REVIVING LEGACY ENTERPRISE SYSTEMS WITH MICROSERVICE-BASED ARCHITECTURE WITHIN CLOUD ENVIRONMENTS

Safa Habibullah[1], Xiaodong Liu[1], Zhiyuan Tan[1], Yonghong Zhang[2], Qi Liu[2]

[1]School of Computing, Edinburgh Napier University, Edinburgh, UK
{s. habibullah, x.liu, z.tan}@napier.ac.uk
[2]School of Automation, Nanjing University of Information Science and Technology, China
{zyh, qi.liu}@nuist.edu.cn

ABSTRACT

Evolution has always been a challenge for enterprise computing systems. The microservice based architecture is a new design model which is rapidly becoming one of the most effective means to re-architect legacy enterprise systems and to reengineer them into new modern systems at a relatively low cost. This architectural style has evolved based on a number of different approaches and standards. However, there are quite a few technical challenges which emerge when adopting microservices to revive a legacy enterprise system.

In this paper, an evolution framework and a set of feature-driven microservices-oriented evolution rules have been proposed and applied to modernise legacy enterprise systems, with a special emphasis on analysing the implications as regards runtime performance, scalability, maintainability and testability. Testing and evaluation have been carried out in depth, aiming to provide a guidance for the evolution of legacy enterprise systems.

KEYWORDS

Microservice, Legacy System, Software Evolution, Cloud Environment

1. INTRODUCTION

The size of a monolithic system increases along with changes in business requirements over time and need for new functionality in a new iterative release. This makes continuous improvement and service delivery extremely difficult, as the entire system has to be tested and redeployed, even with a minor change in its code. Besides, it is difficult to scale a monolithic system when different services have conflicting resource requirements. To tackle these pitfalls of monolithic systems, which are designed as a unit of tightly coupled, entangled services, microservice architectures have been endorsed by software development communities in recent years.

As one of the recommended solutions to evolving legacy monolithic systems, microservice architectures decompose a monolithic system into small distinct services instead of deploying it as a single unit. This improves software development agility, making it easier to expand and scale up. This also results in faster deployment in cloud environments and allows systems to interoperate efficiently. Moreover, each microservice is responsible for a defined task, and all microservices can be linked together through a common Application Programming Interface (API).

However, nothing is perfect, so do microservice architectures. There are challenges in engaging such architectures in development and real-life deployment. These challenges must be addressed and their root causes must be well-studied. As such, a foundation for the successful refactoring of monolithic applications into microservice architectures will be founded.

This paper presents a progressive extension of our previous research [1] on evolving a legacy enterprise system using a microservice architecture. The key innovative contributions of this study include:

- Adopting feature-driven microservice-specific transformation rules to modernise a legacy enterprise system, and
- Evaluating a legacy monolithic architecture with a microservice architecture for industrial adoption with an emphasis on performance, maintainability, scalability and testability.

The rest of this paper is structured as follows. Section 2 presents an analysis of the related work. Section 3 presents the microservice-based evolution approach. A case study is described in Section 4. The implementation of the legacy system evolution and its deployment in the cloud environment is presented in Section 5 and Section 6, respectively. Section 7 presents a detailed description of the test and the analysis. Conclusions are drawn and future work is presented in Section 8.

2. RELATED WORK

Microservice represents the latest advance of the Service-Oriented Architecture towards higher system agility, better modularity and maintainability. The microservice-based architecture represents the latest approach to developing new applications or restructuring legacy systems. It is believed to be more effective as compared to the older service-oriented architecture in a number of ways: for instance, maintainability, reliability, scalability, and agility [2]. An in-depth study has therefore been carried out to discover the state-of-the-art as regards to microservices architectures. Several projects involving this architecture have been analysed from a spectrum of views.

2.1. Microservices Architecture and Design

The term 'microservice' indicates the use of an architectural style which is focused on system agility [3]. This approach unravels the challenges of a single large monolithic development through dividing the functions of the application into independent services which are small enough to be manageable. These services may be written in a number of diverse languages and also might use diverse data storage methods. There are, however, still a number of challenges in terms of fully comprehending this type of architecture. Balalaie et al. [4] recommended a set of patterns describing the kinds of introductory repositories which can be used for the microservice migration process. Diverse patterns were specified in relation to the decomposing of legacy systems, and the discovery of the recommended resolution was deliberated upon.

Brown & Woolf [5] attempt to show how microservices ought to be designed, the ways they can fit into a larger architectural picture, also the ways in which they can be built so that they operate efficiently. Their study deals particularly with matters relating to microservices efficiency, systems design, and microservices design. According to Taibi et al., the microservices architectural pattern can be determined via a catalogue [6]. In accordance with their systematic mapping study, they showed the drawbacks and benefits of each pattern, so that developers can select that the most appropriate for their purposes. For practitioners however, this process of identifying patterns or, indeed, not being able to identify a pattern, is unclear since the process has not actually been implemented.

In relation to the microservices architecture migration process, Carrasco et al. [7] introduced what he termed ‘bad smells’, i.e., situations which should trigger caution, with the purpose of assisting developers understand how to proceed with their designs and either deal with, or evade, these pitfalls.

