Journal of Theoretical and Applied Information Technology
15th June 2021. Vol.99. No 11
© 2021 Little Lion Scientific

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

BENCHMARKING MICROSERVICES ARCHITECTURE IN IMPROVING USER EXPERIENCE

[1]HERALDO YUSRON P, [2]NAUFAL RIZQULLAH P H, [3]BENFANO SOEWITO
[1,2] Computer Science Department, School of Computer Science, Bina Nusantara University, Jakarta, Indonesia 11480
[3] Computer Science Department, BINUS Graduate Program, Master of Computer Science, Bina Nusantara University, Jakarta, Indonesia 11480

E-mail: [1]bsoewito@binus.edu, [2]heraldo.purwantono@binus.ac.id, [3]naufal.hidayat@binus.ac.id

ABSTRACT

Technology is an important thing that must be considered, technology continues to be developed in order to help people to work more efficiently. In every company, one of their main goals is to achieve maximum efficiency, with efficient work, it could reduce the cost necessary and increase the productivity of its business process without sacrificing the quality of their byproduct. One of them came from the terms of communication between employees. How many companies use web applications as a medium to communicate, but the current adopted architecture is mostly still monolithic. A commonly known monolithic architecture has some limitations, especially when applications tend to be more complex such as slow access speeds because programs are running simultaneously in one mass architecture system, small changes to the system require an entire monolithic to be rebuilt, and limited scalability can occur. Therefore, it is proposed to use microservices for internal web applications. Microservice has recently gained popularity among developers since 2014. Because many companies that have implemented this technology can maximize their profits and get a better user experience due to better access speed capabilities. Therefore, in this study, we try to fix the problem and implement a microservice architecture in the hope of providing a better user experience and increasing productivity for their employees. Thus, we need to compare both architectures using comparable benchmarks and try to prove that microservices can lead to better performance and user experience.

Keywords: *Benchmark, Microservices, Software Architecture, Monolithic, DDD*

1. INTRODUCTION

In this era, technology is a necessity that everyone must master without exception. Over time, technology continues to develop from year to year. Technology is made to facilitate human work in various fields. Therefore, Humans continue to innovate and seek the latest technology that humans can use to make their work easier. For example, in the corporate sector, the company itself has a lot of data processing, the data that can be processed can be millions or even tens of millions. In data processing, a media or third party is needed that can be a bridge to share data according to employee needs. In this research, the third party is a web applications application. Web applications are used to make it easier for employees to select data that they can use to reprocess them into data that can be useful in other fields. Of course, web application does not escape technological developments, humans innovate and continue to develop web application to make them easier to use and to be more efficient. In general, a web application has a main function that can be used by users, for example in a company. Companies can use a web application to transfer raw data which will be processed by the user. However, in some cases companies make many web applications for users to use because they have different forms of data, but actually the web application has the same function, so the user must have a different account to access each application. In this case, according to the researcher, it is very ineffective because the user has many accounts to access data. Therefore, researchers created a web application that combines several of these applications using the microservices method. According to Johannes Thönes in his research said that microservices are microservices, are small applications that can be implemented independently, scaled independently, and tested independently and have one responsibility. It is sole responsibility in the original sense that there is one reason to change and / or one reason to be changed. But the other axis is single responsibility in the sense that it only does one thing and one thing and can be easily understood [1].

Based on the description in the background, the research problem we willing to solve can be identified based on follows:

- ✓ How to combine multiple application that are used as third parties to transfer data by users with one account
- ✓ How to implement microservices into a web application

2. LITERATURE REVIEW

According to the one who pioneered the term of "microservice" [2], it is described as an approach to developing a single application as a suite of small services, each running its own process and communicating with lightweight mechanism, often as an HTTP resource API. While other [3], defined it as "distributed application where all its modules are microservices" or "Microservice are small, autonomous service that work together" as described by [4]. This architecture model allows developers to build application as suits of services. As well as the fact that each service is independently deployable and scalable, thus also provides a firm module boundary, even allowing for different services to be written in different programming language and teams. This new form of architectural style was established due to an increased frustration from user especially developer when they were using monolithic architecture style. Especially, as more application is being deployed when small changes made to a small part of the application, requires entire monolithic to be rebuilt and deployed. Over time, difficulties occur in maintaining a good modular structure and making it harder to keep changes that ought to only affect one module within that module. Scaling requires scaling of the entire application rather than parts of it that require greater source [2].

There aren't any formal definition yet that represent the microservice architectural style, but according previous paper [2], [3], we found out that these researches describes some of its characteristic. Common characteristic of microservices architecture include:

- ✓ Componentization via services—the application is developed upon smaller independent services that runs different processes, applied boundaries on its resources, and communicating through lightweight mechanism
- ✓ Organized around business capabilities—built by cross-functional teams.
- ✓ Focus more on products rather than projects—prioritize more on business capabilities, not the software as a set of functionality to be completed.
- ✓ Smart endpoints and dumb pipes—choreographed using simple REST protocols in order to avoid other complex orchestration protocols
- ✓ Decentralized governance and data management—can be developed using different technologies and data management infrastructure
- ✓ Infrastructure Automation—built by teams with extensive experience on continuous delivery and integration
- ✓ Design for Failure—capable of detecting failure quickly and automatically restore service
- ✓ Evolutionary Design—design pattern can adapt dynamically in response to service changes

Another characteristic that has been defined by [4], that makes microservices different are "small and focused on doing one thing well" and "autonomous". Which these characteristics were also mentioned in previous paper [2], [3].

