sustainability

Article

SaaMES: SaaS-Based MSA/MTA Model for Real-Time Control of IoT Edge Devices in Digital Manufacturing

Sanghoon Do [1], Woohang Kim [2], Huiseong Cho [2] and Jongpil Jeong [1,*]

[1] Department of Smart Factory Convergence, Sungkyunkwan University, 2066 Seobu-ro, Jangan-gu, Suwon 16419, Korea

[2] Systems Management Engineering, Sungkyunkwan University, 2066 Seobu-ro, Jangan-gu, Suwon 16419, Korea

* Correspondence: jpjeong@skku.edu; Tel.: +82-031-299-4260

Abstract: As a software delivery model, Software as a Service (SaaS) has attracted considerable attention from software providers and users. Most traditional companies are shifting their businesses to an SaaS model. SaaS development is a very complicated process and its success depends on architectural design and development. A Manufacturing Execution System (MES) was used at the expense of licensing fees for features not used in the On-Premise environment, although the features used vary depending on the manufacturing environment. In an SaaS environment, MES is applied with a function-specific container approach through a Microservice Architecture (MSA) to select and employ only the necessary functions. Furthermore, as the number of customers of virtualized applications increases in SaaS-based services, complexity and operating costs increase; thus, Multi-tenancy Architecture (MTA) technology, which serves all customers through a single instance of the application is crucial. Thus, in this study, we investigate the MTA approach and propose a suitable MTA for the manufacturing execution system. Real-time response is crucial to achieving a cyber-physical system of digital manufacturing in SaaS-based MES. Furthermore, SaaS-based big data analytics and decision-making cannot meet the needs of numerous applications in real-time sensitive workplaces. In this study, we propose an SaaS-based MSA/MTA model for real-time control of Internet of Things (IoT) Edge in digital manufacturing (SaaMES), an architecture of SaaS-based MES with MSA and MTA to meet vulnerable workplaces and real-time responses in Cloud environments. The analysis is used by applying the Autoencoder and Generic Adversarial Networks analysis model to IoT Edge for the connection between the Cloud environment and work site to enable real-time response and decision-making through communication using OPC-UA and small-scale analysis.

Keywords: SaaS; manufacturing execution system; MSA; MTA; IoT edge

check for updates

Citation: Do, S.; Kim, W.; Cho, H.; Jeong, J. SaaMES: SaaS-Based MSA/MTA Model for Real-Time Control of IoT Edge Devices in Digital Manufacturing. *Sustainability* **2022**, *14*, 9864. https://doi.org/10.3390/su14169864

Academic Editors: Maurizio Bevilacqua, Sara Antomarioni, Giovanni Mazzuto, Giulio Marcucci and Antonella Petrillo

Received: 12 June 2022
Accepted: 8 August 2022
Published: 10 August 2022

1. Introduction

One of the reasons why Cloud Computing technology is focused on manufacturing environments is the Internet of Things (IoT). With the development of [1], the cost and flexibility of building an On-Premises IT infrastructure that stores and accumulates vast amounts of digital data generated on the shop floor at the individual factory level is inefficient. Another reason is that many Cloud Computing services provide various additional services such as large-capacity big data analysis and artificial intelligence, and if they are utilized, they can be more helpful in implementing smart factories. As a typical software delivery model, Software as a Service (SaaS) allows third-party providers to provide software as a service rather than a product to many users over the Internet [2]. In addition, with the advancement of IoT, system design with Microservice Architecture (MSA) is suitable to provide rapid change and dynamic resources due to many issues such as technology heterogeneity, resource constraints, and interoperability [3] and Multi-tenancy Architecture

(MTA) has been adopted as a key technology component to transition to SaaS architecture [4]. Combining IoT and SaaS technologies can improve integration between physical resources and Cloud services.

SaaS application is a service through a subscription model, and SaaS's provider is an application that hosts one integrated service in the Cloud and makes it available to multiple users over the Internet [5]. SaaS is appropriate for third-generation software engineering and is considered a package of development, execution automation runtime resource sharing management, and security [2]. The benefits of moving to an SaaS business model are as follows. First, the application of predictable long-term subscription-based revenue models improves business value. Second, rapid and flexible technology implementation and response improve agility. Third, the transition from On-Premise to code-based improves maintenance productivity. Fourth, rapid upgrades accelerate innovation. Fifth, the user's barriers to entering the service experience are lowered, and the overall customer experience is improved due to the competitive usage fee, thereby improving the value of the service. Currently, it is serviced by many SaaS applications; examples include email services, Cloud storage services, collaboration tools (Slack, Notion, Zoom, etc.), and YouTube [6].

Compared with traditional on-premise software, SaaS provides three key benefits [7]. First, because it is not a large initial investment, but rather a usage-based payment approach, the monthly investment is relatively low. Second, by removing specialized personnel who manage your software and underlying infrastructure, you can reduce human and financial pressure on your company. Third, you can save money and time by instantly accessing the latest applications and services without having to bother with development, installation, and testing. Each application domain has numerous SaaS vendors, so customers have more flexibility in IT decisions. Since you have not invested in infrastructure, you can always move to another vendor if needed [8]. In the long run, the advantage of the SaaS transition in terms of cost and profit can be confirmed not only in the case of the previously applied companies but also in the model announced by the technology-as-a-service playbook 2016 (TSIA). The Fish model in Figure 1 demonstrates that in the early stages of business transformation, the return curve temporarily falls below the cost curve due to investment costs, but eventually, over time, operating costs decrease, revenue growth increases, and revenue lines rise.

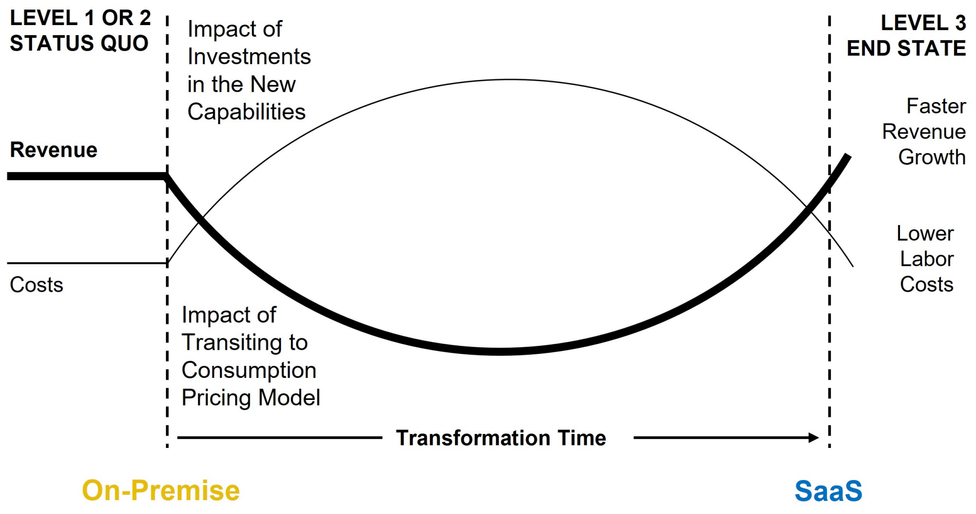


Figure 1. Fish Model at Technology-as-a-service Playbook 2016 (TSIA).

There are two reasons why an MES design based on SaaS requires an MSA. First, it is easy to respond to rapid changes. Unlike traditional application developments, SaaS provides granular services based on the purpose of the service and the ever-changing needs of the customer. As a result, SaaS must deliver higher quality services than On-Premise systems. Existing On-Premise-based MES mainly used the form of building and releasing as a single package, but MES with MSA for each module and class can respond sensitively to customer requirements and deliver only necessary services to the container, so SaaS-based services are cost-effective. MSA [9] is a distributed system solution after Service-Oriented Architecture (SOA). MSA has greater advantages in application system design, functional module development, and subsequent maintenance work compared with traditional SOA architecture. The traditional way we used it was a monolithic architecture in which the system was lumped together. Because the entire application is one, the entire application needs to be built with even a small modification, resulting in longer build and test times, and increasing the use of certain features, making it impossible to selectively scale, and a single service can impact all services [10]. MSA is attracting attention to solve these problems. MSA is a method of designing a single program into a small combination of services by dividing it into each component. Each component is implemented in the form of a service and communicates with other services using the API. Each service is an independent server, and because it is not dependent on other components, it deploys independently. In addition, each component is developed as an independent service, allowing for partial scaling, even as the usage of certain services increases. It is suitable for the SaaS subscription model by quickly responding to customer requirements through service segmentation and distribution processing, and selecting optional functions [10,11].

Second, distributed processing is needed due to the explosion of data. MES is most sensitive to accuracy and works in real-time on the shop floor. Since everything in smart factories is data-driven, problems can occur in the production process if the connection is not smooth, and solutions should be prepared. Furthermore, it is necessary to increase the manufacturing utilization of such big data because it accumulates a vast amount of data collected in real time through sites, markets, and social networks, and can benefit more from business analysis and artificial intelligence. For this, it is essential that real-time processing is smooth [12]. The weakest part of Cloud-based SaaS is real-time functioning. To compensate for this, the network field's development has been repeated. Today, the network-side solution to ensure real-time Cloud-based services is considered a 5G network. However, the speed of the network is not the only solution. There may also be issues with network equipment and Cloud services in the field. Research has been conducted to optimize services and flexibly select services in the manufacturing Cloud for IoT and big data processing [13]. MSA is not just an architecture for application segmentation. Research on the micro-service-based IoT system was also conducted to process big data [3]. A system close to the MSA is the IoT Edge. Optimization and segmentation of SaaS applications are important, but optimization and segmentation on the shop floor are also important. In addition, if these two areas are vertically integrated, it becomes the overall MSA of the Cloud and shop floor.

The use of IoT Edge was noted for real-time use of big data in the Cloud base [12]. Cyber-Physical Systems (CPS) are systems that are divided between the cyber and physical worlds connected over a network and are generally associated with the concept of IoT [14]. The development and many applications of distributed industrial CPS IoT applications have significantly changed manufacturing systems and provided an opportunity to further optimize and refine automation systems to encapsulate them. IoT and Internet services present a vision of a smart factory where CPS monitors physical processes, creates virtual copies of the physical world, and makes distributed decisions [15]. The realization of CPS for factory automation requires tools on the shop floor that can support distributed systems. The focus has shifted from traditional single, professional-based, independent engineering tools and methods to a unified Cloud-based tool/system infrastructure. Smart objects, detection, and operational capabilities of smart factories are being developed and applied to various smart sensors, devices, and facilities during manufacture along with the development of ultra-small sensors aimed at small, low-cost, low-power, and high-precision. Through this, vast amounts of data have been digitized at the same time as IoT development. For distributed processing, IoT Edges are connected to the shop floor and serve as intermediate hubs for transmission to Cloud-based SaaS systems. Actively utilized

IoT Edges can be directly connected to the shop floor and the Cloud, reduce network usage, and provide very resilient and powerful capabilities to handle manufacturing big data. In addition, after gaining some knowledge or insight through big data, real-time measures are taken to improve productivity or reduce losses, compensating for the lack of real-time performance of Cloud-based SaaMES.

There are two reasons why it is most efficient to separate applications and efficiently distribute IoT data through SaaMES' MSA. The first reason is the efficiency of subscriptions [15]. The tasks required vary depending on the characteristics of the manufacturing site and the level of users, so they are divided by task so that users can subscribe efficiently. The second reason is the efficiency of data processing and virtualization [16]. Because the data used by each user varies from service to service, the process of processing internal data and IoT data is also distributed by task, which facilitates maintenance and additional virtualization of applications.

The last technology needed to perform SaaS-based services is MTA. In a system, a tenant is a user who uses shared resources. In SaaS, it is an architecture used by multiple tenants to share and use one application [17]. Tenants can be tenants of one user or organization or multiple tenants for the same purpose. Because the application is shared and used, the requirements of other tenants can be shared and provided. Even if multiple tenants use a common application, the service cannot conflict with each other because it is managed separately from the SaaS base to the container [18]. Depending on the applied MTA, the application and database can be shared and used and depending on the setting, they can be used separately from tenants [19]. When designing SaaS-based systems, MTA must be performed according to the nature of the application. Depending on the nature of each system that shares or isolates applications and databases from each other, you should initially consider the power of the On-Premise to SaaS transition. In this study, the characteristics of SaaS transition are divided into five groups to be considered when establishing power, an MTA suitable for MES is found, and strategies are established for transition.

SaaMES is SaaS based, and its biggest advantage is that it can effectively reduce costs. With MSA, you can select and use only the services you require, reducing costs and waste of resources by not putting unnecessary services in containers. In this study, we summarized the considerations when switching from On-Premise to SaaS with MTA, and proposed the optimal MTA approach for MES to reduce the complexity of instances and reduce costs. We also made it easy to install and upgrade by allowing services to multiple users in a single instance. We reduce the use of resources in the network and Cloud by preprocessing and entering various large numbers of data through IoT Edge, ensuring real-time performance so that production progress is not disrupted in the event of Cloud problems. We confirmed that simple analysis models are possible on the IoT Edge and demonstrated that executing anomaly detection at the IoT Edge is faster in terms of time than performing in the Cloud, and real-time performance is secured, allowing quick decision-making.

This paper is configured as follows. Section 2 explains the related work. Section 3 lists the concepts that support the proposed SaaMES approach, presents an architectural analysis centered on methodological MSA and MTA, presents a standard model appropriate for MES, and demonstrates real-time centered on IoT Edge. Section 4 explains the experimental description and result on IoT Edge for securing SaaMES real-time performance. Finally, Section 5 discusses the conclusions and future research plans.