In accordance with the survey that was carried out amongst professionals in Germany, Knoche et al. pointed out the major motivations which guide companies and developers towards adopting a microservice architecture [8]. One of these motivations was the desire for high elasticity and scalability. In addition, the lack of developers who have the necessary skills was highlighted as one of the major obstacles to the adoption of microservices. That survey only covered Germany, so we cannot generalise it to other parts of the world.

In [9], a different method discussed on how to portion a microservice. Then, a Domain Driven Design (DDD) method used for portioning a microservice into a set of the subdomain. This technique guides the developers in sizing the microservice; and how each microservice will have a specific function and a single database to be implemented later.

2.2. Evolution into Microservices Architecture

Dragoni et al. [10], described a real-world case study of a mission critical system - the FX (Foreign eXchange) core system at Danske Bank. This is constructed using a legacy system architecture, to address the major concerns involving the scalability of the system. It was revealed that using, in addition, a microservices architecture might be a promising methodology for reducing the code complexity. Furthermore, a number of microservices could be decoupled via the use of service discovery. As pointed out in [10], switching to a microservices architecture brings about enhanced scalability through the application of several techniques, for instance load balancing, horizontal scaling, and cache and clustering methods [10]. The FX system can be improved through the implementation of the aforementioned methods for the purpose of supporting additional qualitative characteristics. A case in point is including additional security with the intention of delivering, to the client, a user-experience which is innovative.

Villamizar et al. [3] described an enterprise case study. The application under examination was established on the cloud podium. Two versions were constructed, one using microservices and the other using a monolithic architecture. The authors analyzed the performance of both architectures in terms of response times - and they recognized that, so far as the microsystems architecture was concerned, additional provisions in terms of performance should be considered for the future. It was pointed out in [10] that the performance downgrade which appears to be involved when moving to a microservices architecture are due to the significantly increased network use and the existence of the container. Gouigoux & Tamzalit [11] dealt with some feedback they encountered after moving the MGDIS SA (a French software vendor editing application) monolithic software to an independent service. Among the benefits they encountered was an up-surge in performance, and this is somewhat contrary to the results from our case study.

Bogner et al. [12] interviewed with an expert from 10 different companies based on Germany; some of these companies have been active in Europe or even globally. The discussion covered three main aspects, the technology that used to implement the microservices, the popular features that play a significant role to adopt a microservice architecture and the impact of the microservice on different software quality attribute. The analysis presents that maintainability was rated as the most improved attribute when moving to the microservice architecture. Related to the performance the participants divided into two groups. The first group has noticed a significantly improved on the response time. While the other group believed the response time would not affect. Also, there is still a debate about security and how to deal with these challenges.

Regarding the performance issues highlighted above, there appears to be no clear upshot; it is therefore necessary to continue to look at the differences between the performance of microservice implemented applications and that of those which use a monolithic architecture.

It is believed that the performance issues differ in relation to a number of factors, for instance, the programming languages used, the host environment, and the container technology.

3. THE MICROSERVICE-BASED EVOLUTION APPROACH

This study focuses on three elements: legacy systems, microservices and cloud computing. The objective is to formulate a scheme that is highly effective and based on evolutionary structures and specific rules concerned with modernising legacy systems. An innovative approach is created which enables the evolving of a legacy enterprise system into one which fits into a lean system framework, supported by cloud computing and microservices.

The suggested architectural design concentrates on how legacy systems can be modernised through the use of a microservices framework and migration to a cloud-based platform. To develop this system further, we identified a set of features-driven microservice-oriented evolution rules for changing monolithic systems into microservice-based systems. Based on this feature-driven development, a set of system requirements have been defined for the migration: performance, functionality, and security are the three main features.

A list of sub features and the relationships between them has been identified. Regarding the features list, a set of rules have been introduced; the intention is that these rules will significantly enhance the functionality and quality of the resultant software system (e.g., in terms of maintainability, modularity, and interoperability). These evolution rules have been documented in a pseudo-code format and each rule has been given a title based on its context.

The rules can be used to deconstruct and reorganize legacy applications into microservices architecture systems. Each rule consists of an assumption which reflects the main problem of the system, an action as the solution, and an impact on system features which explains the concerns that might arise when the rule is applied. A detailed definition of the rules (e.g., the single responsibility principle, the separate database principle, the computational task rule, the I/O processing rule, and the throughput rule), along with the framework of evolution processes, has been presented in [1]. Table 1 provides an overview of the rules.

Table 1. Overview of Microservice-Oriented Evolution Rules

Rules	Action and Impact
Decomposing legacy system to microservice	Applying this rule will improve modularity and simplify scaling a particular microservice to meet the new demands and requirements. However, separate microservices add complexity and new problems, including network latency.
Single responsibility principle	Decomposed legacy system into small service, with a specific function, by applying the single responsibility problem concept. This rule will improve scalability and has a consistent code base.
Separate database	Keeping a separate data store for each microservice helps the developer choose the database type based on the microservices' needs. It can introduce some distributed data management challenges
Computation task	Breaking a monolithic system into a microservice should decrease response times and to improve computation tasks.
I/O processing	In order to enhance the legacy system response time, the system will be deconstructed to a set of microservices. This will improve response times by reducing unnecessary I/O processing.
Throughput	To improve the legacy system network throughput, the system will be constructed into a set of services. This will increase the number of the throughput (i.e. messages processed by) of microservices by reducing the number of the memory intensive functions.