Microservice in some of its implementations are commonly being compared with Service-Oriented Architecture (SOA) and Domain-driven Design (DDD). In fact, at the time when there was absence of any standardized definition for microservice, SOA and DDD concept are widely used to develop microservices. And most of the papers even said that MSA are inherited from SOA with a bounded context that adapted from DDD concept [3], [5]–[8]. Before we jump right on to microservices architecture structure, we will be discussing about how microservices beforehand related to its predecessor's architecture, concept and support system. And how those mentioned architecture influenced microservice to be as it is today

2.1. Monolithic Architecture

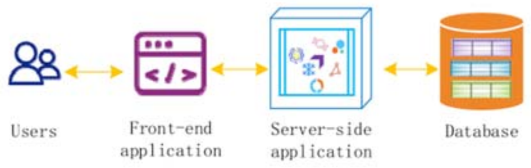

Figure 1. Monolithic Architecture [9]

As mentioned previously in this section, monolithic application is built as a single unit. Enterprise Application are commonly built upon three main parts: a client-side user interface, a database, and a server-side application. The server-side application which handle HTTP requests, execute domain logic, retrieve & update data from database, and select & populate HTML views to be sent to the browser. This server-side application can be called a monolith - a single logical executable. Any changes to the system requires the entire monolith to be rebuilt and deployed [2]. Especially, as application tends to become more complex, its weaknesses appear. For example, high complexity, poor reliability, limited scalability, and hindering technological innovation. As shown above in Fig. 1, which embodied a traditional monolithic architecture. First, the user interacts with the front-end. Then, the front-end application redirect user request to a container that hosted a software instance, which then connects to the database to complete those requests [9].

2.2. Domain-Driven Design (DDD)

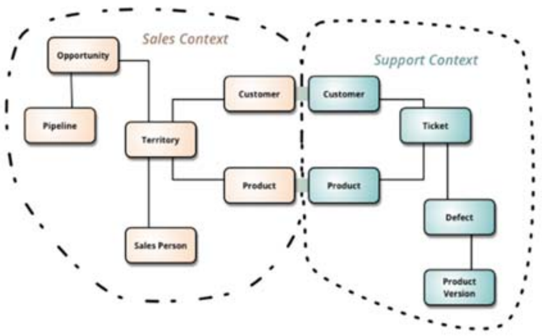

Figure 2. Bounded context concept on Domain-Driven Design [10]

Domain-Driven Design (DDD) has gained more acknowledgement as a result of raising popularity on microservices technology [5]. The main idea of DDD is the binding of its domain to the implementation [11]. Its strategic design is focused more on dealing with large models and teams by dividing them into different bounded context and being explicit about their interrelationship as shown at Fig. 2 [10]. By popularizing its bounded context concept, each component in the system only exists within its bounded context. More microservice implementation done by using this approach due to its well-established set of practices that enables modelling complex systems. So, the following bounded context is a prominent way to start designing microservice, which allows a loosely coupled microservices design [5]

2.3. Microservice Architecture (MSA)

As we also previously said that microservices has some characteristic of dumb piped, flexibility and loosely coupled characteristic. And by means "loosely coupled" will make the change to a service would not bother or require a change on another service, which differentiate it with monolithic that required the whole systems to be rebuilt [4]. Unfortunately, there aren't any standardized microservice architecture, because each microservice architecture was built specially to fulfil its own business requirements. According to various papers that have been found [7], [9], [12]–[14], these papers shows similar form of architectural pattern for microservices. Take Instagram apps for example, Instagram will use a separate service for each action: *Share* (Move a photo from a device), *Like* (A method for incrementing internet karma), *Follow* (Subscribe to a particular user's photos), *Search* (Find photos based on criteria), *Register* (Create a user). These common concern from each of these services separating its data. The separation happens for retrieving user data and how to authenticate user. Authentication often happens frequently and universally by many services. With every call on most of the apps/services contain information about an authenticated user (such as token). This action needs to be looked up quickly from many different

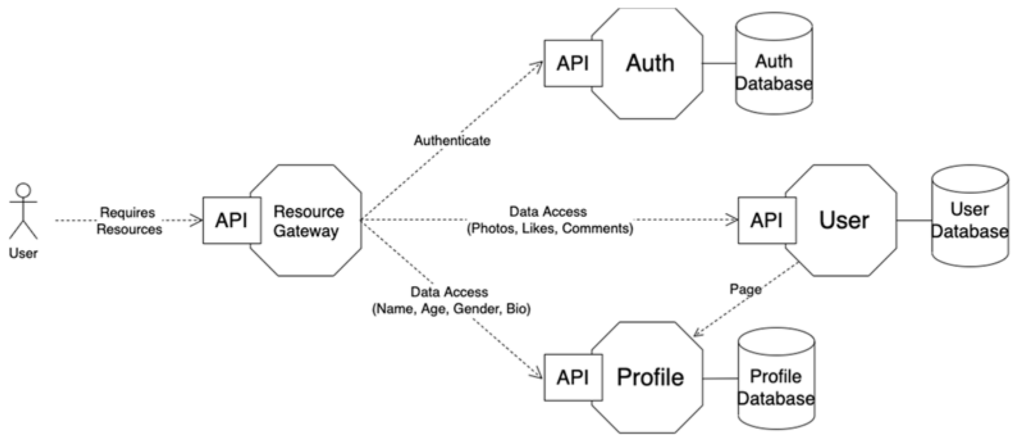

Figure 3. An example of microservices architecture

services. However, the user entity, with entities of name, preferences, and email address probably needs to be accessed less frequently. Thus, while user data and authentication are coupled, it is probably a good idea to keep them separated [15]. By using the same architectural concept that Liu proposed in his paper [9], we try to implement their microservice architecture to form our own version of Instagram's microservices architectural pattern shown in Fig. 3 With Auth API that process user authentication service such as token retrieved from authentication database. Profile API, which retrieve information about user entities from Profile Database. And lastly, User API, which also retrieve user photos, follows, likes, etc. from user database. It is possible that User API might also be divided into several breakdowns based on photos, follows, like, etc. to have more granularity, since "smaller is better for microservices". But, such example is only intended to give readers an overview of how microservice architectural pattern was implemented for each service.

