International Journal of Computer Information Systems and Industrial Management Applications.
ISSN 2150-7988 Volume 16 (2024) pp. 001-009

Microservices-based Architecture to Improve the Enrollment Process of State Schools

Jashir R. Chirre Escate[1], Giorgio G. Gamarra Gómez[2], H. David Calderon-Vilca[3] and Flor C. Cárdenas-Mariño[4]

[1] Software Engineering Department, Universidad Nacional Mayor de San Marcos,
Carlos German Amezaga s/n, Lima 01, Perú
jashir.chirre@unmsm.edu.pe

[2] Software Engineering Department, Universidad Nacional Mayor de San Marcos,
Lima, Carlos Germán Amezaga #375, Perú
12200189@unmsm.edu.pe

[3] Software Engineering Department, Universidad Nacional Mayor de San Marcos,
Lima, Carlos Germán Amezaga #375, Perú
hcalderonv@unmsm.edu.pe

[4] Operations Research Department, Universidad Nacional Mayor de San Marcos,
Lima, Carlos Germán Amezaga #375, Perú
fcardenasm@unmsm.edu.pe

***Abstract*: This research proposes a microservices-based architecture to improve the enrollment process of state schools in times of pandemic. Microservices allow an application to support higher request concurrency and fault tolerance in a scenario where many users require to perform a web-based enrollment process. The development process and the tools used for the proposed architecture are also detailed. Through 4 types of performance tests, it is demonstrated that the architecture achieved an average efficiency of 177% compared to a traditional monolithic architecture. Therefore, the study concludes that the implementation of a microservices architecture for the enrollment process of state schools is reliable, efficient and functional.**

***Keywords*:** microservices, enrollment, education, system, platform, distance learning system**.**

I. Introduction

Before the pandemic, some schools carried out the enrollment process manually and in person. The student and his parents approached the school to provide their information to the school management area, the secretary is in charge of obtaining the data provided by the parents and the student, she is also in charge of receiving the payment and finally she is in charge of store all the documentation delivered.

This face-to-face form of enrollment generates problems when there are a large number of people to enroll. In this situation, the time it takes to complete this process increases considerably, which generates long queues outside the school and great dissatisfaction on the part of the students. the parents. One of the reasons why the time to carry out the registration process increases is because there is only one administrative staff in charge of this process.

In Peru, since the year 2020 due to the global pandemic, classes and educational management became impossible to perform in person, so in order not to jeopardize the national education, the use of electronic devices and digital media were prioritized as a solution for educational and logistical processes of national schools [1].

In all countries, after declaring a State of Emergency due to the COVID-19 crisis, governments have implemented a series of strict measures to deal with this situation. Part of these measures includes the closure of schools and higher education centers, both public and private, as well as all World Cultural and Natural Heritage sites.

The emergency generated by the pandemic highlighted the need to strengthen the science, technology and innovation system, which is key to providing a knowledge base for the government's response in all areas, from diagnostic analyzes to the social organization of confinement.

Within this context, educational modernization is speeded up by the rapid development of the Internet, resulting in online education gradually becoming a new educational model [2].

Educational institutions have chosen to transfer face-to-face education to virtual education, as a consequence the use of different platforms has increased enormously, student enrollment platforms allow socioeconomic and academic data to be recorded, which at the time of consultations and lists they can take longer due to the number of records, so it is

necessary to design new platforms with modern technologies to speed up computational processes and transactions.

Currently, there are classic technological systems that help educational management, but these systems struggle to maintain normal business operations with their traditional monolithic architectures, which are being overtaken by current requirements [3].

As higher education has expanded rapidly in the past decade, many universities' information systems have struggled to maintain normal business operations with their traditional monolithic architecture and service-oriented architecture, leading to implementation problems. of the system, difficulties in the implementation of the system, low levels of utilization of resources, service obligation.

Microservices are considered as an approach to develop a single application as a set of small services, each running its own process independently, communicating with each other with lightweight mechanisms, and can be written in different programming languages and use different data storage technologies. data for better performance.

The concept of microservices architecture has recently emerged driven by the industry and is gaining increasing popularity. The approach refers to a way of designing software as a set of small independent services that cooperate with each other using lightweight communication protocols. Services are built and deployed individually, and are usually supported by continuous integration and deployment processes, the concepts of which are explained in later sections.

Therefore, through research, several proposals for educational and management platforms based on a microservices and cloud architecture have been developed as a response to the challenges faced by education in these times of pandemic and the need to digitalize the classic processes of teaching and management [4].