This paper focuses on evaluating the proposed framework and the evolution rules by applying them in a case study. New microservices have been identified and developed, and each service has been deployed in the AWS cloud. Also, each service has been tested from the source code stage to deployment.

4. Case Study

In this paper, we present a case study which examines a monolithic system called 'RosarioSIS' [13], which we analyzed and then evolved into a new microservice-based architecture. The new system was fully implemented and was deployed on a cloud platform. This section presents an introduction to the original monolithic RosarioSIS system first, followed by a description of its evolution into a microservices architecture.

The application is a school management system which is available across several schools for the management of the school, the staff, the students and grades. The application offers many services such as payment, etc. The new microservices architecture was built using a Laravel and swagger API framework.

However, in order that the case study remained relatively simple, the application was designed to support only the four most popular services. The first service, 'School' is for managing the school itself. The second service 'Student' is responsible for adding, updating, and deleting student records. The third module is the User microservice, which presents user profiles and teacher programs. The last service 'Grades' presents the student grades and manages the GPA.

4.1. RosarioSIS Monolithic Architecture

RosarioSIS [13], an open source legacy Student Information System (SIS), has been chosen for this case study. It runs as a publicly accessible web application. The prime objective of RosarioSIS is to provide schools with a platform whereby they can manage their staff and students. It is presented as a number of different components via which the school admin can manage teachers, students, attendance, fees, events, courses, and resources.

Furthermore, there are several roles, such as teacher, staff member, payroll, administrator that can be registered with the system by the school admin. Every role has a certain number of duties associated with it, as also determined by the school admin. Also, each student will have their own web panel from which they can manage their leave, their fees and other important details.

The current system is built in PHP 5.3, which is not widely supported, and the application has a 3-tiered architecture; this makes the system an excellent candidate for the adoption of the microservice architecture. The architecture of the monolithic application is shown in Figure 1.

4.2 RosarioSIS Microservices Architecture

The ultimate goal of adopting a microservices architecture is to benefit from the single responsibility and independent deployment of each service. However, we must think carefully about how to design and build a microservice in isolation and then test all the services together in the implementation stage before the final release. It is also critical to think properly about the scope and size of each microservice. To construct a microservices architecture, the system was analysed in depth so that the entire architecture was understood.

The process that was used to deconstruct the system into a microservices architecture was as follows:

1- Understand the code and define the system's boundaries.

2- Build a class diagram for the existing system which helps to understand the relationships between the system classes and plays an essential role in determining the service.

3- Determine the problems that the design of the microservices architecture should address.

4- Identify services from the monolith by applying a domain-driven design (DDD). DDD is a technique in which a business domain is deconstructed into smaller functional components and describes the independent steps of the problem which comprise the bounded context; a bounded context includes the details of each domain, such as the domain model, the data model, and the application services [14].

5- Understand the shared database schema and break up the database in order to extract the tables and place each in an isolated independent database to be used later by one of the microservices.

6- Recognize and determine which rules are going to apply. For this case study, three rules have been considered necessary to implement: the decomposing legacy to microservices rule, the single responsibility principles (SRP) rule, and the separate database rule.

The first step is to reconstruct the existing system in order to discover, from the code, what the service elements are. The microservices were derived by applying the first and second rules: decomposing the legacy system into microservices (rule 1) and then the SRP rule (rule 2). Implementing a set of persistent and cohesive services can be achieved by using the SRP rule.

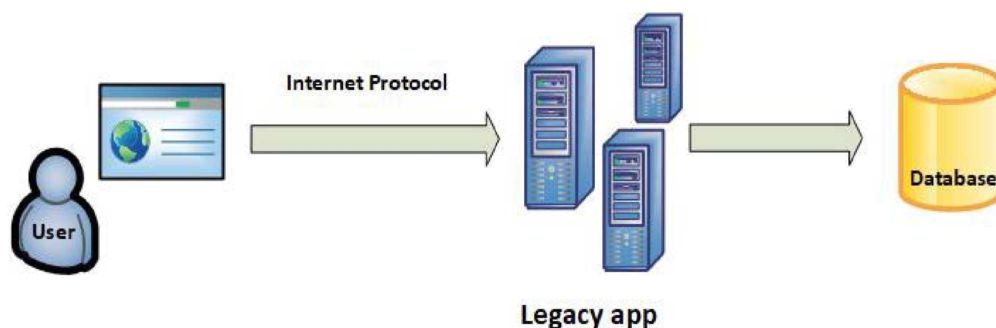


Figure 1. Monolithic architecture

Each service has a solo specific function. This rule makes clear boundaries of each microservices and indicates where code changes should go. Bounded context is an efficient way of designing a microservice.

In the case study, using DDD gives us an idea of how to figure out the boundaries of the problem in various types of RosarioSIS legacy systems, such us user management, educational, and food service systems. Breaking the bigger context into smaller chunks provides a clear idea how data will move from one component to another. As microservices have to be isolated, it is critical that each component remains in their own bounded context and has an obvious responsibility. For instance, there is a student module in RosarioSIS, so a student management is required. Also, each student has to associate with their parents, so there is a parent's profile in this subdomain. The same process has been applied to the other modules until a clear domain and subdomains of the RosarioSIS legacy system have been identified. The bounded context of student, school, user, and grade is presented in Figure 2,3,4,5.

