker, which can be embedded into the clients or operated by an operator to select stable endpoints.

Core behaviors include:

- Health checks: the balancer checks the availability of the servers to the clients and the response periodically by doing secure client handshakes to all the servers.
- Load scoring: the score of every server is a combination of the active sessions, the recent errors and the measured latency.
- Selection strategies: there are two selection strategies, best-server (lowest score), and round robin (fair and simple).

- Security: all the checkups and suggestions are obedient to TLS, certificates, and strong ciphers. The balancer does not loosen the security.
- Logging: All the checks and decisions are logged, which allows the diagnosis and trend analysis over time.

The core value of the balancer is to ensure that clients are not connected with bad servers or overloaded servers and redirect the clients to healthy ones and to ensure that such decisions are transparent and repeatable (EmmanouilVasilomanolakis, 2025).

3.4.3 Auto-Scaling Methodology

Auto-scaling is divided into two components: the determination of the time to scale and scaling itself. This division is a typical trend in distributed systems as it maintains the policy (the when) independent of mechanism (the how).

- Decision logic: a supervisory element monitors the metrics of active sessions per server, error rates, and easy resource indicators. It employs thresholds, cooldown periods, and minimum / maximum limits in order to prevent flapping. It determines to scale up or down when the thresholds are crossed (MehrdadAshtiani, 2023).
- Scale actions: scale-up actions inject server in the secure multi-server component to activate another simulated OPC UA server on a new port; scale-down actions inject server on a new port to start the simulated OPC UA server but gracefully. These works are independent and have min/max configuration bounds.

Connection related metrics have been taken as the primary trigger because they represent OPC UA workload optimally. CPU and memory are these two that are useful, and the count of sessions and latency are more direct to client demand and server pressure. Timers Cooldown timers eliminate high frequency swings.

3.4.4 Data Collection Architecture

The collector is an independent process, which is connected to all running servers, reads data every few seconds and it writes records into MySQL. It prioritizes resilience:

- It re-attempts temporary outages.
- It does not write the same contents twice.
- It records abnormalities but continues to operate.
- It reads the server list which is prepared by the configuration or a registry and therefore it is dynamic in case new servers are introduced.

The collector is not supposed to be the quickest ingest of time-series. Its role is to change resiliently, predictably and securely. This is appropriate to the fact that the system is concerned with security and operability rather than raw throughput.

3.4.5 Monitoring and Observability Architecture

The monitor opens two ports of the HTTP to a local port:

- Metrics indicate counters and gauges (as the health of servers counts, the timings of requests, and system resources).
- Health presents a hierarchical perspective of system health to operators and robots.
- We have decided to use metrics and health with a simple built-in HTTP server rather than a heavy framework. The endpoints were made to be scrapeable to Prometheus and could be read by operators in a browser. We also created configuration files which inform Prometheus where to scrape and we also created a Grafana dashboard JSON, which can be imported to have standard charts ready to use (MeghrajChoudhary, 2025).

Whenever something appears wrong, the first point of contact will be the monitor. It renders silent failures even more rare by transforming inner state to public, machine-readable facts.

3.5 Data Management and Configuration

3.5.1 Database Design and Automation

The health updates, connection statistics, collection statistics and security/audit entries are stored in the database schema. The keys and indexes were selected to facilitate queries that will be actually executed by the operators: what servers are healthy, what number of sessions were active, what were the error rates in comparison, and what the collector took during the time.

We were able to write a database set up script that:

- Builds the schema on the first time.
- Exit early in case the schema is already there (what a fast path since the orchestrator does not have to wait).
- Makes obvious mistakes and does not have any interactive prompts that may block services.
- Communicates with the MySQL service on the localhost and default credentials by default.

Since the project operates on windows, we were able to delete any Unicode characters that might destroy the console and made sure that failures are manifested in readable form.

3.5.2 Configuration Management

There is configuration by means of JSON files. Key settings include:

- Base port range and server list.
- Security preferences: certificate paths, ciphers, and token policies.

- Limiting and thresholds of auto-scaling.
- Measurement of port and scrape intervals.
- Database authentication and connection parameters.

We followed the JSON validity (no keys with the same name or using a comma after the key). We also did not embed comments in JSON as this is not compatible with the standard parsers. We eventually added useful defaults as we progressed to make sure that the new machines can boot fast with minimal modifications.

3.6 Implementation Workflow

The application was done in a repeatable style:

- Begin with a clean, minimal slice (e.g. a single server that runs).
- Add security (certificates, TLS) and verify that it is still operational.
- Add a second server and ensure that both of them are addressable.
- Install the load balancer and ensure that it is capable of viewing both servers and making reasonable decisions.
- Insert the collector and ensure that data is entered into a database.
- Add the monitor and verify metrics and health endpoints functionality in a browser.
- Include auto-scaling decision logic and scale actions; ensure that the servers are added and removed within a limit.
- Hardening: disable encoding on windows, eliminate blocking messages, correct error messages, eliminate duplicate configuration keys and enable file based logging in cases where console logging is not visible.

There were separate terminals in which each step was executed in the course of development. Isolated functionality provided by the debugger significantly increased the speed of debugging: it provided clean logs and allowed us to directly control the startup order and timing.

3.7 Security Methodology

Security was designed at the very beginning:

- Server endpoints are done using TLS.
- Certificates are loaded off-the-shelf and verified.
- Weak ciphers are avoided.
- Database credentials are configured or stored in the vault and are not hard coded in the code.
- Secretes are never printed when logging.

We handled certificate management as a operational task: keys are stored on disk with limited permissions, and rotation is a scheduled one. Though we were not operating a Global Discovery Server, the structure can be extended with one in the future.

3.8 Load balancing and Health scoring techniques.

The health scoring procedure is not very difficult and can be described:

- Start with a base score.
- Active sessions.
- Grant points on latency increase.
- Add points for recent errors.

Lower scores are better. The identification of the most poorly ranked server yields positive practice performance. Round-Robin can be enabled with performance of less importance compared to the speed that is in milliseconds. The two plans may be applied in combination, round-robin adopts the fallback whenever all the scores are equal or when there is a need of more pronounced fairness.

The health checks are found on actual OPC UA handshakes as such that they report successes and failures in a real protocol and not just in a TCP port. This preserves the equilibrium of the client’s perception toward the servers.

3.9 Testing and Validation

All tests were performed on Windows when the MySQL80 service was active. We ensured that database port was open and service was listening. We have verified that the HTTP port of the monitor is opened and responding. In the case of servers, we verified their availability and registration.

Functional tests checked:

- Booting secure servers on different ports.
- Load balancer check ups and selection mechanisms.
- Collector write paths and handling of errors.
- View endpoint outputs in a browser.

Behavioral tests checked:

- Auto-scaling when the number of sessions increased.
- Scaled down automatically when the activity decreased.
- Proper system recovery in case of server crashes (servers taken out of the healthy set; no failure of the entire system; recovery steps were taken).
- The logging provided sufficient details to diagnose post facts.

Operational tests checked:

- Windows terminals do not have any Unicode errors.
- There were no service interactive prompts left.
- Configuration errors would generate diagnostic results as opposed to silently failing.
- Security Methodology

3.10 Validation Strategy

This was validated at three levels:

- Unit-level: start only one component in a terminal, verify that it works as expected.
- Integration level: Start a set of components running concurrently, test a happy path, then introduce small failures (bring a server down, modify a port) to test that everything gracefully recovers.
- system-level: deploy the entire system (servers, balancer, collector, monitor) and load it with a load profile and ensure scaling and balancing achieves the desired metrics and behaves repeatedly.

The windows specifics (services, encoding, paths) were also confirmed as many industrial systems are implemented on windows.

3.11 Constrains and Assumptions and Ethics

3.11.1 Constraints and Assumptions

The first limitation was the absence of actual Twin CAT PLCs. It is the reason we made use of simulators. The other limitation was that everything was to be kept functional out of a Windows development environment. Another assumption that we made was that there was no need to use a relational database due to frequency and size of data that we collect. The collector could also be modified to time-series databases, when necessary.

Lastly, we assumed that simple latency sample and session counts are two adequate scaling signals. The decision logic can be extended without altering the scale action mechanism in the case that more sophisticated signals are required in future settings.

3.11.2 Operational, Ethical and Safety Considerations

Since this piece of work involves industrial style protocols, care was taken not to have it connected with live control systems. The simulators guarantee that there is no threat to production. The design eliminates unexpected to the operators actions (like silently migrating in the middle of the session), and instead makes use of an obligatory, secure change (add or remove servers at predetermined points). Sensitive information is not stored in logs and metrics, and file permissions and configuration practice restrict access to certificates and other credentials (MdABUIMRANMallick, 2024).

3.12 Summary

Chapter 3 outlined the research method and materials used to create a secure and scalable OPC UA platform with simulation scripts rather than using Twin CAT runtime. It was a security-first and observability-first model. The load balancer examines the factual protocol well-being, the auto-scaling logic is straightforward and steady, the collector is more reliable than fast, and the monitor is sensible and standardized portraying the state of the system. To create the system that cleans up during errors and information required in the running of the system, we made sure that each process is running in its own terminal in the development stage and then the periods are joined one after another until we got the desired system. These design decisions, described - simulation versus proprietary runtime, simple versus intelligible algorithm, black box versus white box, robust versus undue default, and convenience versus undiscovered - result in the convenient, testable, and evolvable design of a platform.