Proposals such as [3], present an Android platform developed with a microservices architecture hosted in the cloud based on the Node.js framework for the creation of REST APIs. This application allows students to develop tasks, teachers to assign them and parents to obtain performance reports, all this with the possibility of adding new features thanks to the modularization of microservices which are developed in several layers to efficiently handle large JSON files. In this proposal we find that they cover teaching processes but do not focus on logistics management such as the enrollment process.

Studies such as [8] propose the design of a public services platform for university management based on microservices architecture, with special attention to the problem of load balancing and its optimization. In this research they use microservices with Spring Cloud and devops. The tests were performed on the Linux operating system and Docker with Swarm 17.03.0 edition where 4 clusters were configured and it was evidenced that the load of each node in the cluster becomes much more balanced, optimizing the performance of each node and improving the overall rate of resource utilization of all clusters, with better load balancing of the cluster. This proposal contains among its modules the enrollment process but using a tool like Docker for containers limits the concurrency support capacity to less than 50 TB per second.

In the case of [9], they proposed an educational portal based on a microservices architecture where the architecture allowed them to make performance comparisons between modules developed in .Net, PhP and Java, concluding that the architecture allowed them to select languages and frameworks for specific tasks, thus significantly optimizing performance compared to a monolithic platform based on a single programming language. Again, we found that this educational portal is focused on the learning processes during student classes, leaving aside logistical management processes such as the enrollment process to one side.

Within the same context [5], they had the proposal of a migration methodology from a monolithic system based on a SOA (service-oriented architecture) architecture pattern that did not allow a separation of independent responsibilities to a microservices architecture where they used Docker containers and the HTTP protocol, finding performance and scalability improvements in the new system. In this research we found as in the other proposals the option of using Docker as a module container, which is a valid approach for non-massive processes, if we wanted support for larger scale concurrency, we would need another type of tool such as Kubernetes.

It was found that the proposals are focused on processes that do not have concurrency problems or a significant number of service requests. In addition, the research was conducted for learning processes but not in logistic processes such as enrollment, which in times of start of class needs to support a massive amount of requests.

Therefore, in this research we propose an architecture based on a microservices architecture to optimize the enrollment process of state educational schools.

In section II we review the state of the art, in section III the architecture design methodology and section IV results and discussion.

II. State of Art

A. Microservices-based platforms oriented to education.

Studies such as [14], propose a software for learning programming languages hosted in the cloud based on a modular microservices architecture using JACK, a framework for modular classification which enables the addition of new courses, and ARTEMIs for automated assessment management for interactive learning, which makes it a scalable solution for learning in this specific area.

On the other hand another proposal with the same objective but taken a little further is the one by [10] presenting a platform where they focus on education and learning of various programming languages using GitPod which is a

cloud service for the configuration of different development environments allowing scalability, but in turn they add the data science research modules using JupyterHub which is a server that allows simultaneous access of multiple users to their notebooks, this for students providing them with modeling tools and big data analysis.

On the other hand, [3] has as a problem of distance education of primary and secondary levels and aims to improve the experience of learning and fulfill tasks in a didactic way. For this reason, the aforementioned research proposes an Android application using Node.js as Backend framework for the creation of REST services based on microservices, modularizing services in Docker containers hosted in the cloud, this application has modules to control the teaching of students and test management.

A proposal that addresses the same problem but for universities is the one by [6], which presents a platform both web and mobile, using Spring boot Java for the creation of Rest services and JVM as a module container.

B. Design and techniques for the development of microservices

[12] in their proposal design the AWS cloud microservices deployment process because it provides a flexible environment and proposes the use of containers using Kubernetes because compared to Docker, Kubernetes is designed to run on a cluster, while Docker runs on a single node.

It also includes Debian image building, various optimization schemes to reduce image size, container orchestration and other activities, which further simplify the deployment process. Separate research [13] and [19] agree on the idea that architecture design in microservices should be performed in cloud containers, in this case Azure is used since unlike AWS, Azure does not lose performance when subjected to large volumes of data (500TB) and they used Docker the most used tool for creating cloud containers.

In the case of [15], it proposes some procedures to follow for the development of a microservices architecture, such as isolating errors using resiliency strategies to prevent errors within a service from cascading by applying resiliency patterns such as circuit breaker and bulkhead. Avoiding coupling between services since among the causes of coupling are rigid communication protocols and shared database schemas.

A similar approach is proposed by [16], who for error handling propose resistance patterns but in this case the Health Endpoint Monitoring pattern and the Retry pattern, all this to deal with unintentional errors. In the same context, [17] and [18] consider that to mitigate error cases an application load balancer should be implemented which balances the load of HTTP and HTTPS traffic at the application layer and allows routing connections to microservices based on IP protocol data. It ensures that a network load balancer can handle millions of requests per second while maintaining very low latencies.