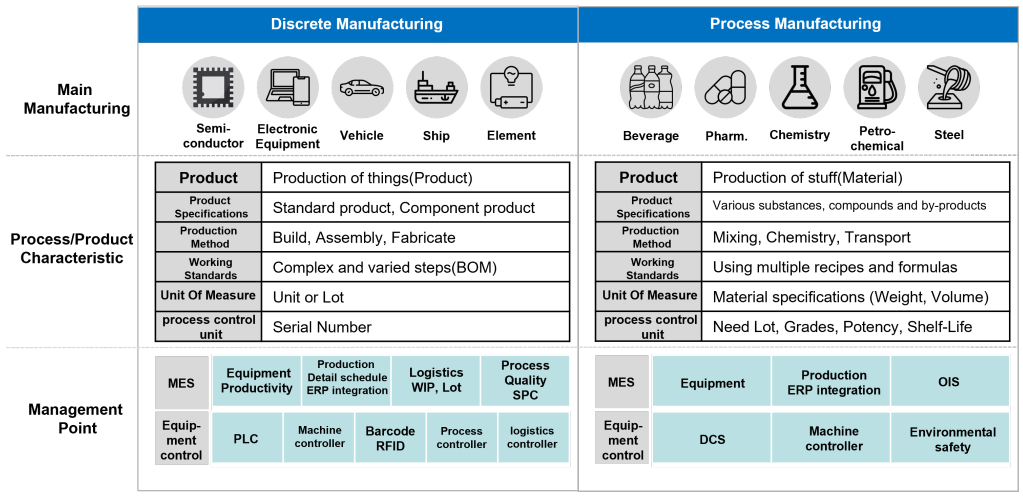
2. Related Work

2.1. Manufacturing Technology

Customizing is not a new concept, it has been investigated and written for more than 40 years [20]. Each tenant has a different requirement for multi-tenant systems [21]. Successful SaaS applications must support customization to meet the requirements of different tenants [21]. Customizing is important to serve a variety of tenants as the flow of business processes, interface methods, and data vary from tenant to tenant [22–25]. However, excessive customizing is a risk that can compromise performance for SaaS services [26,27].

For successful SaaS application deployment customizing, it must meet the requirements of different tenants and be applied to the entire architecture [21,26]. Chung et al. said that software customizing has long been a research challenge, but SaaS features such as multi-tenancy and cost awareness have brought new challenges to SaaS customizing [22]. Meetzner et al. explained how SaaS providers automatically create process-based customizing tools from the variability points of SaaS applications [28]. Lee et al. developed and implemented a multi-tenant web application framework that customized SaaS by tenant, industry, and department [29].

Miracom INC's MES solution in Korea has a large customer base. We examined customized sources of MES and databases for each customer to also analyze user trends. The manufacturing execution system can be classified into two manufacturing industries by referring to Oracle Manufacturing's data, as shown in Figure 2. It can be divided into discrete manufacturing and process manufacturing, and the production approaches and menu usage patterns differ from each other. Most of the assembly industries belong to discrete manufacturing, which has the same unit of input and output and can be viewed as an EA in the process. Process manufacturing is a manufacturing industry that has different units of input and output in its process and mostly includes chemical processes. Thus, to avoid customizing by a customer, it is separated into two applications, discrete manufacturing, and process manufacturing versions, making it easier to maintain and operate in the future.



	Discrete Manufacturing	Process Manufacturing
Main Manufacturing	Semi-conductor, Electronic Equipment, Vehicle, Ship, Element	Beverage, Pharm., Chemistry, Petro-chemical, Steel

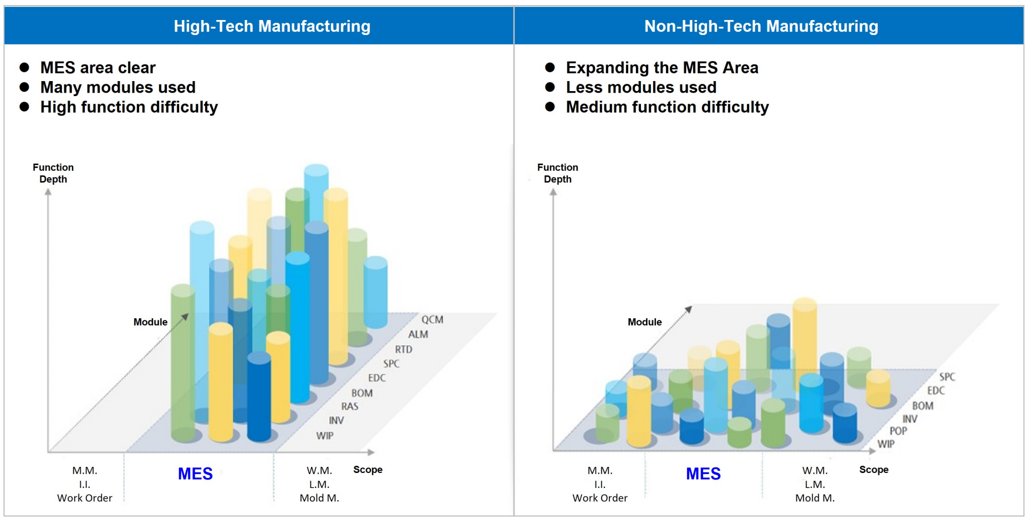
Process/Product Characteristic	Discrete Manufacturing	Process Manufacturing
Product	Production of things(Product)	Production of stuff(Material)
Product Specifications	Standard product, Component product	Various substances, compounds and by-products
Production Method	Build, Assembly, Fabricate	Mixing, Chemistry, Transport
Working Standards	Complex and varied steps(BOM)	Using multiple recipes and formulas
Unit Of Measure	Unit or Lot	Material specifications (Weight, Volume)
process control unit	Serial Number	Need Lot, Grades, Potency, Shelf-Life

Management Point	Discrete Manufacturing	Process Manufacturing
MES	Equipment Productivity; Production Detail schedule ERP integration; Logistics WIP, Lot; Process Quality SPC	Equipment; Production ERP integration; OIS
Equip-ment control	PLC; Machine controller; Barcode RFID; Process controller; logistics controller	DCS; Machine controller; Environmental safety

OIS : Operation Information System, DCS : Distributed Control System, PLC : Programmable logic controller
Source : Oracle Manufacturing

Figure 2. Classification of Manufacturing.

Figure 3 is a statistical survey of the functional depth and scope of each module by manufacturing technology actually used by customers in Miracom INC's MES solution. It can be seen that the modules used are different for each manufacturing technology, and the functional depth and range of the modules are also different. There are also differences in modules used between factories by the same manufacturing technology. Through the corresponding statistics, MSA for each module of MES was conceived by synthesizing by module function.



M.M. : Material Management, I.I. : Import Inspection, W.M. : Warehouse Management, L.M. : Laboratory Management, Mold M. : Mold management
WIP : Work In Process, BOM : Bill of Material, INV : Inventory, RAS : Resource Allocation and Status, EDC : Engineering Data Collection
SPC : Statistical Process Control, RTD : Real-Time Dispatch, ALM : Alarm Management, QCM : Quality Control Management, POP : Point Of Production

Figure 3. Functional Depth and Scope of MES.

2.2. MTA

Tenants refer to their own environment that Cloud service users have. Multi-tenancy is the ability of SaaS applications to support multiple tenants using a single server and service instance so that all tenants feel like dedicated servers [30]. MTA is an architectural design in which a software instance of a service provider shares components, including data models, servers, and database layers, to multiple testers [31]. Multi-tenancy is a crucial feature of SaaS software and a metric for SaaS vendors' success [32]. There are three levels of multi-tenancy, namely, data model multi-tenancy, application multi-tenancy, and overall multi-tenancy [33]. Data model multi-tenancy is a model in which tenants use the same database and use tenant-specific applications. Application multi-tenancy is a model that shares the same application and accesses tenant-specific databases. Full multi-tenancy is a model in which all tenants use the same application and database [33]. Multi-tenancy offers numerous benefits. Multi-tenancy reduces maintenance costs by obtaining information from various tenants. It also facilitates resource management, deployment, and maintenance [32]. Considering the benefits of multi-tenancy, most organizations will convert single-tenant SaaS applications into multiple tenants. Kang et al. proposed a multi-tenant SaaS platform architecture that provides an SaaS application that controls multi-tenants through configuration information [30].

2.3. IoT Edges in Manufacturing

With the development of IoT, processing large amounts of data is the main challenge of IoT research [34]. Big data, including various IoT information and methods, allows quick decision-making in the manufacturing environment [35]. IoT and big data transmitted to the Cloud in a manufacturing environment can be used for flexible and agile decisions [13]. Cloud computing has gained attention because it can process and store large amounts of data at an efficient cost [36]. However, with the explosive growth of data (zeta-byte data) and IoT connections, it is inefficient and impossible to process all data by sending it to the cloud [37]. Edge Computing (EC) has been proposed as a way to break away from the centralization of the Cloud by distributing large amounts of data [37]. The key to the EC is to reduce network traffic by performing calculations on the shop floor [37]. Yang et al. enabled the processing of various big data by the edge of high-performance gateways [12]. It was also expected that the success of the manufacturing IoT would provide insight into the manufacturing industry through the processing of big data and real-time response [12].

2.4. MSA

Compared to monolithic architecture, MSA is a lightweight architecture divided into smaller services [38]. Microservices are business-centric, and the services can be developed, distributed, tested, and operated independently [38]. The MSA consists of blocks such as infrastructure communication as well as business, and each block is connected to lightweight communication when configured independently [39,40]. Since microservice is an independent lightweight business process, it is easy to modify the workflow and can respond to customer requirements [40,41]. While MSA has cost advantages, it should be used for the greater benefits of decentralization and virtualization. For example, a single process, such as monolithic, extends to replication on multiple servers for replication. However, with microservices, it is easy to expand virtualization due to decentralization because it can be replicated within one server [38,42]. Furthermore, the ability to deploy autonomous services independently and benefit from Cloud flexibility and quick resource delivery makes the Cloud-based platform easy to use for MSA development [43,44]. The author of [3] proposed a self-adaptive architecture for microservice-based IoT systems due to performance, stability, and resource management issues that are emerging as IoT advances.

3. SaaMES: SaaS-Based MSA/MTA Model

In this study, we present a SaaMES architecture. This architecture is divided into two primary themes. First is SaaS-based MES architecture. It is crucial to apply MSA and MTA to develop SaaS-based MES architecture. Thus, we apply MSA and MTA to explain the architecture appropriate for MES. Second, securing real time using IoT Edge and detecting abnormalities through simple analysis. For Cloud-based SaaS services, the role of IoT Edge is critical in manufacturing execution systems that need monitoring and control of manufacturing processes in real time. We introduce experiments on the anomaly detection model through a small-size dataset and measure the processing time, including each analysis in the Cloud and IoT Edge for the availability of real-time performance.

3.1. System Architecture

As shown in Figure 4, we propose SaaMES. It is divided into SaaS-based MES part of the Cloud, IoT Edges, and Manufacturing Asset and Objects part of the shop floor. Each tenant can access MTA's own system through the Portal App. You can use the app that you charged through the MSA. In the Digital Twin Agent, the OPC UA Agent collects data and sends it to tenant-specific systems. Depending on the settings of each tenant, the settings of various Manufacturing Assets and Objects are stored through Properties/Methods, and connected to IoT Edges on the shop floor through Telemetry to receive data. Properties/Methods communicate with tenant-specific DBs for real-time settings. In addition, the changes in IoT Edges on the shop floor through Telemetry are matched in both directions to match the settings to the shop floor in the tenant-specific DB. Data on changes in the shop floor other than IoT data are entered into service without going through the Digital Twin Agent in a way using MQTT. Big Data Analytics & AI Service is mainly responsible for large-scale analysis, fine-tuning parameters, and upgrading the analysis model of IoT Edges. Offline modeling performs AI analysis with big data or pre-training of big data to perform data analysis on the shop floor. Parameter Optimization optimizes parameters to increase the accuracy of the analysis of Offline Modeling. In Reconstruction/Upgrading, the analysis is performed with optimized parameters. In addition, the pre-trained analysis model is sent to IoT Edges to enable small-scale analysis models on the shop floor.

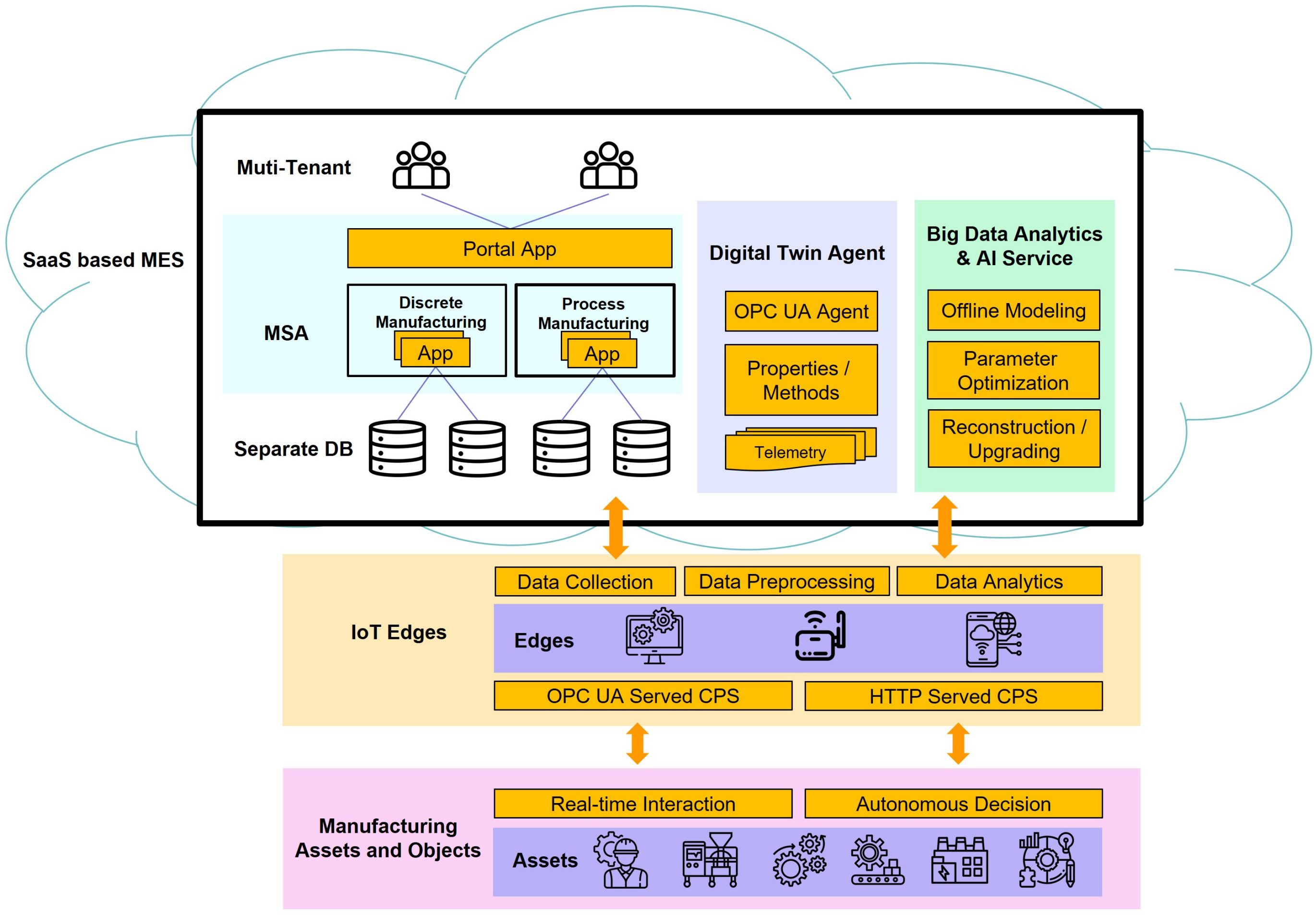


Figure 4. SaaMES Architecture.

