# Architecture of intelligent system for webservices scaling

Andrey Kupin[1], Dmytro Zubov[2], Maxim Kosei[1] and Vladyslav Holiver[1]

[1] *Kryvyi Rih National University, Vitaly Matusevich 11, Kryvyi Rih, 50027, Ukraine*
[2] *University of Central Asia, 125/1 Toktogul Street, Bishkek, 720001, Kyrgyzstan*

## Abstract

The key aspects of web services development, administration structures, and the use of machine learning technologies for server optimization are explored. The tendencies of web services development, scaling options, importance and basic concepts of microservice architecture are considered.

The article highlights the general principles of artificial intelligence, machine learning, and deep learning and their impact on the functionality of web services. To enhance the operation of web services, an architecture of an intelligent system for automatic scaling is presented and machine learning algorithms with increased reliability are elaborated. The article optimizes the performance of such a system. Methods for detecting abnormal system behavior are proposed, which allows preventing failures or a decrease in overall performance.

## Keywords

Microservices architecture, scaling, artificial intelligence, machine learning, deep learning, pattern, API, DevOps, CI/CD, PBW, PAD, Docker, One-Class SVM.1

## 1. Introduction

The research deals with the issue of developing effective models, methods, and algorithms for scaling web services in modern information systems based on machine learning to ensure stable operation of servers when the load changes.

A detailed analysis of the relevance of the problem, the task statement, and the main research directions were defined by the authors in their previous work [1]. In particular, this article presents the necessary architectural and algorithmic solutions.

The development trends of web services at the present stage have been significantly influenced by the following events:

- emergence of cloud platforms (2010s): In the 2010s, cloud-based platforms for developing web services, such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform, were launched. These platforms provide infrastructure and tools for deploying, managing, and scaling web applications. Web analytics and performance optimization services, cloud storage, mobile applications, etc. have also appeared;

- spread of microservice architecture (since the 2010s): One of the current trends in web development is the use of microservice architecture for web services. Instead of creating monolithic applications, developers break down functionality into small, independent components that can be deployed, scaled, and managed separately. This allows for greater flexibility, faster development and deployment, and easier integration with other services;

- expansion of the capabilities of artificial intelligence and the Internet of Things (since the 2010s): Recently, web services have started to use artificial intelligence to automate routine tasks, analyze data, and improve user experience. Web services are also being developed to connect to the Internet of Things, allowing physical devices to be controlled over the network.

Preliminary analysis [2-8, 15-17] shows that there is a lack of research in this area. This is especially true when it comes to identifying effective models, methods, techniques, and algorithmic hardware and software for reliable management of servers on the global Internet. With this in mind, the purpose of this article is to justify the choice of a rational architecture and develop

*ICST-2024: Information Control Systems & Technologies, September 23-25, 2023, Odesa, Ukraine.*
kupin@knu.edu.ua (A. Kupin); dzubov@ieee.org (D. Zubov); kosei@knu.edu.ua (M. Kosei); holivervlad@gmail.com (V. Holiver)
0000-0001-7569-1721 (A. Kupin); 0000-0002-5601-7827 (D. Zubov); 0009-0008-3445-5675 (M. Kosei); 0000-0002-8276-5992 (V. Holiver)

algorithms for an intelligent web service scaling system based on Microservices Architecture (MSA).

The observed literature thoroughly discusses the architectural patterns in MSA and common machine learning models. However, there is a significant gap in research regarding the application of machine learning techniques specifically for scaling Docker-based microservices within MSA. This article aims to address this gap by developing and justifying an intelligent system architecture and algorithms focused on optimizing the performance and reliability of such systems.

Traditional scaling methods for web services typically rely on threshold-based policy rules. While effective, these methods have limitations. Incorporating machine learning into the scaling process offers significant benefits, including improved adaptability, more efficient resource management, and better performance prediction.

## 2. Types Of Scaling

There are two main types of scaling that are used to provide growth in resources and system performance (Figure 1):

- Vertical Scaling: This type of scaling involves increasing the capacity of hardware such as processors, memory, and disk. With vertical scaling, one single server can handle more tasks or process more data. For example, increasing the amount of RAM or upgrading to a more powerful processor.

- Horizontal Scaling: This type of scaling is adding other servers or nodes to the system. It spreads the load across many physical or virtual servers to provide more processing power and availability. Horizontal scaling is often used in cloud environments and distributed computing systems.