C. Tests and metrics performed on microservices-based platforms

For learning platforms, the results of the proposal of [11] were based on performance tests using the JMeter tool obtaining stress test metrics, load test metrics ,compile time metrics, response time and resource utilization tests, hosting the architecture on a Linux Ubuntu 16 server, intel core i7, 32gb RAM DDR4, 2TB SSD, comparing the proposal with a traditional monolith learning system hosted on a Linux Ubuntu server, intel core i5, 32gb RAM DDR4, 1TB SSD, concluding that the proposed architecture is on average 253% more efficient than the traditional monolith architecture of a classical platform.

On the other hand, [10] and [20] used Sentry as a performance monitoring and error tracking tool, in which the proposals based on microservices presented a medium level of availability by generating failures in correlative module requests, preventing the application from becoming inoperable, while a traditional monolith architecture resulted in a null level of availability by generating failures in HTTP services.

In contrast to the type of metrics in the research of [3], the results of Fenglong's proposal were based on user surveys targeting parents and students, with the results showing that students and parents were in a large percentage to take the education from home and not face-to-face as the platform was reliable enough.

III. Architecture design methodology

To design the architecture based on microservices we have considered the tools: a Registry to annotate and discover the microservices and the tool we use is Eureka Server; Additionally, we have used a Load Balancer to load balance the microservices and the tool we selected is Ribbon; To manage the fault tolerance of the microservices we used a Circuit Braker as a tool, we chose Hystrix, an API gateway was also necessary for a single point of access for the microservices and finally a Log center to save the logs.

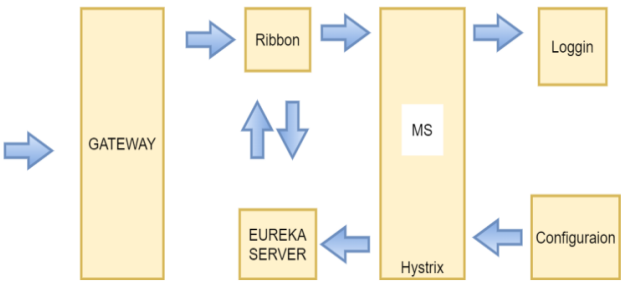

Figure 1. Solution architecture

Proposed architecture

The web application is implemented in 2 parts, the first part being the Frontend implementation which is implemented with Vue.js and React.

The second part corresponds to the Backend implementation, which is implemented in the spring boot framework, using the Java programming language in its version 8.0. The spring boot framework allows using the whole model of microservices oriented architecture together with spring cloud that allows giving the necessary tools to execute the microservices correctly. Among the tools we used Registry to register and discover the microservices, for our case we used Eureka Server.

We also implemented a Config Server that allows to register a centralized configuration of the microservices. Additionally, a Load Balancer was configured to make a load balancing of the microservices and the most appropriate tool that we selected is Ribbon.

To manage the fault tolerance of the microservices, a Circuit Braker was implemented, which in our case we will use Hystrix; an API gateway was also needed for a single point of access of the microservices and finally a Log center to store the logs in a centralized point. Each microservice implements a design pattern of N layers, in addition to exposing APIs that are responsible for transmitting the information in Json format to the Front end of the application.

The means of communication between the microservices is via HTTP protocol using Restful Web services with POST, GET, PUT and DELETE verbs, in addition to XML serialization.