3.2. MSA in SaaS-Based MES

The advantage of SaaS-based services is that the initial investment and maintenance costs are low. However, if we customize and service SaaS-based services for each factory, we cannot take that advantage. Thus, not individual customization at the beginning of development, but general customization with factory know-how by industry and MSA, which divides each service, are crucial technologies to service SaaMES. In this study, we propose a standard architecture by applying the classification of the manufacturing execution system using the manufacturing approach and MSA. When switching from On-Premise to SaaS service, building and distributing as one image in the existing approach does not take advantage of SaaS. Previously, different versions of the service were offered by managing each customer's version. However, while switching to SaaS, it is critical to apply MSA and identify the characteristics of each customer to offer customized services. In addition, with the development of IoT, many data with each characteristic such as production, quality, logistics, and facilities are generated, and smart factories are being developed with the aim of fully automating factories through that data. By modularizing each characteristic through MSA, it is possible to manage more efficiently through resource management distributed within SaaS. As shown in Figure 5, virtual servers can be divided and provided on a customer-by-customer basis by the management of tenants in SaaS. The basic purpose of SaaS is to reduce costs by fully using the services that the service provider offers. However, the approaches employed by each factory are slightly different, and there are numerous places where the process is independently established and employed according to domain know-how. This is also why versions of the service were managed differently for each user in one solution, as shown in Figure 5. To become a universal system, it is essential to offer common functions and provide different support through domain know-how. Furthermore, it is crucial to change the type of work on the site so that customization does not produce a version appropriate for a specific plant and the process of a manufacturing execution system divided by the manufacturing industry can be applied. It is possible to generate optimal productivity by leading manufacturing innovation by applying optimized processes to the manufacturing field.

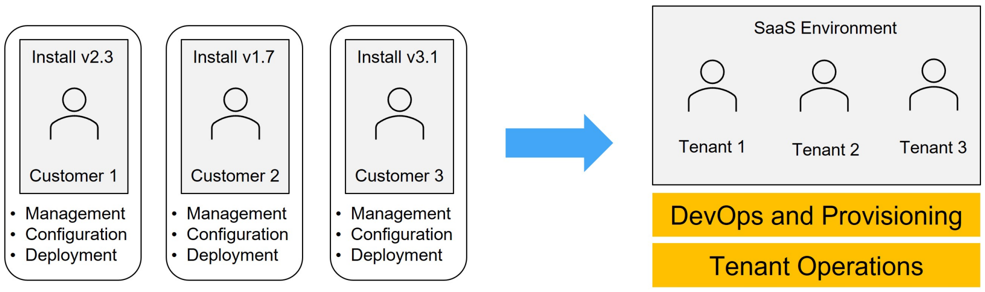


Figure 5. SaaS Model with MSA and MTA.

When SaaS-based MES is combined with functional MSA, the following advantages can be assured. Users pay as much as they register and employ only the functions they need, and wasted resources because of functions that are not used are reduced in terms of Cloud management. Furthermore, since containers are virtually mechanized and employed for each service, non-disruptive services are possible in builds and deployments. Thus, Cloud service can be managed more effectively than the existing On-Premise approach with the MSA approach.

When switching from On-Premise, the first thing to think about is to separately define what can be used in common and design it by collecting common functions. Applicationally, a common function should be created so that only each module can be referred to for use when it is executed. Likewise, in the database, the standard information table commonly used should be separated and managed by MDM. MDM is a basic application, so it is an essential module to select and use. Therefore, baseline informativeness is designed to be managed by MDM. The next is the separation of work areas. Basically, it separates the areas of work based on the statistics actually used in Figure 3. The important point here is that each module should be able to perform independent tasks. The reason is that even if only one of the modules of SaaMES is selected, it should be possible to execute. However, since production must be possible for the purpose of SaaMES, PM, QM, and MDM are essential systems. When it is separated by task, the DB connection of each module must be conceived independently. The reason is that DBs used may be different from each other, so they should be scalable. For example, since there may be an external system-linked service, it should be configured and managed so that it can be configured for each module. Resources for each module can be set up by checking the shop floor for each tenant.

Looking at Figure 6, the Backbone Platform is commonly applied. Each customer employs a different amount of service, and the Expansion Platform can be employed by adding a platform that lacks the Backbone Platform. In the application area, Master Data Management is provided as the default application. Additionally, by applying the MSA, the modules were divided by tasks and main purpose. It is divided into Process Management, Quality Management, Resource Management, Logistics Management, Artificial Intelligence Management, Work Place, and Resource Planning. The application may be selected and serviced according to the user's usage. The Backbone Platform is an area provided by default. Cloud infrastructure is an IaaS that connects networks, virtualization, servers, and storage in the Cloud. SaaS Engine/Management is an area where container services and monitoring are possible. Data Integration is an area where data collected by the Digital Twin Agent and interfaced data are collected, loaded, processed, and stored. Development Platform offers development environments such as IDE, UI Framework, and DevOps. The AI/Analytics Platform offers an analysis engine and model development environments such as analysis algorithms and data visualization. Common Platform offers basic services such as user rights, authentication, logging, search, and encryption. Catalog Platform offers catalogs such as APP/UI, data, analysis models, and APIs. The User Portal provides platform home configuration and service UI Portal for each user. The Market Place Portal provides users with the ability to purchase and use MSA-enabled modules in real time. Expansion Platform is a service offered by the Backbone Platform, which is provided, only for users who want additional features or enhanced specifications. For example, if we want more than basic AI functions, we can receive additional functions through service contracts. Furthermore, if the service is not provided because of the basic specifications, additional services are provided as an advantage based on the Cloud, so stable services can be employed.

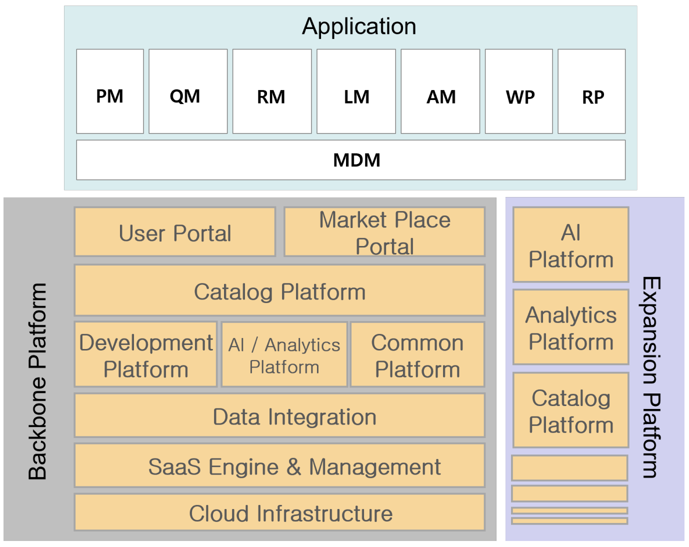


Figure 6. MES Model of SaaMES.

3.3. MTA in SaaS-Based MES

As for the multi-tenant characteristics, a single-tenant application is a structure in which software is installed for individual tenants, and a multi-tenant application is a structure in which tenants share software and use it together. Therefore, it is a structure that can be expected to reduce the overall operating cost (such as license and operating personnel costs).

- Cost reduction
- Easy to analyze and fuse data
- Simplify the software updater process
- Focus on business processes
- Delivers efficient, sustainable scalability
- Service failures affect all tenants when providing single instance-based multi-tenants
- Restricts the delivery of specialized features per tenant and increases complexity in delivery

The advantage of the multi-tenant approach is that it can increase resource utilization, while it affects all tenants in the event of a service failure. Therefore, in order to prepare for such a failure, hedging measures such as distributing applications to multiple instances should be considered.

The main considerations in terms of application design are as follows. First, it is necessary to provide a means to distinguish tenants who are using the application from within the application. Second, the behavior of one tenant should not affect the behavior of another tenant (transaction isolation). Third, the application should present a programmatic means to enable the isolation of data by the tenant, and the application should be logically and physically partitioned (data isolation). Fourth, through application partitioning, different levels of service for each tenant, differences in many characteristics must be provided.

SaaMES manages the multi-tenant as a configuration. A plan to efficiently manage various tenant configurations should be prepared, and when changes occur, they should be reflected at runtime. A configuration file of a typical single tenant application exists inside the application.

MTA is crucial for SaaS-based services. Multi-tenant architecture refers to the principle of building a software architecture for applications implemented to accommodate multiple tenants in the same infrastructure. If the application upgrade process itself has numerous instances, complexity and operating costs increase as the number of customers increases. Thus, in a single instance virtualized application, a multi-tenant application that can serve all customers with the same OS, the same SW, and the same specifications become the standard for SaaS. This eliminates the problem of modifying and redeveloping applications for new customers. Furthermore, the time-to-market of the service can be secured to provide applications to meet the needs of the market as a subscription-based service. In other words, MTA allows one logical database instance to be shared, business logic can be applied to all users, and these technologies are crucial for providing SaaS. In this study, to determine the appropriate service type for MES, the service types for each multi-tenant were classified into five types, as illustrated in Figure 7, and the characteristics of each type are summarized in Table 1.

Figure 8 shows the SET method's architecture in the physical independence method. Physical resources for each tenant, which are previously capacity-calculated in consideration of users, can be accessed by assigning an access URL for each tenant without SET configuration and modification of existing applications.

Figure 9 shows the architecture of the provisioning method in the physical independence method. The independent infrastructure configuration minimizes the modification of existing applications and adds tenant identification logic and routing logic. Provisioning with the same source and image (machine.container) per tenant is effective for management.

Figure 10 shows the DB separation architecture in a logically independent method. Configure an independent DB for each tenant and add tenant identification logic and routing logic to the existing AP. Existing data can be used as it is, existing AP modifications are minimal, and DB security is outstanding.

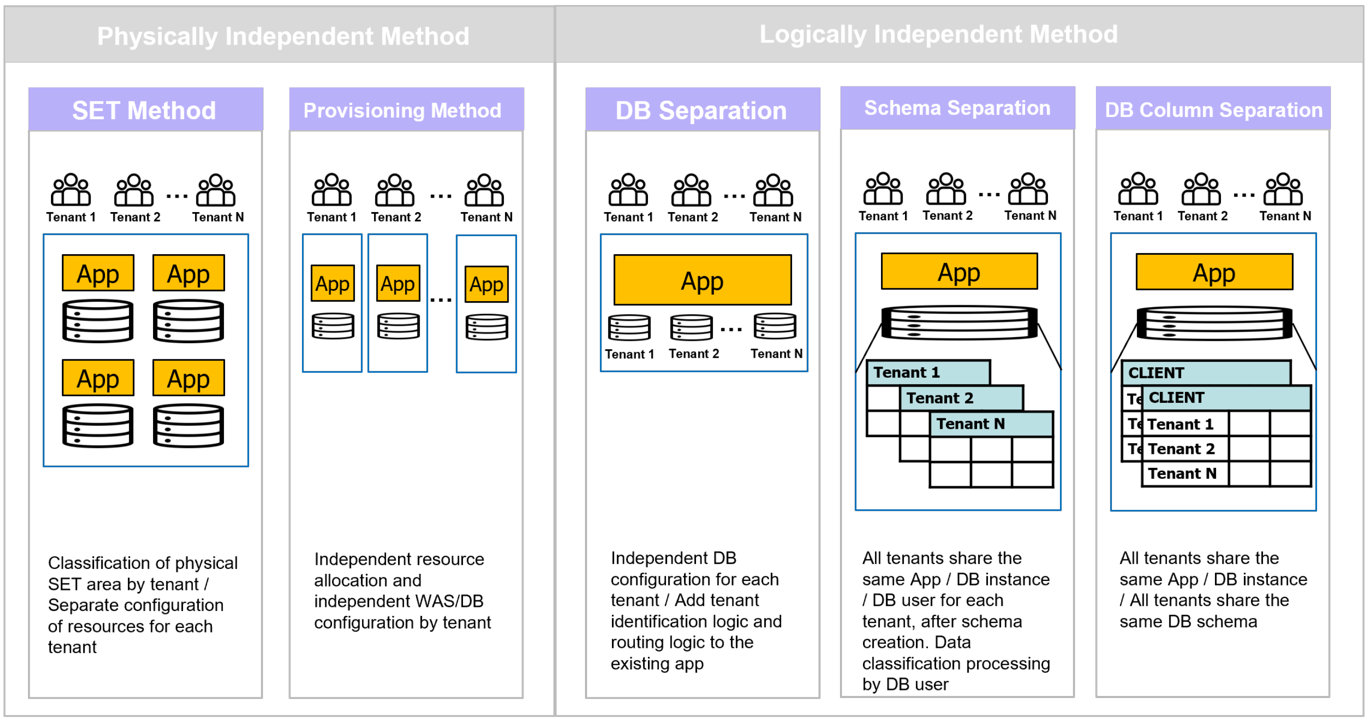

Physically Independent Method
Logically Independent Method
SET Method
Provisioning Method
DB Separation
Schema Separation
DB Column Separation
Tenant 1
Tenant 2
Tenant N
App
CLIENT
Classification of physical SET area by tenant / Separate configuration of resources for each tenant
Independent resource allocation and independent WAS/DB configuration by tenant
Independent DB configuration for each tenant / Add tenant identification logic and routing logic to the existing app
All tenants share the same App / DB instance / DB user for each tenant, after schema creation. Data classification processing by DB user
All tenants share the same App / DB instance / All tenants share the same DB schema

Figure 7. Five Categories of MTA.