2.4. Benchmarking Microservices

Benchmarking as paper [16] mentioned, "Is the process of measuring quality and collecting information on system states.". Benchmarking can also be applied to measure different software versions, configurations, system alternatives, or deployments [17]. We've search for paper related to microservices with keywords include *assessment, performance, benchmark, benchmarking* across various journals. Seeing that microservices research are relatively immature [18], and some paper stated that benchmarking microservices is hard because each of the application requires their own custom benchmark which also need to be assessed repeatedly as the service evolves [19], [20]. Above all hindrance and limitation in finding related research available online, we finally found some. And conclude that benchmarking a microservice application is still possible and could give a tremendous contribution to researchers, practitioners and communities in this field due to its adequacy.

Research that discusses *benchmarking* or non-functional performance assessment criteria of a *microservice*. Which one of them [17], where the study uses a pattern-based approach to assess non-functional *microservice* more easily, while still considering the quality of complex interactions. Assuming that a *microservice* exposes the REST API, described in a machine-understandable way, and allows *developers* to model interaction patterns from abstract operations that can be mapped to that API. Required parameter values are provided at runtime and possible data-dependencies between operation are resolved. They implemented their approach in a prototype, which then being used to demonstrate the low effort applicability of their pattern benchmarking approach to three open-source microservices. As a result, their work shows that pattern-based benchmarking of microservices is feasible and opens up opportunities for microservices providers and tools for developers.

Other research [20] proposes, discusses, and illustrates the use of an initial set of requirements that may be useful in selecting a community-owned architecture benchmark to support repeatable microservice research. In order to fill a lack of repeatable empirical research on the design, development, and evaluation of microservice applications. By using selected possible benchmark candidates amongst five open source microservices applications, the conducted assessments cover up architecture, DevOps, and General contexts. Although, their early results indicate that none of the five applications analyzed is relatively mature to be used as a community-wide research benchmark, each one of them may already be useful to fulfill the needs and promote the reproducibility of specific empirical studies. In hope for a better requirement in benchmarking microservices, they expect that their research constitutes as an 'ideal' benchmark for conducting empirical microservice research.

Another research we found offers an assessment for microservices architecture qualitatively rather than quantitatively [3]. This study stated that, even though microservice was the latest architectural trend and capable to solve various problems associated with monolithic architectures, there still some significant disagreement on when should microservice architecture applied. Likewise, how it may be implemented effectively. In respect of limited empirical research on the topic, this study identifies and discusses a range of opportunities and challenges associated with the microservice application and implementation. The findings reviewed an in-depth interview with 19 ICT architects with significant experience in large corporate systems, middleware, service-oriented architecture, and some of limited extent microservices.

Research conducted by [13] stated that microservice architecture, from its architectural perspective impose a number of relevant challenges related to their high degree of distribution and decoupling. Things such as measuring, controlling, and system architecture quality assurance are of paramount importance. The paper proposed an approach for specification, aggregation, and evaluation of software quality attribute related to the architecture of microservice-based system. This proposed approach allows developer to (i) produce architecture model, either manually or automatically via recovering techniques, (ii) contribute to a well-specified ecosystem and automatically computable software quality parameter for MSAs, and lastly (iii) continuously measure and evaluate the architecture of their system by (re-)using the software quality parameter defined in the ecosystem. This approach is implemented by using Model-Driven Engineering techniques and has been validated by assessing the maintainability of a third-party, publicly available benchmark system.

3. METHODOLOGY

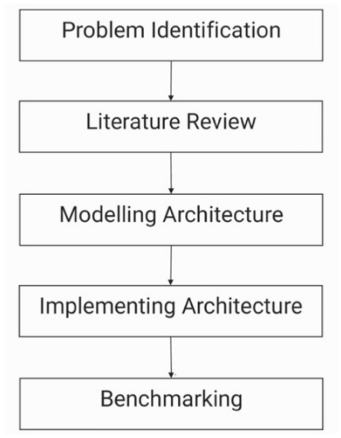

Figure 4. Research Stages

In this research, assessing a microservices application will be built upon several stages as shown in Fig. 4. Those stages should be taken into consideration based on the importance of each stage for this research, which will be explained explicitly as follows.

3.1. Problem Identification

This study is brought up by a problem that came from a company that uses web application for their internal usage. Just by accessing this internal web application, we saw a problem that fuels our concern. This web application requires different pages and accounts for each different service as shown in Fig. 5. For example, when user A needed to collect different data from service 1 and 2. He required to login to a webpage that contains the service 1 to access the data, and then opens up another tab and open another webpage that contains service 2, login, and then access the data. It was a user experience nightmare, and we've heard some of the employees complain about this kind of access that demand more time to switch between each web pages which hamper their productivity. According to their experience, we trying to analyze the reason why this company uses that kind of web page architecture. Afterwards, we suggest a new web page architecture that could perform better in terms of user experience, maintainability, and performance. "Better" by all mean is we trying to compare previous architecture over recent one and define a parameter that could be a reference in comparing those two.