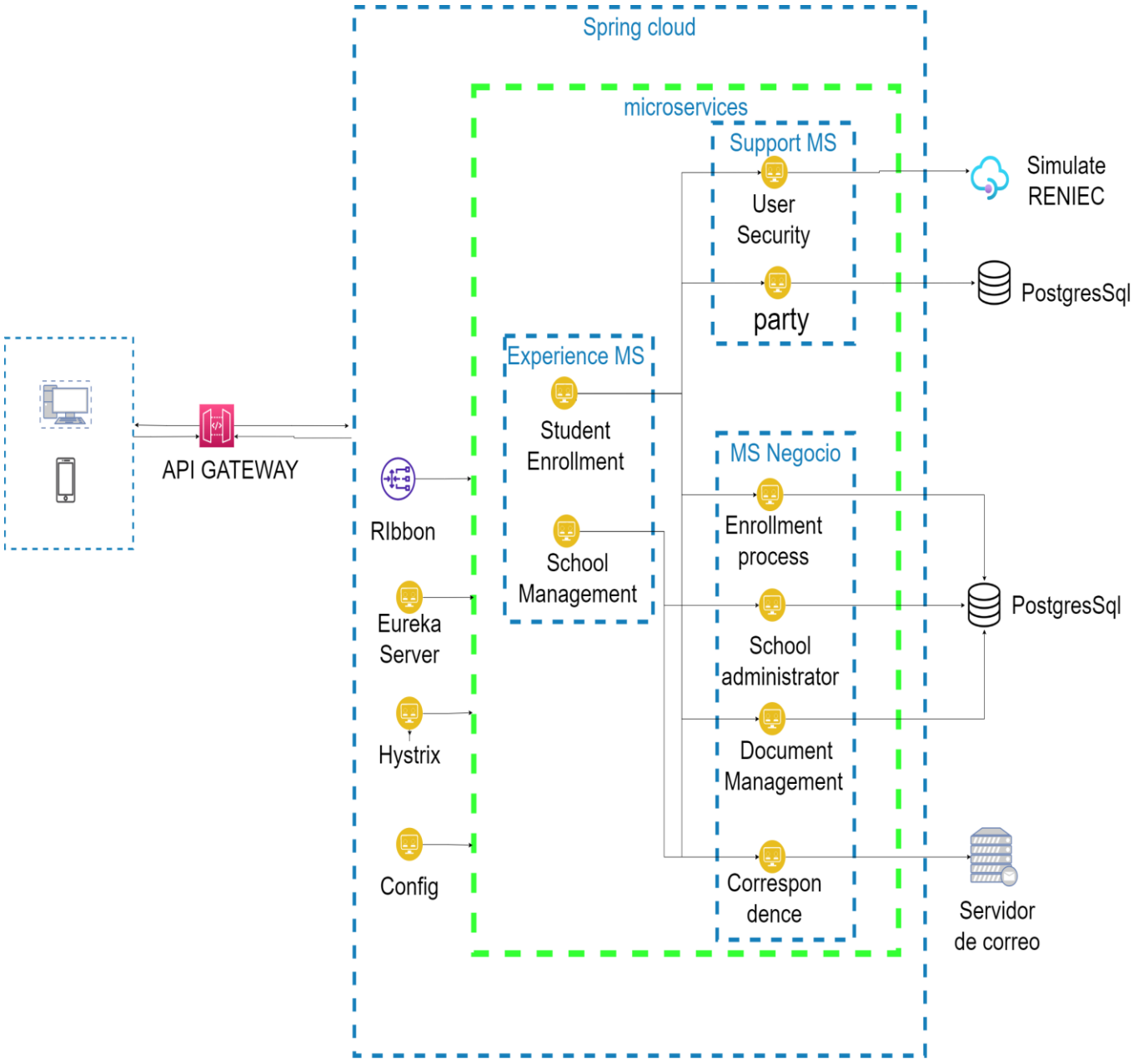

Figure 1. Solution architecture

For data storage, the relational database PostgresSql was used, which can store the data that would be the core of the application, also a function was added for the storage of shared documents or images that corresponds to scans of documents or photos if necessary.

The architecture of the solution based on microservices was divided into 3 components, the first being an experience microservice, the second a support microservice and finally a business microservice.

Experience microservices

The experience microservices are the microservices in charge of the orchestration of services, that is to say, they are in charge of consuming other services that elaborate an independent task in such a way that it generates a correct flow of service consumption to complete the objective of the system.

These microservices cannot connect to a database and if required, a support microservice is used to achieve it. The first one is called "student enrollment", this microservice is in constant communication with the front end application and is in charge of receiving HTTP requests and delivering the responses in Json format to these requests, besides performing the orchestration of the microservices that interact with the enrollment process of the students with their respective study center, and also the orchestration of the microservices that interact with the enrollment process of the students with their respective study center, and also the orchestration of the microservices that interact with the enrollment process of the students with their respective study center.

This microservice is the one related to the Fronted enrollment module. The second business microservice is "School Management" which, like the previous microservice, is in charge of orchestrating the microservices in charge of the CRUD processes of the schools.

Support microservices

The support microservices are those in charge of performing support processes, mostly to lighten the processes corresponding to the business objective, such as database connections or the consumption of a business microservice. For the proposed architecture we have 2 support microservices, the first microservice is called "User security" in charge of user validation by connecting to the Registro Nacional de Identificación y Estado Civil (RENIEC) to verify the existence of the person.

When the microservice of experience consumes this service, it provides the data obtained from RENIEC, the obtaining of these is by means of the consumption of a SOAP API that gives us an answer in XML format and which was transformed to Json format which is the standard format in which we return the answers through our REST APIs.