Table 1. Advantages and Disadvantages of Key Transition Procedures.

Type	Key Transition Procedures	Advantage	Disadvantage
Physically Independent SET Configuration Method	(1) Physical SET area classification by tenant/resource configuration by tenant/physical resource composition after capacity calculation by SET considering the number of users (2) Configure access by granting access URLs by tenant without modifying the AP (3) Keep your existing data intact	- Fewer modifications to existing APs - Tenant physical independence - Less interference impact - Improved security by tenant	- Increased operational /management costs - Tenant addition takes long - Deploy separately by tenant
Physically Independent Provisioning Method	(1) Independent resource allocation and independent WAS/DB configuration by tenant (2) Adding tenant identification and routing logic to an existing AP (3) Provision solution packages in machine or container image configurations (4) Keep your existing data intact	- Modifying existing APs minimal - Tenant Physical Independence - Less interference impact - Improved security by Tenant - Easy to add Tenant	- Expensive to operate /manage - Deploy separately by tenant
Logically Independent DB Separation	(1) Tenant-specific independent DB configuration (2) Adding Tenant Identification and Routing Logic to an Existing AP (3) Keep your existing data intact	- Modifying existing APs Minimal - Improve DB Security	- Expensive to operate /manage - Deploy separately by tenant
Logically Independent Schema Separation	(1) Share all tenant identical WAS/DB instances (2) Data classification processing by DB user after generation of tenant DB user and schema (3) Transfer existing data to DB user-specific tables for use (4) Mastery tables such as reference information are shared by all users regardless of DB user (5) Utilize common modules for tenant-specific DB access to minimize modification of existing APs	- Low operating /management costs - Tenant logical independence - Easy to scale WAS/DB - Single deployment - Minimum AP modification compared to DB column base - Easy to manage data	- Many modifications to existing APs - Mixed data - Relative security vulnerability - A lot of interference effects
Logically Independent DB Column Separation	(1) Share all tenant identical WAS/DB instances (2) All tenant same DB schema sharing (3) Data classification by tenant ID after configuring DB table based on tenant ID column (4) Reflect additional tenant IDs to existing data (5) Mastery tables such as reference information are shared regardless of tenant ID (6) Add existing AP tenant ID-based data processing logic (reflect SQL tenant ID)	- Low operating /management costs - Tenant Logical Independence - Easy to scale WAS/DB - Single distribution	- Many modifications to existing APs - Mixed data - Relative security vulnerability - A lot of interference effects

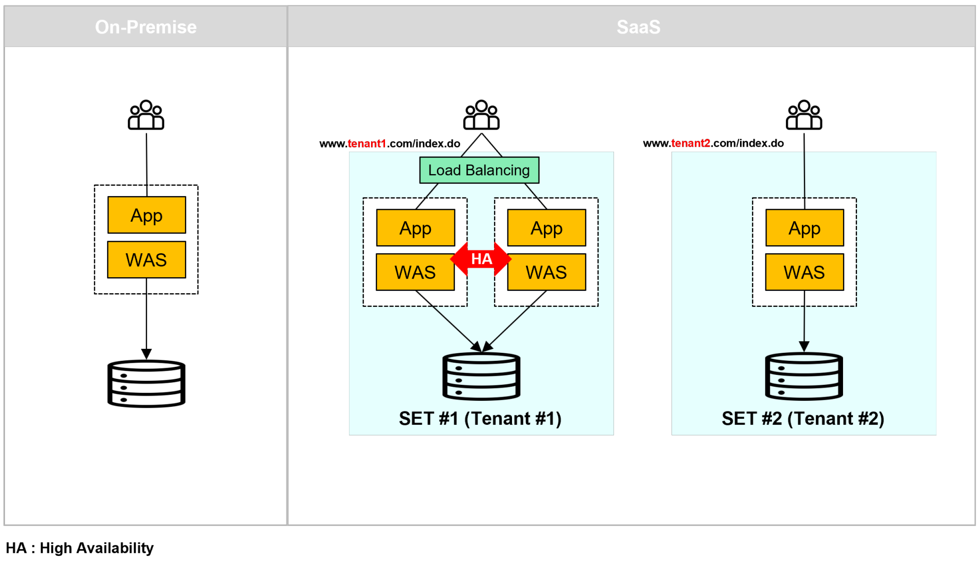


Figure 8. SET Method.

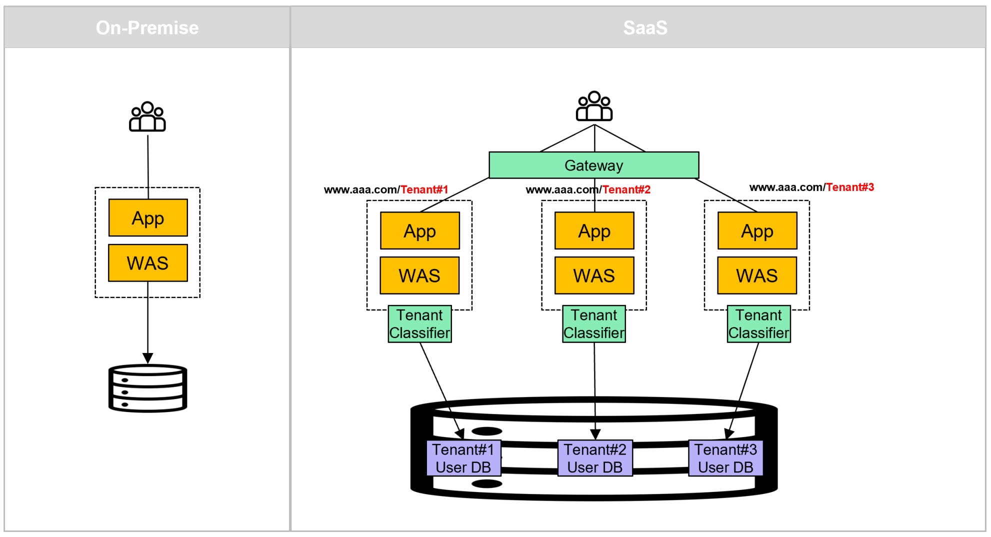


Figure 9. Provisioning Method.

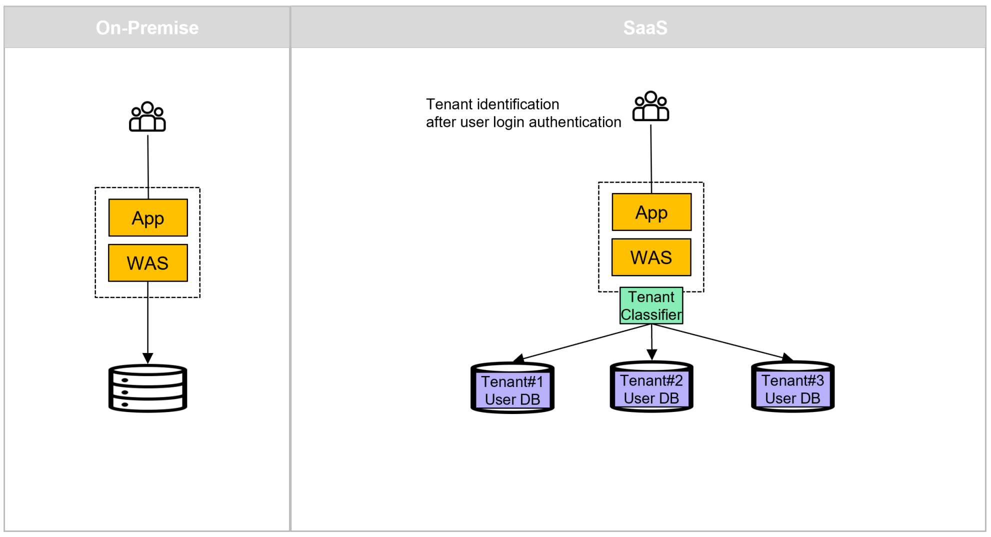
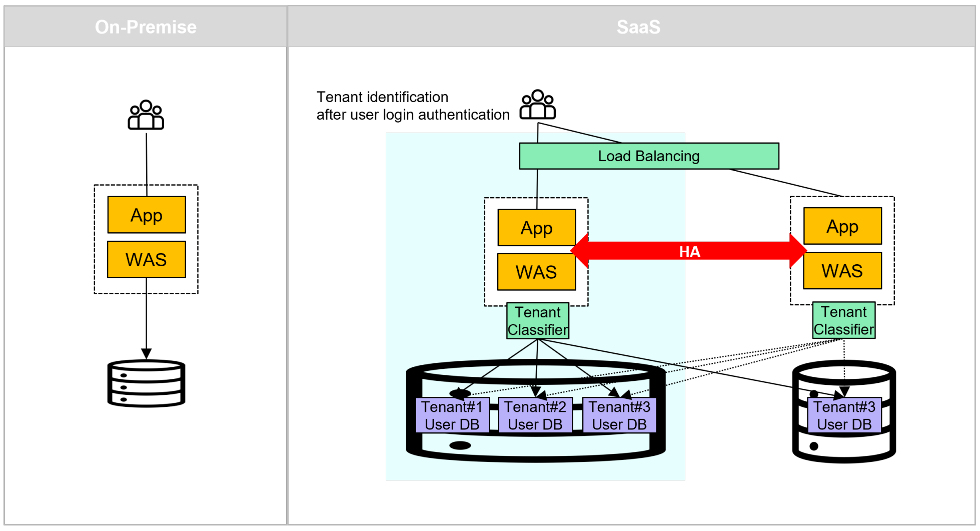


Figure 10. DB Separation.

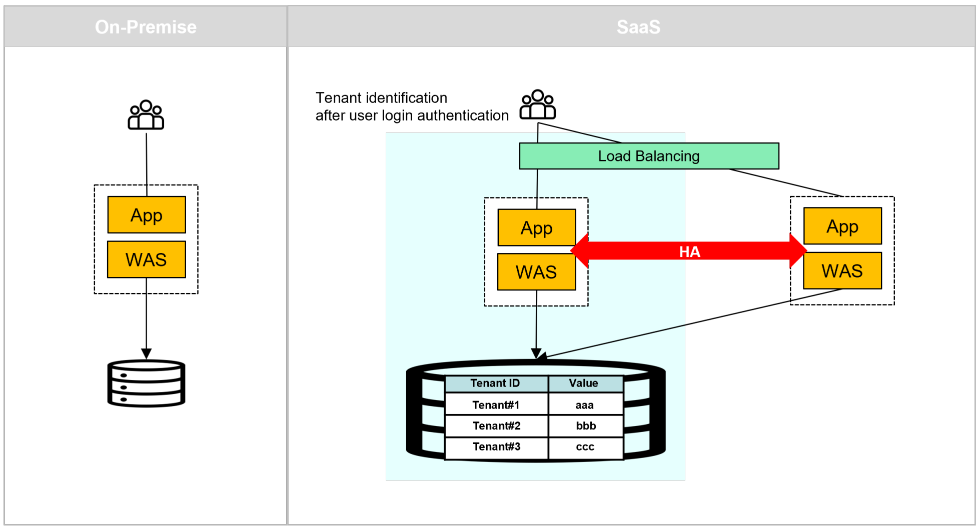
Figure 11 illustrates the architecture of DB schema separation in a logically independent method. All tenant identical WAS/DB instances are shared, DB users and schemas for each tenant are created, and then data are processed for each DB user.



HA : High Availability

Figure 11. DB Schema Separation.

Figure 12 shows the architecture of DB column separation in a logically independent method. Share all tenant identical WAS/DB instances, share the same DB schema within the DB, and process data based on tenant ID.



HA : High Availability

Figure 12. DB Column Separation.

As demonstrated in Table 2, it is essential to establish an SaaS strategy and select a conversion type according to the cost, time, operation efficiency, business requirements, and technical maturity of resource separation and sharing.

Table 2. MTA Transition Strategies for Individual On-Premise System.

