IJIST
P ISSN : 2618-1630
E ISSN : 2909-6130

Transformation of Monolithic Applications towards Microservices

Zaigham Mushtaq, Najia Saher,Faisal Shazad, Sana Iqbal, Anam Qasim
Faculty of Computing, Islamia University, Bahawalpur, Pakistan.
Correspondence: Zaigham Mushtaq (zaigham@iub.edu.pk).

Citation | Mushtaq. Z, Saher. N, Shazad. F, Iqbal. S, Qasim. A, "Transformation of Monolithic Applications towards Microservices: A Review" International Journal of Innovations in Science and Technology, Vol 4, Issue 1, pp: 1-18, 2022,
Received | Dec 17, 2021; Revised | Jan 9, 2022 Accepted | Jan 18, 2022; Published | Jan 22, 2022.

The traditional monolithic approach is widely employed in centralized software development, deployment, and reusability, as the modules are tightly connected, causing several challenges in programming. The study utilized different techniques for the easy transformation of several running monolithic applications to micro services including, Angular 2, REST API, Web application and several other architectural approaches are discussed. This review paper highlights the significance of microservices and the transformation of monolithic applications towards microservices. As multiple software applications are an integral part of a traditional monolithic application, the modules cannot be extended separately, and different modules cannot use various technology stacks. So, monolithic source code must be migrated to the microservice platform in order to extend `the lifecycle of applications in today's environment. However, due to structural complexity, scattered application logic, and dependency upon external framework libraries, the transformation towards a microservices platform is quite challenging. A Microservice architecture is a container of loosely coupled independent services making a flexible system. In this study, potential areas for the transformation of monolithic application source code are highlighted. Furthermore, key challenges and open research issues in this area are highlighted, requiring the research community's attention. The study concludes that Microservices are not a one-size-fits-all solution for every challenging situation. Monolithic transformation requires significant amount of time and effort on the part of everyone in the business.

Keywords: Transformation; Monolithic; Microservices

Introduction

The terminology "microservice architecture" refers to a tactic that assimilate independent services coupled [1]. This paradigm supports the accumulation of scalable and maintainable platform for microservices. In ordinary environments, all the business components are packed together in a centralized traditional application [2, 3], and excessive code is dispersed and released as a whole; this development and deployment style is known as monolithic, as shown in Figure 1.

In traditional monolithic environments, all the business components are packed together in a centralized traditional application [2, 3] and excessive code is dispersed and released as a whole; this development and deployment style[4] is renowned as monolithic. To provide high availability and flexibility, the monolithic application is established as a whole, with load balancing handled by a load balancer at the front end. When a performance bottleneck arises in an application, it is usually due to restraints in one of the modules rather than the system's overall components. Relevant components or modules can't be replicated for extending monolithic architectural systems [5], and spreading the full application over numerous nodes is a waste of resources. Also, more problems are associated with monolithic systems as sometimes code is vulnerable or a lot of re-engineering is required.

Moreover, large application systems have complex maintenance in terms of continuous integration and release[6]. Drawbacks of monolithic systems include tight coupling between components, as a single application handles every task, less reusability, large codebase, tough for developers and QA to understand the code and business knowledge, less scalability, more deployment, and restart times.

Therefore, Microservices [7] accompany a few advantages, for instance, Figure 2 shows the reality that microservices are developed and deployed individually, allowing for more horizontal scalability and flexibility in different environments [8] and development team designs that are more efficient. Also, this system is easy to maintain, available and can easily be invoked. It's no surprise, then, that major internet companies Google, eBay, Netflix, and others have made significant attempts to transit from monolithic architectures to microservice-oriented application landscapes. Thus because of numerous advantages, there is a need for time to change monolithic systems to microservices systems.

Bottleneck components illustrate the applications of microservice in the microservices architecture [2, 9], these components can be deployed in several copies to overcome performance and scalability issues. Figure 3 shows a microservices technological architecture that can be utilized to address issues such as large project teams, iteration of an intricate and inefficient software update, and so on.

Additionally, the trend of migrating monolithic applications [2] to microservices is steadily expanding, transforming monolithic legacy applications into a microservice architecture. Microservices provide for the continuous supply and deployment of large, sophisticated systems that are easier to test and deploy. They communicate and exchange data using the lightweight HTTP protocol, moreover microservices are always available can be invoked and it uses simple language for invocation. Also, microservice's [1, 2] technological architecture is a good option for applications requiring high-concurrency and high-capacity systems.

Nowadays, internet systems including Amazon, Netflix, Google, IBM, Uber, Alibaba, and more firms have made the transition from monolithic [2] to microservice design. There are a range of technologies available in the form of microservices applications (XML, Ruby, Python, and Java etc.) [9] Multiple languages support acts as a framework for the transformation of monolithic systems.

Considering the benefits explained above it is need of time to migrate to microservices architecture. In this paper materials and method describes the main characteristics on which transformation is based, the most up-to-date methodologies and technologies for moving from monolithic to microservice architectures will be given, Discussion section further discusses tools and their details, proposed research design and the challenges and impact of migrating to microservices.