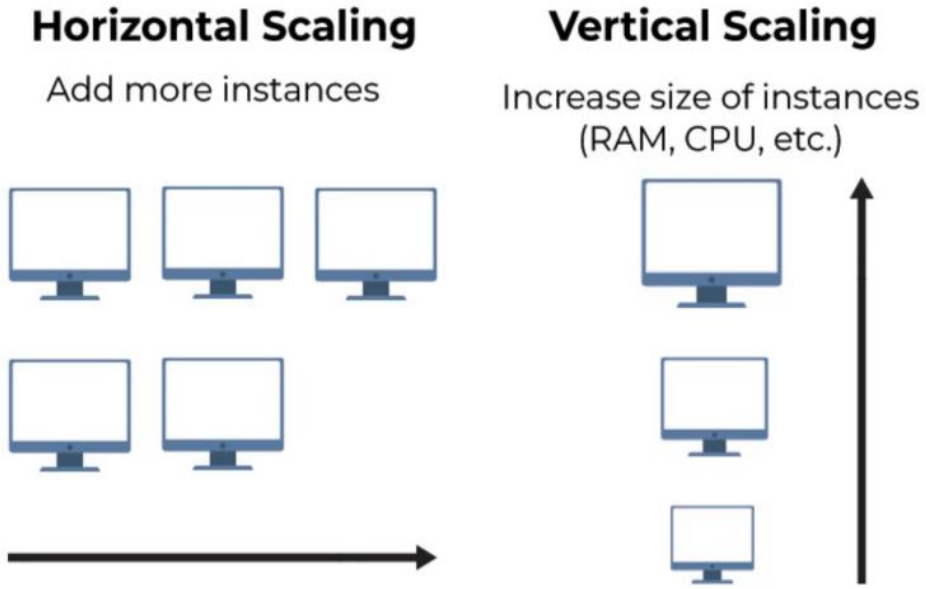


Figure 1: The main types of web service scaling

## 3. Web Service Architecture

Web service architecture can be organized using two main approaches: monolithic architecture and microservice architecture (Figure 2). Monolithic architecture is a traditional approach to web application development in which all application features are located in a single software module, usually a monolithic application or a monolithic server. In a monolithic architecture, all code, database, and logic are located in a single application, which facilitates development and deployment.

The advantages of a monolithic architecture include ease of development and testing, no problems with interactions between components, and reduced infrastructure costs. However, monolithic applications can become difficult to scale and develop in large projects, and they can be less flexible in introducing new features.

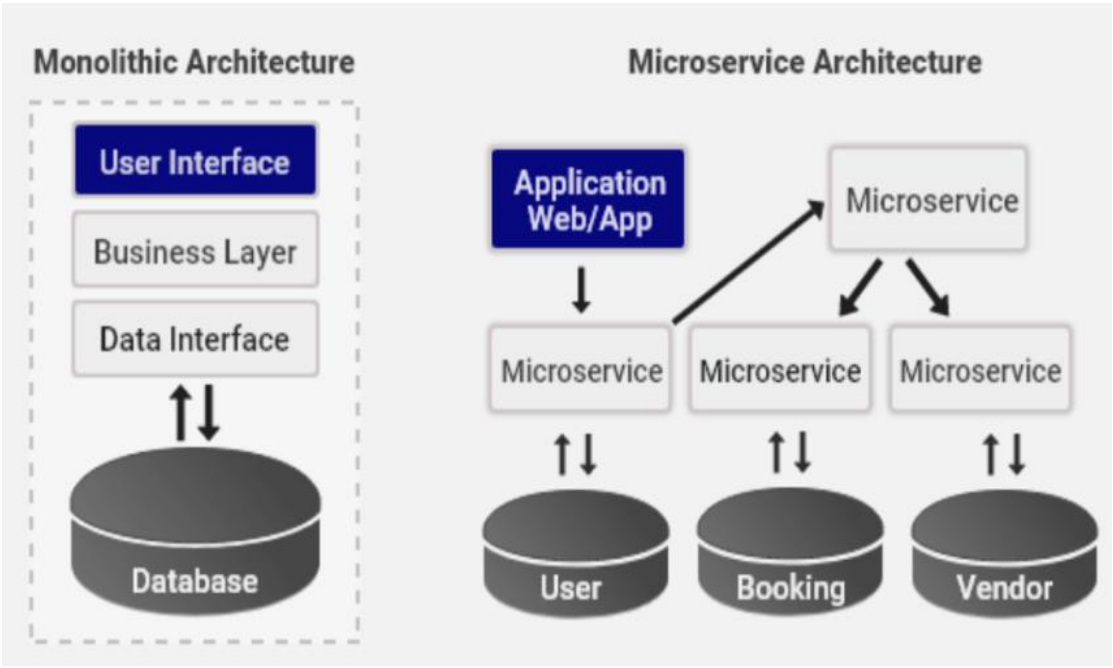

Figure 2: Monolithic and Microservices Architecture [9]

Microservice architecture (Figure 3) is an approach where a large web application is broken down into small, independent services that work together using lightweight communication mechanisms such as APIs. Each service is responsible for limited functionality and has its own database.

Microservice architecture provides greater modularity, scalability, and flexibility in web application development. Each service can be independently developed, scaled, and maintained. In addition, microservices can use different technologies and programming languages, which gives developers more freedom to choose technologies.

However, the microservice architecture also has its challenges, including the complexity of interactions between services, configuration, and monitoring management, and greater complexity in implementing and managing multiple services.