SaaS Strategy	Multi-Tenant Implementation	Infrastructure/Availability Configuration	Service Operations
Physically Independent (Requirements) - Data security - Minimum failure impact - Quick transition	(AP) - Minimum AP source modification - Source machine/container image-based provisioning (easy to deploy) - Configure separate deployments per tenant (Data) - No data changes	(Infrastructure) - Configuring independent resources by tenant - Peak time/maximum user based sizing (Available) - Traditional redundancy configurations (WEB/WAS/DB redundancy)	(User authentication) - Accessing URLs by tenant - User authentication by tenant URL (Charging) - Independent configuration resource - based billing
Logically Independent (Requirements) - Cost savings - Efficient resource management - Flexible scalability	(AP) - Adding tenant identification logic - Reflect logic processing by tenant - Configuring a single deployment (Data) - Tenant DB configuration data transfer - Reflects data processing by tenant	(Infrastructure) - Single instance WEB/WAS/DB configuration - Initial sizing considering maximum tenants (Available) - Flexible scaling as tenants change - Tenant increase/decrease Scale-out/in configuration	(User authentication) - Sccess to the same URL, login - Identify tenants after user authentication (Charging) - Charging based on service or resource usage (cost efficient)

SaaMES strategically applies the logically independent DB separation and DB schema separation approaches in parallel for performance and security. Since the manufacturing plant's data are sensitive for security, DB is provided separately for each company. However, if several factories within the same company provided services, the factory is divided into DB schemas and access by each user is possible. Each company does not need to separate schemas from one DB, but as IoT technology develops and big data use increases, data may increase and cause speed reduction when inquiring in one schema, so it is appropriate to logically divide the storage space.

As for the multi-tenant application resource configuration plan, the components of the application consist of the WEB, WAS, Connector (WEB $\leftrightarrow$ WAS), Identification System, and Integration System by layer. In the tenant application, these factors should be properly partitioned to provide tenant-specific services so that they do not interact. Application partitioning affects scalability, availability, and management. The largest range of tenant-specific separations is one subscription tenant and group multiple tenants in a subscription, which separates tenants by subscription of a multi-tenant application. Tenant separation on H/W and S/W is one tenant per Cloud service (VM), group multiple tenants in a Cloud service (VM), and group multiple tenants in a WEB/WAS/other elements.

As for the multi-tenant data source configuration plan, if a multi-tenant application has a different data source for each tenant, the application should be able to find the appropriate data source at runtime according to the tenant information. In addition, if a new data source is created, it should be recognized and available to the application. For a typical Java Enterprise application, WAS uses a database and Connection Pool to connect, and the application (Framework) uses the connection pool declared in WAS to look up as JNDI. In other words, it takes the form of using a pre-declared data source in WAS. In order to provide each service, access to data sources by tenants must be considered. This approach requires binding and using the appropriate data source according to the tenant ID within the application in a way that each tenant has a separate data source (account). In other words, the data source must be dynamically routed. We look at these Dynamic

Datasource Routes based on the Java Framework, Spring. In the case of data access, the routing method of finding the appropriate data source according to tenant information per request would be suitable for OLTP applications, and swapping data sources according to tenant information is an easy way to process data from one tenant.

Finally, for a multi-tenant log configuration scheme, logging should have separate log files per tenant, and certain tenants should not be able to access log files from other tenants. Since multiple tenants use the service for multi-tenant applications, the information of the tenant must also be output when the log is output. This should only be valid while the request is being processed and should not affect between threads. In addition, if a log file corresponding to a tenant is routed and each file/directory is classified, log file management becomes easy. For this logging, logback, and log4j use a Mapped Diagnostic Context (MDC). It can be used for logging by setting the tenant ID to the MDC at the start of the request and removing it from the MDC at the end of the request. Setting up when a request starts and removing it when it ends is only valid while the request is being processed, thus preventing the use of false data even in WASs that recycle the thread. When managing logs by the tenant through Logback, it should not be forgotten that the MDC should set it up at the beginning of the request and erase the setting at the end of the request. WASs that share ThreadPool can cause incorrect references.

Considering the above method, SaaMES applied with MTA is applied by applying a logical DB separation and provisioning method. Then a method for multi-tenant testing and monitoring is needed to ensure that it is applied well. In actual application, the method was defined and performed. In addition to functional/non-functional testing of general applications, additional tests such as access control of resources by the tenant, customization by the tenant, and backup/recovery of tenant-specific data are required. The test of a multi-tenant application will be the same as the single-tenant application and will be divided into functional and non-functional tests. In addition, a test specialized in a multi-tenant application is required. The multi-tenant application requires testing for isolation in all respects. In addition, testing in a runtime environment for tenant-specific customization is required. In general, customization involves code modification or application redistribution, but in multi-tenant applications, these modifications should not affect other tenants. Therefore, a test item is needed in terms of multi-tenant application development and operators.

3.4. Real-Time Control in IoT Edges

With the development of IoT, centralized methods are inefficient. Therefore, IoT Edge has been proposed for distributed processing on the shop floor. Simply the distribution method of the shop floor does not completely improve centralization. As a result, MSA can be deployed in SaaS-based MES to select and use per-tenant action usage and simplify resource management by distributing shop floor data across modules. In addition, if the data of a particular task grows rapidly, it can efficiently manage resources by adding modules to docker containers, an important technology in the Cloud. We propose IoT Edges because there are more network traffic and connection pools due to the rapidly increasing digitized data of the shop floor. The aspect of resource efficiency is also important, but the real-time aspect is also an important factor in creating a real-time-based CPS of the developed smart factory.

In Figure 13, a node is a set of general PC levels in the area of IoT Edges. Data transmitted via OPC-UA and HTTP in real time from Manufacturing Assets and Objects are collected and preprocessed, and then refined data is sent to SaaS-based MES to reduce the load on SaaS and network resources. The Digital Twin Agent exists to receive data transmitted from the IoT Edge by the OPC-UA. Each type stored in Properties/Method is split into Telemetry and remotely connected, and the received data is collected from the OPC-UA Agent and sent to the promised module for each application divided into MSA. The data sent to each module are processed by each business process and entered into a separate DB. The Digital Twin Agent connects to IoT Edges with the promised definition with the settings stored in the MDM of the application and sends them according to the data transfer rules. When the recipe settings of the Asset are changed from MDM or the parameters of Big Data Analytics & AI Service analysis results are verified by the AM module and the settings are stored in MDM, the Digital Twin Agent sends the changed settings of MDM to IoT Edges. The recipe settings required for the Asset are sent from IoT Edges to ensure that production is not disrupted.

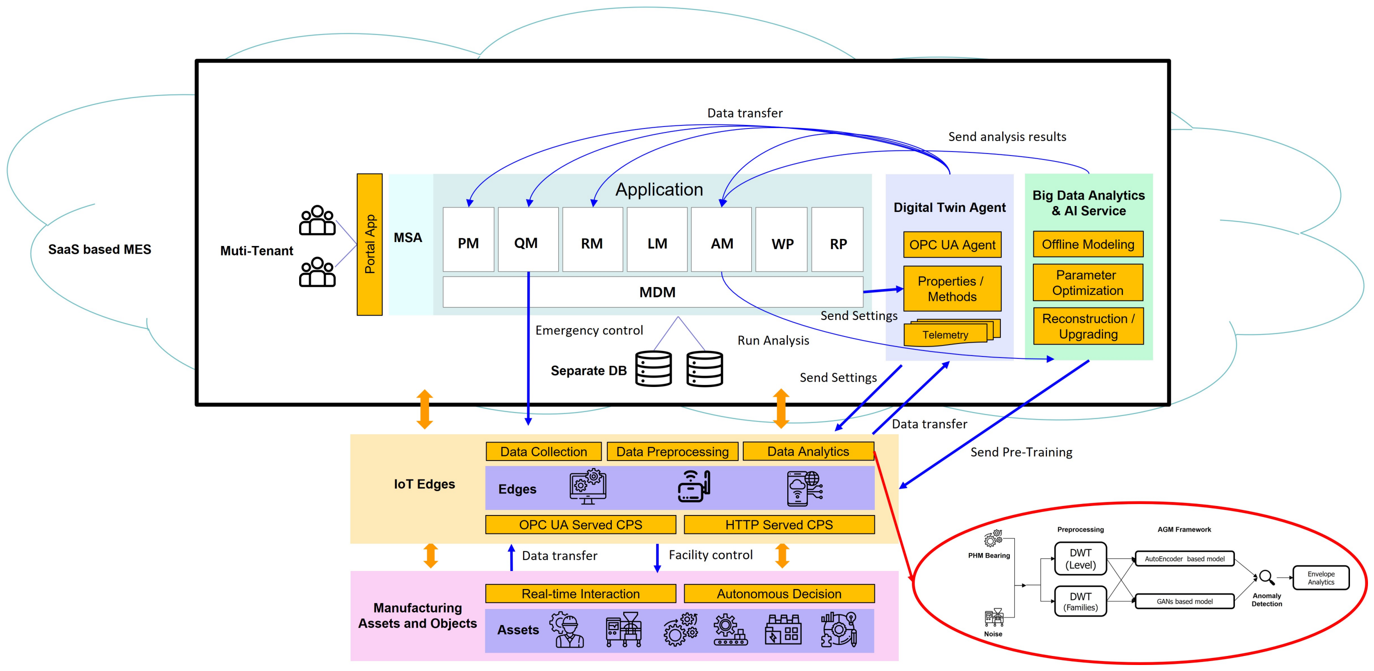


Figure 13. Overall Roles of IoT Edges.

In the process of preprocessing data, IoT Edges detect state anomalies in the Asset and alarms and controls data outside the Upper Specification Limit (USL)/Lower Specification Limit (LSL) in Data Analytics. Small-scale data analysis is possible on IoT Edge. Large-scale analysis and initial learning models are run on Cloud; however, in the case of small-scale analysis where the learning model is completed, it can be processed by IoT Edge to control assets in real time. Furthermore, the individual reference information needed for production progress is stored on IoT Edge in case of unavoidable connection to SaaS-based MES, so that the production process is not disrupted in case of an emergency, and the server sends loaded data to prevent loss of production data.

The role of IoT Edge in SaaS-based MES is a crucial area of SaaMES for securing real-time because of efficient data volume, network resources, and network connections. Launch SaaS-based MES with MTA and MSA, and confirm that data is transmitted to each module and transmitted to the module that became MSA. A processing time comparison experiment using anomaly detection through small-scale analysis is conducted to confirm the need for IoT Edge. We modeled and examined the vibration of the bearing with anomaly detection, and experimented with whether real-time control is possible through real-time anomaly detection in IoT Edge, as illustrated in Figure 14.

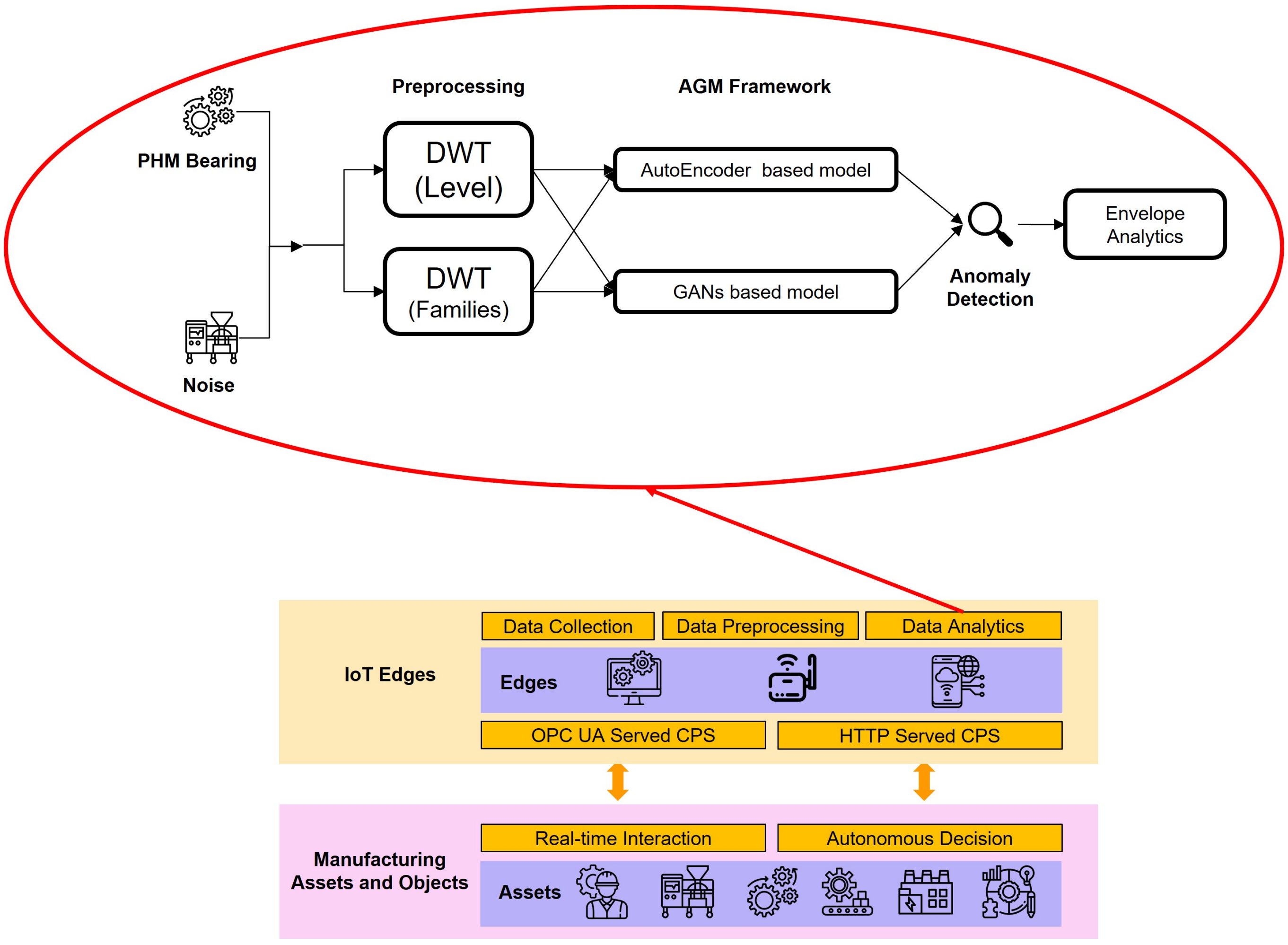


Figure 14. Small-scale Analysis Model through Bearing Vibration in IoT Edge.