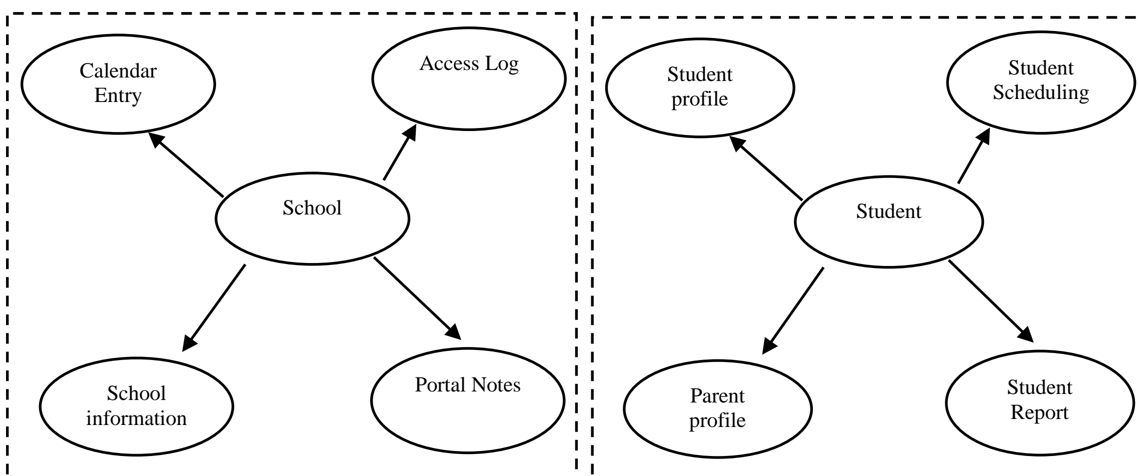


Figure 2. School context

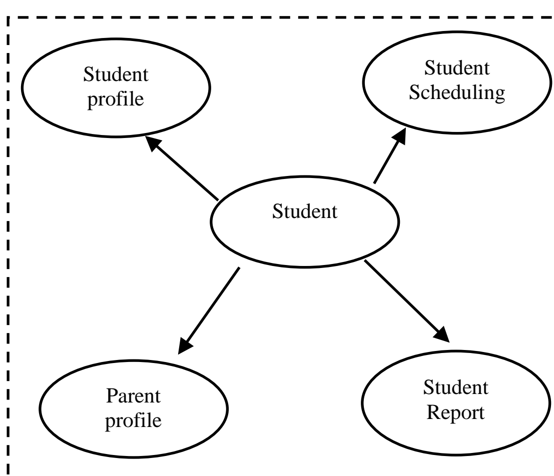


Figure 3. Student context

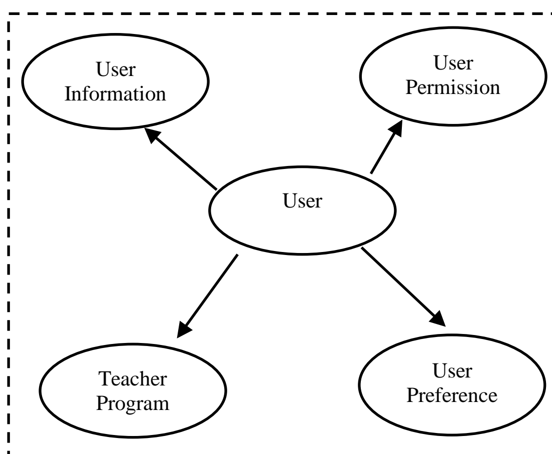


Figure 4. User context

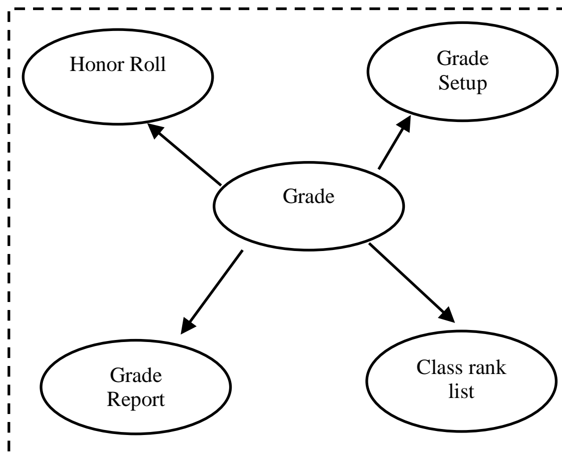


Figure 5. Grade context

The next step is to decide which data is related to which problem. This will lead us the most important part in any application: data management. In a microservice architecture, to achieve independent and loose coupling microservices, a separate database per microservices (rule 3) is applied. This means that all the microservices databases are independent of each other. A change in one database should not affect any other.

In our case study, we break the database based on the domain and figure out all the information present in a specific domain. Different domains are identified, and we then break each domain into smaller parts based on relevant data.