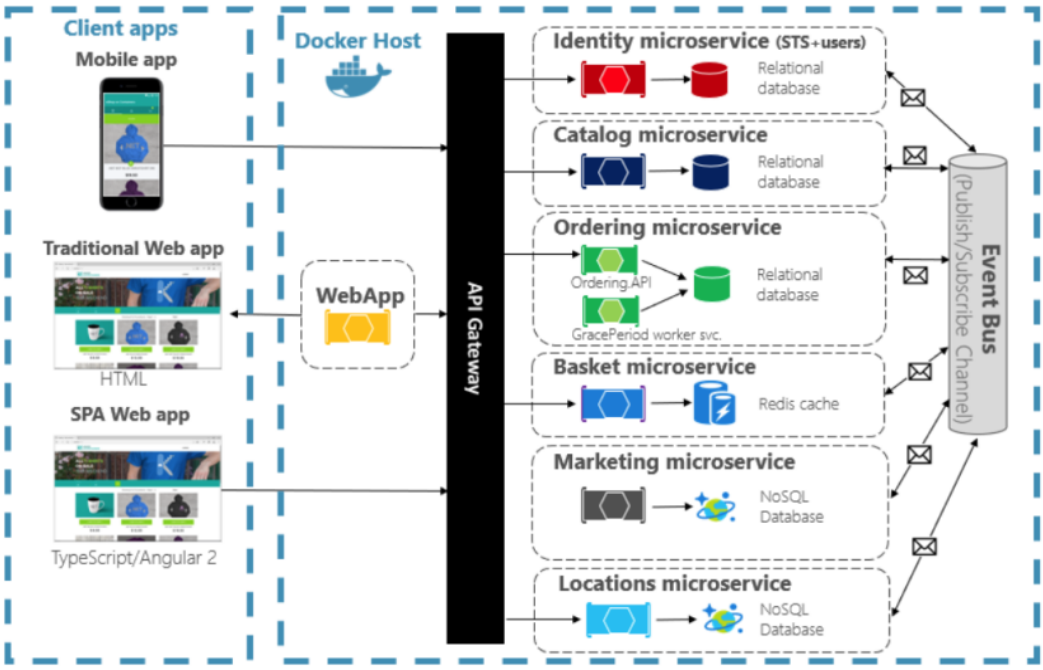

Figure 3: Example of Microservices Architecture

The advantages and disadvantages of MSA compared to monolithic architecture are discussed in more detail in Table 1.

Table 1
Comparison of MSA with monolithic architecture

| Characteristic | MSA - architecture | Monolithic architecture |
|---|---|---|
| Structure | The high degree of autonomy. The system functions are divided into independent, slightly connected parts with a smaller code volume. | Lack of autonomy. System functions are tightly coupled in one large block of code. |
| Portability | Very high. | Very limited portability. |
| Reusability | Highly reusable. | Very limited code reusability. |
| Modularity and scalability | Highly modular and scalable. | Limited modularity and difficult to scale. |
| Time to market | The start time to market depends on the readiness of individual services. The more code is reused, the shorter the time. If the system's microservices are developed from scratch, the time is usually longer than for a monolithic architecture. | Long time to market, especially in large systems. Shorter time to market in small and simple systems. |
| Release cycle and updates | Very short release cycle, rapid implementation of changes and updates. | The long and typically very laborious release cycle for new versions and updates. |
| Initial costs | Usually high. It depends on the size of the system. Initial costs are offset by operational cost savings. | Typically low. They become larger in large corporate systems. |
| Operational costs | Low. Easier to maintain and operate. | High. Difficult to maintain and operate. |
| Complexity | High | Low |
| API control | High | Low |
| Structural data integrity | Decentralized databases, so maintaining data integrity is more challenging. | Centralized database, making it easier to maintain data integrity throughout the system. |
| Performance | Typically lower. | Typically higher. |
| Security | More security issues | Less security issues. |
| Implementation in software development organization | Hard to implement depending on the organizational structure. Requires adoption of flexible development and DevOps (CI/CD, etc.). Organizational transformation may be needed, which can take a long time to achieve. | Easy to implement. Minimal organizational transformation is required, if any at all. |
| Fault tolerance | Typically higher. | Typically lower. |

Microservice architecture (Figure 3) is better designed for scaling than monolithic architecture for the following reasons:

- Decentralization. Microservices are distributed across multiple servers, which makes them more scalable than monolithic applications that run on a single server. This means that you can easily add or remove servers as needed to maintain the desired performance.
- Isolation. Each microservice is isolated from the others, which means that you don't need to scale the entire application if there is a significant load on just one microservice. This also means that you can scale microservices independently of each other, which can be useful for cost optimization.
- Layer architecture. Microservices are often built using a layered architecture, which allows you to scale the application using different technologies for each layer. For example, the storage tier can be scaled horizontally and the processing tier can be scaled vertically.

## 4. Microservice Architecture (MSA)

MSA is a technique for creating a complex system from a set of smaller applications, each of which is designed to perform a specific limited function.

These minor applications (or services, or microservices) are developed independently of each other and can function independently of each other. Each microservice has an API interface to communicate with other microservices in the system.

The way these individual microservices are organized together determines the functionality of the larger system.

To comprehend the value of microservices and the challenges that come with developing an MSA, it is important to understand how microservices interact and communicate with each other.

This interaction can be linear or non-linear.

In a linear interaction (Figure 4), microservices transfer data to each other sequentially, processing it in the system. Input data is always transferred to the first microservice, and output data is always generated by the last microservice in the system.