The second support microservice is "Party Data", this microservice is responsible for storing in the database the data that has been validated by RENIEC. The reason for this action is to reduce the number of calls to RENIEC services, in addition to reducing the cost of using this service.

Business microservices

The business microservices are those in charge of connecting to databases, SOAP or REST connections with providers or communicating with some core program that the business is using. For the proposed architecture we have 4 business microservices, the first one being the microservice called "Enrollment Process" which is responsible for containing all the vital business logic, i.e., in this microservice is concentrated all the information relating to the enrollment of students and the respective rules and validations.

This microservice receives all the information obtained from the Front end, which is transformed by the "student enrollment" experience microservice so that it can be processed in this business microservice, in addition to a connection to databases to store the student enrollment records in their respective study centers.

The second microservice is called "School administrator", this microservice has the logic to maintain the schools throughout the life cycle of the schools, that is, it is a CRUD that is responsible for adding, updating and deleting schools in the database. This microservice is consumed by the experience microservice "School Management" which receives and transforms the data obtained from the Front end in order to use this service.

The third microservice called "Document Services" which is responsible for document storage, these documents were stored in pdf and also supports image storage to store scans of documentation that cannot be digitized.

The fourth microservice is called "Correspondence", this microservice is using the JAVA mailing library and connecting to a mail server to send a notification of successful enrollment to the mail used during the enrollment process.

Storage (PostgreSql)

For the web service a database manager was used, the one we selected for this architecture proposal was Postgresql in its version 14. The database manager "Postgresql" was used to help build the storage of security information, user data, system enrollment information, CRUD information from the schools, etc.

Additionally, we opted for the strategy of using 2 databases, one exclusively for the storage of business information, and another multichannel one that can be shared by several microservices in order to be scalable over time.

Load balancer (Ribbon)

In a distributed environment, services need to communicate with each other. The communication can either happen synchronously or asynchronously. Now, when a service communicates synchronously, it is better for those services to load balance the request among workers so that a single worker does not get overwhelmed.

Load balancing is the process of distributing traffic among different instances of the same application. To create a

fault-tolerant system, it's common to run multiple instances of each application. Thus, whenever one service needs to communicate with another, it needs to pick a particular instance to send its request. There are many algorithms when it comes to load balancing:

- Random selection: Choosing an instance randomly
- Round-robin: Choosing an instance in the same order each time
- Least connections: Choosing the instance with the fewest current connections
- Weighted metric: Using a weighted metric to choose the best instance (for example, CPU or memory usage)
- IP hash: Using the hash of the client IP to map to an instance

Spring Cloud Ribbon is a library that allows communication between different processes whose main feature is to provide different algorithms to perform client-side load balancing. In addition to client-side load balancing, Ribbon provides other useful functions such as the following:

- Integration with Eureka Server: this integration allows obtaining information from the services registered in Eureka and performing load balancing between them.
- Fault tolerance: with Ribbon it is possible to determine which services are up or down dynamically and thus act accordingly to guarantee the service.
- Load balancing strategies: with Ribbon you can set up standard and custom load strategies by providing several strategies for load balancing such as RoundRobin Rule, Availability Filtering Rule or Weighted Response Time Rule. It also allows you to adjust the strategy according to particular needs.

Circuit Breaker (Hystrix)

The purpose of the Circuit Breaker pattern is different than the Retry pattern. The Retry pattern enables an application to retry an operation in the expectation that it'll succeed. The Circuit Breaker pattern prevents an application from performing an operation that is likely to fail. An application can combine these two patterns by using the Retry pattern to invoke an operation through a circuit breaker. However, the retry logic should be sensitive to any exceptions returned by the circuit breaker and abandon retry attempts if the circuit breaker indicates that a fault is not transient.

A circuit breaker acts as a proxy for operations that might fail. The proxy should monitor the number of recent failures that have occurred, and use this information to decide whether to allow the operation to proceed, or simply return an exception immediately.

Hystrix is a library that implements the Circuit Breaker pattern. Hystrix allows us to manage interactions between services in distributed systems by adding latency and fault tolerance logic. Its purpose is to improve the overall reliability of the system, for this Hystrix isolates the access points of the microservices, thus preventing cascading failures through the different components of the application, providing fallback alternatives, managing timeouts, thread pools, etc.

API Gateway

Zuul is an edge service that allows dynamic routing, load balancing, monitoring and request securitization. For practical purposes Zuul is a server composed of filters, each of which is focused on a specific functionality. Zuul is configured as the entry point to the microservices ecosystem and was in charge of routing, balancing and securing the requests received by the microservices.