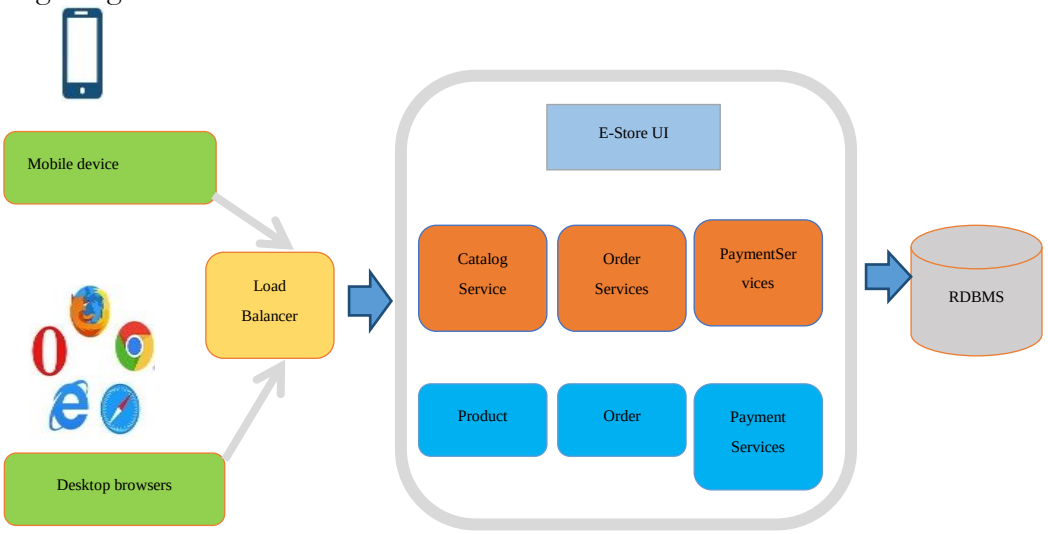


Figure 1. Monolithic Architecture

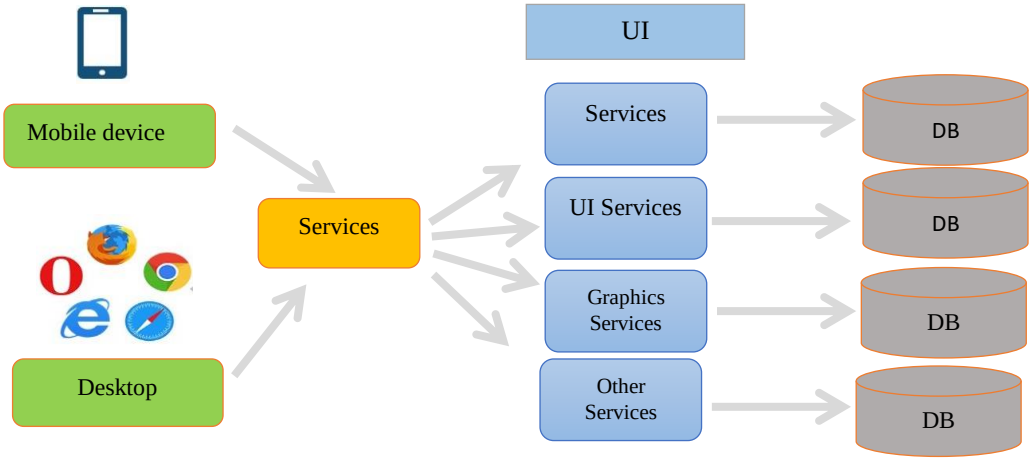


Figure 2. Microservices Architecture

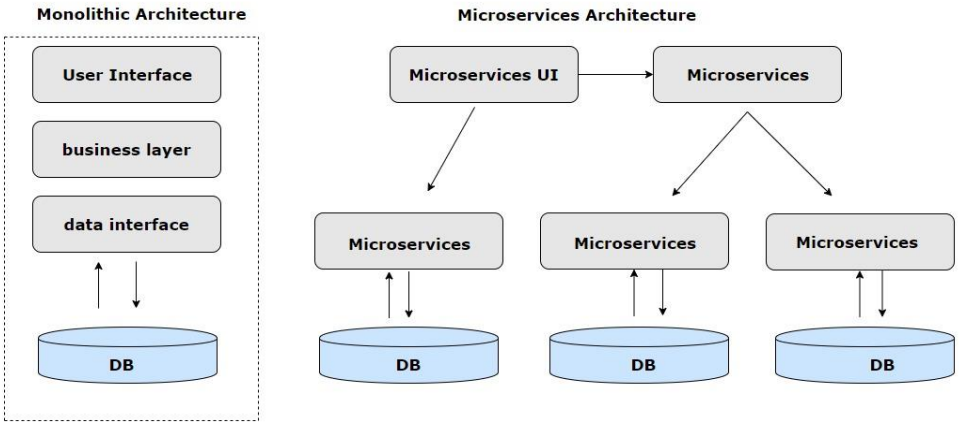


Figure 3. Difference Between Monolithic and Microservices Architecture
https://journal.50sea.com/index.php/IJIST/Transformation-of-Monolithic-Application