3.2. Literature Review

In this stage, we search for various studies related to software architecture especially microservices which recently has been trends among companies, communities and researchers. We also mentioned other predecessor's software architecture that pioneered microservices to be microservices that generally being used today. Related predecessors' architecture that became the constituent parts of microservices such as Service-Oriented Architecture (SOA), Domain-Driven Design (DDD), and its support system such as Development and Operation (DevOps) as mentioned in section II. From that various related studies, we also found out that those studies simultaneously mentioned the benefits of microservices over other former software architecture because of its adaptability. For many companies that have been using microservices architecture for their services, it became one of the main factors that bolsters up their tremendous business growth and outpaces their competitors. Not to mention, Amazon, Netflix, Instagram which have been implementing microservice, saw potential use on this kind of architecture in the future. Therefore, we try to conduct this study more about microservices architecture rather than other architecture.

3.3. Modelling Architecture

Before we develop the application, first we need to model the architecture based on its authorization and role using UML's use case diagram. We use *Use Case Diagram*, because previous architecture was built based on its role, authorization and service provided. Thus, we saw this type of modelling language suits the architecture. As we also had mentioned, there are 2 use case diagrams that we're going to make including the company's own architecture and microservices architecture shown on Fig. 5. The company's architecture describe that each user requires to login into different website for each different available service. And the microservices architecture only requires user to login one time to access all available services.

3.4. Implementing Architecture

After the architecture design was made, we then implement the designed architecture to web application using PHP supported with Laravel framework. We prefer PHP as our implemented application because this company's web architecture is also using PHP as its preferred language. This same language implementation will support each assessment to provide a better comparison and benchmark between the two.

3.5. Benchmarking

Based on what is seen from the problems that occur in a company, in this stage, we will assess each architecture using the empirical approach that has been used in [20] using parameters that are determined based on the context. Where these parameters was based on the case of how microservices application being developed and deployed by various practitioners and industry experts [2], [4], [20]. As described in Aderaldo's works [20], this specified parameter reflect how typical microservices applications are currently being developed and delivered into production, as reported by practitioners, and industry experts.

3.5.1. Requirements related to architecture

These parameters reflect the ideal characteristics of the benchmark for microservices from the perspective of its architectural nature.

P1: Architectural Model: A typical application for microservices consists of many tiny independently deployable services that can communicate asynchronously and indirectly during runtime. These features make it hard for a developer to completely understand the integration points and obligations of all resources within the overall framework architecture, based on its source code alone. At runtime, a well-documented benchmark for microservices should provide an explicit view of its key service components and their possible communication channels. In order to help software engineering researchers in better understanding, exploring and evaluating the architectural design decisions and compositional runtime topologies of the benchmark, such a view is necessary.

P2: Pattern-based Design: The advantages of a pattern-based software architecture have long been recognized by the software engineering community, such as ease of maintenance and reuse. A number of industry-tested architecture trends have already been proposed to facilitate the development of scalable and robust applications for microservices in this regard. Some of the most common microservice patterns are Circuit-breaker, API Gateway and Service Discovery. Therefore, the use of such patterns is expected in the design of a microservices benchmark that is representative of how applications for microservices are currently being developed and delivered to production environments in the real world.

3.5.2. Requirements related to DevOps

These parameters reflect the need of the industry to adopt core programming practices of the DevOps continuous delivery pipeline for a production-ready microservices framework.

P3: Easy Access from a Version Control Repository: The use of a Version Control System (VCS) is a crucial aspect of the development of any modern software, even more so in a distributed setting. There is a mandatory prerequisite for any microservices benchmark candidate to use a public distributed VCS such as GitHub or Bitbucket as its key software repository, as it enables software engineering researchers and practitioners to have easy access to the source code of the benchmark and release history.

P4: Support for Continuous Integration: Continuous integration is a software development practice in which, with any code commit sent to the version control system, new code developed on a developer's computer is automatically merged with the current software code base. It is the duty of continuous integration software such as Jenkins and TeamCity to construct a new application build and to alert the developer team of the outcomes of the build. These tools can also cause additional functions, such as code quality management and checking, to be performed.

P5: Support for Automated Testing: Automated research instruments such as Cucumber and Selenium are able to conduct experiments, record their findings and equate them with previous test runs. It is possible to run experiments conducted with these methods continuously, at any time.

P6: Support for Dependency Management: It is the duty of a dependency management program such as Maven or NPM to automatically download and install all external software objects (e.g., modules, libraries) needed to create a given software product locally. Those tools have a particular notation, called a manifest file, for defining those dependencies. Dependencies defined in a manifest file are usually downloaded from a central repository of software

P7: Support for Reusable Container Images: Applications for microservices, as offered by public cloud vendors, are usually implemented in a virtualized infrastructure. Developers typically rely on lightweight containerized virtualization tools, such as Docker, to build reusable container images of the entire software stack and execution environment needed to execute each application portion in order to speed up deployment. This helps