After splitting the function of the monolithic into different microservices, we prioritise the services to be built and ensure the main services are available. As a result, four services (µS1, µS2, µS3, and µS4) were derived from the legacy PHP code. Each service addresses a specific business scope and they are fully decoupled from each other.

To adopt the microservices architecture, it is important to decide on the number of microservices to be built. For this case study, we selected the above four microservices.

Each microservice is developed as an independent Laravel MVC framework. The gateway was developed as a light web application which receives requests from end-users (browsers) through the Internet and consumes the private services offered by the microservices (µS1, µS2, µS3, and µS4) through REST. The message interchange protocol which connects the browsers to the gateway, and the gateway to each microservice, is JSON. The gateway does not store any information. The microservice architecture will be deployed on the Amazon ECS cloud platform using the deployment illustrated in Figure 6.

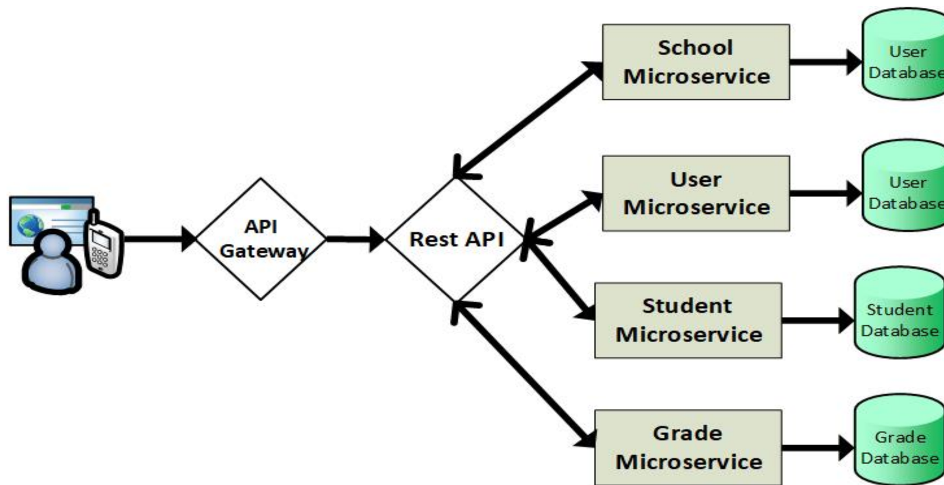


Figure 6. Microservice architecture

5. IMPLEMENTATION

The RosarioSIS legacy system was implemented as a set of web applications. Its architecture is very monolithic with rather modularity. It can be accessed through an intranet with no Internet access (i.e., offline).

- The web application. This application was developed using PHP 2.10.2, The relational database used was PostgreSQL.
- The front-end application. This application was developed in HTML-5, CSS 3, and jQuery.

In contrast, after the transformation into microservice-based architecture, the microservices architecture operated by the AWS cloud is implemented as four independent applications:

- The microservices µS1, µS2, µS3, and µS4. These applications were developed using PHP 7.1, and the Relational Database is MySQL 5.0.12.
- The gateway application. This application was developed using the Swagger API.

6. DEPLOYMENT IN A CLOUD ENVIRONMENT

Comparing the infrastructures which support each architecture, first, the monolithic architecture and the microservices architecture were deployed using Amazon ECS. ECS runs containers on a cluster of Amazon EC2 (Elastic Compute Cloud) virtual machine instances. It handles installing containers, scaling, monitoring, and managing these instances through both an API and also the AWS Management Console. It simplifies the view of EC2 instances to a pool of resources, such as CPU and memory. The specific instance a container runs on, and the maintenance of all instances, is handled by the platform. This service is deployed to a Cluster of ECS container instances that provide the pool of resources needed to run and scale the application. Additional services can be deployed to the same Cluster. Amazon ECS, or in fact any container management service, aims to make this provision of resources as simple as possible, abstracting away many of the complexities of running infrastructure at scale [15].

The legacy RosarioSIS monolithic architecture is deployed in the AWS cloud. The instance details in given in Table 2.

Table 2. Monolithic Instance Details

Instance type	vCPU	CPU Credits /hour	Mem (GiB)	Storage	Network Performance
T2.micro	1	6	1	EBS-Only	Low to Moderate

The new RosarioSIS microservice architecture is operated by the cloud customer. The four applications are deployed as shown in Table 3.

Table 3. Microservice Instance Details

Instance type	vCPU	CPU Credits /hour	Mem (GiB)	Storage	Network Performance
T3.small	2	24	2	EBS-Only	Up to 5

7. TEST AND ANALYSIS

All the tests were undertaken using a test plan relating to the JMeter tool. This tool provides for all the performance aspects of testing such as the measurement of the error rate, the throughput and the average response time. These features work as a guide to understanding what will happen when the number of users is increased.

We started the comparison by performing a targeted test for each microservice to identify the maximum number of users that each service could handle simultaneously. Then, we executed the performance test for the monolithic architecture to calculate the number of requests it could handle at the same time, in a specific ramp up period.

The development of the application for the case study allowed us to compare the performance of each architecture by executing different stress tests. The corresponding results from the experiments are described below.