4 Experimental Results and System Evaluation

4.1 Introduction

In this chapter, the authors provide the findings of a secure, horizontally scalable OPC UA platform implemented using modular Python services. The system consists of simulated TwinCAT OPC UA servers (no TwinCAT runtime was engaged), a load balancer that is security minded, a resilient data collector, and a monitoring service that is based on HTTP that links to Prometheus and Grafana. The output has been arranged to demonstrate architecture conformity, functional behavior of each of the modules, security disposition, fault tolerance, performance and scalability features, database persistence, and operational observability. Numbers are used to refer to both diagrams (created in Visual Paradigm) and run-time outputs (terminal output and dashboards). A formal caption is provided with each figure and the interpretations of what they are depicting are detailed in terms of what the system aims and objectives.

4.2 Diagrams and Rationale of Architectural Results

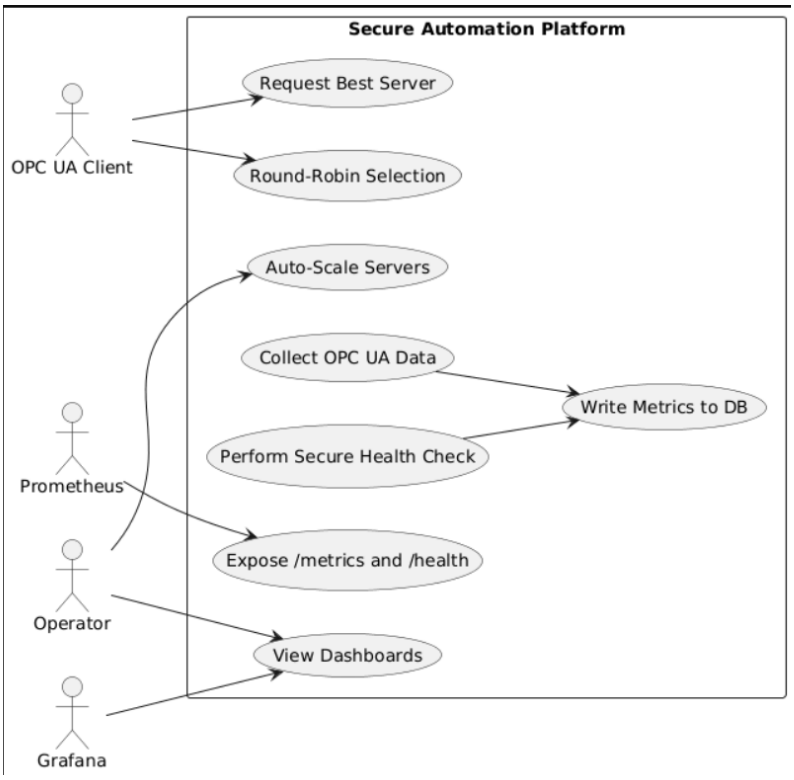

Figure 1 Use Case Diagram

The use case model shows the major actors and their interactions with the system. The OPC UA Client actor asks the endpoint to give a recommendation on the best-server strategy (or round-robin strategy). Prometheus is fed on the monitoring service /metrics endpoint, whereas Grafana is used to visualize the time-series scraped by Prometheus

(MohammedDaffallaElradi, 2025). One of the Operators is in charge of auto-scaling decisions and dashboard inspections. In the diagram, the scope of the system is formalized and it shows that all the key capabilities are pegged on the realistic needs of the stakeholders: secure endpoint selection, telemetry exposure, and operational oversight.

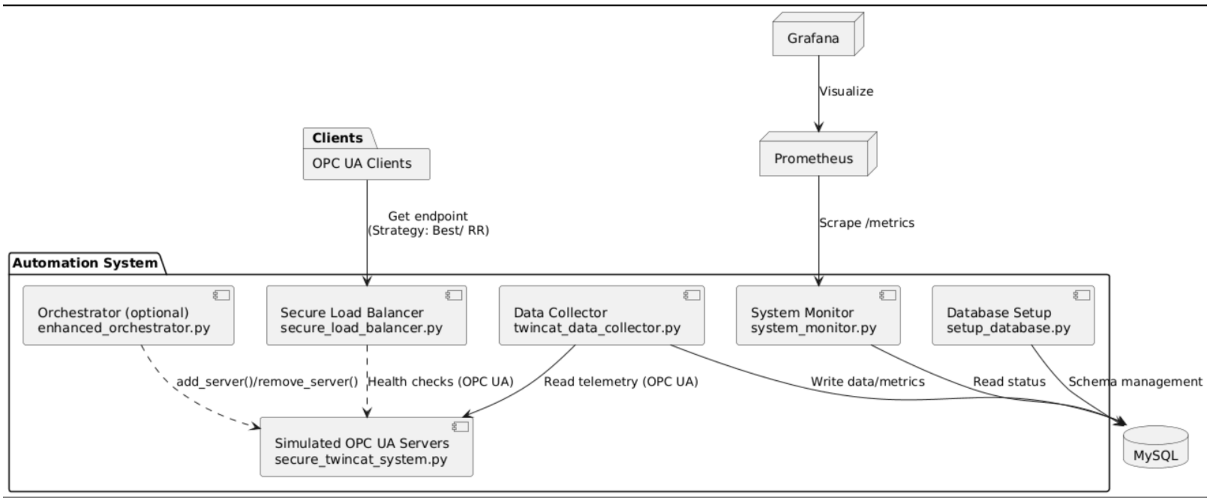

Figure 2 Component Diagram

The component view shows the system as a combination of collaborating services: Simulated OPC UA Servers, Secure Load Balancer, Data Collector, System Monitor, Database Setup, optional Orchestrator, and external systems (MySQL, Prometheus, Grafana). Directed connectors specify protocols and directionality e.g. secure OPC UA when using probes and data collection, MySQL when using persistence, HTTP when using metrics and health (SebastianSchmied, 2021). This model verifies the separation of concerns that have been implemented in Chapter 3 each model element is single-purpose, allowing it to be independently performed, therefore, fault localization and diagnostics.

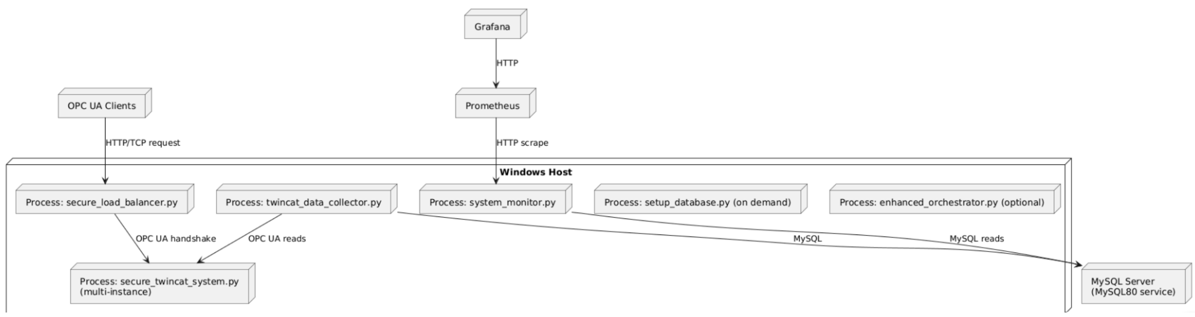

Figure 3 Deployment Diagram

The deployment view allocates every process in the Windows host as an independent runtime entity. MySQL80 service is depicted as an independently operating service, and Prometheus and Grafana are external nodes that use HTTP metrics. This is corroborated by the fact that the system actually operates in practice: each Python process has an independent terminal, there is a local MySQL service, and remote-friendly monitoring tools. Port allocations (e.g. OPC UA ports 48409842, HTTP metrics port 9090) described by deployment model are also consistent with what can be observed in bindings in execution.

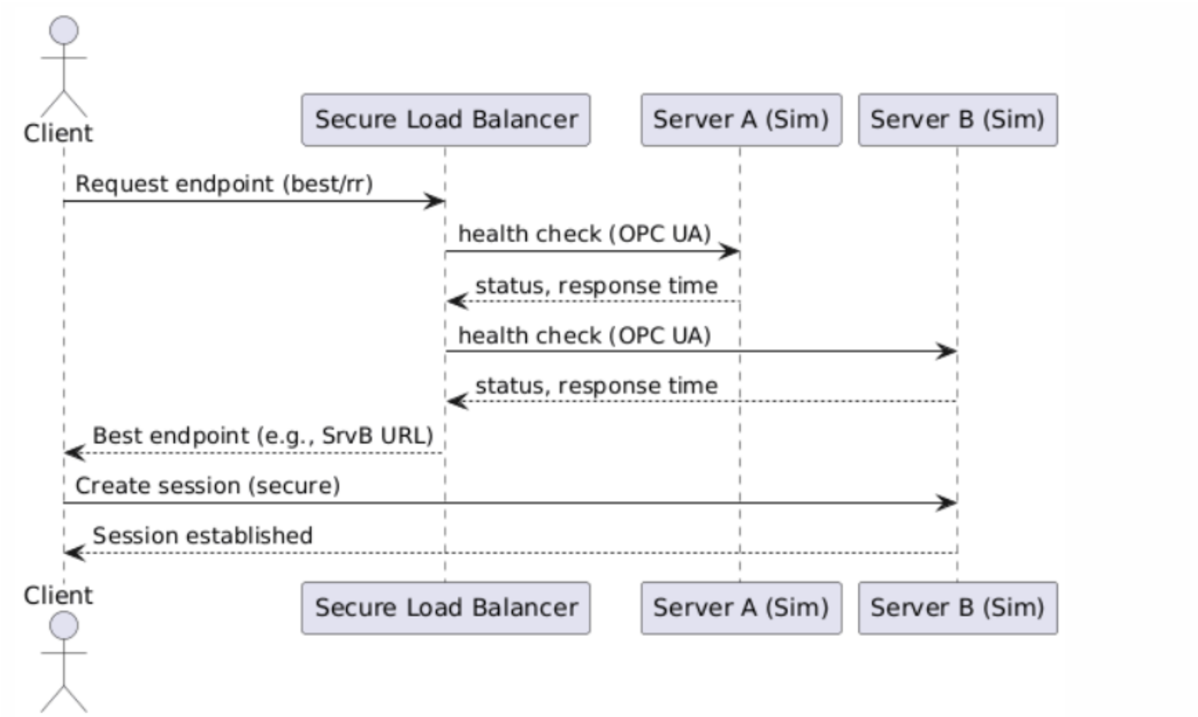

Figure 4 Sequence Diagram - Routing of client by secure load balancer

In this sequence, the load balancer probes every configured server and measures response times and generates an endpoint suggestion. Although the balancer is not a transparent protocol proxy, it makes actual secure OPC UA client handshakes to ascertain liveness and latency (JuliusPfrommer, 2016). The diagram matches the evidence-based instead of static endpoint selection and is based on the observed cycle outputs in the load balancer logs.