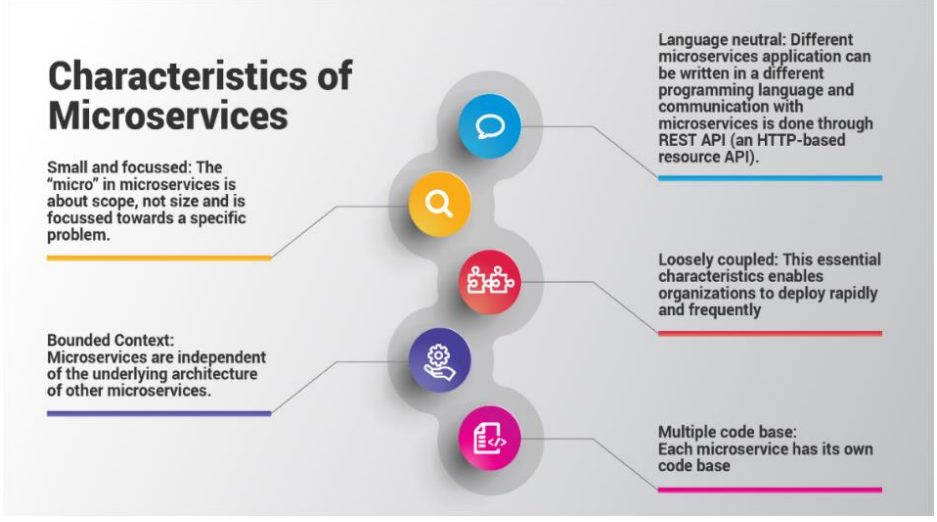


Figure 4. Characteristics of Microservices [10]

The concept to bring the vision [11] of microservices to life[8], it is based on the following characteristics in Figure .

Small and Focused.

The architectural style of microservices [12] Figure 2 shows a strategy for building a single application out of a number of small services. These are well understood and focused on a specific problem.

Lightweight.

Microservices [13] are well-designed and matched to a single business capability can perform only one function. As a result, microservices with smaller lightweight footprints are one of the common features as shown in Figure 4 that can be seen in most implementations.

Language Neutral.

Different microservices applications are written in different languages and microservices [10] can be communicated through a language-independent interface which is Rest API.

Loosely Coupled.

If changes to one system's design, implementation, or behavior do not cause changes in the other, the systems are loosely linked. When this comes to microservices in Figure 2, if a change to one microservice causes an almost rapid change to all other microservices, coupling may happen [10] that collaborate with it directly or indirectly. Loosely coupled services [1] make it easier to implement continuous integration and deployment scenarios.

Multiple Codebases

Each Service can have an independent codebase and CI/CD tooling sets as illustrated in Figure 2. Services[12] are built around business capabilities, independently deployable and packaged in code, each running in its processor codes[7].

The implementation of these characteristics in Figure 4 are also very adaptable in terms of adjusting and enhancing the knowledge-based element that is being examined. The execution engine simply needs to be enriched with more data-dependent rules after the data sources have been further integrated. This includes cases in which continuous data streams must be polled in real time. This way efficient transformation[8] of monolithic applications can be done. Figure 5 describes the main tools/techniques towards microservices that if implemented smartly will cope with challenges transformation to microservices architecture[12] will give the best results.

Figure 5. Tool used in Transformation

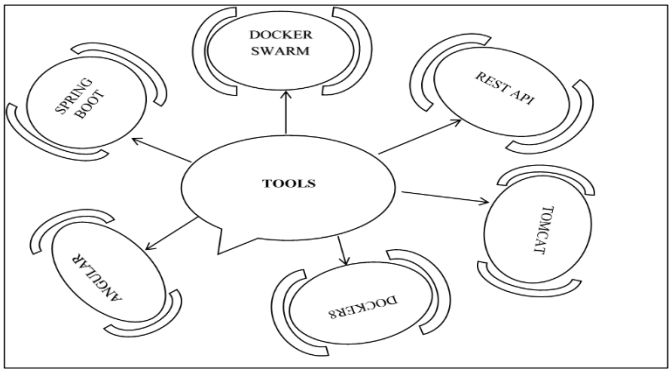


Table 1. Transformation of Monolithic Applications to Microservices Techniques/Tools: Literature Review

S #	Technique/Tool Algo	Model	Analysis	Language	Case Studies	Problem definition	Future work
1.	PPTAM+ Tool [21]	Continuous assess degradation system, avoids performance regression.	Dynamic analysis	Java code	Open-source DSL Framework	Transitioning microservices may not end up with the same or a better performing system	Integrating advanced statistical tools in R scripts like genetic algorithms can evaluate architectural choices.
2.	Software clustering algorithm SarF [9]	Identify candidate micro services	Static analysis	Java code, COBOL	Spring Boot Pet Clinic.	Manuals effort is required analyzing many dimensions	execution timing and revisions need to be added to improve candidate's identification
3.	Angular2 [16]	Mapping model bw Java classes/ proposes a fitness function to measure service quality	Dynamic analysis	Java with spring boot.	MVC framework	Complex task to identify and classify the existing service layer.	Advanced analysis such as service descriptions including its usages, dependency graphs can exclude.
4.	Docker 8 [20]	Tip-of-ice berg programing model/ Demonstrate tree shaking, sand-boxing approach	Statistic analysis, dynamic analysis	java, python, Ruby++, .net	Industry example node.js on IoT project	Architectural adjustments are required to fulfill functional and non-functional requirements.	stronger connection ad hoc reuse and its impact in the design in terms of adaptation effort.
5.	NSGA-III (algo) [30]	Measure quality indicators IGD & HV/ analyze both closeness and diversity of web services.	Dynamic analysis	JavaScript PHP,	JMetal1 in version 5.9	Closeness problem and diversity of a Pareto front.	Inverted Generational Distance (IGD) can measure convergence and closeness between PF known and Pftrue.

