MEKONG UNIVERSITY SCIENTIFIC JOURNAL

https://tapchikhdhcl.edu.vn/mkusj/ P-ISSN 3093-3218
https://vjol.info.vn/dhcl-e/index E-ISSN 3093-3358

PERFORMANCE EVALUATION OF MICROSERVICES-BASED IOT APPLICATIONS

Mai Nhu Uyen Le[1], Duc-Thien Phan[2], Le Dinh Phu Cuong[1*], Vo Quoc Tuan[3], Phan Ho Duy Phuong[4], Nguyen Hoang Tu[5], Quoc-Cuong Pham[6]

[1]Yersin University of Da Lat
[2]Nam Dinh University of Technology Education
[3]University of Phan Thiet
[4]Mekong University
[5]Hanoi University of Industy, Vietnam
[6]Ho Chi Minh City University of Industry and Trade, Vietnam

DOI: https://doi.org/10.65934/mkusj.2026.02.1143
[*]Email: cuongldp@yersin.edu.vn
Received: 13/04/2026 *Reviewed: 16/05/2026* *Accepted: 28/06/2026*

ABSTRACT

Microservices architecture has emerged as a promising approach for developing scalable, flexible, and maintainable Internet of Things (IoT) applications. However, evaluating the performance of microservices-based systems remains a critical challenge due to the distributed nature of their architecture and the increasing demands of modern software applications. This study investigates the performance characteristics of microservices-based IoT applications through benchmark testing under containerized deployment environments. The proposed architecture was evaluated using Docker-based benchmarking tools to assess key performance indicators, including memory throughput, storage read and write performance, load handling capability, and operational speed. The experimental results demonstrate the effectiveness of the microservices architecture in supporting distributed IoT applications while providing insights into system performance under different workload conditions. The study highlights the advantages of adopting microservices architecture for IoT software development, particularly in improving scalability, maintainability, and deployment flexibility. It also discusses practical challenges associated with implementing microservices-based systems and identifies future research directions for enhancing the performance and reliability of distributed IoT applications.

Keywords: *microservices architecture; Internet of Things (IoT); Docker; performance evaluation; benchmarking; distributed systems.*

Introduction

Introduction to Microservice Architecture

Microservices are a new trend rising fast from the microservices have been identified, it is difficult to have a clear view of existing research solutions for architecting microservices. The microservices architecture (Fig. 1) consists of a small collection of autonomous services. Each service is self-contained and needs to implement a single business function within a limited context. Boundary contexts are a natural division within an organization that provides an explicit boundary in which a domain model resides [1]. Microservices are small, independent, and loosely coupled. A single small team of developers can create and maintain services. Each service is a separate code base and can be maintained by a small development team [1]. Services can be provided independently. The service is responsible for maintaining its own data or external state. This is different from the traditional model where another data layer handles data persistence [1]. Services communicate with each other using a well-defined API. The details of the internal implementation of each service are hidden from other services. It supports multilingual programs. For example, services do not have to use the same technology stack, library or framework [1].

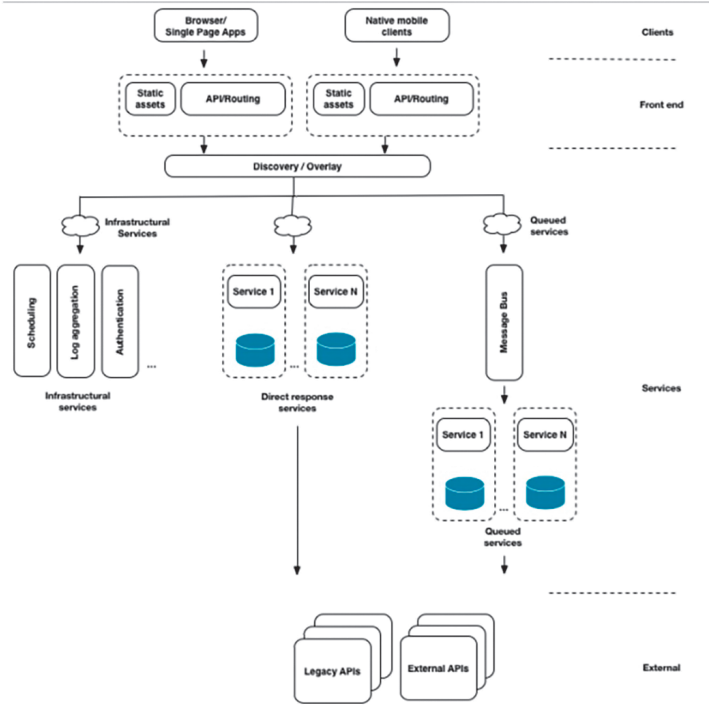

Figure 1. Microservice Architecture Reference Diagram

Although, the microservices architecture is an easier helping hand for software developers. This architecture develops a strategy in the process of software development. The architecture involves other components which are mentioned [1]:

Management/orchestration: This component is responsible for deploying services to nodes, identifying failures, rebalancing services between nodes and more. This component is usually an off-the-shelf technology like Kubernetes and is not specifically designed [1].