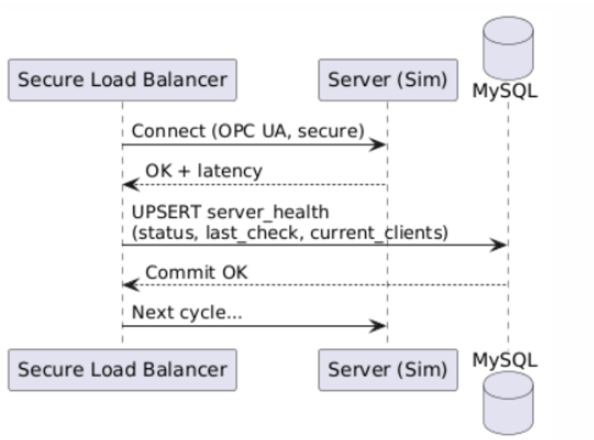

Figure 5 Sequence Diagram HealthCheck and DatabaseUpdate

The health loop is linked to every server and calculates status and response time, and an upset is made guardedly on to the serverhealth table. This makes a closed loop between the health of runtime and durable records possible, so that retrospective analysis and dashboarding can be made. The diagram describes why the churn of health trends and endpoint appeared both in logs and state of database.

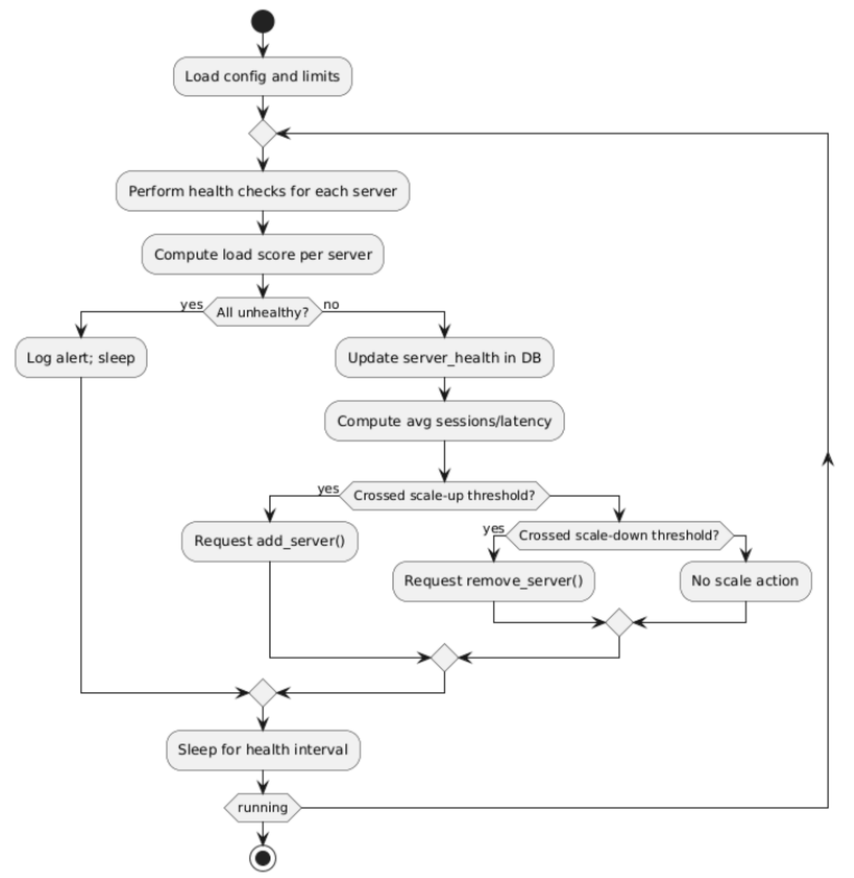

Figure 6 Activity Diagram

The flow of activities represents the rhythm of the operations: load setting, conduct health checks, score servers, update persistence, evaluate thresholds, and (not mandatory) instigate scaling operations. Oscillation is avoided by the existence of cooldowns and bounds. This model is based on real behavior in long-runs: periodic steps producing new health and selection summaries, and no flapping at constant load.

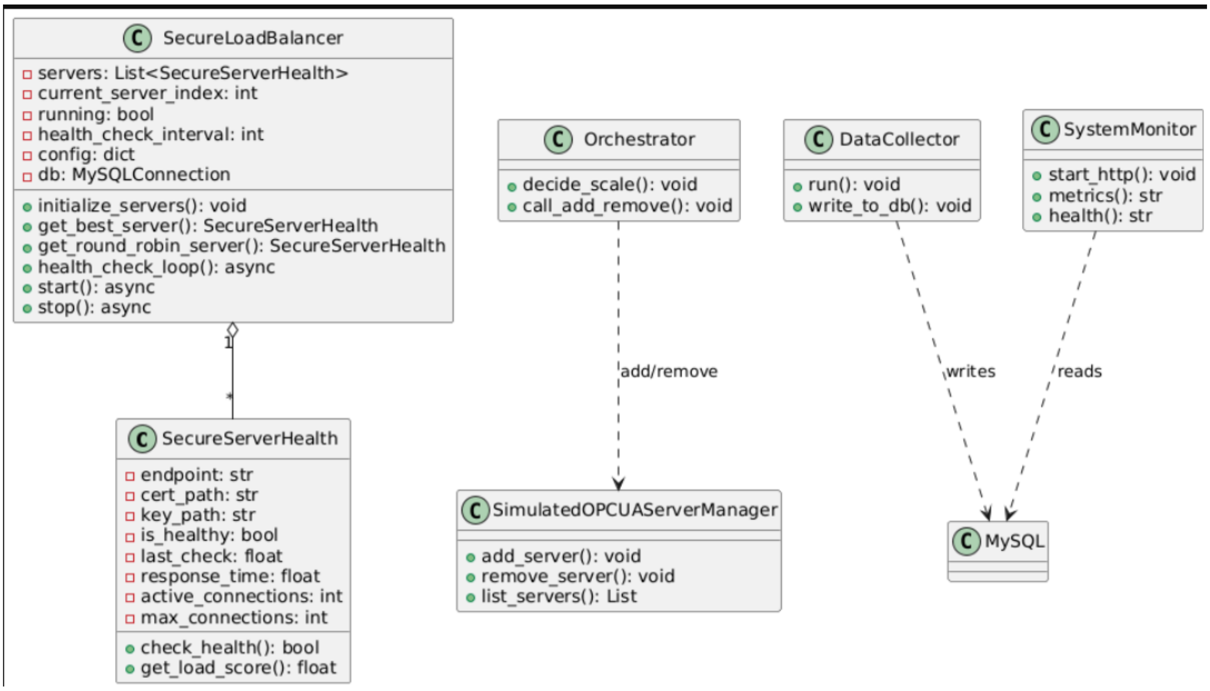

Figure 7 Class Diagram

The program structure applied in the implementation is abstracted as the class view. The SecureLoadBalancer is the aggregation of SecureServerHealth instances; a server manager has add/remove operations to control many server instances; the data collector and system monitor represents the encapsulation of data acquisition and observability issues. This organization is what makes the maintainability and debuggability as we see it in execution, all capabilities are local and side effects are observed in organization specific logs.

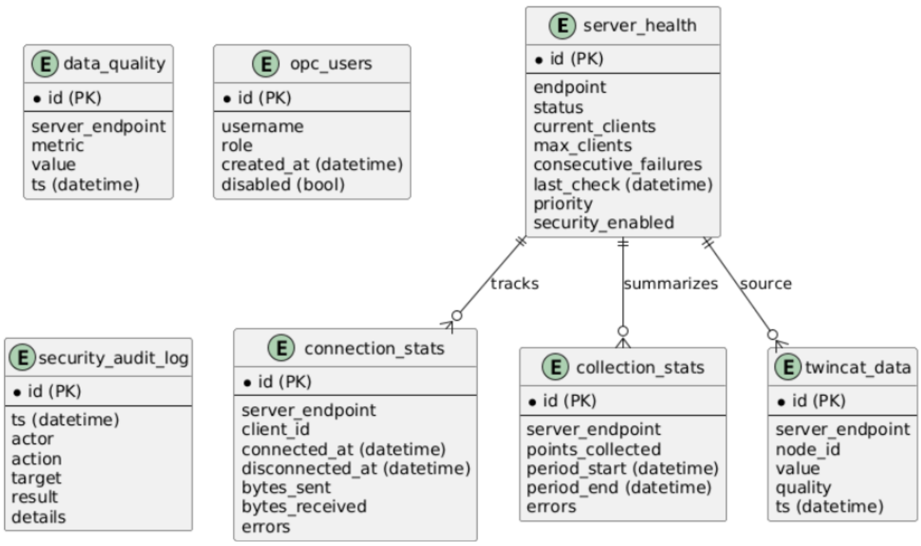

Figure 8 ER Diagram

The ERD specifies the database schema upon which health persistence and data collection is based: serverhealth, connectionstats, collectionstats, twincatdata and others. The columns selected (endpoint, status, counts, lastcheck, etc.) are operational requirements to find out the liveness, capacity and recent changes. The degree of normalization in the schema helps accommodate the common read patterns (e.g. latest health by endpoint) without writing patterns that are difficult to comprehend.