7.1. Performance Tests

To test and compare the performance and infrastructure of the four microservices, we defined the same scenario for all the microservices and configured the response time. The first scenario was designed so that 40% of the requests made to µS1, µS2, µS3, and µS4 would be consumed. In the second scenario, each service received 50% of the requests. The third scenario sent 75%

of the requests to all the services. The next scenario was planned to receive 85% of the requests. The last scenario was designed to receive 90% of the requests.

To execute the stress tests for each service, JMeter was used to simulate a constant workload; this depended on the scenario and the number of requests per second which had been configured. Also, the duration was defined to be between 0.5 and 5 sec. In the course of the performance tests, both the monolithic, and the microservice architecture was operated by the Amazon ECS cloud.

The stress tests were executed with the intention of identifying the maximum number of requests supported by the microservices. This number was determined by increasing the number of requests for each scenario until the application began to generate errors or the time defined for responses from µS1, µS2, µS3, or µS4 were not met. The results of the performance tests performed on the microservices architecture are shown in Figures 7, 8, 9, and 10.

We executed stress tests for the monolithic architecture, based on the number of requests per second supported by the microservices architecture. The performance tests were executed with the goal of identifying the maximum number of requests supported by the monolithic architecture in Amazon AWS. This number was calculated by increasing the number of requests for each scenario until the application began to generate errors or the time limits defined for responses from the monolithic architecture were not being met. The results of the performance tests executed on the monolithic architecture are shown in Figure 11.

These results show that the monolithic architecture provides better performance than the microservices architecture. The reason for this is that we use only one instance for each microservice, i.e., one CPU for each microservice. However, if we were to increase the number of instances for each service, we would achieve better performance.

There are two types of requirements which must be adhered to when building a microservices architecture. The first comprises the functional requirements. The second comprises the non-functional requirements - also known as the quality attributes. The following subsections show how a microservices architecture can improve the quality attributes of an enterprise system.

7.2. Security

To understand the microservice security regime implemented here, we must first look at the way security works in monolithic applications; this will help us to see the differences between the authentication and authorization mechanisms of the RosarioSIS monolithic system and of the RosarioSIS microservices system.

In a monolithic application, the purpose of authentication is always to verify the identity of a user. In addition, authorization manages what a user can or cannot access (in other words, permissions). Also, the data which is passed between the client and the server can be encrypted. Usually, the user enters a username and password through a web browser. Then, the server verifies these given credentials. A 'session' is created, and this is stored in the database. A cookie and session id will be kept in the web browser (client side). The session will be removed from both the web browser and the sever side once the user logs out.

In contrast, the microservice API authentication, here, has been implemented with the use of Laravel passports. This represents a form of token-based authentication. The user enters a username and password. Then the server verifies these user credentials and generates a token. The token is stored on the client side. The server verifies the token (a JSON web token) and returns the required data. Once the user logs out the token is destroyed. By applying this technique, we make sure that the service user is allowed to access each specific service that they require.