API gateway: The API gateway is the entry point for the client. The client calls the API gateway instead of calling the service directly. The API gateway forwards the call to the appropriate service on the backend [1].

Summary, the microservices architecture has several advantages applicable in the entire cycle of software development. Hence, it is essential to get over the background of Microservices Architecture.

Background

In recent years, the microservices development paradigm has been used to describe the evolutionary process for developing software applications as modular, independently deployable packages of services [2-3]. The main purpose of this paradigm is to divide business processes into small, independent units called services [4]. Each service has a container, a programming language, a process, data storage and a communication mechanism. This paradigm provides guidelines for software developers and an implementation effort by providing a lightweight, flexible and scalable process for building and running applications [4].

In addition, microservices architectures have many other advantages over monolithic software architectures [5-6-7]. In this way, enterprises can develop more adaptable and reusable software solutions quickly and efficiently. Microservices technology is becoming more popular and some companies are adopting microservices-based architectures such as Amazon, Netflix and Uber to improve business process performance and gain agility and stability [7]. In this regard, the benefits of microservices are also compatible with IoT for the project managers and decision-makers.

Some of the reasons that depict the benefits behind using Microservice Architecture:

Agility: Microservices are deployed independently, making it easier to manage bug fixes and feature releases. We can update the service without redeploying the entire application and roll back the update in case of problems. Many traditional applications can block the entire

release process if a bug is found in any part of the application. New features may be waiting for fixes to be integrated, tested, and released [1][8].

Focused and limited team: Microservices should be small enough to be built, tested, and deployed by a single functional team. The smaller the team size, the more agile we are. Large teams tend to be less productive due to slower communication, increased administrative overhead, and reduced agility [1][9]

Mini code base: In monolithic applications, code dependencies tend to get intertwined over time. To add new features, we need to touch the code in many places. The microservices architecture minimizes dependencies by not sharing code or data storage, making it easy to add new features [1][10].

A combination of technologies: Teams can choose the technology that best suits their service and combine the technology stacks as needed [1] [11].

Containment of errors: Even if a single microservice is not available, the entire application is disrupted as long as all upstream microservices are designed to handle the error correctly there is no [1].

Scalability: Services can be scaled individually, so you can scale subsystems that require more resources without scaling the entire application. Orchestrators such as Kubernetes and Service Fabric allow you to pack higher density services into a single host for more efficient use of resources [1].

Data separation: Updating the schema is much easier, as only one microservice is affected. In a monolithic application, different parts of the application can access the same data, making changes to the schema dangerous, which can make updating the schema very difficult [1].

Introduction to IoT

The development of Internet technology in recent years has led to the rise of the Internet of Things (IoT). The IoT is defined by [10] as a network of things that has clear element identification and is built into software intelligence, sensors, and ubiquitous connections to the Internet. IoT applications allow you to connect physical objects called things to a variety of smart devices to monitor, store, process, and analyze the data we collect [12]. Things are computing devices, from ordinary household items to sophisticated industrial tools. Currently, there are more than 8 billion connected IoT devices around the world and the number is expected to nearly triple by 2030 to more than 25 billion. Today, multiple domains and applications are using the IoT to offer better services, including B. Healthcare, Business Analysis, Automotive Industry, Smart City, Smart Agriculture, Energy Management, etc. [13-14]. The IoT plays an important role in these applications and areas by providing multiple solutions that can improve people's lives.

However, the benefits of IoT are limited by many serious security issues and privacy threats [15-16]. In fact, connected devices within large IoT infrastructures are exposed to numerous security attacks such as distributed denial of service (DDoS) attacks, malware and ransomware, botnets, and phishing attacks. Therefore, implementing security and protection measures for our business environment based on IoT systems should be a top priority. Several technical solutions have been proposed to address security issues in the IoT environment. Among these solutions we can mention the adoption of architectures based on microservices. Microservices technology is one of the leading technologies in computing that is designed to provide services in a decentralized and independent process. Microservices-based architecture enables different heterogeneous and distributed application entities to be independently developed, deployed, and scaled, making it a trend for IoT application development today. The integration of microservices technologies within the IoT architecture enables the improvement of various aspects [17-18], such as: i) continuous provisioning, testability and deployability, ii) business function reuse, iii) improving scalability and performance etc.

1. Literature Review

Microservices based applications can be designed to minimize their attack surface by only executing specific functions and executing them only when needed to keep less unused functions active. In addition, Microservices can provide a higher level of isolation for IoT applications.

In this section, we provide a summary of the authors' previous work, peer-reviewed literature on microservice architecture challenges, performance evaluation of microservice architectures, and intrusion detection tools. In any case, we relate the peer-reviewed literature to the contributions of this article.