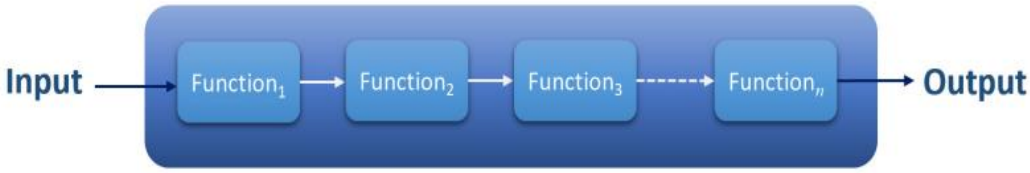


Figure 4: Linear interaction of microservices

In almost most existing systems, the interaction is non-linear (Figure 5).

In a nonlinear microservice interaction, data is distributed among different functions in the system. Input data can be passed to any function in the system, and output data can be generated by any function in the system.

Let's consider nonlinear interaction using a practical example in a typical e-commerce system (Figure 6).

Each function in the "Placing an Order" process is a microservice. After a customer places an order, an API call to the "Add/Update Customer Information" microservice is triggered to save or update customer information. This microservice is solely responsible for managing customer information based on the data it receives from the API call.

Also, another API call to the "Verify Payment" part of the process is executed at the same time.

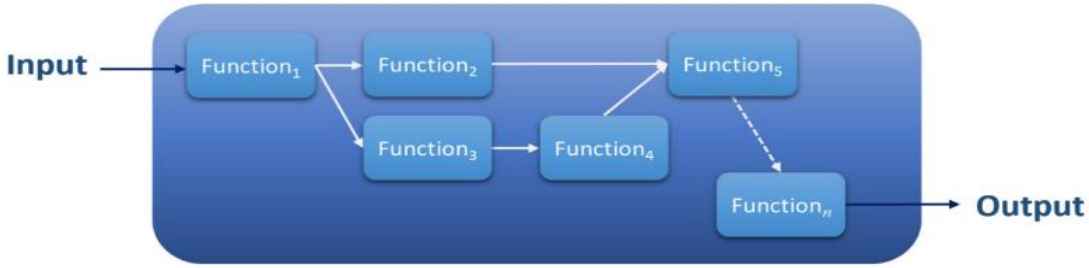


Figure 5: Nonlinear interaction of microservices

This call will be directed to the "Process PayPal Payment" or "Process Credit Card Payment" microservice, depending on the type of payment specified in the API call. It's worth noting here how the payment verification process is split into two different microservices, each of which is specifically designed for a specific payment function.

This provides flexibility and portability of these microservices to other parts of the system or another system if necessary.

After the payment is processed, other microservices in the system receive API calls to fulfill the order.

This example shows how modular and flexible the MSA system design could be.

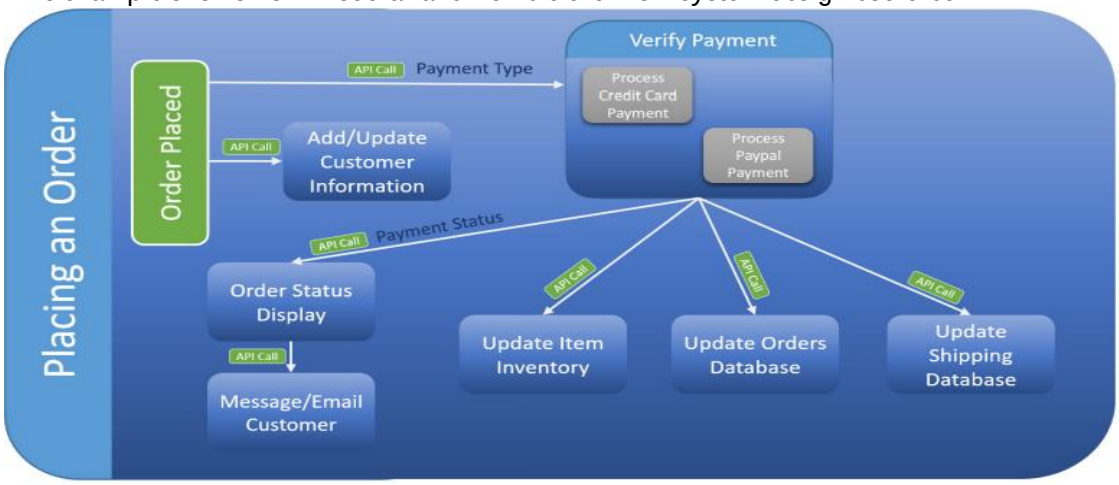


Figure 6: An example of a nonlinear interaction of microservices

## 5. Artificial Intelligence (AI), Machine Learning (ML) And Deep Learning (DL)

Despite the recent rise in popularity of Artificial Intelligence (AI) and Machine Learning (ML), the field of artificial intelligence has existed since the 60s of the XX century. With the emergence of various AI subfields, it is important to be able to distinguish them from each other and understand what they mean and include.

First, AI is a general field that encompasses all the subfields we see today, such as ML, Deep Learning (DL) (Figure 7), and others.

Any system that perceives or receives information from the environment and performs actions to maximize rewards or achieve its goal is considered an AI system.

This is very common in robotics today. Most of our machines are designed so that they can collect data through their sensors, such as cameras, sonars, or gyroscopes, and use the collected data to perform a particular task efficiently.

This concept is very similar to how humans behave. Humans use their senses to gather information from the environment and, based on the information they receive, perform certain actions.

AI is a vast field, but it can be broken down into different subfields, one of which we know today as ML. What makes ML unique is that this field works to create systems or machines that can learn and improve their models without explicit programming.