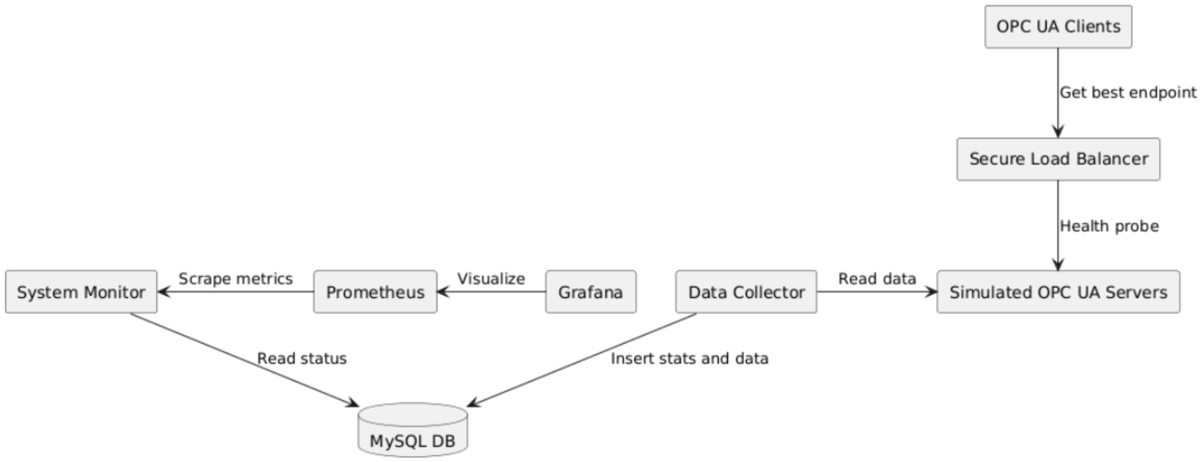

Figure 9 Data flow diagram

The DFD is a summary of data flow: clients request the optimal endpoint, the balancer requests servers, the collector requests telemetry, the monitor requests metrics/health, Prometheus requests metrics, and Grafana requests visualization. The model provides us with a succinct picture of the runtime indicating that each module only communicates what is needed, to the intended consumers, via clearly defined interfaces.

4.3 Module Results

4.3.1 Simulated OPC UA Server Layer

A simulated server layer gives the parallel secure endpoints that constitute the capacity pool of the system. All server instances connect to different ports and provide secure OPC UA semantics of session establishment. Since there is no TwinCAT runtime involved, the simulations provide the behavior of OPC UA that is necessary to verify upper-level system functions: secure session creation, bare-minimal address-space functionality (e.g. necessary to support health and data collection) and controlled injection of latency or failure to test.

When at steady state, all the configured servers are healthy and will allow secure client connections to the load balancer (health checks) and the data collector (sampling). This situation attests to the fact that horizontal capacity at the server tier is achieved. Isolation of the processes guarantees that the failure of one server will not affect the availability of other servers. To restart a halted server will be discovered in the next health cycle and returned to the healthy pool. The fact that the number of running server instances can be altered and that the platform is able to scale horizontally by managing the number of processes and ports, as well as that upstream components can be manipulated without a manual reconfiguration, makes this possible.

The fault behavior of the layer is also as expected. Upon the failure of a server, connection requests sent by the balancing load balancer will fall fast and the request is logged; the subsequent loop will leave the server in the unresponsive state until it is responsive. These changes are observable in the health summaries of the load balancer as well as in the metrics of the monitor, which provide a direct relation between the availability of the endpoints and the health metrics throughout the system.

Figure 10 Simulated OPC UA Servers - Console Output

Simulated OPC UA Servers - Console Output (securetwincatsystem.py) illustrates the output of the securetwincatsystem.py script in the console window.

This figure is expected to portray one console image of the simulated OPC UA servers when starting and in steady-state. The predicted console output is: a successful connection of all of the servers with their own endpoint (opc.tcp://localhost:4840, 4841, 4842), verification of secure endpoints, and preparedness to accept secure sessions. All servers are healthy in steady state. In case one server can be stopped and then restarted deliberately, the resulting transitions in process lifecycle can be seen in the console output, still without affecting the rest. This number confirms the existence of horizontal capacity to the server layer, and confirms that the simulation scripts are rich enough to represent TwinCAT runtime in controlled and re-producible experiment.

4.3.2 Secure Load Balancer

A load balancer has periodic health checks, composite load score and two strategies of selection best-server and round-robin. It has a simple design due to the intention of making it explainable and predictable.

Initialization behavior is similar to the configuration: a fixed set of endpoints is loaded (or possibly extended with a scalable configuration file), per-endpoint health structures are initialized, and an attempt to connect to MySQL to store data is made best-effort. The states of success and failure are recorded explicitly and do not inhibit the functioning. The platform generates a secureloadbalancer.log file and ensures long-term evidence between runs.

The basic loop is the core health loop which works with an interval of a specified time (30 seconds by default) (geeksforgeeks, 2025). The balancer of the load within every cycle is a secure OPC UA client handshake to all configured endpoints. Even the handshake is a realistic test of liveness and responsiveness since it does not just perform a TCP test. The round trip time is calculated and noted as a response time in each endpoint. The health state of per-endpoint is updated (healthy/unhealthy) and a composite score of load which is the sum of normalized active connections and response time (broken to avoid unbound values) is computed. The completion of every cycle gives a brief status: the number of healthy servers, response times, load scores, and a chosen endpoint server of the best server. This is the last period snapshot that is the operational signal which would be used in routing clients (MarcLadegourdie, 2022).

Service of health state is made as a guarded insert into the serverhealth table. With the database accessible, with each cycle each endpoint will write or update the corresponding row with its current status and lastcheck time; with unreachability the load balancer records a warning, will keep on with the computation of health, and restart writing once connected again. This decoupled persistence scheme does not connect database outages and health computation. The persisted data also offers a long-lasting audit trail of endpoint transitions of state and allows capacity planning, as well as forensics, over longer runs.

The operating behaviors of the strategies used in selection are different. In best-server selection, the balancer prefers endpoints with lower response times and lower load scores; the shorter client connection latency in the face of heterogeneous or temporary pressure of the servers. As a load balancer, round-robin selection offers even distribution between healthy endpoints irrespective of insignificant variation in performance. Operational priorities: fairness or responsiveness may influence the choice of the strategy. Under mixed conditions, best-server selection not only avoids degraded nodes by design but also round-robin prevents fairness but will include all healthy nodes in the rotation (Istikmal, 2023).

The straightforwardness of the strategy and the readability of the logs render the behavior of the load balancer as transparent to the operators and auditors. The design does not contain obscure heuristics and the selection results can be easily checked with respect to the measured response times and documented health condition.

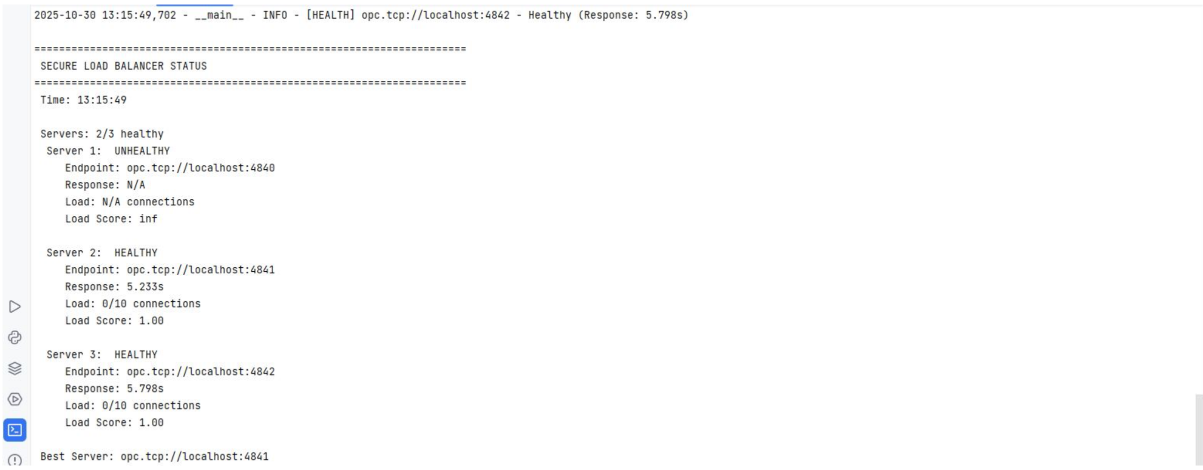

Figure 11 Secure Load Balancer -Console Output

This figure provides one console image of the load balancer following the startup and during a normal health cycle. The output is expected to contain the banner of the initialization, the list of configured endpoints, the confirmation of the health-check cadence (30 seconds default), and the list of healthy endpoints and its response time per endpoint and calculated load score. The suggested endpoint is shown by the summary line in view of the best-server strategy. In case the database is reachable, the console logs state MySQL connection and periodical upserts to the server health table, in case the database is not reachable, the health loop is not blocked and a warning is printed. This number shows evidence-based endpoint selection, sustainable reporting of health and graceful degradation in the case of persistence failure.