Previous Work of The Authors

The current article is an extension of the previous work published [23-24] and also includes the content of the two-page material [25]. The paper presented a quantitative approach based on domain-based metrics to evaluate different microservices architecture deployment alternatives and compared them in bare metal and virtual environments. This paper extended previous work [25-27] to define a new methodology for evaluating microservices deployment alternatives-also referred to as architecture configurations. In [25], we introduced a metric to assess software scalability. This metric uses requirements definition, high-level architecture modeling, and system measurement results to assess the system architecture's ability to meet performance requirements as load increases. In [27], we presented an approach for evaluating telecommunication systems using Markov approximations. This approach uses operational data and a resource-based Markov state definition to derive an efficient test suite, which is then used as the basis for a domain-based System Under Test (SUT) reliability assessment. Markov approximation is used to estimate the probability of occurrence of each test case at steady state. In this way, the test suite can be effectively reduced and focused on the performance test cases that are most likely to represent production usage. In domain-based load testing, the input domain is the workload, e.g. in terms of arrival rate or number of concurrent users. The total workload is divided into subsets that are related to the probability of occurrence of individual workload situations [27]. The tool paper [26] proposed a fully automated framework for implementing this approach. This approach enables experimental replication in different contexts and visualization of results using a dashboard (e.g. mobile device). This paper builds on all previous work and further includes: (i) new experiments that include security attacks using the Mirai tool [28], (ii) new experiments in a virtualized environment (UNIBZ) that use a different operational profile and additional CPU and memory resources , (iii) sensitivity analysis of performance threshold in experiments.

Microservice Architectural Challenges

In [23] it presents a comprehensive review of the literature on the architectural challenges of microservices. The authors focus on challenges, descriptions of architecture and their qualitative attributes. They found that most of the current research on microservice architecture quality attributes has focused on scalability, reusability, performance, rapid agile development, and maintainability. Furthermore, it presents a systematic overview of the current research on microservices and their application in cloud environments. They found that microservices research is still immature and that further experimental and empirical evaluation of the application of microservices in cloud environments is needed. Their literature survey also identified a need to develop automation tools for microservices. In this paper, we address some of these issues: (i) presenting a scalability assessment methodology that can be integrated into tool automation, and (ii) conducting experiments to validate the proposed methodology.

Performance Assessment of Microservice Architectures

In [29] address the problem of selecting more appropriate performance metrics to activate auto-scaling actions. Specifically, they investigate the use of relative and absolute metrics and propose a new auto scaling algorithm that is able to reduce the response time by a factor between 0.66 and 0.5, when compared to the actual Kubernetes' horizontal auto-scaling algorithm. In this paper, we introduce a new Domain-based metric that captures: (i) Scalability testing results in the SUT using several architecture deployment configurations, (ii) expected production usage derived from operational data analysis. Therefore, the metric represents the system's ability to satisfy scalability requirements for the evaluated workload situations and architecture deployment configurations.

2. Methodology

Microservices have evolved into an

architectural style for developing decentralized applications. Architecture deployment configuration performance-example: B. From a deployment choice perspective-difficult and needed to align our system for production use (Fig. 3). This paper shows how to use operational profiles to generate load tests and automatically evaluate the pass/fail criteria for scalability of microservices configuration alternatives. This approach provides domain-based metrics for each alternative. It can be used, for example, to make informed decisions about alternative choices and to perform production monitoring on performance-related system properties.

In this research paper, we present a quantitative approach to assessing the performance of alternative microservices deployments. This approach uses the results of automated performance tests to quantitatively evaluate the deployment configuration of each architecture against the domain-based metrics shown in this document.

Methodology (Fig. 1) shows the proposed methodological steps and framework architecture for scalability evaluation. We combined the analysis of operational profile data with the results of performance tests to create a domain metric dashboard. The dashboard shows the scalability of the system in terms of distributing operational profiles in a production environment. Define H to be an empirical distribution of workload conditions and performance results in a load test environment. Operational profile data is used to estimate the probability that a situation for each production workload will occur. Scalability requirements are used to evaluate the deployment configuration of each architecture.

The resulting quantitative score is a metric from 0 to 1 that assesses the suitability of a particular architectural alternative for performance in a defined workload situation. Experiments two types of experiments evaluate and apply the proposed approach. First, we calculated the domain-based metrics introduced for 12 different configurations based on three different values: two different memory allocations, two different CPU allocations and the number of microservices replicas. The experiment was conducted in two data center environments.

For each environment, we have identified an architectural deployment configuration that provides the best value for domain-based metrics. Importantly, in both environments, increasing the number of replicas of services evaluated or the percentage of CPU allocation does not guarantee the performance gains evaluated by domain-based metrics. Additional experiments were performed in one of the environments as a basis for later investigating the sensitivity of domain-based metrics to configured scalability requirements. Next, we evaluated the impact of security attacks on domain-based metrics in scenarios with and without security attacks.

This paper states:

Scalability assessment methodology: A new quantitative approach and framework for assessing alternative configurations of microservices architectures.

Experimental validation: Experimental validation of the proposed approach to assess scalability with and without attacks.