ML does this by collecting data, known as training data, and trying to find patterns and regularities in that data to make accurate predictions without being explicitly programmed to do so. ML uses different methods to learn from data, and these methods are chosen depending on the problems to be dealt with.

The approaches used in ML are traditionally divided into three broad categories:

1. Supervised Learning (SL);
2. Unsupervised Learning (UL);
3. Reinforcement Learning (RL).

SL helps to understand the relationship between input and output data. One typical example of SL is predicting the price of a house in a certain city.

Data is collected on existing houses, namely their characteristics and current prices (training set), and then the patterns between the characteristics of these houses and their prices are studied.

After that, you can take a house that is not part of the training set and use the model you built to predict its price based on its characteristics.

UL involves learning the structure of data using grouping or clustering methods. This method is often used for marketing purposes.

For example, a store wants to divide its customers into different groups to effectively tailor its products to different demographics.

It can obtain the purchase history of its customers; study this data to determine purchase patterns, and recommend certain products or services that might be of interest to them, thereby maximizing its profits.

Before looking at DL, which is a subfield of ML, it is important to understand what Artificial Neural Networks (ANN) is.

Taking the neurons in the brain as an example, ANNs are models that consist of a network of interconnected nodes, also known as artificial neurons. They contain a set of inputs (Input), hidden layers (Hidden Layer) connecting the neurons, and an output node (Output) (Figure 8). [10]

Each neuron has an input and an output that can be transmitted throughout the network. To calculate the neuron's output, the weighted sum of all inputs is taken, multiplied by the neuron's weight, and usually a shift parameter is added.

This process continues until the last layer is reached, which is the output neuron. A nonlinear activation function, such as a sigmoid function, is applied to obtain the final prediction.

The resulting predicted value is input into the cost function. This function shows how well our network is learning.

The value of the cost function is used to backpropagate errors through all layers back to the first layer by adjusting the weights of the neurons. This allows us to create powerful models that can perform tasks such as handwriting recognition, gaming AI, etc.

In some cases, ANNs can be very powerful, but there are serious drawbacks that limit their application:

- Black Box: ANNs can be hard to interpret, making it complicated to understand how they work and why they make certain predictions. This can make it difficult to debug ANNs and trust their results.

- Computational cost: Training an ANN can be computationally expensive, especially for large and complex networks. It may require specialized hardware such as GPUs and can take a long time to train.

- Overfitting: ANNs are prone to overlearning, which means they can learn the training data too well and fail to generalize to new data. This can lead to poor performance on real-world examples.

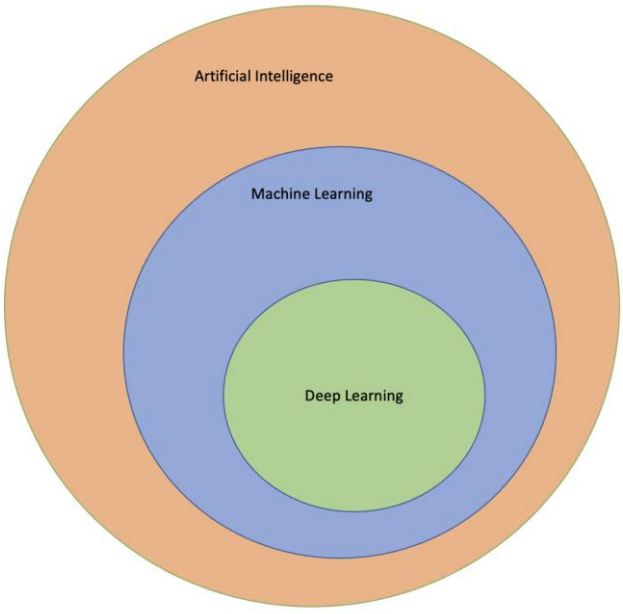


Figure 7: Relationship between AI, ML, DL

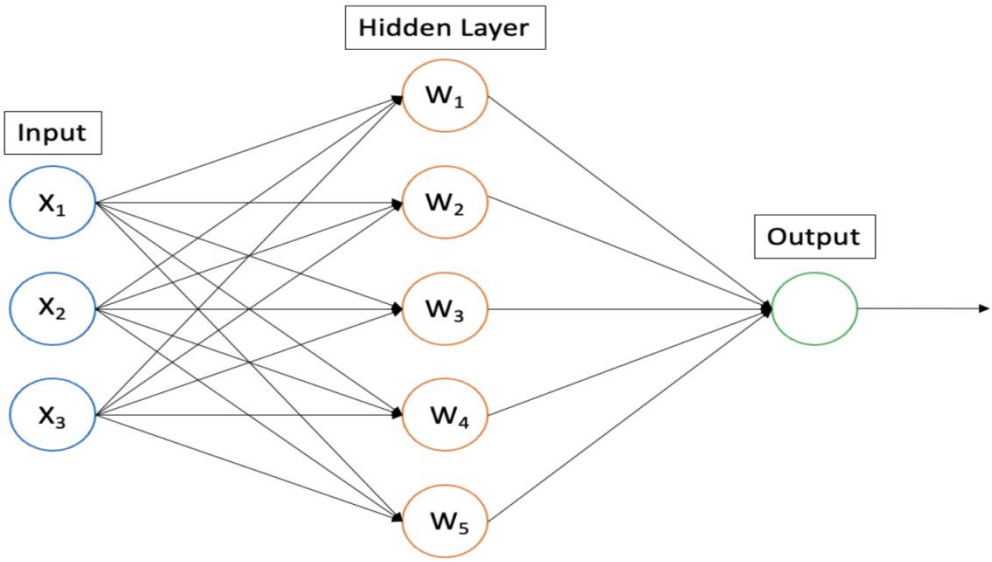


Figure 8: Artificial Neural Networks - ANN

This is where DL comes into play.