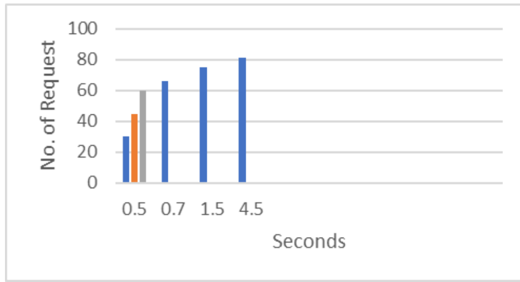


Figure 7. Performance result for student service

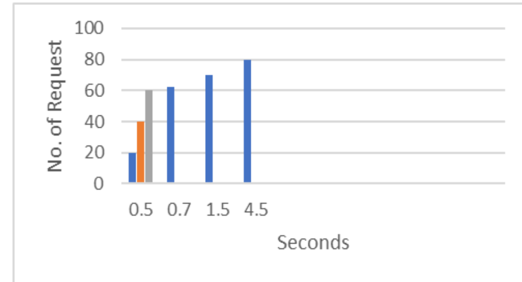


Figure 8. Performance results for user

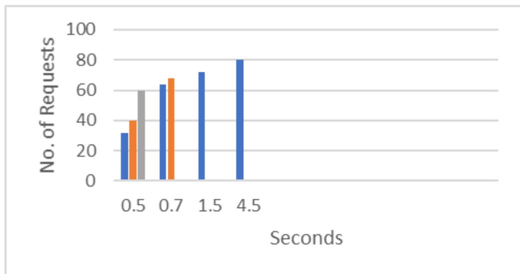


Figure 9. Performance results for grade service

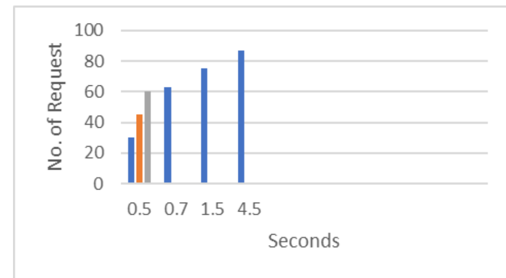


Figure 10. Performance results for school

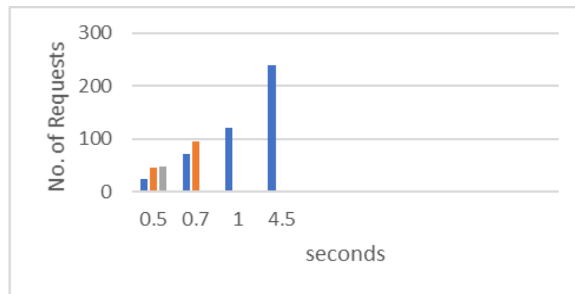


Figure 11. Performance result for monolithic

7.3. Scalability

The scalability and performance issues relating to microservices architectures are entwined with each other because of the effects each on the efficiency of the system. To improve microservices performance, we need the microservices to react to changing workloads. However, other significant factors must be considered when building a scalable microservice. The first issue is that of determining the load growth in term of requests per second by applying load testing (testing a microservice with higher traffic loads). This type of test helps to predict how the system can be expected to behave under heavier and increasing loads and provides a perspective on what the traffic loads are at peak and off-peak hours, and also where the bottlenecks are. It also assists in working out whether there are any issues which may lead to unavailability of the service.

Another factor is resource allocation, and how efficient usage of the resources (CPU, memory, data storage) may be enabled for each microservice by utilizing a container

technique which allows for automatic resource allocation for each service. The scaling of microservices can take place during runtime as each microservice is deployed independently of the other microservices, based on the most efficient use of resources and on changes in the workload [16].

7.4. Maintainability

The RosarioSIS monolithic system is complex and tightly coupled and indeed this is one of the main reasons we started to look at breaking up the system into a number of smaller units, i.e., the microservices, each of which focuses on only one business function and aims to deliver it well, so as to achieve a perfect loose coupling. These loosely coupled components are the key to improving the maintainability of RosarioSIS services. These smaller ('micro') systems are much easier to understand, change and test. The RosarioSIS microservices are clearly separated from each other as 'independent units', and there is no necessary sharing of information between one service and another. Loose coupling means that the RosarioSIS microservice design is more flexible and so will more easily facilitate future changes: error fixes and/or new functionality [10].

7.5. Testability

All of the components in the RosarioSIS microservices architecture are separate and isolated; so, testing can be scoped and also isolated. Since RosarioSIS microservices are autonomous and loosely coupled, testability is much improved in relation to the legacy RosarioSIS system. Moreover, regression testing for a specific microservice is much easier than it is for the testing of a particular functionality of the RosarioSIS monolithic system; with microservices it is possible to change part of the system and isolate it in order to test it independently from the rest of the system [10].

8. Conclusion and Future Work

In this paper, a microservice-based evolution framework has been proposed and evaluation by applying the feature-driven microservice evolution rules to the RosarioSIS legacy system. We have analyzed the differences in the performance testing of the monolithic as compared to the microservices based systems in relation to pre-specified time periods and have observed how both of these systems behave under heavy loads. Other qualitative attributes have been analyzed in relation to the adoption of microservices for the RosarioSIS system, including scalability, maintainability and testability.

It is concluded as a result of the experiments that the microservices architecture has significant value in terms of solving the problems that may arise in legacy enterprise applications.

In terms of future work, the evolution rules will be further enhanced to satisfy the needs of more complex, very large, enterprise systems. These enhanced rules will be applied to a very large enterprise system to evaluate the efficiency and scalability of the feature-driven evolution framework and the microservice-based evolution rules.

Acknowledgements

This work has received funding from the Royal Society of Edinburgh, UK and China Natural Science Foundation Council (RSE Reference: 62967_Liu_2018_2) under their Joint International Projects funding scheme.

References

[1] S. Habibullah, X. Liu, and Z. Tan, An approach to evolving legacy enterprise system to microservice-based architecture through feature-driven evolution rules, *International Journal of Computer Theory and Engineering*, 10(5), 2018.

[2] S. Daya, N. Van Duy, K. Eati, C. M. Ferreira, D. Glozic, V. Gucer, M. Gupta, S. Joshi, V. Lampkin, M. Martins, et al., *Microservices from Theory to Practice: Creating Applications in IBM Bluemix Using the Microservices Approach*. IBM Redbooks, 2016.

[3] M. Villamizar, O. Garcés, H. Castro, M. Verano, L. Salamanca, R. Casallas, and S. Gil, Evaluating the Monolithic and the Microservice Architecture Pattern to Deploy Web Applications in the Cloud, *the 10th Computing Colombian Conference (10CCC)*, Bogota, 2015, pp. 583-590.

[4] A. Balalaie, A. Heydarnoori, and P. Jamshidi, Microservices migration patterns, *Technical Report no. 1 TR-SUTCE-ASE-2015-01,* Automated Software Engineering Group, Sharif University of Technology, Tehran, Iran, 2015.

[5] K. Brown and B. Woolf, Implementation patterns for microservices architectures, *Proceedings of the 23rd Conference on Pattern Languages of Programs (PLoP'16)*, Monticello, USA, Oct. 2016.

[6] D. Taibi, V. Lenarduzzi, and C.Pahl, Architectural patterns for microservices: a systematic mapping study, *the 8th International Conference of Cloud Computing and Services Science (CLOSER'18)*, Funchal, Portugal, March 2018.

[7] A. Carrasco, B. van Bladel, and S. Demeyer, Migrating towards microservices: migration and architecture smells, *Proceedings of the 2nd International Workshop on Refactoring*, Montpellier, France, 2018, pp. 1-6.