Each request sent to Zuul passed through the filters that compose it, which depending on the characteristics of the request can, for example reject, it for security reasons, register it for monitoring purposes, route it to a certain instance of a microservice according to the configured filters.

By default, Zuul uses Ribbon to locate, through Eureka, the microservice instances to which it will route the requests to be executed within a "Hystrix Command", thus integrating all the components of the architecture and taking advantage of all the benefits provided by the spring cloud ecosystem.

Config Server

Eureka is a service that allows other microservices to register in its directory. When a microservice registered in Eureka starts, it sends a message to Eureka indicating that it is available. The Eureka server will store the information of all registered microservices, as well as their status. Communication between each microservice and the Eureka server is done via heartbeats every "X" seconds. If Eureka does not receive a heartbeat of a given type after 3 intervals, the microservice is removed from the registry.

In addition to keeping track of active microservices, Eureka also offers other microservices the possibility to "discover" and access other registered microservices. Therefore, Eureka is considered a microservice discovery and registration service.

Tracing

Zipkin was originally developed at Twitter, based on a concept of a Google paper that described Google's internally-built distributed app debugger – dapper. It manages both the collection and lookup of this data. To use Zipkin, applications are instrumented to report timing data to it.

Zipkin is a very efficient tool for distributed tracing in the microservices ecosystem. Distributed tracing, in general, is the latency measurement of each component in a distributed transaction where multiple microservices are invoked to serve a single business usecase.

Distributed tracing is useful during debugging when lots of underlying systems are involved and the application becomes slow in any particular situation. In such cases, we first need to identify which underlying service is actually slow. Once the slow service is identified, we can work to fix that issue. Distributed tracing helps in identifying that slow component in the ecosystem.

Internally it has 4 modules:

- **Collector** – Once any component sends the trace data, it arrives to Zipkin collector daemon. Here the trace data is validated, stored, and indexed for lookups by the Zipkin collector.

- **Storage** – This module store and index the lookup data in backend. Cassandra, ElasticSearch and MySQL are supported.
- **Search** – This module provides a simple JSON API for finding and retrieving traces stored in backend. The primary consumer of this API is the Web UI.
- **Web UI** – A very nice UI interface for viewing traces.

To improve tracing we integrated Zipkin with Sleuth to improve the tracing. Sleuth is another tool from the Spring cloud family. It is used to generate the trace id, span id and add this information to the service calls in the headers and MDC, so that It can be used by tools like Zipkin and ELK etc. to store, index and process log files.

As it is from the spring cloud family, once added to the CLASSPATH, it automatically integrated to the common communication channels like:

- requests made with the RestTemplate.
- requests that pass through a Netflix Zuul microproxy.
- HTTP headers received at Spring MVC controllers.
- requests over messaging technologies like Apache Kafka or RabbitMQ etc.

IV. Results and Discussion

The proposed microservices-based architecture has been subjected to 4 types of performance tests, such as load, compilation, response time and resource utilization tests, using the JMeter tool which allowed us to perform the tests. The tests were performed by comparing it with an enrolling system developed with a monolithic architecture.

The tests were performed on a computer with a third generation Intel Core I3 processor with 2.63 Ghz frequency, with a capacity of 8Gb of DDR2 ram memory, hard disk with 20Gb of space that was destined for the database. At the same time, it should be noted that the same equipment worked as a server, database manager, and e-mail server.

For the stress tests given the low performance of the hardware in possession was limited to tests of 10 and 20 samples because when using more samples, the equipment reached its maximum operating point and produced errors due to lack of memory and among others.

Result tables

Table 1. Compilation comparison

Architecture	Best result (seconds)	Average result (seconds)
Monolith	30	35
Microservices	12	16

It was found that, for the compile time comparison, the proposal was found to be 218% faster compared to a traditional monolithic system.

Table 2. Comparison of the response time of 10 requests.

Architecture	Best result (seconds)	Average result (seconds)
Monolith	7.731	8,671
Microservices	4.93	4.875

On the other hand, the response time of 10 concurrent requests resulted in a 177% faster response time for the microservices architecture proposal.

Table 3. Comparison of load test of 20 requests.

Architecture	Best result (seconds)	Average result (seconds)
Monolith	13,709	15,799
Microservices	6,062	10,037

As for the load test by sending 20 simultaneous requests, the proposal showed better performance by performing the task 156% faster than the monolith architecture.

Table 4. Comparison in memory usage

Architecture	Best result	Average result
Monolith	-	92%
Microservices	-	96%

Finally, in the resource utilization test, it was shown that the proposal requires 96% of cpu usage, which is higher than the monolith architecture, because the microservices architecture requires greater complexity and more resources for its operation.