4. Experiment and Results

4.1. Experiment Environments

A test environment was configured to experiment with Figure 13. Python was the computer language employed in the experiment, and the algorithm was tested on computers equipped with the following equipment in the IoT Edge and Cloud environments, as shown in Table 3.

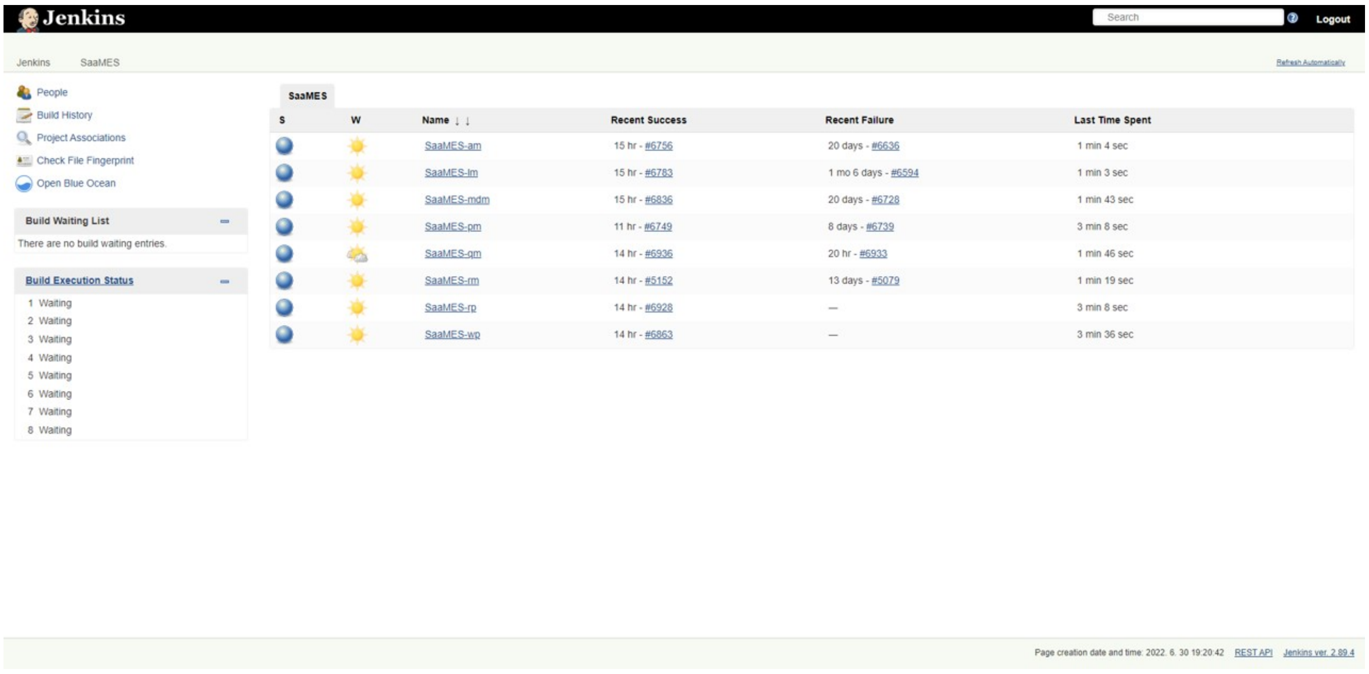
In this experiment, SaaMES is executed to transmit data to each application module whose data became an MSA through IoT Edge. When a configuration change occurs, IoT Edge detects it and reflects it in Asset, and it runs a small-scale analysis on IoT Edge. It proceeds with additional execution of IoT Edge. The purpose of this additional experiment is to measure and compare the processing time of transmission between the sensor and IoT Edge in real time, the anomaly detection through analysis on IoT Edge and Cloud, respectively, and the transmission of the analysis result to the shop floor.

Table 3. Test Environment.

	CPU	RAM	GPU
IoT Edge	AMD Ryzen 5 3600 6-Core Processor 3.60 GHz	16.0 GB	NVIDIA GeForce GTX 1660 SUPER
Cloud	Intel Xeon E5-2686 v4 vCPU 8 2.30 GHz	61.0 GB	NVIDIA Tesla V100

4.2. Deploying SaaMES

According to Figure 15, each module of SaaMES applied with MSA was containerized and registered with Jenkins. A build is called a build when it is ready to be uploaded to the server. The deployment is called the deployment, which is uploaded to the server and made available to the user. As it becomes an MSA, the timing of distribution is fast and repetitive to respond to changes quickly. So I think it is essential to use automatic distribution tools in MSA-enabled applications. This program is one of the automated deployment tools that automatically deploys from build to deployment, and also allows scheduling, making it easy to deploy in MSA.



Build Waiting List

There are no build waiting entries.

Build Execution Status

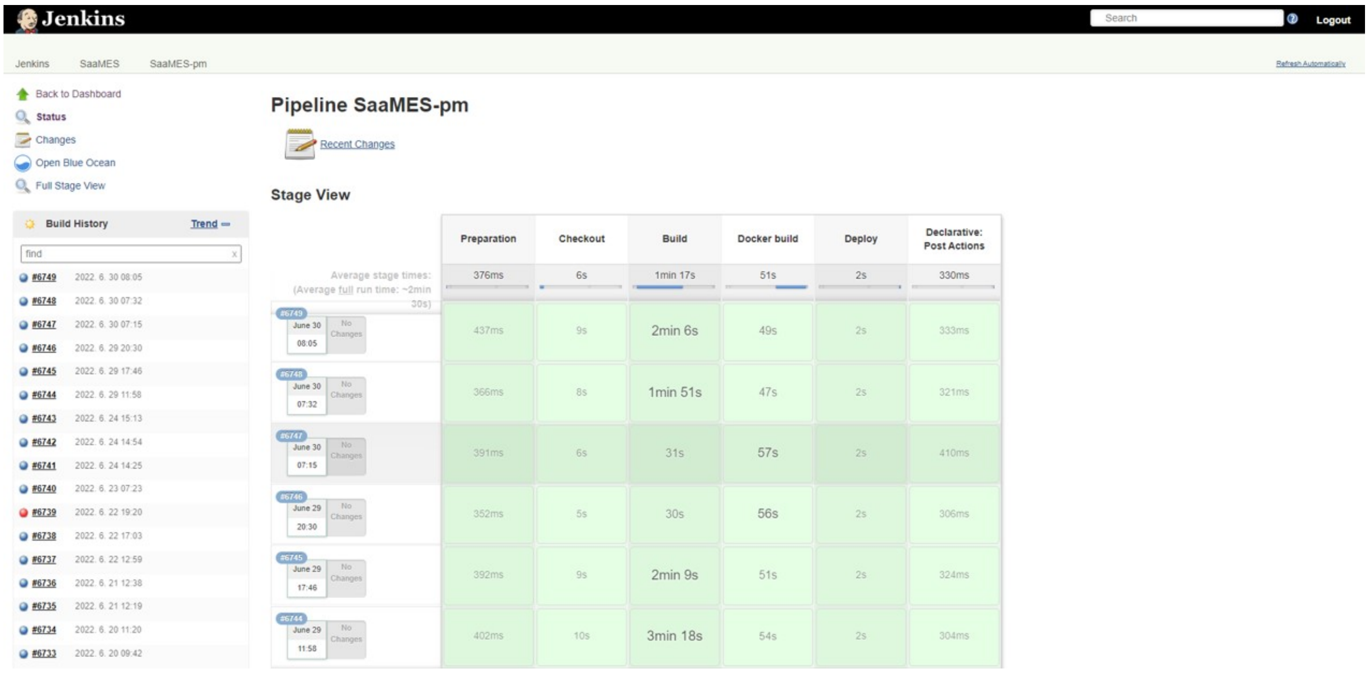
1 Waiting
2 Waiting
3 Waiting
4 Waiting
5 Waiting
6 Waiting
7 Waiting
8 Waiting

S	W	Name ↓ ↓	Recent Success	Recent Failure	Last Time Spent
●	☀	SaaMES-am	15 hr - #6756	20 days - #6636	1 min 4 sec
●	☀	SaaMES-lm	15 hr - #6783	1 mo 6 days - #6594	1 min 3 sec
●	☀	SaaMES-mdm	15 hr - #6836	20 days - #6728	1 min 43 sec
●	☀	SaaMES-pm	11 hr - #6749	8 days - #6739	3 min 8 sec
●	⛅	SaaMES-qm	14 hr - #6936	20 hr - #6933	1 min 46 sec
●	☀	SaaMES-rm	14 hr - #5152	13 days - #5079	1 min 19 sec
●	☀	SaaMES-rp	14 hr - #6928	—	3 min 8 sec
●	☀	SaaMES-wp	14 hr - #6863	—	3 min 36 sec

Page creation date and time: 2022. 6. 30 19:20:42 REST API Jenkins ver. 2.89.4

Figure 15. Modularized and Registered SaaMES in Jenkins.

As shown in Figure 16, it is automatically deployed from the build. Each of the eight modules made up of MSA takes an average of 90 s to execute. Initially, when using the same tool before applying the MSA, the execution time was about 290 s. However, as MSA is applied and the amount of source is divided, distribution execution time is also reduced, so rapid distribution is possible, although there is not much difference. In addition, it is easier to respond to rapid changes as only the modules in question can be modified and distributed by applying MSA. When it was a previous application, it was burdensome because it had to distribute the whole thing if there was a problem with one task. Nevertheless, SaaMES to which MSA is applied can be said to adapt to change quickly. In addition, resource management is easy for each module, making resource management more efficient than before MSA was applied.



Pipeline SaaMES-pm

Build History Trend

#6749	2022. 6. 30 08:05
#6748	2022. 6. 30 07:32
#6747	2022. 6. 30 07:15
#6746	2022. 6. 29 20:30
#6745	2022. 6. 29 17:46
#6744	2022. 6. 29 11:58
#6743	2022. 6. 24 15:13
#6742	2022. 6. 24 14:54
#6741	2022. 6. 24 14:25
#6740	2022. 6. 23 07:23
#6739	2022. 6. 22 19:20
#6738	2022. 6. 22 17:03
#6737	2022. 6. 22 12:59
#6736	2022. 6. 21 12:38
#6735	2022. 6. 21 12:19
#6734	2022. 6. 20 11:20
#6733	2022. 6. 20 09:42

Stage View

	Preparation	Checkout	Build	Docker build	Deploy	Declarative: Post Actions
Average stage times: (Average full run time: ~2min 30s)	376ms	6s	1min 17s	51s	2s	330ms
#6749 June 30 08:05 No Changes	437ms	9s	2min 6s	49s	2s	333ms
#6748 June 30 07:32 No Changes	366ms	8s	1min 51s	47s	2s	321ms
#6747 June 30 07:15 No Changes	391ms	6s	31s	57s	2s	410ms
#6746 June 29 20:30 No Changes	352ms	5s	30s	56s	2s	306ms
#6745 June 29 17:46 No Changes	392ms	9s	2min 9s	51s	2s	324ms
#6744 June 29 11:58 No Changes	402ms	10s	3min 18s	54s	2s	304ms

Figure 16. Deploying SaaMES in Jenkins.

4.3. Data Processing Process

SaaMES tested the process of processing data from the Asset. The method was performed in the same order as in Figure 17. Receive data from the Asset. It is collected from IoT Edge and inspected through simple analysis through pretreatment. IoT Edge sends data to each module through the Digital Twin Agent. The module processes the data and stores it in the DB. Transmitting data according to the module is distributed through the Catalog Platform to the common framework. This is the normal order of data processing. However, there are two cases when there is a problem. First, the Asset is controlled as soon as a problem is identified in Data Analytics in IoT Edge. Second, after the data is processed on the module, it is analyzed by various methods such as statistical analysis or 6 sigma, and emergency control is performed directly on the IoT Edge when emergency control is required when abnormal signs of the line are identified. All control of the shop floor is possible through IoT Edge. When a parameter value is changed through a data input value, a value through MDM is delivered to the IoT Edge through the Digital Twin Agent, and the IoT Edge delivers the setting value required for the Asset or changes the setting value to the IoT Edge. The IoT Edge has Asset connection parameters or settings that need to be analyzed.

Next, a test with an analysis model was conducted. As in Figure 13, small-scale analysis is performed on IoT Edge with a pre-trained model in Big Data Analytics & AI Service. This test was conducted on predictive preservation with bearing data in the next subsection, considering that small-scale analysis would be possible in IoT Edges due to many advances in PC performance. This experiment is aimed at enabling IoT Edge to expand its reach in MES, saving Cloud resources while ensuring real-time.