Our methodology is designed to support automatic evaluation of architectural deployment configurations. This gives measurements for each configuration. It is so-called domain-based.

Metrics: Quantify the ability of configurations to meet scalability requirements under a particular operational profile. Define workload status as an abstract concept that represents the results of operational data analysis for each application domain. In particular, the number of concurrent users is the focus of this white paper. However, transaction rates are available for other application domains, such as banks. For complex systems under test, such as the microservices architecture learned in this document, it is difficult to assess resource saturation because multiple types of services are running. Multiple software and hardware resources, virtualization engine, load balancer, CPU, memory operation, I/O and network. Thus, the proposed system ables to assess the performance of the system for different architectural configurations of complex microservices architecture.

Steps for computing Domain based Microservices Architecture assessment:

Collecting operational data based on normal system usage in a certain time interval for example, HTTP requests.

The next step is analysis the operational data, which is the quantitative estimation of probably occur a certain workload situation, for example number of concurrent users.

The last step is to find the automated generation of load test cases for architectural deployment configuration under evaluation.

In detail explanation for the above steps is given below:

Operational Data Analysis:

For each workload situation, the operational profile describes the probability of occurrence p(λ), which is estimated by the relative frequency of occurrence f(λ) in the SUT. (i) How do we identify the workload situations to test using the operational profile? What is the representativeness of the selected tests in terms of production usage? Using the operational data, we first create an operational profile as the frequency of occurrence of the workload situations found in the system at a certain time t. The workload situations are aggregated in bins, the final operational profile represents the frequency of occurrence of such bin values, and the test suite coverage the criteria is based on these values.

Experimental Impact:

Under this step, we have defined the test cases with respect to following elements mentioned below:

a. Load test sequence that we can get by selecting workload situations which the bins of operational profile of SUT.

b. Load Test specification includes workload situation and has a choice for architecture deployment configuration.

c. The basic requirements define the pass / fail criteria for each service's tests based on performance metrics. There are no specific assumptions about such performance indicators. The example used in this task is the average response time of the service to the deployment configuration of the reference architecture.

d. A take a look at case includes a fixed of experiments achieved in line with a load take a look at specification and evaluated towards a baseline requirement. In this work, we achieved 60 experiments for every take a look at case.

Experimental Execution of Domain based metric

$$D(\alpha, S) = \left| \sum_{i=1}^{z} p(\lambda i)\xi(\lambda i) \right.$$

Figure 2. Formula used for Domain-based metric [20]

Finally, the configuration's overall Domain-based metric may be assessed in relation to a test suite S defined by workload circumstances. The likelihood of occurrence corresponding to the workload scenario is denoted by p(i).D(p, S, I) = p(i) s(i) is the domain-based measure for each workload circumstance. The quantitative assessment that results is a number between 0 and 1 that may be used to compare the performance of various architectural deployment configurations. D(α, S) was assessed as 0.615 in the running example, and the contribution of the test case given at the conclusion was 0.142, i.e. 0.19 with 74.81%, where 0.19 is the frequency of occurrence of the workload situation = 100 in the operational profile (Fig. 3). The domain-based metric's contribution. Plots may be used to show the distribution of workload situations in a load test run. The figure like this shows how well a certain system deployment configuration meets the fail/pass criterion. For two architecture deployment configurations, the graphs at the workload conditions of the load test sequence demonstrate gaps between the total probability mass (outside polygon, light blue) and the measured measurements (inside polygons). The impact of the measured performance decline on the Domain-based metric is shown by these gaps.

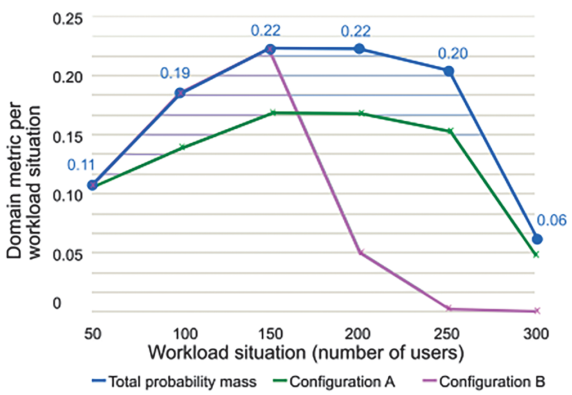

Figure 3. Graphical Representation between Domain metric per workload situation and number of users[23]

Architectural Framework for Methodology

The Fig. 4 [23] represents the architectural framework that has been used for performing Domain-based metric assessment. The framework architecture consists of following compartments that are present below:

An analytical component that collects a production system's operational profile and computes the baseline probability of finding the system in a certain condition as a workload situation [23].

A system for creating and running load tests with various architectural deployment configurations in order to gather system performance against a baseline (fail/pass criteria)[23].