6.	Steinmetz architecture[31]	Graph clustering algorithm/ methodology aggregates three dimensions into a single graph	Dynamic analysis	java code	java based applications from GITHUB	Decomposition process is a significant challenge.	specific patterns may be detected where this methodology falls short.
7.	FX-Agents Approach [25]	The approach used Web Service GUI (WSGUI) Engine, which does allow dynamic GUI Generation.	Dynamic analysis	WSDL	Event Planning System using BSD and WSGUI	Shortcomings of current Web service standards like WSDL and SOAP.	FX-Agent's approach may not identify the deficiencies of WSDL and associated technologies.
8.	SSA, SCGA, CSDA techniques [12].	Static analysis and dynamic analysis model/ evaluate the degree of dependence, and through function clustering	Static analysis Dynamic analysis	web application server tomcat 8.5.20, Spring Boot 1.5.7	12 applications of open-source java-based projects	Lack of consideration of the runtime dynamic characteristics, completeness and accuracy of the static analysis.	Evaluation of candidate microservice sets obtained by different divisions is needed.
9.	MOEA 4 MBPL approach [14]	Extract Feature Models on multi-objective evolutionary algo.	Dynamic analysis	java, PHP, COBOL	six micro services-based systems	Manual approach to support design Microservices -Based Product Lines (MBPLs).	Non-functional properties, as well as the creation of other architectural models is needed.
10	MST Clustering Algo [1]	Graph-based clustering/ meta information from monolithic code bases to construct a graph represents	Static analysis, Dynamic analysis	Java, Ruby, and Python	21 open-source projects	Informal migration patterns & techniques.	fine-granular software artifacts improve the granularity and precision

Many strategies and tools for transforming monolithic software programs have been presented in Table 1 which presents the best knowledge on transformation tools and techniques.

Tools for the Transformation of Monolithic Applications Towards Microservices

A tool support is intended to make developing, deploying, and running applications easier transformation to microservices. The developer can use tools to package an application with all of its components, including libraries and other dependencies, and deploy it as a single package[15]. It Creates the environment for an automated workflow and verify following overviews given in Figure 5 about some tools used in microservices transformation[4].

Angular2

Angular2 [16] is a web application development platform for mobile and desktop devices. Using the Angular tool, any template may be translated to code that is properly optimized for today's JavaScript virtual machines. It is used as a frontend tool[16]. This framework was created to help solve problems that arise when working with single-page applications. It began with the development of a model for converting Java classes into microservice concepts. After that, it shows how to use a fitness function to determine service quality.

REST API

A REST API [17] is an easy-going and undisturbed software intermediary that allows two applications to talk to each other (API or Web API) that follows to the limitations of the REST architectural style and allows collaboration with RESTful [2, 18] web services. RESTful API [17] was introduced before Microservices that makes it easy to build loosely coupled Microservices.

Web Application Server

Tomcat [19] combines different applications in one package, users get a web server (that can manage HTTP requests/responses) and a web container (gears a Java Servlet API, also known as a servlet container). It is not a developed Java EE application server, despite the fact that it is commonly referred to be one (it does not appliance the whole Java EE API). Any Web application distribution [12]may be accomplished in several ways using the Tomcat server in microservices architecture (Figure 2).

Spring Boot for Microservices Identification

Spring Boot[19] is employed in the backend. With Spring Boot[12], Microservices allow to start small and iterate quickly. As a result, it has established itself as the de-facto standard for Java microservices. It simplifies the development of standalone, production-ready Spring-based applications that may be "simply run". Spring Boot's[19] multi purpose-built capabilities make it simple to create and deploy microservices at scale.

Docker and Docker Swarm

Docker[20] is a software platform that makes it simple for software developers to implement the use of container into their development process. Containers are a considerably better solution for a microservices design than VMs just in terms of efficiency. Docker is a free and open-source software platform that runs on a range of operating systems, making it accessible to developers working on a variety of platforms.

Techniques & Approaches Proposed for the Transformation Towards Microservices Platform.

Many organizations are approaching the objective of migrating to a microservices architecture from a monolithic one. Microservices promise faster processing and delivery, as well as the flexibility to pivot when market need shifts. Therefore, from our experience, the following fundamental techniques\approaches in Figure 4 will ensure that this revolutionary journey begins with an effective strategy.

SCGA, CSDA Technique

SCGA, CSDA Technique[12] suggests decomposing monolithic applications vertically into a subset of business-driven services. This technique[12] determine the degree of dependence, the coupling between functions was used. It performs static and dynamic analysis on monolithic apps to determine their static structure and runtime behavior.

PPTAM, PPTAM+ Technique

PPTAM, PPTAM+Technique[21] amalgamate an Application Performance Monitoring (APM) tool that gathers performance data and stack traces and shows conceptions of the results to stakeholders. By the time, PPTAM Tool [21]is also been improved such that it can detect performance reduction when switching from monolithic to microservice systems [4] developed and launched using DevOps.

SArF (Clustering Algorithm Tool)

SArF (Custering Algorithm Tool)[9] utilizes graph clustering [2] for the program dependencies that are static, as well as program and data dependencies. This algorithm [22] can collect software programs and data that are closely connected. It breaks down a system into manageable collections of software objects (e.g., programs, data). This decomposition[8] can be used to represent the system's architectural information and high-level abstraction interpretations.