Discussion

The results achieved are similar to the results of the proposal of the researcher [11], who obtained with the same JMeter tool comparing a monolith system and his proposal with microservices reaching an average performance level of more than 200% which exceeds our proposal which obtained an average of 180%. However, it should be noted that the researcher used the high-performance server of AWS (Amazon Web Services) which allowed him to obtain the exposed result, finally it should be highlighted that his research does not focus on enrollment processes but on learning processes.

Regarding the research of [20] who present a platform both web and mobile, using Spring boot Java for the creation of Rest services and JVM as a module container. This did not contemplate a scenario of massive concurrency, in which our proposal was submitted, and we developed specific tools for this casuistry, where we implemented an orchestrator that manages the traffic of requests and if at some point the concurrency comes to indispose a service, our balancer was in charge of pointing to a new available server.

The proposals of [17] and [21] include a balancer as our proposal to ensure the availability of the service modules, where we agree that it is the best solution for massive loads in a given time.

Our proposed microservices-based architecture was compared with a monolithic system of enrollment and was submitted to 4 types of performance tests, such as load tests, compilation, response time and resource utilization tests, using the JMeter tool which allowed us to perform the tests. On the other hand, other researchers found new ways to test their research, as is the case of [8] and [22], who opted for user satisfaction surveys using Likert scale, measuring the results by percentages at the end.

Researchers [10] and [23] add to this type of tests using a satisfaction survey but in this case the satisfaction scale CSAT (Customer Satisfaction Score).

Research [7], [11], [24] and [25], propose a microservices architecture focused on education in learning processes such as interactive modules that facilitate student learning. While our microservice proposal is also focused on education, it also presents an architecture for management modules such as the enrollment process of schools which has to support a large concurrency in times of beginning of the school year and even more in a scenario of global pandemic as we are living in this 2021.

Conclusions

This research proposed an architecture based on microservices to improve the enrollment process of state schools. From the review of the state of the art it was found that the use of microservices is the most appropriate for the project since monolithic architectures do not have the availability, fault tolerance and performance that microservices do.

Having tested the architecture by submitting it to different performance tests, it was concluded that the architecture obtained an outstanding performance, and that it can be applied to solve the problem of massive concurrency in times of school enrollment. It is evident that there is still more work to be done in terms of development and user experience.

The next steps to be taken will be linked to the implementation of this microservices architecture approach to different processes that require a high level of availability in concurrency peaks, such as what happened in the middle of the pandemic.

References

[1] “Resolución Viceministerial N° 088-2020-MINEDU”. https://www.gob.pe/institucion/minedu/normas-legales/466186-088-2020-minedu (consultado nov. 13, 2021).

[2] H. Zhao, Y. Jiang, y X. Zhao, “Design and research of University intelligent education cloud platform based on Dubbo microservice framework”, en 2020 5th International Conference on Mechanical, Control and Computer Engineering (ICMCCE), Harbin, China, dic. 2020, pp. 870–874. doi: 10.1109/ICMCCE51767.2020.00191.

[3] Y. Fenglong, R. Changning, Z. Minghui, Y. Diankang, y W. Yujie, “An Android Learning Platform in Elementary and Secondary Education Based on Micro-Service Architecture”, p. 8.

[4] Y. Berkunskyi, K. Knyrik, T. Farionova, y T. Smykodub, “Using Microservices in Educational Applications of IT-Company”, p. 4, 2017.

[5] F. Auer, V. Lenarduzzi, M. Felderer, y D. Taibi, “From monolithic systems to Microservices: An assessment framework”, Inf. Softw. Technol., vol. 137, p. 106600, sep. 2021, doi: 10.1016/j.infsof.2021.106600.

[6] L. Huang, C. Zhang, y Z. Zeng, “Design of a public services platform for university management based on microservice architecture”, Microsyst. Technol., vol. 27, núm. 4, pp. 1693–1698, abr. 2021, doi: 10.1007/s00542-019-04474-4.

[7] J. Cao, “Design on Deployment of Microservices on Container-based Cloud Platform”, J. Phys. Conf. Ser., vol. 1624, p. 062008, oct. 2020, doi: 10.1088/1742-6596/1624/6/062008.

[8] A. Fellah y A. Bandi, “Microservice-based Architectures: An Evolutionary Software Development Model”, pp. 41–32. doi: 10.29007/1gx5.

[9] A. Bucchiarone et al., Eds., Microservices: Science and Engineering. Cham: Springer International Publishing, 2020. doi: 10.1007/978-3-030-31646-4.