DL can be categorized according to the following key features:

- Hierarchical composition of layers: Instead of having only fully connected layers in the network, we can create and combine several different layers consisting of nonlinear and linear transformations. These different layers play a role in extracting key features in the data that would otherwise be difficult to find in an ANN.

- End-to-end training: The network starts with a method called feature extraction. It analyzes the data and finds a way to group redundant information and identify important features of the data. The network then uses these features to learn and make predictions or classifications using fully connected layers.

Distributed representation of neurons: With feature extraction, the network can group neurons to encode a larger feature of the data. Unlike ANNs, no single neuron encodes everything. This allows the model to reduce the number of parameters it has to learn from while retaining key elements in the data.

DL is widely used in computer vision. Due to advances in photo and video capture technology, it has become very difficult for ANNs to learn and recognize images with high accuracy. The reason is that when using an image to train a model, you need to consider each pixel as an input parameter of the model. For example, a 256x256 image has more than 65,000 input parameters. Depending on the number of neurons in a fully connected layer, the number of parameters can reach millions. With such a large number of parameters, there is a chance of overfitting and training can take a very long time. With DL, you can create a group of layers called Convolutional Neural Networks (CNNs). These layers are responsible for reducing the number of parameters that the model needs to learn while preserving the key features of our data. With these additional elements, we can learn how to extract certain features and use them to train our model with high efficiency and accuracy (Figure 9).

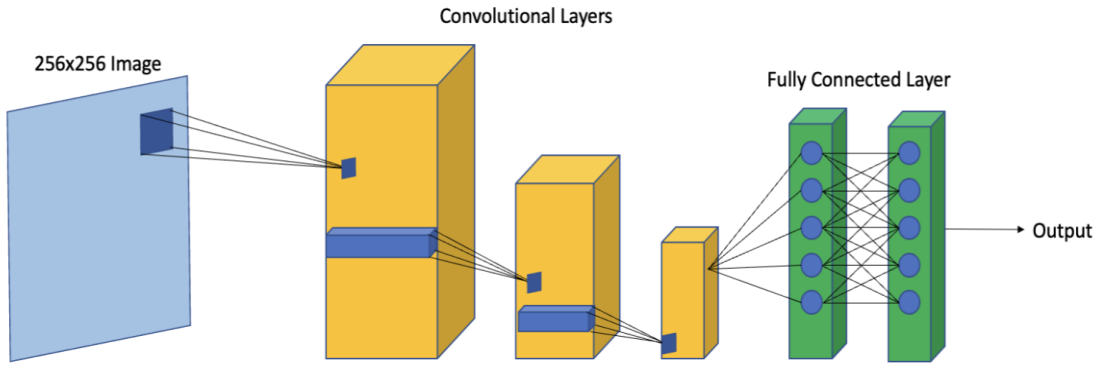


Figure 9: Convolutional Neural Networks – CNN

# 6. Algorithms of AI Web Service Scaling System Based on MSA

Commonly used for web service autoscaling is the Threshold Rules Policy, which consists in setting certain thresholds or limits that, when exceeded or reached, cause resources to automatically scale to ensure optimal system performance and reliability.

The use of ML techniques can greatly improve web service autoscaling strategies, especially for large and complex systems, as such systems often have a large number of parameters and loads that change in a very dynamic way.

In such systems, patterns emerge that are the result of recurrence or similarity in the data, interactions between system components, or the way the system processes the data.

These patterns can be detected using various machine learning algorithms, which can significantly improve the efficiency and scalability of the system, as well as ensure that the system as a whole performs more optimally.

As mentioned, there are many areas in the MSA system where artificial intelligence can be used.

The focus will be on two main potential areas of improvement (Figure 10), which are implemented by individual additional AI services. The first is to increase the system's response speed in the event of microservice failure or performance degradation. The second area of improvement is the introduction of the proactive role of the Circuit Breaker.