MST Clustering Algorithm

MST Clustering algorithm [23] comprises of three extraction stages: monolithic, graphing, and microservices stage[24]. To generate a graphical representation of the monoliths, the coupling solutions, on the other hand, depend on (meta-)information from monolithic code bases[4] in a refactoring situation, which are then evaluated by the clustering method to give endorsements for possible microservice contenders [9] .

FX-Agents Approach

FX-Agents [25] constructed a Business Service Directory (BSD) based on Info master that provides dispersed search using logic as "glue. "The FX-Agents approach[25] is used in the microservices conversion to identify the insufficiencies of WSDL[26] and related technologies, then to discourse them with the acceptance and flexibility of Declarative Logic.

Maturity Model for Smart WS

The maturity model [27]determines Smart WS quality and usefulness. It connects various types of devices and real-world objects using Web Standards (WS), allowing them to become part of the Internet (WWW)[27].

MOEA4MBPL (approach)

MOEA4MBPL[14]is based on evolutionary algorithms with many objectives. The NSGA-II[28] and SPEA2[29] was used to evaluate six microservices-based systems by using

MOEA4MBPL[14]. Multi-objective evolutionary algorithms are utilized in this method. The method proved successful in identifying FMs with good precision and recall trade-offs, as well as meeting all microservice requirements.

NSGA-III Algorithm

NSGA-III algorithm [30] is used to measure quality indicators e.g. IGD and HV. Quality indicators enable us to analyze both closeness and diversity of web services. It is a Search-Based Microservice Identification algorithm[13].

Semi-Automated Approaches.

Semi-Automated Approaches [31] establishes a paradigm for mapping Java classes to microservice ideas. Following that, it presents a fitness function for determining service quality. Also this approach[32] developed a toolkit for analyzing monolithic systems and recommending the most effective approaches to divide functionality into a group of microservices.

Steinmetz Architecture

Steinmetz Architecture [31] aggregates the three dimensions into a single graph that we cluster into microservice candidate recommendations. It is a semi-automatic microservice decomposition technique[33].

Clustering-based Technique

Clustering-based Technique [31] defines function[34] that measures a microservice's quality by looking at its capacity to deliver consistent service and its dependency with other microservices in the resulting architecture.

Tip of the ice berg Programming Model

Tip of the ice berg Programming Model [20] that performs additional research to show a powerful link among ad hoc monolithic application reuse and its influence on design in relations of amendment effort. It employs both static and dynamic analytic approaches [20] that offered actual evidence for opportunistic software's function.

SCRUM, KANBAN Approach

SCRUM, KANBAN Approach[35] in which the background of migrations to microservices, a qualitative study on intents, strategies, and problems were recommended. Based on real-world industry systems, it examines the revolution procedure from monolithic architectures to Microservices.

Using the above technologies for migration to a microservice-based architecture legacy systems have been migrated by organizations [8] to achieve modernization. There are open challenges[21] as well as gaps in the use of these technologies/tools that are essential for both practice and research. For instance, diverse microservices can be freely joining or provide the same functionality in many forms, but may get deprecated or offline at any point in time. This particular problem is concerned with dealing with change and evolutionary variety [27] of microservices.

Further, a microservice-based architecture [12] permits to effortlessly accomplish and coordinate such a blend. These services promise a number of advantages, including reduced maintenance work, greater availability, shortened incorporation of advanced features, allowed nonstop delivery and DevOps, better configurability management, and shorter time to market. On the other side, microservices are highly compatible, empowering developers to integrate

functionalities of different systems that are not executed with the same technologies. For example, applying a complex business rule may require harmonization among a Java, a PHP, and a COBOL application. So, by using these tools, microservice-based systems enable for reprocess and personalize. The above-mentioned tools/techniques are presented in the [Table 1]below, which provides an overview of monolithic system[36].

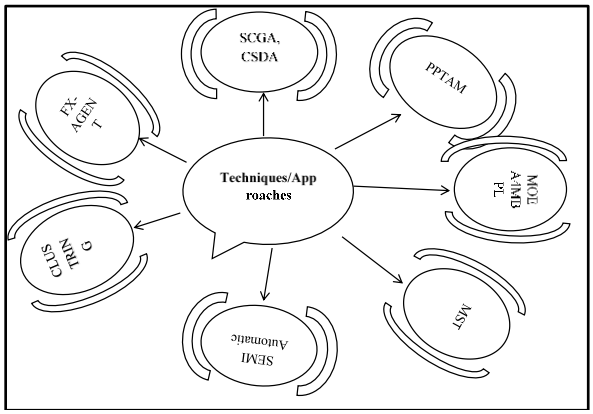


Figure 6. Proposed Techniques & Approaches for the Transformation Towards Microservices Platform from Monolithic Applications

Discussion

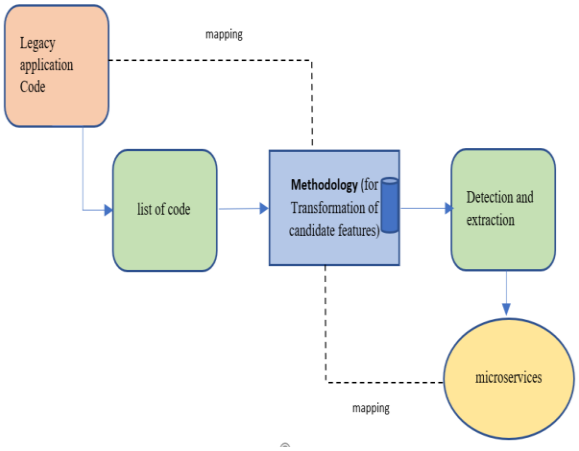
The main topic of discussion in this section is that none of the recently mentioned techniques/tools provide a general, extendable, and broader solution for analyzing the transformation of monolithic in microservice architecture at different degrees of abstractions. The presentation of various applications is incomplete and should be expanded. Existing techniques/tools/methods are challenging to extend and understand. The analysis in [Table 1] shows that monolithic transformation to microservices is still a challenge[36] for the software engineering community. However, it is contented that less complicated model is needed to help transform monolithic applications as it is simpler to carry out, the analysis is shown in Table 2.