Table 1. Benchmark Assessments Parameter

Context	Parameters		Assessment Rationale
Architecture	P1:	Architectural Model	Represent how he application should provide enough architecture view for each services elements
	P2:	Pattern-based Design	Represent how the application should be designed based on renowned microservices architectural pattern
DevOps	P3:	Easy Access from a Version Control Repository	Represent how the application repository should be easily accessible from a public version control system
	P4:	Support for Continuous Integration	Represent how the application should provide support for at least one continuous integration tool
	P5:	Support for Automated Testing	Represent how the application should provide support for at least one automated testing tool
	P6:	Support for Dependency Management	Represent how the application should provide support for at least one dependency management
	P7:	Support for Reusable Container Images	Represent how the application should provide reusable container images for at least one container technology
	P8:	Support for Automated Deployment	Represent how the application should provide support for at least one automated deployment tool
	P9:	Support for Container Orchestration	Represent how the application should provide support for at least one container orchestration tool
General	P10:	User Experience required steps	Represent how the application deals with the user experience according to user's step
	P11:	Page Runtime	Represent how much time required for the application to run the webpage
	P12:	Alternate Versions	Represent how the application should provide alternate implementation in terms of programming languages and/or architectural decisions
	P13:	Community Usage and Interest	Represent how the application should be easy to use and of interest to its target research community

the program to be conveniently deployed separately from the underlying physical system in the same virtual environment (e.g., developer machines, production servers)

P8: Support for Automated Deployment: A standard microservice application's implementation configuration can vary greatly across different execution environments (e.g., development, staging, production). Changing the execution environment would also entail changing its source code if such variations were incorporated in the implementation of the program. This will, of course, cause delivery of applications a very challenging and error-prone process. Developers typically define environment-dependent configuration details in objects external to the source code that are then used by automatic deployment tools such as Chef and Ansible to mitigate this issue. These tools typically have a systematic and centralized means of defining the numerous forms in which a microservices program could be deployed at runtime.

P9: Support for Container Orchestration: A delightful feature of containers is that they can be planned and orchestrated instantaneously on top of either physical or virtualized computing infrastructure. Three of the most widely used container orchestration software are Docker Swarm, Kubernetes and Mesos. These tools offer automated system assistance to overcome some key issues in the implementation of applications for microservices, such as infrastructure discovery, load balancing and rolling updates.

3.5.3. General requirements

These parameters represent general benchmark characteristics which, from a technological point of view, are not compulsory but which would give contribution to community and researchers in enhancing insight for developing microservices architecture.

P10: User Experience required steps: Especially for this study, assessment based on user experience related to each required step will be conducted. This assessment was held in purpose of seeing how the user will then interact with the deployed application. Former web application as stated before, require user to login into different web pages to use each service provided. And in contrast, we develop another web application using an implementation of microservices that might provide a better user experience. And in this assessment will provide a more comprehensible comparison for identifying how well the recent developed architecture in compare of previous architecture in term of its user experience.

P11: Page Runtime: Generally, a quality of a software engineering is commonly related to its

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

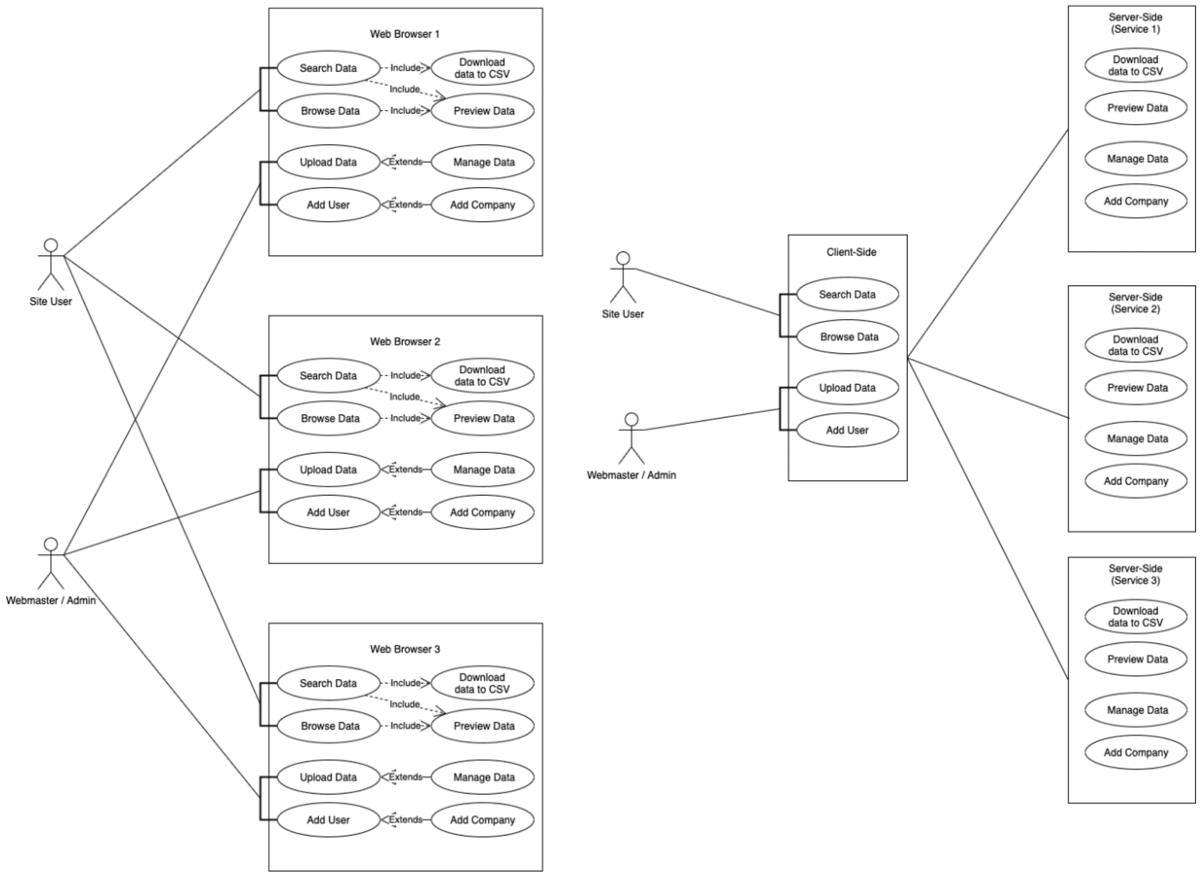

Figure 5. Use Case Diagram of X-Company's (Left) and Proposed Microservices Architecture (Right)