4.3.3 Data Collector

The collector confirms the overall data route of secure OPC UA endpoints into the relational data-base. It is concerned with reliability under intermittent failures as opposed to the highest ingestion throughput.

In a normal state, the collector will work in a cyclic manner, through the active endpoints, creating secure sessions, reading configured nodes, and writing to the database. Connection establishment, successful reads, and completed inserts are confirmed using console output. As time goes by the resulting database tables indicate a progressive build up of records at the predetermined sampling rate. Interactions with the collector are made idempotent where needed and record exceptions with enough context (identity of endpoint, phase of an operation and type of exception) so that, in the case of behavior that is not per the norms, help diagnose what is going on.

The short term errors, which occur on the set up of connections, the closing of secure channels or brief unavailability of endpoint are gracefully managed. The collector records warnings and only omits the failed step or interval, and continues at the next scheduled cycle. Such strength is a sign of the fact of the existence of operational networks where momentary packet loss or micro-restarts are not something that should destroy the work of the system. The resiliency of the collector in a long run provides a constant data stream despite sometimes disruptions.

In the case whereby the database is offline, the collector can record failure and reconnect logic based on the retry policy. The overall aim is to prevent termination of the processes, maintain logs of cause and time and restart standard insertion when the database is made available. Such a behavior is in line with the overall fault tolerance strategy of this platform: compute and cache or retries locally, and restart when dependencies are ready.

```
2025-10-30 13:18:01,883 - opcua.client.ua_client.Socket - INFO - opening connection
2025-10-30 13:18:11,942 - opcua.client.ua_client.Socket - INFO - Thread started
2025-10-30 13:18:11,970 - opcua.uaprotocol - INFO - received header: Header(type:b'ACK', chunk_type:b'F', body_size:20, channel:0)
2025-10-30 13:18:11,992 - opcua.client.ua_client.Socket - INFO - open_secure_channel
2025-10-30 13:18:11,995 - opcua.uaprotocol - INFO - received header: Header(type:b'OPN', chunk_type:b'F', body_size:123, channel:10)
2025-10-30 13:18:12,015 - opcua.client.ua_client - INFO - create_session
2025-10-30 13:18:12,083 - opcua.uaprotocol - INFO - received header: Header(type:b'MSG', chunk_type:b'F', body_size:2914, channel:10)
2025-10-30 13:18:12,133 - opcua.client.ua_client - INFO - activate_session
2025-10-30 13:18:12,173 - opcua.uaprotocol - INFO - received header: Header(type:b'MSG', chunk_type:b'F', body_size:84, channel:10)
2025-10-30 13:18:12,178 - __main__ - INFO - [COLLECT] Connected to opc.tcp://localhost:4841
2025-10-30 13:18:12,179 - opcua.client.ua_client - INFO - read
2025-10-30 13:18:12,198 - opcua.uaprotocol - INFO - received header: Header(type:b'MSG', chunk_type:b'F', body_size:70, channel:10)
2025-10-30 13:18:12,235 - opcua.client.ua_client - INFO - read
2025-10-30 13:18:12,283 - opcua.uaprotocol - INFO - received header: Header(type:b'MSG', chunk_type:b'F', body_size:70, channel:10)
2025-10-30 13:18:12,296 - opcua.client.ua_client - INFO - read
2025-10-30 13:18:12,327 - opcua.uaprotocol - INFO - received header: Header(type:b'MSG', chunk_type:b'F', body_size:70, channel:10)
2025-10-30 13:18:12,381 - opcua.client.ua_client - INFO - read
2025-10-30 13:18:12,397 - opcua.uaprotocol - INFO - received header: Header(type:b'MSG', chunk_type:b'F', body_size:70, channel:10)
2025-10-30 13:18:12,418 - opcua.client.ua_client - INFO - read
2025-10-30 13:18:12,457 - opcua.uaprotocol - INFO - received header: Header(type:b'MSG', chunk_type:b'F', body_size:63, channel:10)
2025-10-30 13:18:12,470 - opcua.client.ua_client - INFO - read
2025-10-30 13:18:12,500 - opcua.uaprotocol - INFO - received header: Header(type:b'MSG', chunk_type:b'F', body_size:70, channel:10)
2025-10-30 13:18:12,528 - opcua.client.ua_client - INFO - read
2025-10-30 13:18:12,580 - opcua.uaprotocol - INFO - received header: Header(type:b'MSG', chunk_type:b'F', body_size:70, channel:10)
2025-10-30 13:18:12,611 - opcua.client.ua_client - INFO - read
2025-10-30 13:18:12,680 - opcua.uaprotocol - INFO - received header: Header(type:b'MSG', chunk_type:b'F', body_size:70, channel:10)
2025-10-30 13:18:12,693 - opcua.client.ua_client - INFO - read
2025-10-30 13:18:12,748 - opcua.uaprotocol - INFO - received header: Header(type:b'MSG', chunk_type:b'F', body_size:70, channel:10)
2025-10-30 13:18:12,757 - opcua.client.ua_client - INFO - close_session
```

Figure 12 Data Collector -Console Output

This number should provide one console image of the collector at constant sampling period. The anticipated content is the successful secure connection to every active endpoint, reads to the target nodes and acknowledgement of the insertion into the database. Under unfavorable situations (transient endpoint unavailability or client-side exceptions), the console indicates warnings and future recovery on the next sampling cycle. This number shows the integrity of the end-to-end data path and recovery against temporary failures without terminating the process.

4.3.4 System Monitor and observability

The monitoring service offers two HTTP endpoints on port 9090, /metrics to allow machine consumption (Prometheus) and a /health endpoint to give human readable summaries. It is its purpose to reveal internal state in a normal form which can be scraped, alerted upon and visualized.

The overhead of the monitoring service remains low after it has been initiated. Counters and gauges that indicate the health of the system are published by the /metrics endpoint, such as the number of healthy servers, the time of the last health cycle, and process metrics. Exported metric names adhere to common conventions, which allows easy access as a Prometheus query and a Grafana panel. The /health endpoint and is to be used to perform the inspection manually, is a structured summary that shows the current state and the time of the last check. Handling the Windows-specific issues, the monitor implements serialization policies which translate non-native types into strings and limits the output produced by the console to text that can be represented in ASCII character set, to avoid the termination of the process because of the encoding errors (AyooluwaIsaiah, 2025).

The monitoring service also aids in the quick installation by offering configuration templates of Prometheus and ready stamp Grafana dashboard JSON. The templates are based on the exposed endpoints of the service, suggested panels, and decrease the integration friction and allow observability similarity among deployments.

Figure 13 Grafana Dashboard

This number is expected to show a single screenshot of the Grafana dashboard which displays Prometheus-scraped metrics of the monitoring service. It will consist of time-series panels of number of healthy servers, average per-cycle response times, the error rates, and the gauges of resources usage. The dashboard supports end-to-end observability: the monitor presents metrics, Prometheus scrapes them, and Grafana displays them, which operators can interpret and use to raise alarms.

4.4 Database Persistence Results

The serverhealth table can be persisted to ensure that the runtime measurements are stored in a long-term fashion and can be accessed without reference to the temporary console output. Fields of an endpoint row are endpoint identity, a status (healthy or unhealthy), current and maximum clients (where applicable), consecutive failure count and a last check timestamp. As these health cycles repeat, the load balancer also updates these fields every time the database is available. No computations would be lost in case of unavailability of a database, and resumes when it comes back up. This use of persistence model is what makes sure that an operator or auditor can rebuild health timelines and match them with selection decisions that have been observed and the behavior of collectors.

The tables related to data collection have periodical inserts in relation to the sampling interval. Temporal slicing and endpoint specific queries can be done with indexing on timestamps and endpoint identifiers. The data stored provides a basis of further analytics, such as anomaly detection, trend analysis and capacity planning, on standard SQL or Grafana panels developed on Prometheus-derived measurements where needed.

4.5 Fault Tolerance and Recovery

Fault Reliability was tested in controlled situations that resemble real world operational destabilizing situations. This conduct in both cases is in line with safe predictable industrial practice.

Termination of the endpoint and recovery were tested by interrupting one of the server instances and monitoring the system-wide response. The next health cycle had the load balancer marking the endpoint as unhealthy, and reducing the number of healthy ones and reconfiguring the recommended endpoint to one of the remaining healthy servers. Upstream services (collector and monitor) were adhered to automatically. When the endpoint had become available again, the subsequent cycle had found it, and the endpoint was accepted back into the pool of the healthy ones. There were no cascading failures and process liveness maintained within the system (Kolias, 2021).

Pause of MySQL service was done to simulate database outages. When outage occurred, the load balancer would still calculate health and immediately generate logs, but would fail to perform upset operations with warnings. On how to configured the collector, the collector would either buffering or skip inserts with context-rich logging. On the recovery of the database, the two components automatically returned to normal operations. This aspect of gradual degradation shows that concerns are properly isolated: compute loops are never blocked by persistence failures, and persistence is restarted without administrative intervention.

Windows-related limitations, such as console encoding and JSON serialization, were also handled by discarding unsupported characters in runtime console-based applications and by making sure that non-native types in a JSON dump are represented as a string. Following these alterations, there were no additional encoding-related terminations and long-term behavior became stable (AshrafSharif, 2022).

4.6 Scalability and Performance

The performance was defined in three aspects: health loop overhead, endpoint responsiveness and collector cadence. Scalability was concerned with the scale up and scale down of the server pool and the responsiveness of the system.