Table 2. Types of analysis and use

Analysis type	Usage
Dynamic	60%
Static	10%
Static/dynamic	30%

This research highlights the key challenges[36] that exist in real-world development environments in many areas, such as analysis, performance, scalability, flexibility, accuracy, and applicability. The difficulties arise when dealing with complex and large systems that span multiple areas and languages. The main difficulty for transformation tools is the tight coupling of required microservices features in different platforms as a modification made to a small section of code might require building and deploying an entirely new version of software. Scaling specific functions of an application, also means you have to scale the entire application. Microservices solve these challenges[37] of monolithic systems by being as modular as possible. As the world is moving towards microservices architecture so, there is a need of time that monolithic applications or legacy code need to be reused. Also, Monolithic

applications[4] need to be transformed towards microservices architecture to achieve better usability of applications in a smart way. However, it is contented that less complicated model Figure 7 that is simpler to carry out is needed to help transform monolithic applications.



Proposed research design outlined in this paper was embedded into a predefined extraction model in Figure 7. It comprised of three extraction stages: the legacy code stage, the methodology implementation stage and the extraction microservice candidate's stage. There were two transformations between the stages: The construction step transforms the legacy code into the program representation, and the extraction step decomposed (detect being as modular as possible. As the world is moving towards microservices architecture so, there is a need of time that monolithic applications or legacy code needed to be reused. Also, Monolithic applications [4] need to be transformed towards microservices architecture to achieve better usability of applications in a smart way. However, it is contented that less complicated model Figure 8 that is simpler to carry out is needed to help transform monolithic applications.

The transformations performed during the steps may differ according to the extraction strategy in use. Furthermore, the following is the description of key challenges which transformation process has to deal with them.

Challenges & Impacts: Transformation of Monolithic to Microservices

In this section Challenges and impacts of transformation of monolithic applications towards microservices platform is discussed.

Challenges

More than 70% of today's technology leaders use the phrase "monolith to microservices" to describe their business. Managing microservices becomes more difficult as the number software increases in the microservices transformation. Increased robustness, enhanced scalability, and faster time to market are all well-documented advantages. Implementing microservices, like any transformational tendencies, comes with its own set of contests. Figure 8 illustrates the topmost challenges that most organizations suffer in their microservices journey.

It is critical that these challenges are thoroughly addressed, or be prepared for one of the following outcomes: the project will never see the light of day, or it will be completed only to find that many of the expected benefits are not realized.

Refactoring

The most challenging things in these situations is separating these services. Refactoring [3] the services out of the monolithic design can require a long time and effort. As a result, the refactoring to microservices [23] should be done in stages. Also, even if it may be faster, new functionalities should not be added to monolithic when implementing it.

Testing

Developers can use a variety of automatic testing [15] techniques to test an application, based on its requirements. Microservices go hand in hand with continuous integration [29] and continuous delivery. Without these two approaches, managing multiple services, their implementations, and validation of the service's behaviors becomes very difficult.

Splitting up

When separating apart the services [22], it's important to keep in mind that the services don't get very fine-grained. Microservices may cause performance [17] disbursal, particularly when communicating across the network. If you communicate via REST over HTTP, for example, each inter-service contact adds burden due to network delay and marshalling and unmarshalling the data. There will be a lot of interactions between the services if they are highly fine-grained, and because each call adds strain, the system may not run efficiently enough.

Integration

Integration [4] across different microservices is one of the most difficult tasks. Since the players may choose to utilize different programming languages while creating services, it is not advisable to attach the interaction between services to a certain technology. It is preferable to use a technology that does not demand the use of a certain programming language.

Data management

Data management [16] is a vital feature of any application. The application's use cases and the database schema that should be used. Microservices allow users to use a variety of database engines. This architecture, known as "database per service", has its private set of tasks. Numerous databases make managing them more difficult, and the organization may not have a clear understanding of the database. However, using a single database for all services is troublesome because the database structure is now tightly coupled. The usage of a single shared database negates many of the advantages of microservices, and thus is not recommended.

Tight coupling

Monolithic applications are being refactored [38] or rewritten using the microservice architectural pattern. Decoupling a monolithic program into discrete modules that each include the components required to execute a particular business function is a common way to achieve microservice architecture. These applications should be loosely coupled[39]so that they can be reused according to specific services. These services usually connect with one another via language-independent APIs such as REST [17]. Biggest disadvantage of virtualization shown heavy load on underlying kernel or server but from past some decades an alternative technology emerges and get popular in a short time [40, 41].

Fault-tolerant

Microservices must be fault-tolerant in design [2, 42]. In a distributed system with many services, it is possible that a service may become overburdened and unable to reply on time, or that the service will go down. The circuit breaker pattern is convenient. The circuit breaker pattern responds quickly to errors and can give a fallback that returns default data rather than waiting for a dependency's answer. When there are enough failures, the circuit breaker will stop making additional calls to the dependent and will instead return an error. Hystrix [42] is a library for distributed systems that provides latency and fault tolerance. This

is simple to use and allow developers to make calls to dependencies with minimal latency and fault tolerance.