[10] N. Chondamrongkul, J. Sun, y I. Warren, “Software Architectural Migration: An Automated Planning Approach”, ACM Trans. Softw. Eng. Methodol., vol. 30, núm. 4, pp. 1–35, jul. 2021, doi: 10.1145/3461011.

[11] N. C. Mendonca, C. Box, C. Manolache, y L. Ryan, “The Monolith Strikes Back: Why Istio Migrated From Microservices to a Monolithic Architecture”, IEEE Softw., vol. 38, núm. 5, pp. 17–22, sep. 2021, doi: 10.1109/MS.2021.3080335.

[12] Freight One, E. A. Zharkov, V. D. Malygin, y MOC IKT, “Intellectual Mathematical Support Software and Inner Architecture of LMS MAI CLASS.NET”, Bull. South Ural State Univ. Ser. Math. Model. Program. Comput. Softw., vol. 14, núm. 3, pp. 46–60, 2021, doi: 10.14529/mmp210304.

[13] J. Kostolny y J. Bohacik, “Development of an Education Information Portal with Microservices”, en 2019 14th International Conference on Advanced Technologies, Systems and Services in Telecommunications (TELSIKS), Nis, Serbia, oct. 2019, pp. 368–371. doi: 10.1109/TELSIKS46999.2019.9002184.

[16] K. Miao, J. Li, W. Hong, y M. Chen, “A Microservice-Based Big Data Analysis Platform for Online Educational Applications”, Sci. Program., vol. 2020, pp. 1–13, jun. 2020, doi: 10.1155/2020/6929750.

[17] T. Hinrichs y H. Burau, “A Scaleable Online Programming Platform for Software Engineering Education”, p. 8.

[18] S. Ghavifekr, A. Z. A. Razak, M. F. A. Ghani, N. Yan, Y. Meixi, y Z. Tengyue, “ICT Integration In Education: Incorporation for Teaching & Learning Improvement”, vol. 2, núm. 2, p. 22.

[19] M. Waseem, P. Liang, y M. Shahin, “A Systematic Mapping Study on Microservices Architecture in DevOps”, J. Syst. Softw., vol. 170, p. 110798, dic. 2020, doi: 10.1016/j.jss.2020.110798.

[20] M. A. Rahman, M. S. Abuludin, L. X. Yuan, Md. S. Islam, y A. T. Asyhari, “EduChain: CIA-Compliant Block-chain for Intelligent Cyber Defense of Microservices in Education Industry 4.0”, IEEE Trans. Ind. Inform., pp. 1–1, 2021, doi: 10.1109/TII.2021.3093475.
[21] V. Bushong et al., “On Microservice Analysis and Architecture Evolution: A Systematic Mapping Study”, Appl. Sci., vol. 11, núm. 17, p. 7856, ago. 2021, doi: 10.3390/app11177856.
[22] M. Moussa, A. Benachenhou, S. Belghit, A. Adda Benattia, y A. Boumehdi, “An Implementation of Microservices Based Architecture for Remote Laboratories”, en Cross Reality and Data Science in Engineering, vol. 1231, M. E. Auer y D. May, Eds. Cham: Springer International Publishing, 2021, pp. 154–161. doi: 10.1007/978-3-030-52575-0_12.
[23] J. Kostolny y J. Bohacik, “Development of an Education Information Portal with Microservices”, en 2019 14th International Conference on Advanced Technologies, Systems and Services in Telecommunications (TELSIKS), Nis, Serbia, oct. 2019, pp. 368–371. doi: 10.1109/TELSIKS46999.2019.9002184.
[24] “Microservices”, martinfowler.com. https://martinfowler.com/articles/microservices.html (consultado nov. 16, 2021).
[25] D. A. Bauer, B. Penz, J. Mäkiö, y M. Assaad, “Improvement of an Existing Microservices Architecture for an E-learning Platform in STEM Education”, p. 10, 2018.

Author Biographies

Jashir R. Chirre Escate Expert Backend developer, passionate about Microsservice Architecture, mathematics and affective computing. He developed with his fellow students a Microservice Architecture.

Giorgio G. Gamarra Gómez Expert in microservices, passionate about new technologies and mathematics. He developed with his fellow students a Microservice Architecture. He is currently looking to open new projects about cloud systems.

Flor Cagniy Cárdenas Mariño PhD in Computer Science, research professor of the "Optimización Matemática y Computacional" Group of the Universidad Nacional Mayor de San Marcos - Peru, advisor of undergraduate and graduate thesis projects related to artificial intelligence.