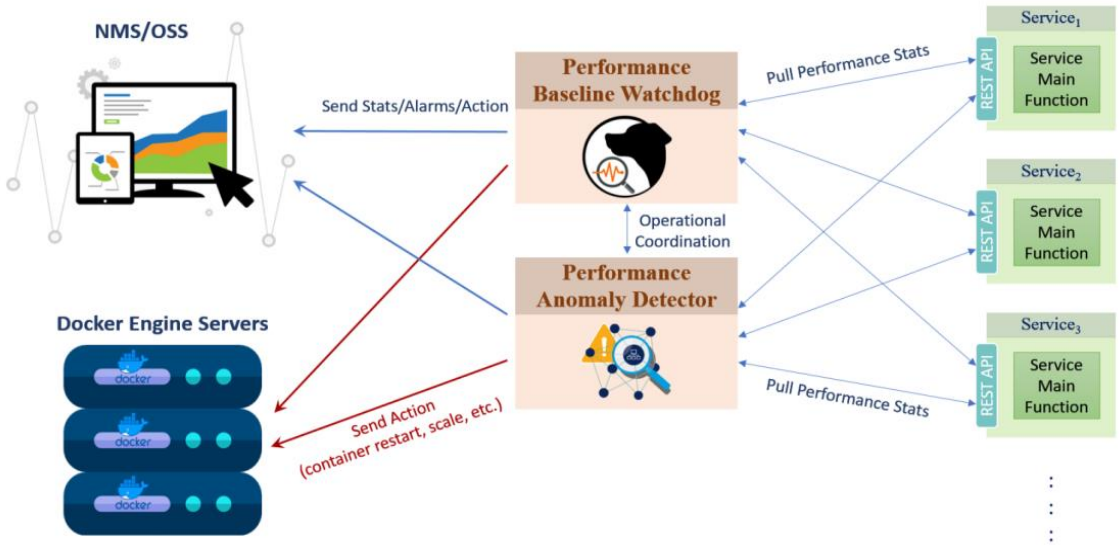


Figure 10: PBW and PAD - Artificial Intelligence Microservices for Enhancing Reliability and Manageability of MSA System

The first AI microservice is called Performance Baseline Watchdog (PBW). PBW is an ML microservice that determines whether the performance of each microservice in the system meets expectations. If the performance of a microservice falls below the expected level by a certain amount, PBW sends an alert to operations support or network management systems. If performance falls even further, PBW sends an alert to the Operation Support System (OSS) or Network Management System (NMS) and can take action to automatically correct the problem.

The second artificial intelligence microservice is the Performance Anomaly Detector (PAD). PAD is a machine learning service that covers the entire MSA system. It analyzes MSA performance patterns and tries to detect any unusual behavior. PAD finds problematic patterns in the behavior of microservices, automatically detects problems before they occur, and proactively acts to resolve them.

The PBW algorithm calculates the expected performance based on the collected performance statistics. The collected performance statistics include API response time statistics, errors or error rates of individual microservices, API response codes, and the load applied to the microservice itself. Predefined actions are triggered depending on how much the microservice deviates from the calculated performance indicator. Based on the PBW configuration, the larger the deviation, the more likely it is that a proactive action will be initiated to try to self-heal. However, in the case of a minor deviation, no self-healing action should be triggered - a system warning informing the system administrator is sufficient.

Table 2
Challenges of the MSA System and PBW.

| Problem | Action triggered by PBW |
|---|---|
| Slow response or timeouts | Scaling microservice vertically or horizontally or restarting microservice |
| HTTP response errors | Checking the status of Apache, Flask, JVM, Docker volumes, SQL service, etc. Restarting the service if necessary. |
| Microservice does not respond (turned off). | Restarting the microservice container. |

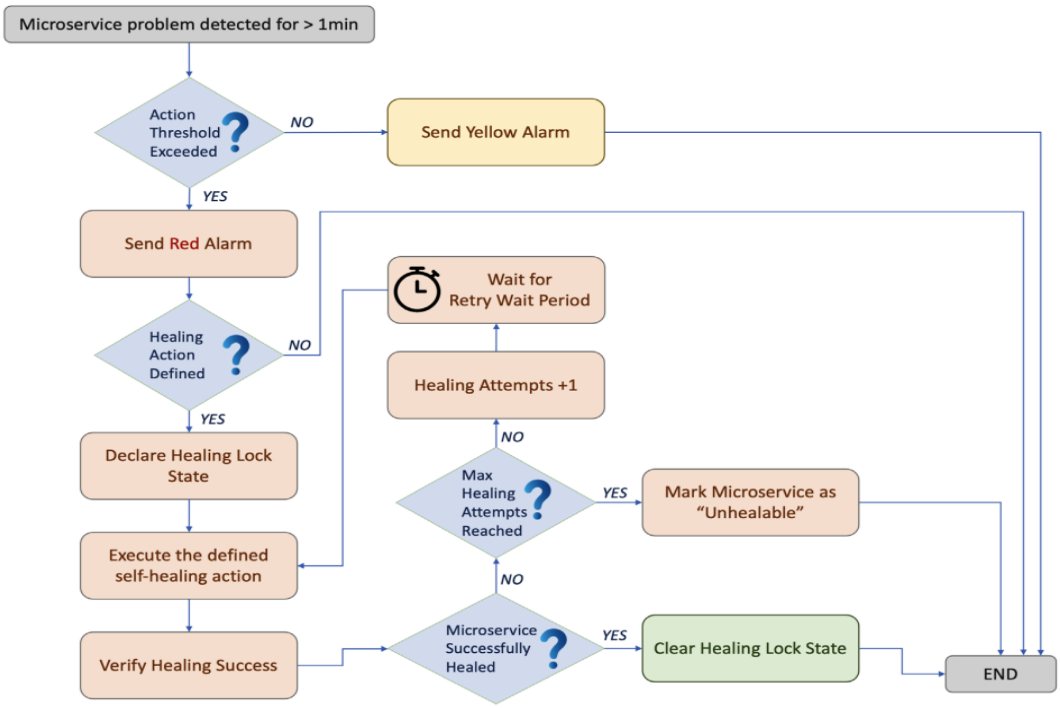


Figure 11: Self-healing Microservices Algorithm

Table 2 shows some of the possible system problems [11-12] that can be encountered during system operation and the actions that the PBW service will take to try to fix the problem, and Figure 11 and Table 3 show the microservice self-healing algorithm.