Health loop was completed with ease and within the stipulated time at pools ranging between three and five endpoints. Each cycle completes a few short lived secure client operations, computationally cheap compared to the 30 second cadence. The localhost endpoint response times were typically measured in milliseconds; they slightly rose when endpoint was started or when simulated latency injection Since the endpoint startups were rapid, endpoint response times rose momentarily too. The best-server strategy addressed these changes by preferring endpoints with a lower measured latency, which enhanced client-perceived responsiveness. Even distribution was achieved with round-robin with high availability with homogeneous conditions.

The collector could keep the sampling frequency during long runs. Jitter of a minor nature due to the occasional retries in the event of briefly occurring errors did not significantly affect the cumulative nature of records or the temporal regularity of sampling. The database schema and indexing were useful in efficient Inspection of the recent records using endpoint and interval (WaelAL-zyadat, 2024).

Scalability had been proven through the addition and removal of server instances. The introduction of one instance increased the healthy pool after the subsequent health cycle, and the elimination of healthy pool decreased the healthy pool. The set of the load balancer selection and the target set of the collector also indicated these changes with no reconfiguration of core services. Such scaling behavior was therefore constrained and predictable: maximum and minimum instance limits can be imposed in policy logic, and cooldown delays can be used to avoid oscillation during load changes (MattyKadosh, 2020).

4.7 Security Validation

Security was confirmed on the boundaries of transport and processes. Both the load balancer and the collector did secure OPC UA session establishment (both in health checks and sampling, respectively). These operations are practiced in the system under TLS and certificate authentication, defense is offered against interception and unauthorized access. Authentication flows and cipher suites are in line with widely accepted industrial practices of secure-by-default deployments.

Handling of credentials is hygienic. Database credentials are out of config, secrets are not hardcoded and logs do not display sensitive values. Certificates and key material access control is assumed in the deployment environment and path names of certificate and keys are quoted without being dumped. The logging is developed to generate usable diagnostics without sensitive information being exposed (MortezaSafaeiPour, 2023).

This has the overall effect of producing a security posture which is both practical and auditable: the transport security is made visible by normal operations, selection and collection is made by secure sessions and operational logs evidence successful and failure security handshakes.

4.8 Usability and Operational Observability

The platform gives more emphasis to the clarity of operations. The modules are separate processes on a Windows host and each of them produces logs of immediate value. The monitoring service reveals both machine readable measurements at metrics and human readable health status at health, where scripted and manual tests can be made. The Grafana dashboard summarizes the most important indicators, which are the number of healthy servers, response time, and the errors rate into one operator view. Long running services are provided with file-based logging to provide continuity in evidence even when a console session has gone away (MaurícioAniche, 2017).

The overall result is the lowering of the average time to identify and diagnose problems. Operators are not only able to see health changes almost in real time, but also are able to correlate those changes with selection decisions and collector behavior, as well as to check historical records in the database or logs to comprehend how to change the state of the system.

4.9 Limitations and Contemplations

Two limitations should be taken into consideration during the interpretation of results. The server layer is virtualized first and is not based on TwinCAT runtime. Although secure OPC UA semantics is faithfully exercised there is no evaluation of hard real-time behavior or vendor specific redundancy features. Second, connection-time balancing is favored over connection stickiness: session migration during the connection is out of scope and generally unsatisfactory with stateful OPC UA subscriptions. Such limitations are aligning with safe and realistic workload operation of telemetry-oriented workloads and do not incur to validity of illustrated security, resilience, and observability.

4.10 Summary

The findings support a secure, scalable, and horizontal OPC UA platform with windows. The architecture reflects design goals; the simulated fleet of servers offers parallel capacity, the fault-tolerant load balancer is based on the evidence-based health and selection cycles, the data collector reliably pushes the telemetry into the relational database, and the monitoring service presents machine and human readable status, which can be used with Prometheus and Grafana. Fault injection verifies gracious degradation in the event of endpoint or database outage and clean recovery in case thereafter. The measurements of the performance indicate that there is low overhead of health cycles, responsive endpoint selection, and constant sampling cadence. A horizontal scaling is achieved by the addition and removal of endpoints that are automatically discovered and adapted by upstream modules. Best practices of security such as secure session, conservative logging, and credential hygiene are followed. A combination of evidence, with the help of diagrams and runtime outputs, demonstrates that the system is capable of obtaining its goals of secure communication, operational resilience, real-time observability, and scalable capacity without referring to the TwinCAT runtime, which makes it a strong base to build on towards the further integration of the industry and research.

5 Analysis and Discussion of Experimental Results

5.1 Overview

In this chapter, the findings in Chapter 4 are analyzed and the effectiveness of the currently implemented system in achieving its desired goals that are: secure operation, modularity, observability, reliability under fault, and horizontal scalability are evaluated. It is based on the structure and the runtime nature of the modules, i.e., simulated OPC UA servers, the secure load balancer, the data collector, and the monitoring service; it identifies the strengths that have been demonstrated, as well as limitations, which are practical in the case of simulated servers use versus a TwinCAT runtime. The implementation of the load balancer (secureloadbalancer.py) is specifically focused, paying attention to the fact that the focus of core logic is on health assessment, endpoint choice, and database persistence.

5.2 Architectural Modularity and Adequacy

The modular nature of the platform complies with optimal practices of distributed, observable systems. All the significant issues lie in distinct long-term processes with a little or no connection: simulated servers will supply capacity; the load balancer will gauge well-being and offer endpoint proposals; the collector will ingest time-series readings into a relational store; and the monitor will promote health and metrics via HTTP. The above separation allowed clean fault boundaries in the results: a pause of a server instance did not interrupt other servers; a database outage did not interrupt the health loop; and the monitoring process was still available regardless of the state of load-balancer or collector. The deployment view (Windows host, independent terminals, MySQL service, and external Prometheus/Grafana) indicates an effective working model and was supported by the runtime behavior (TayyabMuhammad, 2022).

Safe horizontal growth is also supported by the architecture. Scaling in or out of server instances can change server capacity without topological recon figurately to other services. It is a valuable characteristic of scalable operational strategies since it minimizes the number of moving components subject to scale action and maintains observability throughout change (Prosper, 2025).

5.3 The Secure Load Balancer Analysis

The load balancer exhibits three critical features: persistent health checking, selection policy based on running measurements and best-effort health state persistence. The behavior observed indicates that these capabilities operate as they are supposed to work at the scale tested; nevertheless, when the implementation is analyzed; it indicates a number of important nuances.

Health probe fidelity. Health checks are used as real OPC UA client handshakes and this is the right way to go with a stateful protocol. This makes sure that healthy is associated with session establishment which is healthy and not just an open TCP port. The default cycle cadence (30 seconds) was stable with no excessive overhead of 3–5 endpoints. Since checks are done in serial fashion, cycle time linearly increases with the number of endpoints; larger fleets would switch to parallel checks (e.g. async gather) to lower the tail latency of health assessment, and to better respond (SohailSaifi, 2025).

Selection policy. The policy includes an active connections and response time factor into one to derive a composite score of load, and a round-robin fallback is provided as well. Practically, the given implementation does not update active connections, hence the policy acts like a latency-based one (along with a health gate). This gave reasonable decisions in the outcomes (preferring minimum response time) but capacity-based balancing was not practiced. The score should also be enhanced by adding a server-side signal on active sessions (e.g., reading a well-known Node on each endpoint), which would bring balancing consideration more in line with real capacity pressure.

Security position when health checks are done. The code promotes the use of certificate paths per server, but it fails to bind the security parameters of the client-side to the opcua.Client instance in health path. Depending on the server policy, the health checks can be successful without the need to establish the identity of the client as implemented. This is okay with simulation and with checking server liveness but it does not strictly test mutual TLS and certificate validation in health loop. Stiffening the client security setting in health checks would give more confidence that healthy is also to be securely accessed as per policy (TaufikHidayat, 2020).

Persistence semantics. The guarded update of serverhealth leaves an audit trail of endpoint liveness, but does not bind computation to persistence. This is an asset: when there was a failure in the database, health computation was not stopped and persistence was restored automatically. Status, client counts, lastcheck are the schema fields that have been captured and sufficient to make simple dashboards, but response time was not captured. Response time (and possibly a moving average) would be added to allow historical latency analysis, as well as improve both dashboards and auto-scaling triggers.

Operator transparency. The end of cycle status summary (healthy count, response time, load score, best server) is a clear evidence used to make decisions. This openness assists in promoting trust in the operators and makes post hoc analysis easier. A small operational problem is that non-ASCII characters are present in the console output; this can result in encoding errors on some windows code pages. The previous mitigation, which is text only, ASCII should be used throughout the prints.

In brief, it can be stated that the load balancer serves the purpose of evidence-based selection and fault-tolerant persistence at the scale under consideration. The principal technical debt items are:

(i) add health check parameters related to clients and TLS

(ii) fill out active connections through the server layer

(iii) persist response time

(iv) automate the health checks of bigger fleets

5.4 Simulated OPC UA Server Layer Analysis

The simulated server layer provided the necessary behavior to be able to evaluate the system level: a variety of secure endpoints, a consistent session initiation to be used in health check and data gathering, and manageable failure cases. The main benefit of the simulation is that it is safe and repeatable in experimentation; the main drawback is the fact that it is not a true representation of vendor runtimes (e.g. TwinCAT) and real network conditions.

Capacity and independence. The horizontal capacity and process-level fault isolation was proved through running multiple server instances and in parallel. Taking one instance out of service was not a degradation of others and putting it back into service was uncovered in the next health cycle. This activity is behavioral and can be projected to anticipated functioning in multi-controller setting (RodrigoDiaz, 2021).