Impacts

Social impacts

In the transformation putting whole organization on the project, may halt or slow down any development as this can disconnect those people from other ongoing development, or longer than project team thinks. Instead of making it a project, it should be an ongoing effort [43].

Economic impacts

The economic impact of such a change is not negligible, and taking such an important decision to re-architect an existing system should always be based on solid information, so as to ensure that the migration will allow achieving the expected benefits. Moving to economics, it was proposed that monoliths benefit from economies of scope, and microservices benefit from economies of scale[44].

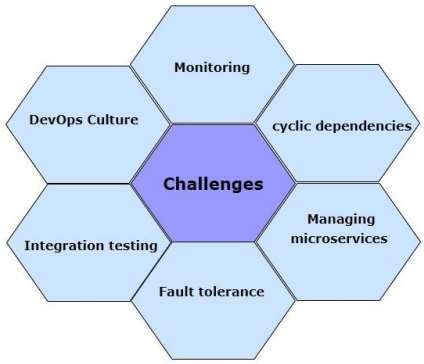


Figure 8. Challenges in Transformation to Microservices

Conclusion

This research provides knowledge about the unique challenges that businesses have to experience transitioning from monolith to microservices, this shift requires a significant amount of time and multiple organizational teams' efforts. Making a distributed and systematic system creates additional issues that must be reported and resolved. Although microservices architecture can be thought of as an advanced software framework, it is more systematic in terms of microservices tooling approaches, and most difficulties can be overcome by using open-source tools developed by software businesses.

However, mentioned strategies and technologies may not be able to solve the problem of loose coupling and code restructuring. The focus on challenges should almost always be on the technical side. The organization's architecture should be similar to its structure. Microservices are not a one-size-fits-all solution for every situation, and the challenges can be tough to overcome in some circumstances. Monolithic refactoring is a significant procedure that involves a significant amount of time and effort on the part of everyone in the business. This change is still feasible. Organizations attempting this change should weigh the costs and advantages of the transition in question, as well as their own concerns. To be successful with microservices, technical and hierarchical problems must be overcome. There is also a demand for tools that may address technical issues and obstacles associated with the transition from monolithic to microservices architecture.

Acknowledgement.
We would like to thank with deep sense of gratitude to our research supervisor Dr. Zaigham Mushtaq for his keen interest, inspiring guidance and endless support with our work at all stages.

Author's Contribution.

Designing The Experiment:	Sana Iqbal, Zaigham Mushtaq, Najia Saher
Performed The Experiments:	Sana Iqbal, Zaigham Mushtaq, Faisal Shazad
Analyzing The Data:	Sana Iqbal, Zaigham Mushtaq, Anam Qasim

Code, Designed the Software or Performed the Computation Work Sana Iqbal, Zaigham Mushtaq, Anam Qasim Work or Revised It Critically for Important Content: Sana Iqbal, Zaigham Mushtaq, Najia Saher, Faisal Shazad

Conflict of interest. Authors has no conflict of interest for publishing this manuscript in IJIST.

Project details. The aim of this research is the reusability of exiting code and transformation towards microservices environment.

REFRENCES

1. Mazlami, G., J. Cito, and P. Leitner, *Extraction of Microservices from Monolithic Software Architectures*. 2017. 524-531.
2. Zahid, M., Z. Mehmmod, and I. Inayat. *Evolution in software architecture recovery techniques—A survey. in 2017 13th International Conference on Emerging Technologies (ICET)*. 2017. IEEE.
3. Jamshidi, P., et al., Microservices: *The journey so far and challenges ahead. IEEE Software*, 2018. 35(3): p. 24-35.
4. Selmadji, A., et al., *From Monolithic Architecture Style to Microservice one Based on a Semi-Automatic Approach*. 2020. 157-168.
5. Tapia, F., et al., *From monolithic systems to microservices: A comparative study of performance. Applied Sciences*, 2020. 10(17): p. 5797.
6. Lapuz, N., P. Clarke, and Y. Abgaz. *Digital Transformation and the Role of Dynamic Tooling in Extracting Microservices from Existing Software Systems*. in European Conference on Software Process Improvement. 2021. Springer.
7. Brito, M., J. Cunha, and J. Saraiva. *Identification of microservices from monolithic applications through topic modelling. in Proceedings of the 36th Annual ACM Symposium on Applied Computing*. 2021.
8. Birchall, C., *Re-engineering legacy software*. 2016: Manning Publ.
9. Kamimura, M., et al. *Extracting Candidates of Microservices from Monolithic Application Code*. in 2018 25th Asia-Pacific Software Engineering Conference (APSEC). 2018. IEEE.
10. srijan, *The Advantages of Microservices*.
11. Dragoni, N., et al., *Microservices: Yesterday, Today, and Tomorrow, in Present and Ulterior Software Engineering*, M. Mazzara and B. Meyer, Editors. 2017, Springer International Publishing: Cham. p. 195-216.