user accessibility, and time are mainly mentioned. Research that using time as its benchmark measurements such as [19], mentioned that by using duration they would enhance their approach with the capabilities necessary for benchmarking entire microservices application. Especially, the ability of resolving complex data dependencies across microservices endpoints. Therefore, we consider that runtime is one of our measurements in benchmarking both architectures.

P12: Alternate Versions: Providing several implementation alternatives is another attractive characteristic for any software engineering benchmark. In the unique case of a benchmark for microservices, this may mean offering alternative implementations of the benchmark microservices using various programming languages (e.g., Java and Node.js) or different architecture designs (e.g., monolithic vs. decentralized). For Software Engineering researchers interested in comparing various microservice architectures in terms of their design decisions and their technical choices, this will be extremely useful.

P13: Community Usage and Interest: A research benchmark's use history illustrates how much its target testing audience has used the benchmark. In addition, the attention of the group is likely to be further drawn by a benchmark that is well defined and easy to customize (for example, to enable integration of external data collection and research tools) and rollout. Of course, using a benchmark for microservices that is simple to use and has already drawn the attention of other researchers in the area not only enhances confidence in the adequacy of the benchmark for new studies, but also enables repeatability of previous benchmark findings.

Mentioned explanation of related parameter will be summarized according to its perspective and rationale in Table. 1.

4. ARCHITECTURE

In assessing both x-company's architecture and our proposed microservices architecture, we'll provide both architecture model to inform readers how the application would be developed in this study as we described previously in modelling architecture

Table 2. Benchmark Assessment Results

	Benchmark Assessment Results			
Par.	**X-Company's Architecture**		**Microservices Architecture**	
P1	😊	The application was provided with an explicit view on its overall architecture	😊	The application was provided with an explicit view on its overall architecture
P2	😊	Database per each service, 3 web application using the same 3 monolithic architecture (monolithic architectures)	😊	Database per each service, 1 web application for client side, 3 web application for different services (Client-Server architecture)
P3	☹	Source code are not publicly available on GitHub	☹	Source code are not publicly available on GitHub
P4	😐	No CI tools used. Although, some CI tools are supported for this apps such as TravisCI, GitHub Actions, and Jenkins	😐	No CI tools used. Although, some CI tools are supported for this apps such as TravisCI, GitHub Actions, and Jenkins
P5	😐	No Automated Testing tools used. Although, it supported with Automated Testing tools such as PHPUnit	😐	No Automated Testing tools used. Although, it supported with Automated Testing such as PHPUnit
P6	😊	Includes Composer as its DM	😊	Includes Composer as its DM
P7	😐	No Reusable Container Images used. Although, it supported with docker for its Reusable Container Images	😐	No Reusable Container Images used. Although, it supported with docker for its Reusable Container Images
P8	☹	No Automated Deployment. Although, it supported with Automated Deployment tools such as Jenkins	☹	No Automated Deployment. Although, it supported with Automated Deployment tools such as Jenkins
P9	😐	No container orchestration used. Although, it supported with Container Orchestration tools such as Kubernetes	😐	No container orchestration used. Although, it supported with Container Orchestration tools such as Kubernetes
P10	😊	After some iteration, it has an average access speed of 415ms after user login. And average speed of 280ms when user switch to another page	☹	Our test results shows that it has an average access speed of 550ms after user login. And average speed of 135ms when user switch to another page
P11	☹	Require around 6 step for user to access data from 2 services	😊	Require around 4 step for user to access data from 2 services
P12	☹	Available only in PHP with CodeIgniter web frameworks	☹	Available only in PHP with CodeIgniter web frameworks
P13	😐	Never previously used before as a use case study. But have a potential community interest and provided with its latest libraries	😐	Never previously used before as a use case study. But have a potential community interest and provided with its latest libraries

at section III. This model will also being used in assessing both architecture based on its architecture's assessments. Use case diagram is being used to model both architectures as shown on Fig. 5. for the former x-company's architecture, and the recent microservices architecture.

As we already told a glimpse of the problem that occurred in this company's web app architecture previously in problem identification at section III, now we will describe more specifically in this section. There was a company that would use the web app as a medium for them to interact and exchange its data and information across divisions internally. This web application was being used by employees to view, retrieve, and convert the data to be reprocessed into new data which will be redistributed to other divisions. Currently, several divisions were using this web application. The web application has its same pattern and function, there were around three or more web applications in each of two divisions that were being used by the employees. Employees required to log in and access each different web application to be able to transfer the data and information. From this occasion, we as researchers concluded that there were divisions that necessitates their employees to use the web app as their main tools to exchange data and information. And there were around one to two web applications for each division. Employees required to access both web applications to transfer data. Therefore, we will build an architecture, especially microservices-based architecture which will be operated on these divisions to overcome the limitation that impedes employee productivity. Microservices aims to carry out activities in accordance with the needs of their respective function and divisions.