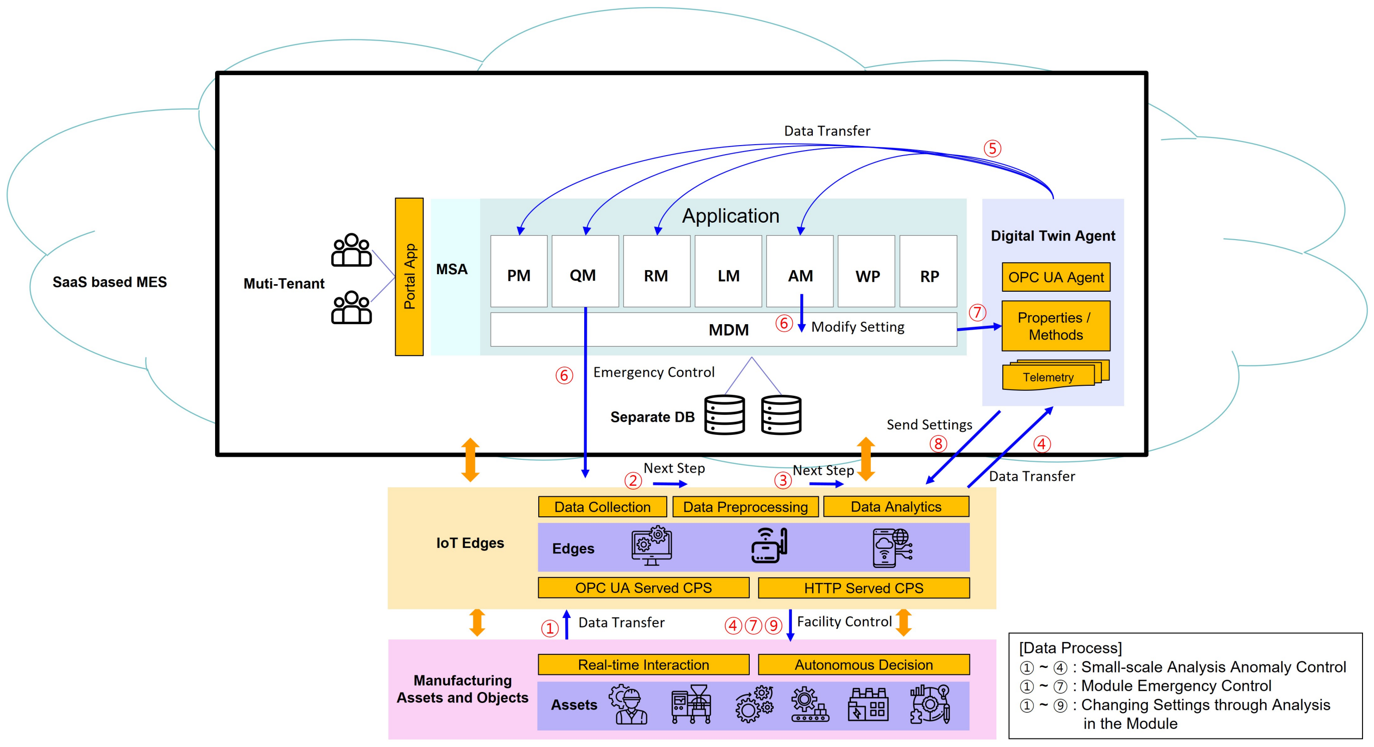


Figure 17. Processing Data in SaaMES.

4.4. Data Preprocessing

For anomaly detection, the "PHM IEEE 2012 Data Challenge" dataset, which is a dataset consisting of the vibration values of rotating bearings, was employed. The dataset consists of 7,172,877 vibration values collected from rotating bearings at 1800 rpm and 4000 N. The sampling frequency is 25.6 kHz, and data are recorded every 1/10 s when the bearing rotates.

If the raw data measured by the vibration sensor is employed as it is, it is challenging to examine it in a frequency form, and there is a high possibility that a false alarm occurs. Thus, it is crucial to enhance the quality of input data through the frequency conversion of raw data. Among the various frequency conversion approaches, discrete wavelet transform (DWT) with high time resolution in the high-frequency domain and resolution in the low-frequency domain was employed. The above features are crucial since the time resolution for rapidly changing high frequencies is more critical to determining the location of the point of change, and the frequency resolution for slow-changing low frequencies is more crucial. Figure 18 shows the result of converting the frequency by applying DWT to raw data.

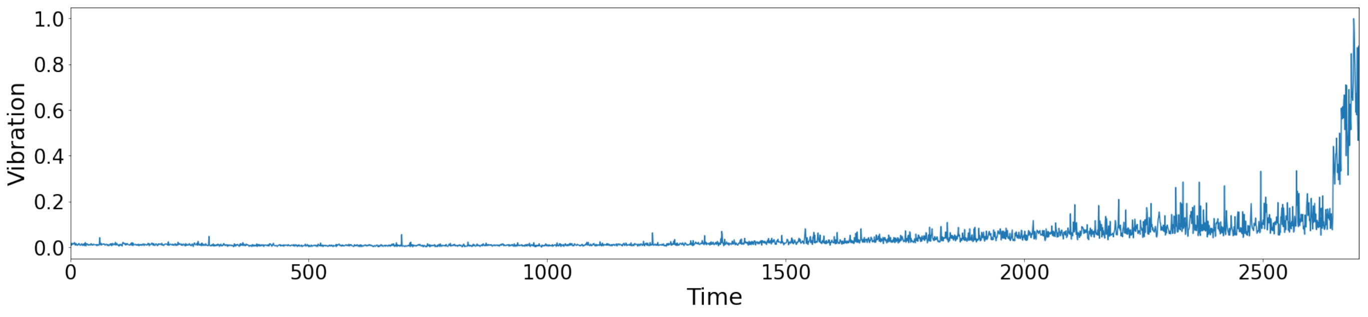


Figure 18. Bearing Vibration Data after DWT Conversion.

4.5. Anomaly Gap Maximization

In the AGM framework, models based on Autoencoder (AE) and Generic Adversarial Networks (GANs) were employed among various types of detection algorithms. Abnormal detection using AE employs reconstruction errors, which occur in the process of compressing and restoring data. An anomaly score means a score that indicates how close each input data point is to abnormal data and employs a reconstruction error as an anomaly score. Thereafter, the optimal threshold is selected based on the obtained anomaly score, and the point with a score above the threshold is judged as abnormal data. The threshold is selected by considering ISO-10816's reference value for the frequency of vibration defects, field abnormalities, and failure experiences. In this experiment, Unsupervised Anomaly Detection on Multivariable Time Series (USAD), a model that performs anomaly detection tasks with unsupervised settings in multivariable time series, was employed. Figure 19 shows the anomaly score measured in the process of compressing and restoring data converted through DWT using USAD.

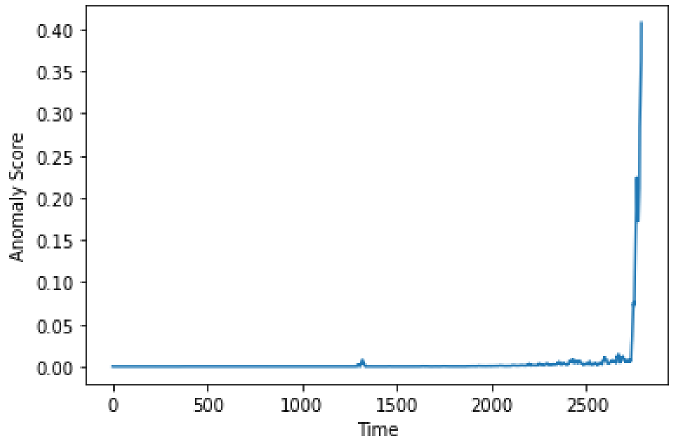


Figure 19. Anomaly Score Calculated by USAD.

Figure 20 shows the result of setting the threshold based on the measured anomaly score and performing the anomaly detection based on the threshold value. As shown in Figure 20, a value corresponding to an area painted in red is a value determined as abnormal data.

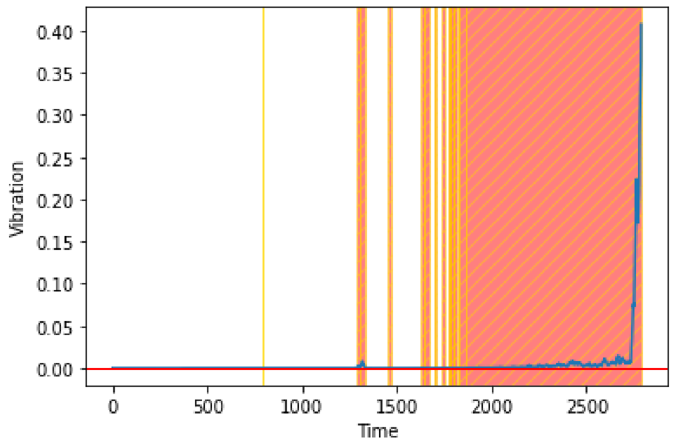


Figure 20. Anomaly Detection Using USAD.

Anomaly detection using GANs is not much different from AE-based anomaly detection; however, there is a difference in obtaining an anomaly score by using the output result of the decriminator along with the generator's reconstruction error. In this experiment, Timeseries Anomaly Detection GAN (TadGAN), a GAN model optimized for time series data anomaly detection, was employed. TadGAN has superior performance compared with other anomaly detection models and is recognized in various fields, so the model was employed in this experiment. Figure 21 shows the TadGAN model's result value for the data converted through DWT.

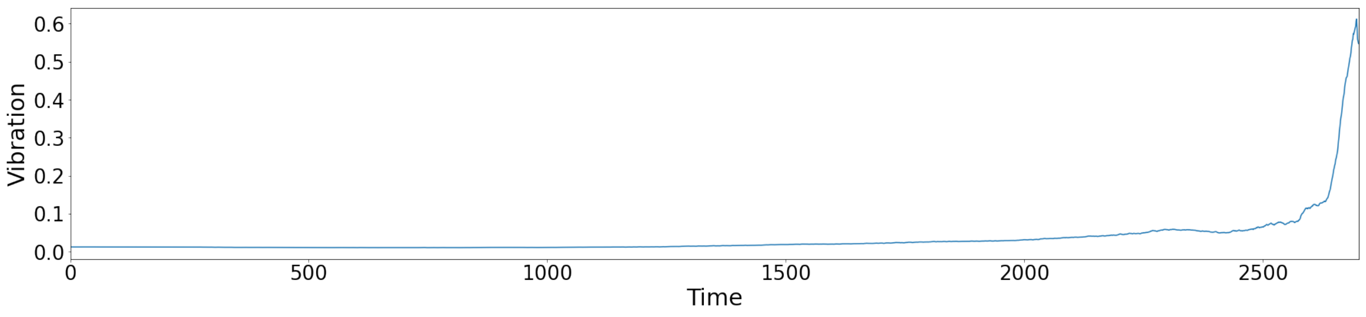


Figure 21. Generated Value of Bearing Vibration by TadGAN.

Figure 22 shows the results of anomaly detection based on the result value of the TadGAN model. Here a value corresponding to an area depicted in red is a value determined as an abnormal value.

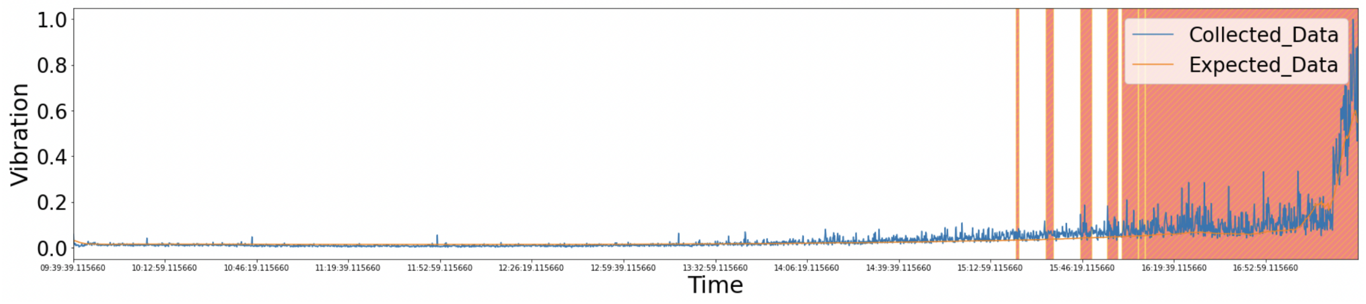


Figure 22. Anomaly Detection Using TadGAN.

4.6. Average Processing Time on IoT Edge and Cloud

To compare processing time in IoT Edge and processing time in the Cloud, the processing time of DWT Transformation and AE and GAN-based anomaly detection were measured in the Cloud and IoT Edge, respectively. Figure 23 shows the processing time needed to process data in real time in IoT Edge and Cloud environments.

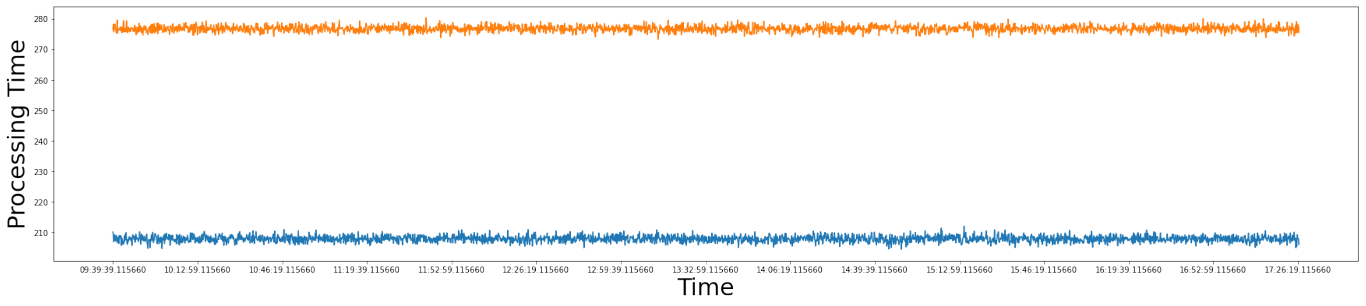


Figure 23. Required Processing Time to Process Real-time Data in IoT Edge (**Below**) and Cloud (**Above**).

At this time, the average processing time needed for DWT Transformation and AE and GAN-based anomaly detection was similar in both IoT Edge and Cloud environments. This is because the analysis of small-scale data employed in the experiment does not need much computing power, so there is no difference in processing speed between PC-class IoT Edge and Cloud. However, in the case of Cloud Computing, it takes additional transfer time to upload data from IoT Edge to Cloud and download analysis results, which increases processing time by about 33.1% compared with IoT Edge Computing. Based on this experiment's results, it is more effective in terms of the real-time processing of small-scale data on IoT Edge. This takes additional time to upload and download data to the Cloud; thus, it is recommended that IoT Edge processes an analysis of small-scale data. Furthermore, the use of IoT Edge in SaaMES is more crucial because if data resources (physical number of Assets and Objects) increase, the average processing time also increases due to increased throughput of network resources and Cloud Computing due to upload/download to the Cloud.