12. Ren, Z., et al. *Migrating web applications from monolithic structure to microservices architecture. in Proceedings of the Tenth Asia-Pacific Symposium on Internetware.* 2018.
13. Yi, J.-H., et al., *An improved NSGA-III algorithm with adaptive mutation operator for Big Data optimization problems. Future Generation Computer Systems*, 2018. 88: p. 571-585.
14. Mendonça, W.D., et al. *Towards a Microservices-Based Product Line with Multi-Objective Evolutionary Algorithms.* in 2020 IEEE Congress on Evolutionary Computation (CEC). 2020. IEEE.
15. Saman, B., *Monitoring and analysis of microservices performance. Journal of Computer Science and Control Systems,* 2017. 10(1): p. 19.
16. Bandara, C. and I. Perera. *Transforming Monolithic Systems to Microservices-An Analysis Toolkit for Legacy Code Evaluation.* in 2020 20th International Conference on Advances in ICT for Emerging Regions (ICTer). 2020. IEEE.
17. Salah, T., et al. *The evolution of distributed systems towards microservices architecture.* in 2016 11th International Conference for Internet Technology and Secured Transactions (ICITST). 2016. IEEE.
18. Apis, N. *Difference between microservices and monolithic architecture.* Available from: https://nordicapis.com/should-you-start-with-a-monolith-or-microse.
19. Macero, M., *The Microservices Journey Through Tools. 2017.* p. 179-265.
20. Mäkitalo, N., et al., *On opportunistic software reuse. Computing*, 2020. 102(11): p. 2385-2408.
21. Janes, A. and B. Russo, *Automatic Performance Monitoring and Regression Testing During the Transition from Monolith to Microservices.* 2019. 163-168.
22. Kobayashi, K., et al. SArF map: *Visualizing software architecture from feature and layer viewpoints.* in 2013 21st International Conference on Program Comprehension (ICPC). 2013. IEEE.
23. Yano, K. and A. Matsuo. *Labeling feature-oriented software clusters for software visualization application.* in 2015 Asia-Pacific Software Engineering Conference (APSEC). 2015. IEEE.
24. Brandes, U., M. Gaertler, and D. Wagner. *Experiments on graph clustering algorithms. in European Symposium on Algorithms.* 2003. Springer.
25. Petrie, C., et al. *Adding AI to web services. in International Symposium on Agent-Mediated Knowledge Management.* 2003. Springer.
26. Curbera, F., et al., *Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI.* IEEE Internet computing, 2002. 6(2): p. 86-93.
27. Maleshkova, M., et al., *Smart Web Services (SmartWS)--The Future of Services on the Web.* arXiv preprint arXiv:1902.00910, 2019.
28. Deb, K., et al., *A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE transactions on evolutionary computation*, 2002. 6(2): p. 182-197.
29. Adham, A., N. Mohd-Ghazali, and R. Ahmad, *Performance optimization of a microchannel heat sink using the Improved Strength Pareto Evolutionary Algorithm (SPEA2). Journal of engineering thermophysics*, 2015. 24(1): p. 86-100.

30. Carvalho, L., et al. *On the Performance and Adoption of Search-Based Microservice Identification with toMicroservices.* in 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME). 2020. IEEE.
31. Löhnertz, J. and A.M. *Oprescu, Steinmetz: Toward automatic decomposition of monolithic software into microservices.* 2020.
32. Maisto, S.A., B. Di Martino, and S. Nacchia. *From Monolith to Cloud Architecture Using Semi-automated Microservices Modernization.* in International Conference on P2P, Parallel, Grid, Cloud and Internet Computing. 2019. Springer.
33. Löhnertz, J. and A.M. Oprescu, Steinmetz: *Toward automatic decomposition of monolithic software into microservices.*
34. Eski, S. and F. Buzluca. *An automatic extraction approach: Transition to microservices architecture from monolithic application.* in Proceedings of the 19th International Conference on Agile Software Development: Companion. 2018.
35. Fritzsch, J., et al. *Microservices migration in industry: intentions, strategies, and challenges.* in 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME). 2019. IEEE.
36. Assunção, W.K., J. Krüger, and W.D. Mendonça. *Variability management meets microservices: six challenges of re-engineering microservice-based webshops.* in Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A-Volume A. 2020.
37. Rajasekharaiah, C., *Microservices: What, Why, and How?, in Cloud-Based Microservices.* 2021, Springer. p. 13-40.
38. Hunold, S., et al. *Pattern-based refactoring of legacy software systems. in International Conference on Enterprise Information Systems.* 2009. Springer.
39. Hasselbring, W. *Microservices for scalability: Keynote talk abstract. in Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering.* 2016.
40. Sehir e N, Shehzad M.A, Aslam M.S, Sajid W and Imran M, "*Optimize Elasticity in Cloud Computing using Container Based Virtualization*". International Journal of Innovations in Science and Technology, Vol 02 Issue 01: pp 01-16, 2019.
41. Saleem.K, Khan.SM "*A Study of Awareness and Practices in Pakistan's Software Industry towards DevOps Readiness*" International Journal of Innovations in Science and Technology Vol 3 Issue 3 PP 102-115, 2021.
42. Hassan, S., R. Bahsoon, and R. Kazman, *Microservice transition and its granularity problem: A systematic mapping study. Software: Practice and Experience,* 2020. 50(9): p. 1651-1681.
43. Auer, F., et al., *From monolithic systems to microservices: an assessment framework. Information and Software Technology*, 2021. 137: p. 106600.
44. Singleton, A., *The economics of microservices. IEEE Cloud Computing*, 2016. 3(5): p. 16-20.