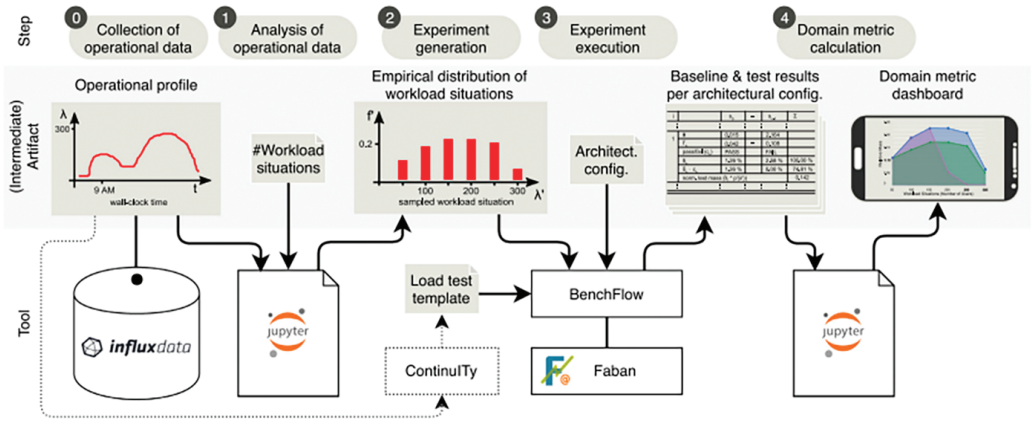

Figure 4. Architectural framework used for performing Domain based metric assessment [24]

Each of these components is controlled by a set of integrated tools. We used Application Performance Monitoring (APM) [24] technologies, such as an open-source tool from the OpenAPM effort, to collect the operating profile. For storing the observed operational data, these solutions frequently use time-series databases, such as InfluxData. Our programme (which is packaged as a Jupiter Notebook4) connects to an InfluxData, collects the raw operational data and calculates the empirical distribution of the workload circumstances (optimal test masses) given in the script. The open-source BenchFlow [25] is used in the load testing infrastructure to automate the deployment of defined experiments based on container-based virtualization using Docker for defined workload circumstances and system configurations.

3. Evaluation Based on Benchmark Tools Results

The load testing instrument that we utilized was BenchFlow [25]. BenchFlow is an open-source framework that automates the performance testing process from beginning to end. BenchFlow incorporates and reuses cutting-edge technologies like Docker and Apache Spark. BenchFlow runs load tests in a reliable manner, collects performance data automatically, and calculates performance metrics and statistics. BenchFlow is also used to ensure that the results are reliable. Users of BenchFlow employ a declarative domain-specific language (DSL) for goal-driven load testing to specify their performance purpose. Declarative templates are supplied for defining test requirements such as test goals, test of types, metrics of interested, test stop circumstances for example, maximum test execution duration and parameter variations during test execution. BenchFlow is a strategy-implementation framework.

Users can describe performance tests declaratively using BenchFlow's DSL [25]. In our situation, we created a load test that explored several system configurations. BenchFlow supports a wide range of variables that can be explored automatically during configuration tests, such as (i) the number of concurrent users, (ii) the amount of RAM/CPU share assigned to each deployed service, (iii) service configurations via environment variables and (iv) the number of replicas for each service. All of the experiments described in this part are defined using BenchFlow's DSL and their automated execution, test execution quality checking and results retrieval are handled by the

BenchFlow framework. Docker containers are used to deliver the environment, with each container implementing a different microservice.

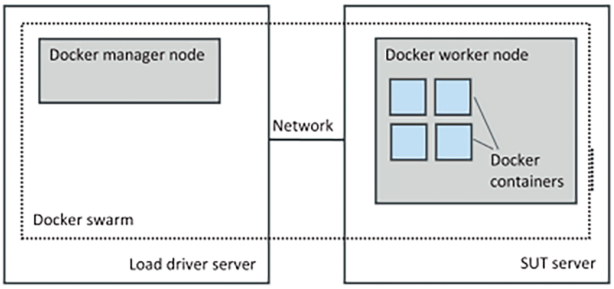

Figure 5. Architectural Overview of HP setup [25]

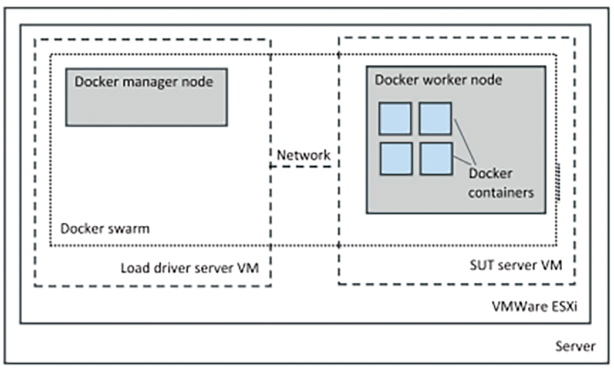

Figure 6. Architectural overview of the setup at UNIBZ, one bar metal machine runs VMWare ESXi that runs over two virtual machines that are configured exactly as in HPI [25]

Containers run as a daemon process on a hypervisor on top of the Docker engine. Fig. 5 displays the HPI deployment, including the Docker Swarm, the Docker management node and the Docker worker node, which contains the Docker containers. The microservice components are executed by these containers. The Dockerswarm is installed on a bare metal computer for the HPI deployment. At UNIBZ, the same deployment is employed. However, there is an extra virtualization layer for UNIBZ installations, as indicated in Fig. 6. The bare metal server there runs VMWare ESXi, which runs two virtual machines in the identical arrangement as at HPI [23].