[8] H. Knoche, and W. Hasselbring, Drivers and Barriers for Microservice Adoption – a Survey among Professionals in Germany, *Enterprise Modelling and Information Systems Architectures (EMISAJ), International Journal of Conceptual Modelling*, 14(1), 2019.

[9] I. J. Munezero, D. Mukasa, B. Kanagwa and J. Balikuddembe, Partitioning Microservices: A Domain Engineering Approach, *IEEE/ACM Symposium on Software Engineering in Africa (SEiA)*, Gothenburg, 2018, pp. 43-49.

[10] N. Dragoni, S. Giallorenzo, A. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, Microservices: yesterday, today, and tomorrow, *in Present and Ulterior Software Engineering*, M. Mazzara and B. Meyer, Eds. Cham: Springer International Publishing, 2017, pp. 195–216.

[11] J. P. Gouigoux, and D.Tamzalit, From monolith to Microservices: Lessons learned on an industrial migration to a web oriented architecture, the *IEEE International Conference on Software Architecture Workshops (ICSAW)*, Gothenburg, Sweden, 2017.

[12] J. Bogner, J. Fritzsch, S. Wagner, & A. Zimmermann, Microservices in Industry: Insights into Technologies, Characteristics, IEEE International Conference on Software Architecture Companion (ICSA-C), Hamburg, Germany,2019.

[13] Francoisjacquet, "francoisjacquet/rosariosis," GitHub. [Online]. Available: https://github.com/francoisjacquet/rosariosis/blob/mobile/INSTALL.md. [Accessed: 25-Apr-2019].

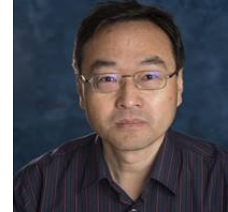
[14] M. Fowler, bliki: BoundedContext. [online] martinfowler.com. Available at:https://www.martinfowler.com/bliki/BoundedContext.html [Accessed 12 Jan. 2019].

[15] Amazon Web Services, Inc.. *Amazon ECS - run containerized applications in production*. [online] Available at: https://aws.amazon.com/ecs/ [Accessed 11 Feb. 2019].

[16] S. J. Fowler, *Production-Ready Microservices: Building Standardized Systems Across an Engineering Organization*. O'Reilly Media, Inc., 2016.

Authors

Safa Habibullah is a PhD candidate in the School of Computing in Edinburgh Napier university, UK. She holds a master's degree in software technology for the web from Edinburgh Napier University, UK. She is a lecturer at the School of Computing, Jeddah University, Saudi Arabia.



Prof. Xiaodong Liu received his PhD in Computer Science from De Montfort University and joined Napier in 1999. He is currently leading the Intelligence-Driven Software Engineering research group in the School of Computing, Edinburgh Napier University. He is an active researcher in software engineering focusing on its emerging themes including pervasive systems, microservices-oriented architecture, and cloud service evolution. He has led 10 externally funded projects as the PI, and published 125 papers in established international journals and conferences, 5 book chapters and 3 research handbooks. He is the inventor of 1 patent and the founder of a spin-out company. He has been the chair, co-chair or PC member of a number of IEEE International Conferences. He is the editorial board member of 4 international journals and editor of 2 journal special issues. He is a Senior Member of IEEE Society.

Dr. Zhiyuan Tan holds a PhD in Computer Systems from the University of Technology, Sydney, Australia. He is a Lecturer at the School of Computing, Edinburgh Napier University, UK. Prior to joining ENU, he held a Postdoctoral Research Fellowship at the University of Twente, the Netherlands and the University of Technology, Australia. His research has been supported by various funding agencies, including the Commonwealth Scientific and Industrial Research Organisation, Australia. His research findings have been published in leading journals, including IEEE Transactions on Parallel and Distributed Systems (TPDS), IEEE Transactions on Computer (TC), IEEE Transactions on Cloud Computing (TCC), Future Generation Computer Systems (FGCS), and Computer Networks (CN). He has won various research awards including the National Research Award 2017 from the Research Council of the Sultanate of Oman. He has engaged in international conferences and journals as technical committee member, organising chair and guest editor.

Prof. Yonghong Zhang is the Vice President of Nanjing University of Information Science and Technology, and the professor in the School of Automation. He has been appointed as the "Summit of the Six Top Talents" Program and the "Young academic leaders" of the Qing Lan Project in Jiangsu Province, China. He is a CMES (Chinese Mechanical Engineering Society) Senior Member, IEEE Member and CMS (Chinese Meteorological Society) Member. His major research interests focus on Dynamics Modelling of Mechanical Systems, Stability Analysis of Complex Systems, Pattern Recognition and Intelligent Systems, Neural Networks and Wavelet Analysis. He has secured over 10 international and national research projects and published over 40 academic papers.

Prof. Qi Liu received the B.S. degree in Computer Science and Technology from Zhuzhou Institute of Technology, China in 2003, and M.S. and Ph.D. in Data Telecommunications and Networks from the University of Salford, UK in 2006 and 2010. His research interests include context awareness, data communication in MANET and WSN, and smart grid. His recent research work focuses on intelligent agriculture and meteorological observation systems based on WSN.