5. Conclusions

In this study, we mention the limitations of the existing On-Premise approach MES and propose SaaMES that combine SaaS-based MES with MSA and MTA, and SaaS combines real-time IoT Edge to maximize its advantages. In addition, resource management is an important task due to the separation of the application through MSA with the development of IoT. SaaS application has the effect of reducing initial development and maintenance costs, eliminating unnecessary functional use for users, and reducing Cloud management costs for providers. Furthermore, it is expected that MSA and MTA will be applied so that users can select and employ only the functions they require individually and serve different customers through multi-tenancy.

To apply SaaS, MTA is the most important thing. Therefore, we define five methods of MTA by dividing it into logical and physical independence and suggest a method of combining DB separation and schema separation of logical independent methods, which are MTA suitable for MES, by considering MES characteristics. We also applied MSA to take advantage of the growth of data and SaaS benefits of SaaS. MES' MSA increases system flexibility and scalability. In addition, with the development of EC, the role of IoT Edges has been expanded and presented, enabling distributed processing not only in Cloud but also on the shop floor. The actual numerical advantage of "real-time response faster" was presented compared with the Cloud environment by applying an analysis model using Autoendcoder and GANs to IoT Edge, which can respond and make decisions through real-time analysis. In other words, it is expected that SaaMES, which complements the lack of real-time Cloud-based SaaS-based MES using IoT Edge, will become the standard for SaaS-based MES in a complicated digital manufacturing environment. As a result, we presented an architecture of SaaMES that presents the vision of SaaS-based MES.

In previous anomaly detection analyses, the analysis results are expected to be inaccurate since the analysis environment of Cloud Computing and Edge Computing using AWS is different. Furthermore, there is a limitation that the advantage of the SaaMES, which is MES architecture with MSA and MTA does not show explicit effects, excluding IoT Edge. Additionally, I felt the need for research on how to measure the capacity of IoT Edges through load testing. Although there are various data on the shop floor, it was difficult to grasp the capacity of IoT Edges because it was not tested with various data. Our future study will focus on demonstrating the limitations of IoT Edge through various analyses, enabling SaaS-based MES to be realized based on measurements of the scale available for analysis on IoT Edge and the architecture of SaaMES presented in the study.

Author Contributions: Conceptualization, S.D. and J.J.; methodology, S.D.; software, H.C.; validation, S.D., H.C. and J.J.; formal analysis, S.D. and J.J.; investigation, S.D.; resources, S.D.; data curation, S.D. and H.C.; writing—original draft preparation, S.D.; writing—review and editing, S.D., W.K. and H.C.; visualization, S.D., W.K. and H.C.; supervision, J.J.; project administration, J.J.; funding acquisition, J.J. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2022-2018-0-01417) supervised by the IITP (Institute for Information & Communications Technology Planning & Evaluation). Also, this work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2021R1F1A1060054).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data used to support the findings of this study can be provided by the corresponding author upon request (jpjeong@skku.edu).

Acknowledgments: This research was supported by the SungKyunKwan University and the BK21 FOUR (Graduate School Innovation) funded by the Ministry of Education (MOE, Korea) and National Research Foundation of Korea (NRF).

Conflicts of Interest: The authors declare that they have no conflict of interest.

References

1. Atzori, L.; Iera, A.; Morabito, G. The internet of things: A survey. *Comput. Netw.* **2010**, *54*, 2787–2805. [CrossRef]
2. Vidhyalakshmi, R.; Kumar, V. Design comparison of traditional application and saas. In Proceedings of the IEEE International Conference on Computing for Sustainable Global Development, New Delhi, India, 12–14 March 2020; pp. 541–544. [CrossRef]
3. Sanctis, M.D.; Muccini, H.; Vaidhyanathan, K. Data-driven Adaptation in Microservice-based IoT Architectures. In Proceedings of the 2020 IEEE International Conference on Software Architecture Companion (ICSA-C), Salvador, Brazil, 16–20 March 2020. [CrossRef]
4. Aleem, S.; Ahmed, F.; Batool, R.; Khattak, A. Empirical Investigation of Key Factors for SaaS Architecture. *IEEE Trans. Cloud Comput.* **2021**, *9*, 1037–1049. [CrossRef]
5. Luit Infotech Article. Difference between the ASP Model and Saa Smodel. October 2013. Available online: www.luitinfotech.com/kc/saas-asp-difference.pdf (accessed on 11 June 2022).
6. Bhardwaj, S.; Sahoo, B. A particle swarm optimization approach for cost effective saas placement on cloud. In Proceedings of the IEEE International Conference on Computing, Communication & Automation, Greater Noida, India, 15–16 May 2015. [CrossRef]
7. Jiang, X.; Zhang, Y.; Liu, S. A well-designed saas application platform based on model-driven approach. In Proceedings of the IEEE 9th International Conference on Grid and Cloud Computing, Nanjing, China, 1–5 November 2010; pp. 276–281. [CrossRef]
8. Mcsi, M. Configurability in saas (software as a service) applications. In Proceedings of the 2nd Annual India Software Engineering Conference, Pune, India, 23–26 February 2009; pp. 19–26. [CrossRef]
9. Fowler, M.; Lewis, J. Microservices—A Definition of This New Architectural Term. 2014. Available online: https://martinfowler.com/articles/microservices.html#:~:text=a%20definition%20of%20this%20new,suites%20of%20independently%20deployable%20services (accessed on 11 June 2022).
10. De Lauretis, L. From Monolithic Architecture to Microservices Architecture. In Proceedings of the 2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Berlin, Germany, 27–30 October 2019; pp. 93–96. [CrossRef]
11. Fan, C.; Ma, S. Migrating Monolithic Mobile Application to Microservice Architecture: An Experiment Report. In Proceedings of the 2017 IEEE International Conference on AI & Mobile Services (AIMS), Honolulu, HI, USA, 25–30 June 2017; pp. 109–112. [CrossRef]
12. Yang, C.; Lan, S.; Wang, L.; Shen, W.; Huang, G.G.Q. Big Data Driven Edge-Cloud Collaboration Architecture for Cloud Manufacturing: A Software Defined Perspective. *IEEE Access* **2020**, *8*, 45938–45950. [CrossRef]
13. Yang, C.; Shen, W.; Lin, T.; Wang, X. IoT-enabled dynamic service selection across multiple manufacturing clouds. *Manuf. Lett.* **2016**, *7*, 22–25. [CrossRef]
14. Lee, E. Cyber physical systems: Design challenges. In Proceedings of the 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing, Orlando, FL, USA, 5–7 May 2008; pp. 363–369. [CrossRef]
15. Dan, M.; Seidmann, A. The pricing strategy analysis for the "software-as-a-service" business model. In *International Workshop on Grid Economics and Business Models*; Springer: Berlin/Heidelberg, Germany, 2008. [CrossRef]
16. Thramboulidis, K.; Vachtsevanou, D.C.; Solanos, A. Cyber-physical microservices: An IoT-based framework for manufacturing systems. In Proceedings of the 2018 IEEE Industrial Cyber-Physical Systems (ICPS), Saint Petersburg, Russia, 15–18 May 2018. [CrossRef]
17. Peter, A. Reference Architecture Model Industrie 4.0 (rami4. 0). In *ZVEI and VDI, Status Report*. 2015. Available online: https://ec.europa.eu/futurium/en/system/files/ged/a2-schweichhart-reference_architectural_model_industrie_4.0_rami_4.0.pdf (accessed on 11 June 2022).
18. Tsai, W.; Zhong, P. Multi-tenancy and Sub-tenancy Architecture in Software-as-a-Service (SaaS). In Proceedings of the 2014 IEEE 8th International Symposium on Service Oriented System Engineering, Oxford, UK, 7–11 April 2014; pp. 128–139. [CrossRef]

19. Zuo, Q.; Xie, M.; Tsai, W. Autonomous Decentralized Tenant Access Control Model for Sub-tenancy Architecture in Software-as-a-Service (SaaS). In Proceedings of the 2015 IEEE Twelfth International Symposium on Autonomous Decentralized Systems, Taichung, Taiwan, 25–27 March 2015; pp. 211–216. [CrossRef]
20. Tsai, W.; Bai, X.; Huang, Y. Software-as-a-service (saas): Perspectives and challenges. *Sci. China Inf. Sci.* **2014**, *57*, 1–15. [CrossRef]
21. Chong, F.; Carraro, G. Architecture Strategies for Catching the Long Tail. pp. 1–20. Available online: https://wenku.baidu.com/view/66361116a4e9856a561252d380eb6294dd88226b.html?re=view (accessed on 11 June 2022).
22. Walraven, S. Middleware and Methods for Customizable SaaS. June 2014. Available online: https://lirias.kuleuven.be/retrieve/274549 (accessed on 11 June 2022).
23. Maria, A.V.; Vazquez, J.A. Business and technical requirements of Software-as-a-Service: Implications in portuguese enterprise business context. *arXiv* **2013**, arXiv:1312.2243.
24. Ali, A.Q.; Sultan, A.B.M.; Ghani, A.A.A.; Zulzalil, H. Customization of software as a service application: Problems and objectives. *J. Comput. Sci. Comput. Math.* **2018**, *8*, 27–32.
25. Tsai, W.-T.; Zhong, P.; Chen, Y. Tenant-centric sub-tenancy architecture in software-as-a-service. *CAAI Trans. Intell. Technol.* **2016**, *1*, 150–161. [CrossRef]
26. Al-Shardan, M.M.; Ziani, D. Configuration as a service in multi-tenant enterprise resource planning system. *Lect. Notes Softw. Eng.* **2015**, *3*, 95–100. [CrossRef]
27. Tsai, W.-T.; Shao, Q.; Li, W. OIC: Ontology-based intelligent customization framework for SaaS. In Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications (SOCA), Perth, WA, Australia, 13–15 December 2010; pp. 1–8. [CrossRef]
28. Mietzner, R.; Leymann, F. Generation of bpel customization processes for saas applications from variability descriptors. *IEEE Int. Conf. Serv. Comput.* **2008**, *2*, 359–366. [CrossRef]
29. Lee, W.; Choi, M. A multi-tenant web application framework for saas. In Proceedings of the IEEE 5th IEEE Fifth International Conference on Cloud Computing, Honolulu, HI, USA, 24–29 June 2012; pp. 970–971. [CrossRef]
30. Kang, S.; Kang, S.; Hur, S. A design of the conceptual architecture for a multitenant saas application platform. In Proceedings of the 1st IEEE ACIS/JNU International Conference on Computers, Networks, Systems and Industrial Engineering, Jeju, Korea, 23–25 May 2011; pp. 462–467. [CrossRef]
31. Bezemer, C.-P.; Zaidman, A. Multi-tenant saas applications: Maintenance dream or nightmare? In Proceedings of the ACM Joint ERCIM Workshop Software Evolution (EVOL) and International Workshop on Principles of Software Evolution, Antwerp, Belgium, 20–21 September 2010; pp. 88–92. [CrossRef]
32. Laatikainen, G.; Ojala, A. Saas architecture and pricing models. In Proceedings of the IEEE International Conference on Services Computing, Anchorage, AK, USA, 27 June–2 July 2014; pp. 597–604. [CrossRef]
33. Kabbedijk, J.; Jansen, S. Variability in multi-tenant environments: Architectural design patterns from industry. In Proceedings of the International Conference on Conceptual Modeling, Brussels, Belgium, 31 October–3 November 2011; pp. 151–160. [CrossRef]
34. AbdelHafeez, M.; Ahmed, A.H.; AbdelRaheem, M. Design and Operation of a Lightweight Educational Testbed for Internet-of-Things Applications. *IEEE Internet Things J.* **2020**, *7*, 11446–11459. [CrossRef]
35. Yang, C.; Shen, W.; Wang, X. The Internet of Things in manufacturing: Key issues and potential applications. *IEEE Syst. Man Cybern. Mag.* **2018**, *4*, 6–15. [CrossRef]
36. Cheng, B.; Zhang, J.; Hancke, G.P.; Karnouskos, S.; Colombo, A.W. Industrial Cyberphysical Systems: Realizing Cloud-Based Big Data Infrastructures. *IEEE Ind. Electron. Mag.* **2018**, *12*, 25–35. [CrossRef]
37. El-Sayed, H.; Sankar, S.; Prasad, M.; Puthal, D.; Gupta, A.; Mohanty, M.; Lin, C. Edge of things: The big picture on the integration of edge IoT and the cloud in a distributed computing environment. *IEEE Access* **2018,** *6*, 1706–1717. [CrossRef]
38. Di Francesco, P.; Lago, P.; Malavolta, I. Architecting with Microservices: A Systematic Mapping Study. *J. Syst. Softw* **2019**, *150*, 77–97. [CrossRef]
39. O'Connor, R.V.; Elger, P.; Clarke, P.M. Continuous Software Engineering—A Microservices Architecture Perspective. *J. Softw. Evol. Process* **2017**, *29*, e1866. [CrossRef]
40. Zimmermann, O. Microservices Tenets. *Comput. Sci.* **2017**, *32*, 301–310. [CrossRef]
41. Shadija, D.; Rezai, M.; Hill, R. Towards an Understanding of Microservices. In Proceedings of the ICAC 2017—2017 23rd IEEE International Conference on Automation and Computing: Addressing Global Challenges through Automation and Computing, Huddersfield, UK, 7–8 September 2017. [CrossRef]
42. Amshidi, P.; Pahl, C.; Mendonca, N.C.; Lewis, J.; Tilkov, S. Microservices: The Journey so Far and Challenges Ahead. *IEEE Softw.* **2018**, *35*, 24–35. [CrossRef]
43. Tekinerdogan, B.; Aksit, M. Classifying and evaluating architecture design methods. In *Software Architectures and Component Technology*; Springer: Boston, MA, USA, 2002; pp. 3–27. [CrossRef]
44. Pahl, C.; Brogi, A.; Soldani, J.; Jamshidi, P. Cloud Container Technologies: A State-of-the-Art Review. *IEEE Trans. Cloud Comput.* **2017**, *7*, 677–692. [CrossRef]