Table 3
Explanation of Terms for Figure 11.

| Term | Description |
|---|---|
| Healing Action | Action taken to fix a failure. |
| Healing Lock State | State of the microservice where only the self-healing algorithm can interact with the problematic microservice. |
| Retry Wait Period | The time to wait when a treatment fails before retrying. The default timeout period before retrying is 2 minutes. |
| Unhealable State | The state in which the microservice is marked as unhealable after its failed attempt to heal itself. |
| Maximum Healing Attempts | The maximum number of attempts made to rectify the microservice before marking it as unhealable. |

PBW uses a linear regression model for training and prediction, while PAD uses a One-Class Support Vector Machine (One-Class SVM [13-14]).

Compared to traditional support vector machines, which are used for classification tasks where the data is labeled, One-Class SVM is designed for situations where only one class of data is available (unlabeled data). Its main goal is to identify and classify normal data points from outliers or anomalies [15-17].

## Conclusion

The paper researches the main aspects of web services development, their structure, and the impact of ML on this field. In particular, the paper considers web services development trends, scaling options, importance, and basic concepts of MSA architecture.

The general principles of artificial intelligence, machine learning, and deep learning and their impact on the functionality of web services are also covered. This helps to understand what technological innovations are used to improve the performance of web services and how machine learning changes their capabilities.

The described approach provides a general idea of the basic principles and trends of web services development and the impact of machine learning on this industry. This is an important basis for further research and implementation of innovations in the field of web services and their connection with machine learning.

Implementation of ML methods in web service autoscaling can provide significant benefits and improve the efficiency of the MSA system.

ML is especially useful for large and complex systems, as it enables the detection of patterns in data and the interaction of system components.

The proposed PBW and PAD algorithms provide the following advantages:

1. Improved system reliability: These algorithms allow the system to respond to deviations in microservice performance and detect anomalies, even before they occur. This allows system operators to take action to fix problems faster and more efficiently, increasing overall system reliability.
2. Performance optimization: Rapid problem detection and automatic correction avoids loss of productivity. Timely response to abnormalities helps maintain system stability and optimal performance, which in turn improves productivity.
3. Preliminary detection of problems: PAD helps detect anomalous patterns or unusual behavior before they can cause serious problems. This allows the system to prevent failures or performance degradation, enabling operators to prepare for potential problems and prevent them from spreading.

## References


[1] S. Semerikov, D. Zubov, A. Kupin, M. Kosei, V. Holiver, Models and Technologies for Autoscaling Based on Machine Learning for Microservices Architecture (2024), in: CEUR Workshop Proceedings, 2024, 3664, pp. 316–330. URL: https://ceur-ws.org/Vol-3664/paper22.pdf

[2] P. Raj, A. Raman, H. Subramanian, Architectural Patterns. Packt Publishing, 2017.

[3] S.Newman, Building microservices: Designing fine-grained systems. Beijing i pozostałe: O'Reilly, 2021.

[4] M. Bruce, P. Pereira, Microservices in action. Shelter Island, NY: Manning Publications Co., 2019.

[5] A. Müller, S. Guido, Introduction to machine learning with python: A guide for data scientists. Sebastopol: O'Reilly Media, 2018.

[6] J. Mueller, Machine learning security principles: Use various methods to keep data, networks, users, and applications safe from Prying eyes. Birmingham: Packt Publishing, 2023.

[7] S. Raschka, Y. Liu, and V. Mirjalili. Machine learning with pytorchand Scikit-Learn: Develop machine learning and deep learning models with python. Birmingham: Packt Publishing, 2022.

[8] M. Abouahmed, and O. Ahmed. Machine learning in microservices: Productionizing Microservices Architecture for Machine Learning Solutions. Birmingham: Packet Publishing, 2023.
[9] Ubuntu server - for scale out workloads Ubuntu, 2023. URL: https://ubuntu.com/server/
[10] A. Kupin. Application of neurocontrol principles and classification optimisation in conditions of sophisticated technological processes of beneficiation complexes (2014), in: Metallurgical and Mining Industry, 2014, 6(6), pp. 16–24. ISSN: 20760507.
[11] J. Brains, PyCharm: The python IDE for professional developers by jetbrains, JetBrains, 2021. URL: https://www.jetbrains.com/pycharm/
[12] DBeaver Community, 2023. URL: https://dbeaver.io/
[13] MySQL, 2023. URL: https://www.mysql.com/
[14] Accelerated Container Application Development, 2023 Docker. URL: https://www.docker.com/
[15] A. Davis, Bootstrapping Microservices with Docker, Kubernetes, and Terraform: A project-based guide. Manning, 2021.
[16] S. Wells, Enabling Microservice Success: Managing Technical, Organizational, and Cultural Challenges. O'Reilly, 2024.
[17] C. Richardson, Microservices Patterns: With examples in Java. Manning, 2018.