Architectural Deployment Configuration

We deployed the SUT utilizing several architecture deployment options, as stated by the methodology's experiment generating stage. The quantity of accessible RAM and the CPU share were the characteristics that differed throughout the different deployment scenarios. RAM configurations were chosen from the options 0.5 GB, 1 GB and CPU shares from the options 0.25, 0.5. We removed this feature in the Docker engine to avoid containers being "killed" during operation due to out-of-memory. We repeated the studies using higher memory architecture configurations (8 and 16 GB). We ran 264 trials with various settings in all.

4. Experimental Results

These findings demonstrate that establishing the ideal deployment configuration for a system necessitates the systematic use of engineering methodologies for quantitative performance evaluation. We discovered that increasing CPU power or the number of Docker container replicas may not increase system performance in the bare-metal scenario (HPI). The Domain-based metric oscillates across a limited range in the virtual environment (UNIBZ). Scaling beyond 0.5 GB of RAM, 0.5 CPU share, HPI and UNIBZ settings, appears to imply that other architecture considerations, such as VMware Hypervisor overhead and I/O bandwidth, may be hurting system performance [23]. These findings back up the idea that practitioners can benefit from using. Hence, by examining predicted operating profile and deployment possibilities in their specific environment, as recommended in this study. Furthermore, these findings imply that bottleneck identification and meticulous performance engineering efforts should be carried out before adding extra resources to the architectural deployment configuration [23].

5. Ongoing Challenges

The convergence of the microservices paradigm and the IoT offers a myriad of benefits, but there are also some trade-offs. Challenges to overcome to increase the adoption of microservices architecture styles in various IoT application. This section introduces the key ongoing challenges that impede the adoption of microservices in the IoT. The brief explanation is mentioned in detail:

Diverse Issue Consistency: Applications based on microservices consist of finely distributed and independent units. These entities are implemented in the different technology and data is stored in

different databases and communicates with each other via machine architecture and programming language independent APIs. This increases the total number of critical points that can be compromised [30].

Container's Susceptible: Containers are a simplified way to develop, test, deploy, and orchestrate microservices in a variety of environments. However, it is an attractive target for hackers due to its multiple suscetpitable [31]. The traffic that results from the communication required for container management and orchestration contains data that is valuable to hackers. Also, most of the key components that make up a containerized environment (hosts, registries, images, etc.) can be potential targets for hackers. Hacked containers can be gateways to other parts of the IoT system, such as system hosts, system remote devices and other interoperable containers.

Cloud security Commination: Cloud Native Computing is an IoT system development approach because it allows us to build, execute and deploy decentralized and scalable entities. However, this approach leads to some security threats [32]. When a large amount of data is input to the cloud, especially public cloud services, hackers can endanger or misuse this data. Data breaches can cause serious damage that can affect an organization's reputation and credibility. It can also affect short-term and long-term profits, resulting in loss of intellectual property and significant legal obligations.

Authorization and Authentication: Microservices-based applications need to be able to verify the authenticity of each service involved in communication. If the service is controlled by a hacker, the rest of the service blockchain can be compromised and maliciously exploited [33]. In addition, when a service receives a message, it needs to know if the message is suspicious and if the source service that issued the message has valid credentials. To address these issues, implement a set of authorization and access control mechanisms (policies, models, protocols, etc.) to meet the security requirements of heterogeneous (multi-cloud, etc.) environments, with centralized security management. We need to be able to provide effective protection of the application used.

Resource restriction: Most IoT devices cannot perform complex processing operations in real time, so they lack the computing resources needed to support the implementation of sophisticated and effective authentication and encryption algorithms [6]. A dedicated security chip can be used to solve this problem, but it increases cost, complexity and power consumption and manufacturers have no incentive to follow this route, especially for consumer devices. Local storage is also limited and IoT devices often rely on cloud storage to store the data they generate. This fact opens the door to additional threats in the form of cloud security threats.

Therefore, the above are the following challenges that we can come across while merging IoT with microservice architecture. IoT is blooming in various industries at a height which can be advantageous for users but at the same time you need to come across multiple challenges.

6. Future Redirections

Under future redirections, we had mentions the subsequent terminologies which provide more security further to Microservices IoT mechanism.

AI/ML based solutions: Artificial intelligence (AI) and machine learning (ML) have been demonstrated in the context of the IoT. Efficiency of analyzing and delivering data collected and stored in cloud infrastructure quickly and accurate prediction and decision. We can also use both AI and ML to improve the security of our IoT ecosystem. This allows us to identify, investigate and predict potential external threats [33]. Through a decentralized offering learning systems, AI / ML-based solutions can improve the security of microservices-based IoT applications. Monitor the flow of data needed to ensure communication and orchestration between these entities container.