Timing and determinism. No hard real-time conditions or recreation of the scan-cycle timing of the PLC are applied to the simulated layer. Consequently, latency measurements indicate client-library overhead, local IPC, and the host OS scheduler more than deterministic control timing. This would be suitable to test the load balancer, collector and monitor, however, validation on a TwinCAT runtime would be necessary in the future to ensure that it behaves in an industrial environment with scheduling constraints.

Security and policy. Simulations helped in the configuration to secure endpoints, however, since the load balancer client health checks did not explicitly add certificates, the system was not testing the harshest mutual-authentication flows in the health path. This is a familiar weakness of the existing implementation and not the simulation itself.

The simulated layer was, in general, a useful substitute to testing the platform. Issues are comprehended and reasonable when conducting research on balancing, persistence, and observability (BengtLennartson, 2012).

5.5 Analysis of the Data Collector

These findings of the collector focus more on resilience and continuity as opposed to raw throughput, which is in tandem with the project goals.

The ability to withstand temporary faults. The collector recorded and re-joined some client intermittent exceptions (i.e. during the closing/open of secure channels) and short-term endpoint unavailability. Cascading failure or termination of the processes were not observed. This is a vital characteristic of field deployments where glitches are short lived.

Cadence and idempotence. The collector continued its sampling period with small jitter in re-tries. The inserts were steadily stored in the database proving end-to-end integrity of the data-path. In places where feasible, collector operations were intended to mitigate possible duplicate writes or be tolerant to them without damaging downstream analytics.

Back pressure and buffering. In the event of a database failure, the collector was graceful and came up again when connected to the database. To prevent memory pressure, explicit buffering and backpressure control can be desired in high-frequency sampling in a production, while the strategy in place is sufficient in the experimented cadence (Kolias, 2021).

Similar to the load balancer, the collector would have access to a more powerful contract around server-side metrics (e.g., the number of active sessions at the servers), which would not only enhance the content of the records stored but also allow more complex scaling policies as well.

5.6 Monitoring Service and Observability Analysis

The monitoring service uncovered two endpoints, which are /metrics and /health, to satisfy both machine and human consumption patterns.

Metrics coverage. The metric set was able to capture the key points: the number of healthy servers, time indicators of health cycles, process level indicators that were enough to fit to the Grafana panels. This allowed verifying the trends observed in the consoles and a basis to raise an alarm.

Rate stability and compatibility. The service was responsive at a steady state. Problems with early serialization and encoding were eliminated through imposing safe JSON serialization and ASCII-compatible console output and no further instability was detected (BeaudenJohn, 2025).

Dashboards and experience of operators. Grafana visualizations gave a unified perspective of the health counts, response-times, and error-rates, which helped not only in diagnosing but also in the retrospective analysis. Ready-to-use configurations reduced the level of integration complexity and made it more reproducible.

The observability stack (monitor → Prometheus → Grafana) fits in both the problem space and scales without friction at the size tested.

5.7 Fault Tolerance, Reliability and Recovery

5.7.1 Induced faults in the system were predictable

Endpoint failure. Clean migration and restoration of a server instance spread well through the system: health changed correspondingly; the suggested endpoint reconfigured; and the remaining servers were left in the collection.

Database outage. Inability to persist did not prevent health computation or health monitoring. Upon receipt of the database, upsets as well as inserts were automatically entered again. This justifies proper separation between compute and persistence issues.

Environment constraints. The problem of windows console encoding and JSON serialization was detected and addressed. Post remediation processes were stable and lasted long.

All of these results are indicative of a strong runtime free of cascade failures and graceful recovery, which are essential attributes of operational environments.

5.7.2 Scalability analysis and performance analysis

Performance was measured in relation to health-loop overhead, endpoint responsiveness and collection cadence; scalability in the number of server instances added and removed to the system and the system adaptive response.

Health-loop overhead. With three or five endpoints, the sequential health loop completed well within the 30-second cadence, and there was much room even in the case of intermittent failures. In the case of larger fleets, it would be recommended to run parallel health checks to ensure that the latency of detection is kept.

Endpoint response time. Response times were in the milliseconds; and they were very low when starting or when simulated load occurred. The best-server approach accurately preferred endpoints with fewer measured latency, and time to establish a session would be better in cases where there was a difference. Round-robin was an even distribution under homogeneous conditions.

Collector cadence. Its retries had small jitters to preserve the interval of the collector. Increasing the rate of sampling or size of a fleet, it can be wise to batch read or write or increase or decrease intervals.

Horizontal scaling. The addition of an instance to the server enhanced healthy capacity in the subsequent health cycle; its removal decreased the capacity respectively. The upstream elements were automatically adapted. This is suitable to deterministic behavior associated with operational scaling policies, especially when used in combination with min/max limits and cooldowns to prevent oscillation (HiroshiMineno, 2002).

This system proved to be performing well at the tested scale and there was a clear way to support fuller deployments with minor improvements (parallel health checks, more detailed metrics and batching).

5.8 Security Analysis

Simulation and current client configuration allowed attaining security goals which were both achieved in part and approximated:

Transport security. The load balancer and the collector were both based on OPC UA client sessions. Nevertheless, the health-check client failed to explicitly deploy certificates and impose mutual TLS on the presented code path, which implies that health liveness could be ensured without the client's authentication under stringent policy. It is advised to make the health path stronger by adding credentialing information of clients, certificate validation, as well as the choice of security policy to use in production.

Hygiene secret management and logging. To externalize database credentials, all logs were not printed with sensitive values, all console output was sanitized to avoid encoding errors. These are good practices that are consistent with operational hygiene and minimize the risk of accidental exposure.

State and session stickiness. Connection-time balancing and session stickiness is intentionally applied in the system. Mid-session migration This is usually unsafe to the OPC UA subscriptions and is a good design decision, as a stateful protocol.

Generally, the security stance is right (encrypted transport, safe logging), and the next step is obvious to implement mutual TLS in the health loop and explicitly verify server certificates.

5.9 Persistence and schema utilization of databases

The serverhealth table was used to record endpoint liveness and last check time (updates) and has formed a long-lasting record of state-to-state transitions. This allowed the correspondence of observed runtime behavior to history maintained. Failing to persist response times crippled historical latency analysis and would be beneficial to add a response time (or a rolling average) field to enhance the analysis value of the data and allow more conditioning to alert or scaling. Idempotent updates can be reasonably modeled by the use of a stable primary key (hash of endpoint), but the endpoint canonicalization needs to be regularly maintained in order to prevent duplicates. Collection tables stored records based on the set cadence and worked with simple endpoint temporal queries.

5.10 Validity and Limitations Threats

There are a number of limitations to external validity:

- TwinCAT vs. simulation run time. No real TwinCat runtime implies that no results are provided to solve PLC scheduling, TwinCAT security policy intricacies, or vendor redundancy groups. This is also valid to the middleware behaviors under evaluation (balancing, persistence, observability), however, hardware-in-the-loop testing would be required to verify fully industrial use.
- Localhost networking. Single host measurements can be a bias to the actual latencies, jitter and packet losses in the real world. There is no representation of network effects like congestion, NAT traversal and variable MTU.
- Sequential health checks. At more significant fleet scales, sequential probing may grow the detection latency and bias comparisons. This bias would be done away with by parallelization.
- Incomplete load signals. In the absence of active connections, obtained by servers, the load score actually reduces to latency. This is tolerable when it comes to initial assessment but constrains the sophistication level of the balancing policy.

These constraints are manageable and give good direction on the future.

6 Conclusion

The platform demonstrates that it is possible and viable to have secure, modular, and observable control of a pool of OPC UA servers even when simulated servers are involved instead of a Twin CAT runtime. The load balancer is evidences-based selection on basis of actual protocol handshakes, the data collector is dependable pipeline to persistent storage and the monitoring service is offering machine-readable, human readable state, which is compatible with Prometheus and Grafana. The fault scenarios (endpoint loss, database outage) were also handled without the cascading failures and ensured the architecture separation of concerns and the focus on graceful degradation.

Three findings stand out:

The safety and integrity can be attained in small steps which are concentrated. Mutual TLS will implement health checks, validation of server certificates and externalized secrets will enhance assurance although it will not require architectural change.

What is required is scale and more advanced signals and not complexity. As the fleets grow in size, parallel health probing and the standard OPC UA diagnostics of using the number of sessions will be used to improve the selection fidelity and responsiveness. Persisting latency and failure streaks will support better dashboards, future auto-scaling policies, alerts and persisting latency and failure streaks.

Operation ability is founded on operation discipline and observability. The predictability of the system is guaranteed under normal conditions and in degraded condition with clear logs, persistent health state, actionable metrics, alerting and runbooks. Service and log rotation (windows specific) wrapping will reduce the danger of operations further.

It is also known to be limited: the layer of server is modeled and not replicated with PLC scan-cycle determinism or vendor-specific options of redundancy; it has been measured on a single host; and the current health loop is not yet enforced using mutual TLS. These are tractable. The proposed changes are evolutionary and are in line with the industry best practice and do not require redesign.

On the whole, this work is a good foundation in the safe and scalable OPC UA operations. It offers load balancing policy which is not complicated to explain, good collection, and complete observability which is required as far as trustworthy industrial data systems are concerned. It is possible to optimize the platform until it reaches production usage with the proposed improvements, which are stronger mutual authentication, parallel health checks, enhanced diagnostics, and automated PKI in the industrial context, and further work on predictive scaling, anomaly detection, and automated PKI on the platform can be conducted.

7 References

1. AnoudAlseiari, 2024. Auto-Scaling Techniques in Cloud Computing: Issues and Research Directions. *Sesors,* 24(17), p. 5551.