5. RESULT

In order to assess whether each benchmark candidate would satisfy each of our proposed parameter, we have preliminarily examined each application based on their provided documentation from their respective software official sites. We also, done some iteration and measures the average test results on some of the parameter to gain overall test results with the intention for a better assessment result. The assessment result is summarized in Table. 2.

According to Table. III, both architectures are not fully satisfied all of our proposed parameters. After some iteration in conducting assessments for both applications, the overall results that we have found stated that microservices tends to have a slower accessing speed than a monolithic based application. These rather contrary results might be caused by our simple task application, not a heavy workload application that runs maybe handle hundreds of services and tons of traffics at a time. Our assessments on both architecture models are visible because we model and develop both architectures in complementary of this study. And both architecture models also had been described in detail in section IV.

Both architectures have their design pattern uniqueness for the pattern-based design assessments. With x-company's architecture based on three different services each have three similar web application provided with their own database. This company's design pattern has a similar design pattern with an architecture that consists of 3 monolithic architecture. And recent microservices architecture for each three different services, it was structured upon four different independent web application that work simultaneously. With one web application that runs client-side to provide visibility or front-end for the end-user to interact with and three web application that runs server-side or back-end with access to a database which then will be sent to the client-side to be displayed. Each of these three server-side is also provided with three different databases based on each service.

Both web application is not available to be accessed publicly, considering one of the web applications was developed for a private company that operates internally and contains confidential information. Thus, this confidentiality was also the main reason why we named the architecture as an "x-company's architecture". And for the microservices architecture, we afraid we also cannot and will not allow the application to be publicly available. Because, it has a similar data pattern and attributes for each of its *Create, Read, Update, and Delete* (CRUD) functions that leads to the company's confidential information. For its continuous integration, both architectures were not provided with continuous integration tools. Nonetheless, there were plenty of CI tools out there that would support this application such as TravisCI [21], GitHub Actions [22], and Jenkins [23]. These supported continuous integration tools will enable developer to collaborate as a team to continuously develop the application. Both applications were not provided with automated testing tools. Yet supported automated testing tools such as Selenium [24] and PHP Unit [25] can be applied to this application to perform special automated testing for each test case unit. These tools can be considered as investments of money and resources in the development process for its versatility. From the dependency management perspective, both applications were developed using a common web application language namely PHP. Both applications were also developed using a PHP-written web framework namely CodeIgniter [26] This chosen framework supports both the company's and microservices architecture and provides us, developers, with a standard way to build a dynamic web application, thus increase developer productivity. Then based on its Reusable Container Images assessments, there were no container images included in both applications. However, platform such as docker was supported with updated official container images for PHP application [27]. Unfortunately, automated deployment was also not included for both applications. But, tools such as Jenkins [23] which are also mentioned as supported Continuous Integration tools for both applications are also provided with automated build, tests, and deployments. Other automated deployment tools like Kubernetes [28] are also available and famously known as an agnostic-language of microservices. In this study, we have not yet developed the web application using either container or orchestration. Thus, orchestration is not implemented in both applications because orchestration tools require an embedded container in it. Which there were not any containers applied on both applications. Even though, orchestration tools such as Kubernetes are available yet suitable to be implemented for a microservices-based application [28].

After we developed both applications, we held some test based on user perspective while they were using both applications. Our test results show that for the previous x-company's application, around 6 steps were required for them to finally access data from 2 different services. While in microservices architecture, each user only required to do around 4 steps in order for them to access data from 2 different services. From both performance perspective, unfortunately microservices was falling behind with an average of 550ms needed while on going home site is loading after the user had already login. While on the company's architecture, it only needed around 415ms for the home page to fully loaded after the user had already login to the web application. Being a self-developed application, as we had also mentioned before, it was built upon PHP language. Using PHP-written web framework namely CodeIgniter. We use the same framework to develop both applications to avoid bias on the benchmark results. By publicly restricted and only developed in complementary to this study, thus this application has never been used before as a study case.

6. LIMITATION

Just like any other research, this research has a number of limitations. One of those obvious limitations are both applications are self-developed, with developers that have a least amount of experience in developing microservices architecture. Other limitation are both of the applications are relatively simple, thus it does not represent features of microservices on its full potential and scale. This rather qualitative study might not cover aspects of microservices that are actually more crucial to be assessed.

7. CONCLUSION

This conducted study mainly offer an initial set of benchmark parameter in comparing two different web application to be used as an empirical software architecture research as well as implementing microservices architecture. Both assessed applications is provided with an illustrated architecture models and also developed using identical language and framework to provide either equal comparison as well as consistent test results. Discussed parameter for the assessments on this study was ranged across architecture, DevOps, and other general requirements related. Our early results indicate that microservices application for a small-scale service application, tends to run slower than the company's architecture. This rather contrary result happens caused by our lack experience in developing a proper microservices architecture and a small scale testing that might not represents microservices in its full potential.

8. FUTURE WORK

Above all, we hope our study can be useful for the community, research, and practitioners that saw this architecture trends as a potential future. As what we also did with [20], by trying to implement their assessments. Thus, also pass the baton from previous study as a start point for the discussion of what constitutes an 'ideal' benchmark criterion for an empirical microservices research.

REFERENCES:

[1] J. Thönes, “Microservices,” *IEEE Softw.*, vol. 32, no. 1, 2015, doi: 10.1109/MS.2015.11.

[2] M. Fowler and J. Lewis, “Microservice,” 2014. https://martinfowler.com/articles/microservices.html#:~:text=In short%2C the microservice architectural,often an HTTP resource API.