Gaining trust: In the context of IoT applications, traditional cloud systems require a third party to build trust between data owners and users. To address this trust issue, dynamic smart contracts can be used to design and implement proxy re-encryption schemes at run time [33]. Smart contract technology is defined as a set of protocols that digitally guarantee contract validation, control, and execution. Smart contracts are widely adopted in various key sectors such as healthcare [34], financial services and supply chains because they

offer many advantages in terms of autonomy, security and accuracy. In the context of IoT applications, smart contract integration eliminates the need for trusted third parties. In addition, proxy re-encryption effectively ensures the visibility of the data between the owner and the user registered in the smart contract. Therefore, the proposed system enables secure sharing and storage of IoT sensor data. This is achieved by encrypting the data before uploading it to the cloud and re-encrypting it before sharing it with users for complete confidentiality.

Blockchain: Blockchain technology can be used as a third party to secure distributed and heterogeneous ecosystems of microservices-based IoT applications. The blockchain has proven its efficiency in this field by offering several features (e.g., tamper-proof, robust encryption, transparency, fast transactions and coordination among billions of connected devices, anonymity, etc.) and benefits (e.g., improved trust and security, susceptibility to manipulation, and fairness). In IoT contexts, a variety of features can be created, hosted and utilized dynamically, such as smart contracts, cryptography algorithms, identity access management platforms, identity-based consensus mechanisms, etc. In IoT contexts, a variety of features can be created, hosted and utilized dynamically, such as smart contracts, cryptography algorithms, identity access management platforms, identity-based consensus mechanisms, etc.

Design for IoT: Ethics is a branch of philosophy that rationalizes our moral judgments and distinguishes between morally right and wrong, justice and injustice. As IoT systems are complex, heterogeneous and large, new ideas and considerations are needed to define appropriate regulations and policies to ensure effective data security and privacy in this complex environment. This must be presented. Therefore, to protect the IoT ecosystem, we need to create an ethical framework to understand what is right, what is wrong and what is good or bad [10]. These frameworks include mechanisms that need to be relevant to heterogeneous ecosystems, including humans, autonomous and self-determining systems, virtual and physical devices and environments.

Security Design Approach: Security by design is an approach to software and hardware development that integrates security from the beginning of the initial development process, not as an addition after an incident caused by a security breach. In the context of IoT applications, adopting concrete and best practices, following practical development guidelines and mapping security requirements to the life cycle of IoT components is an important consideration for implementing a security-by-design approach [33]. That allows IoT components to remain reliable and operational from design to service life. This promising approach needs to be further investigated, especially by investigating and implementing the proposed security patterns for microservices-based architectures.

Conclusion

In this research paper, we have performed a Domain based metric microservice architecture assessment, to evaluate the performance of CPU and I/O. The paper introduces briefly about microservice architecture, also the impact of microservice architecture along with IoT on the user. The approach for the assessment relies on few steps. They consist of operational data analysis, experiment generation and the experimental execution. Our Domain based metric is calculated for each microservice option, which is specified as an architectural deployment configuration. The metric (0-1) represents the deployed configuration's ability to satisfy performance criteria for the predicted production use load. We used our technique in numerous deployment scenarios, including a large bare-metal testing environment and a virtualized environment. The solution made use of an advanced load test automation tool to automate the deployment of Docker containers. Using a baseline calculation of these needs, we add to the state of the art by automatically calculating baseline performance requirements in a baseline run and analyzing pass/fail criteria for the load testing. Furthermore, we were able to fully automate our technique, a framework that can be used in any real-world production setting. We discovered that in auto-scaling cloud environments, careful performance engineering activities must be performed before additional resources are added to the architecture deployment configuration, because increased workload at the bottleneck resource may

result in significant performance degradation. Our proposal is to validate the performance of CPU and RAM at a node and for evaluation which we are using benchmarking tools like Docker containers.

Funding: This work was supported by China Scholarship Council (grant number 2018GXZ020784).

Conflicts of Interest: Authors must declare all and any conflicts of interest.

REFERENCES

Aderaldo, C. M., Mendonça, N. C., Pahl, C., & Jamshidi, P. (2017). Benchmark requirements for microservices architecture research. In Proc. 2017 IEEE/ACM 1st Int. Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering (ECASE).

Alshuqayran, N., Ali, N., & Evans, R. (2016). A systematic mapping study in microservice architecture. In Proc. IEEE 9th Int. Conf. on Service-Oriented Computing and Applications (SOCA).

Antonakakis, M., et al. (2017). Understanding the Mirai botnet. In USENIX Security Symposium.

Avritzer, A., et al. (2019). PPTAM: production and performance testing based application monitoring. In Companion of the ACM/SPEC Int. Conf. on Performance Engineering (ICPE).

Avritzer, A., et al. (2020). Scalability assessment of microservice architecture deployment configurations: A Domain-based approach leveraging operational profiles and load tests. Journal of Systems and Software.