2. AshrafSharif, 2022. *Steps to Take if You Have a MySQL Outage.* [Online] Available at: https://severalnines.com/blog/steps-take-if-you-have-mysql-outage/

3. AxelBusboom, 2024. Automated generation of OPC UA information models — A review and outlook. *Journal of Industrial Information Integration,* 2(39), pp. 100-602.

4. AyooluwaIsaiah, 2025. *Python Monitoring with Prometheus (Beginner's Guide).* [Online] Available at: https://betterstack.com/community/guides/monitoring/prometheus-python-metrics/

5. BeaudenJohn, 2025. AI-Driven Predictive Analytics for Server Health Monitoring. *AI-Driven Predictive Analytics for Server Health Monitoring.*

6. BengtLennartson, 2012. Methods for Reliable Simulation-Based PLC Code Verification. *IEEE Transactions on Industrial Informatics ,* 8(2), pp. 267-278.

7. DaviOliva, 2022. *Getting Started with OPC UA in Beckhoff TwinCAT 3.* [Online] Available at: https://www.dmcinfo.com/blog/17976/getting-started-with-opc-ua-in-beckhoff-twincat-3/

8. DimitrisKanellopoulos, 2022. Dynamic Load Balancing Techniques in the IoT: A Review. *Symmetry ,* 12(14).

9. EmmanouilVasilomanolakis, 2025. Drifting Away: A Cyber-Security Study of Internet-Exposed OPC UA Servers. *Conference: International Workshop on Traffic Measurements for Cybersecurity (WTMC)At: Venice, Italy.*

10. geeksforgeeks, 2025. *Load Balancing Algorithms.* [Online] Available at: https://www.geeksforgeeks.org/system-design/load-balancing-algorithms/

11. Goodwin, M., 2024. *What is site reliability engineering (SRE)?.* [Online] Available at: https://www.ibm.com/think/topics/site-reliability-engineering

12. GuilhermeSerpaSestito, 2018. Performance Comparison Between OPC UA and MQTT for Data Exchange. *Conference: 2018 Workshop on Metrology for Industry 4.0 and IoT.*

13. HariSetiyani, 2022. Load Balancing Analysis Using Round-Robin and Least-Connection Algorithms for Server Service Response Time. *Applied Technology and Computing Science Journal,* 2(5), pp. 40-49.

14. HaroldCastro, 2023. Observability and Monitoring Using Prometheus and Grafana in Cloud Setups. *Observability and Monitoring Using Prometheus and Grafana in Cloud Setups.*

15. HassaneTahir, 2025. DevOps for Optimizing Database Management: Practice Implementation, Challenges and Comparison with the Context-Based Approach. *Computer Science & Engineering An International Journal ,* 15(1).

16. HiroshiMineno, 2002. On Web Server Performance Modeling for Accurate Simulation of End User Response Time. *On Web Server Performance Modeling for Accurate Simulation of End User Response Time.*

17. Istikmal, 2023. Analysis of Load Balancing Least Connection and Shortest Expected Delay Algorithm for Web Server Using Kube-Proxy on Kubernetes. *ELKOMIKA Jurnal Teknik Energi Elektrik Teknik Telekomunikasi & Teknik Elektronika,* 11(2), p. 439.

18. JianPeng, 2016. User profiling in intrusion detection: A review. *Journal of Network and Computer Applications ,* 72(2).

19. JoãoFigueiredo, 2019. A Literature Survey on Open Platform Communications (OPC) Applied to Advanced Industrial Environments. *A Literature Survey on Open Platform Communications (OPC) Applied to Advanced Industrial Environments,* 5(8), p. 510.

20. JohnOlusegun, 2025. ACHIEVING HIGH AVAILABILITY AND FAULT TOLERANCE IN CLOUD-BASED SYSTEMS. *ACHIEVING HIGH AVAILABILITY AND FAULT TOLERANCE IN CLOUD-BASED SYSTEMS.*

21. JonathanKua, 2022. Performance Analysis of OPC UA for Industrial Interoperability towards Industry 4.0. *Iot,* 3(4), pp. 507-525.

22. JuergenJasperneite, 2013. Scalability of OPC-UA down to the chip level enables “Internet of Things”. *Conference: Industrial Informatics (INDIN), 2013 11th IEEE International Conference on.*

23. JuliusPfrommer, 2016. RESTful Industrial Communication with OPC UA. *IEEE Transactions on Industrial Informatics,* 12(15), p. 1.

24. KeithStouffer(NIST), 2015. *Guide to Industrial Control Systems (ICS) Security.* [Online]
Available at: https://csrc.nist.gov/pubs/sp/800/82/r2/final

25. Kolias, C., 2021. Vulnerabilities and Attacks Against Industrial Control Systems and Critical Infrastructures. *Vulnerabilities and Attacks Against Industrial Control Systems and Critical Infrastructures.*

26. LiangCheng, 2015. Operating System Security Policy Hardening via Capability Dependency Graphs. *Lecture Notes in Computer Science.*

27. MarcLadegourdie, 2022. Performance Analysis of OPC UA for Industrial Interoperability towards Industry 4.0. *IOT,* 3(4), pp. 507-5022.

28. MattyKadosh, 2020. LB Scalability: Achieving the Right Balance Between Being Stateful and Stateless. *LB Scalability: Achieving the Right Balance Between Being Stateful and Stateless.*

29. MaurícioAniche, 2017. Log-based software monitoring: a systematic mapping study. *PeerJ Computer Science.*

30. MdABUIMRANMallick, 2024. Navigating the Cyber security Landscape: A Comprehensive Review of Cyber-Attacks, Emerging Trends, and Recent Developments. *Navigating the Cyber security Landscape: A Comprehensive Review of Cyber-Attacks, Emerging Trends, and Recent Developments.*

31. MeghrajChoudhary, 2025. *Proactive Server Health Monitoring: Avoid Downtime Before It Happens.* [Online]
Available at: https://middleware.io/blog/server-health-monitoring/

32. MehrdadAshtiani, 2023. An auto-scaling approach for microservices in cloud computing environments. *An auto-scaling approach for microservices in cloud computing environments.*

33. MohammedDaffallaElradi, 2025. Prometheus & Grafana: A Metrics-focused Monitoring Stack. *Journal of Computer Allied Intelligence,* 3(3), pp. 28-39.

34. MortezaSafaeiPour, 2023. A Comprehensive Survey of Recent Internet Measurement Techniques for Cyber Security. *Computers & Security,* Volume 128, pp. 103-123.

35. PaulGrassi, 2017. Digital Identity Guidelines: Authentication and Lifecycle Management. *Authentication and Lifecycle Management.*

36. PremkumarGanesan, 2022. OBSERVABILITY IN CLOUD-NATIVE ENVIRONMENTS CHALLENGES AND SOLUTIONS. *OBSERVABILITY IN CLOUD-NATIVE ENVIRONMENTS CHALLENGES AND SOLUTIONS.*

37. Prosper, D., 2025. Comparative Analysis of Vertical vs. Horizontal Scaling in Cloud Environments. *Comparative Analysis of Vertical vs. Horizontal Scaling in Cloud Environments.*

38. PurshotamSinghYadav, 2023. Automation of Digital Certificate Lifecycle: Improving Efficiency and Security in IT Systems. *Journal of Mathematical & Computer Applications.*

39. Quattrocchi, G., 2024. Autoscaling Solutions for Cloud Applications Under Dynamic Workloads. *IEEE Transactions on Services Computing,* pp. 1-17.

40. RamuhalliPradeep, 2022. LWRS Development of a Cloud-based Application. *Affiliation: Idaho National Laboratory.*

41. RodrigoDiaz, 2021. Security Information and Event Management (SIEM): Analysis, Trends, and Usage in Critical Infrastructures. *Sensors,* 21(14), pp. 47-59.

42. SalvatoreCavalieri, 2013. Analysis of OPC UA performances. *Computer Standards & Interfaces ,* 36(1), pp. 165-177.

43. SebastianSchmied, 2021. A framework for monitoring multiple databases in industries using OPC UA. *Journal of Ambient Intelligence and Humanized Computing,* 12(14), pp. 1-10.

44. SebastianSchmied, 2021. A framework for monitoring multiple databases in industries using OPC UA. *Journal of Ambient Intelligence and Humanized Computing ,* 12(4), pp. 1-10.

45. SohailSaifi, 2025. *The Load Balancer Algorithm That Adapts to Your Application's Behavior.* [Online]
Available at: https://medium.com/@sohail_saifi/the-load-balancer-algorithm-that-adapts-to-your-applications-behavior-44718c958685

46. StenGrüner, 2016. RESTful Industrial Communication with OPC UA. *IEEE Transactions on Industrial Informatics ,* 5(12), p. 1.

47. SubrotaMondal, 2023. Toward Optimal Load Prediction and Customizable Autoscaling Scheme for Kubernetes. *Mathematics,* 11(12), pp. 26-75.

48. TaufikHidayat, 2020. Load Balancing Network by Using Round Robin Algorithm: A Systematic Literature Review. *Jurnal Online Informatika ,* 4(2).

49. TayyabMuhammad, 2022. A Comprehensive Study on Software-Defined Load Balancers: Architectural Flexibility & Application Service Delivery in On- Premises Ecosystems. *A Comprehensive Study on Software-Defined Load Balancers: Architectural Flexibility & Application Service Delivery in On- Premises Ecosystems.*

50. WaelAL-zyadat, 2024. Web application performance assessment: A study of responsiveness, throughput, and scalability. *International Journal of ADVANCED AND APPLIED SCIENCES,* 11(9), pp. 214-226.