[3] S. Baškarada, V. Nguyen, and A. Koronios, “Architecting Microservices: Practical Opportunities and Challenges,” *J. Comput. Inf. Syst.*, vol. 60, no. 5, pp. 428–436, 2020, doi: 10.1080/08874417.2018.1520056.

[4] S. Newman, *Building Microservices*. O’Reilly Media, 2015.

[5] M. S. Hamzehloui, S. Sahibuddin, and A. Ashabi, “A study on the most prominent areas of research in microservices,” *Int. J. Mach. Learn. Comput.*, vol. 9, no. 2, pp. 242–247, 2019, doi: 10.18178/ijmlc.2019.9.2.793.

[6] H. Vural, M. Koyuncu, and S. Guney, “A systematic literature review on microservices,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 10409 LNCS, no. July, pp. 203–217, 2017, doi: 10.1007/978-3-319-62407-5_14.

[7] D. Taibi, V. Lenarduzzi, and C. Pahl, “Architectural patterns for microservices: A systematic mapping study,” *CLOSER 2018 - Proc. 8th Int. Conf. Cloud Comput. Serv. Sci.*, vol. 2018-Janua, no. Closer 2018, pp. 221–232, 2018, doi: 10.5220/0006798302210232.

[8] L. De Lauretis, “From monolithic architecture to microservices architecture,” *Proc. - 2019 IEEE 30th Int. Symp. Softw. Reliab. Eng. Work. ISSREW 2019*, pp. 93–96, 2019, doi: 10.1109/ISSREW.2019.00050.

[9] G. Liu, B. Huang, Z. Liang, M. Qin, H.

Zhou, and Z. Li, "Microservices: architecture, container, and challenges," *2020 IEEE 20th Int. Conf. Softw. Qual. Reliab. Secur. Companion Microservices*, pp. 629–635, 2020, doi: 10.1109/qrs-c51114.2020.00107.

[10] M. Fowler, "Bounded Context," 2012. http://martinfowler.com/bliki/BoundedContext.html (accessed Jan. 19, 2021).

[11] R. H. Steinegger, P. Giessler, B. Hippchen, and S. Abeck, "Overview of a Domain-Driven Design Approach to Build Microservice-Based Applications," *Third Int. Conf. Adv. Trends Softw. Eng. (SOFTENG 2017)*, no. April, pp. 79–87, 2017, [Online]. Available: https://www.thinkmind.org/index.php?view=article&articleid=softeng_2017_4_30_64138%0Ahttps://www.researchgate.net/publication/316492773_Overview_of_a_Domain-Driven_Design_Approach_to_Build_Microservice-Based_Applications.

[12] N. Dragoni *et al.*, "Microservices: Yesterday, today, and tomorrow," *Present Ulterior Softw. Eng.*, no. June, pp. 195–216, 2017, doi: 10.1007/978-3-319-67425-4_12.

[13] M. Cardarelli, A. Di Salle, L. Iovino, I. Malavolta, P. Di Francesco, and P. Lago, "An extensible data-driven approach for evaluating the quality of microservice architectures," *Proc. ACM Symp. Appl. Comput.*, vol. Part F1477, pp. 1225–1234, 2019, doi: 10.1145/3297280.3297400.

[14] C. Richardson, "Microservice Architecture pattern," 2018. https://microservices.io/patterns/microservices.html (accessed Jan. 20, 2021).

[15] T. Hunter II, *Advanced Microservices*. 2017.

[16] D. Bermbach, E. Wittern, and S. Tai, *Cloud service benchmarking: Measuring quality of cloud services from a client perspective*. Cham: Springer International Publishing, 2017.

[17] M. Grambow, L. Meusel, E. Wittern, and D. Bermbach, "Benchmarking microservice performance: A pattern-based approach," in *Proceedings of the ACM Symposium on Applied Computing*, Mar. 2020, pp. 232–241, doi: 10.1145/3341105.3373875.

[18] A. Avritzer, V. Ferme, A. Janes, B. Russo, H. Schulz, and A. Van Hoorn, "A quantitative approach for the assessment of microservice architecture deployment alternatives using automated performance testing," 2018, Accessed: Jan. 21, 2021. [Online]. Available: https://www.docker.com/.

[19] M. Grambow, E. Wittern, and D. Bermbach, "Benchmarking the performance of microservice applications," *ACM SIGAPP Appl. Comput. Rev.*, vol. 20, no. 3, pp. 20–34, 2020, doi: 10.1145/3429204.3429206.

[20] C. M. Aderaldo, N. C. Mendonça, C. Pahl, and P. Jamshidi, "Benchmark Requirements for Microservices Architecture Research," *Proc. - 2017 IEEE/ACM 1st Int. Work. Establ. Community-Wide Infrastruct. Archit. Softw. Eng. ECASE 2017*, no. November, pp. 8–13, 2017, doi: 10.1109/ECASE.2017.4.

[21] "TravisCI," *TravisCI*. https://travis-ci.com/.

[22] "GitHub Action," *GitHub*. https://docs.github.com/en/actions.

[23] "Jenkins," *Jenkins*. https://www.jenkins.io/.

[24] "Selenium," *Selenium*. https://www.selenium.dev/.

[25] "PHP Unit," *phpunit.de*. https://phpunit.de/.

[26] "CodeIgniter," *codeigniter*. https://www.codeigniter.com/.

[27] "Docker," *docker*. https://www.docker.com/.

[28] "Kubernetes," *kubernetes*.