Avritzer, A., & Weyuker, E. J. (1995). The automatic generation of load test suites and the assessment of the resulting software. IEEE Transactions on Software Engineering.

Barker, C. (2016). Mirai (DDoS) source code review. Medium.

Casalicchio, E., & Perciballi, V. (2017). Auto-scaling of containers: the impact of relative and absolute metrics. In 2017 IEEE 2nd Int. Workshops on Foundations and Applications of Self Systems (FASW).

Chitturi, A. K., & Swarnalatha, P. (2020). Exploration of various cloud security challenges and threats. In Soft Computing for Problem Solving. Springer.

Dey, N., Mahalle, P. N., Shafi, P. M., Kimabahune, V. V., & Hassanien, A. E. (2020). Internet of things, smart computing and technology: a roadmap ahead. Springer International Publishing.

Dmitry, N., & Sneps-Sneppe, M. (2014). On micro-services architecture. International Journal of Open Information Technologies.

Driss, M. (2022a). Req-WSComposer: a novel platform for requirements-driven composition of semantic web services. Journal of Ambient Intelligence and Humanized Computing.

Driss, M. (2022b). WS-ADVISING: a Reusable and reconfigurable microservices-based platform for effective academic advising. Journal of Ambient Intelligence and Humanized Computing.

Driss, M., Aljehani, A., Boulila, W., Ghandorh, H., & Al-Sarem, M. (2020). Servicing your requirements: An fca and rca-driven approach for semantic web services composition. IEEE Access.

Francesco, P. D., Malavolta, I., & Lago, P. (2017). Research on architecting microservices: trends, focus, and potential for industrial adoption. In 2017 IEEE Int. Conf. on Software Architecture (ICSA).

Hajjaji, Y., Boulila, W., Farah, I. R., Romdhani, I., & Hussain, A. (2021). Big data and IoT-based applications in smart environments: A systematic review. Computer Science Review.

Li, D., Deng, L., Cai, Z., & Souri, A. (2020). Blockchain as a service models in the Internet of Things management: Systematic review. Transactions on Emerging Telecommunications Technologies.

Lu, D., Huang, D., Walenstein, A., & Medhi, D. (2017). A secure microservice framework for IoT. In IEEE Symposium on Service-Oriented System Engineering (SOSE).

Maheswari, M., & Karthika, R. A. (2022). A Novel Hybrid Deep Learning Framework for Intrusion Detection Systems in WSN-IoT Networks. Intelligent Automation and Soft Computing.

Microsoft. (2022). Microservices architecture style. Azure Architecture Center. https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices

Mohanta, B. K., Jena, D., Satapathy, U., & Patnaik, S. (2020). Survey on IoT security: challenges and solution using machine learning, artificial intelligence and blockchain technology. Internet of Things.

Ponnusamy, V., et al. (2022). IoT Wireless Intrusion Detection and Network Traffic Analysis. Computer Systems Science and Engineering.

Ragupathi, T., Govindarajan, M., & Priyaradhikadevi, T. (2022). Class Imbalance Handling with Deep

Learning Enabled IoT Healthcare Diagnosis Model. Intelligent Automation and Soft Computing.

Rayes, A., & Salam, S. (2019). Internet of things (IoT) overview. In Internet of Things from hype to reality. Springer.

Razzaq, A. (2020). A systematic review on software architectures for IoT systems and future direction to the adoption of microservices architecture. SN Computer Science.

Riazul, I. S. M., Kabir, M. H., Hossain, M., Kwak, D., & Kwak, K. S. (2015). The internet of things for health care: a comprehensive survey. IEEE Access.

Sequeiros, J. B., Chimuco, F. T., Samaila, M. G., Freire, M. M., & Inácio, P. R. (2020). Attack and system modeling applied to IoT, cloud, and mobile ecosystems: embedding security by design. ACM Computing Surveys (CSUR).

Sultan, S., Ahmad, I., & Dimitriou, T. (2019). Container security: issues, challenges, and the road ahead. IEEE Access.

Sultana, T., et al. (2020). Data sharing system integrating access control mechanism using blockchain-based smart contracts for IoT devices. Applied Sciences.

Tapia, F., et al. (2020). From monolithic systems to microservices: A comparative study of performance. Applied Sciences.

Vural, H., Koyuncu, M., & Guney, S. (2017). A systematic literature review on microservices. In Proc. ICCSA. Springer.

Wisetsri, W., et al. (2022). Electricity Theft Detection and Localization in Smart Grids for Industry 4.0. Intelligent Automation and Soft Computing.

Yarygina, T., & Bagge, A. H. (2018). Overcoming security challenges in microservice architectures. In 2018 IEEE Symposium on Service-Oriented System Engineering (SOSE).

Yu, D., Jin, Y., Zhang, Y., & Zheng, X. (2019). A survey on security issues in services communication of microservices-enabled fog applications. Concurrency and Computation: Practice and Experience.