Disclaimer

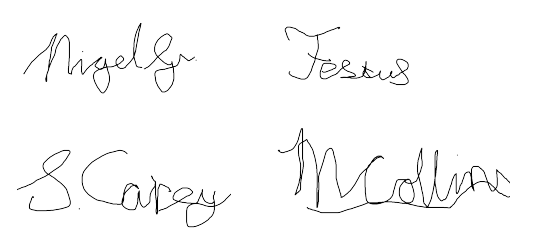
A report submitted to Dublin City University, School of Computing for module CA4009: Search  
Technologies, 2019/2020. I understand that the University regards breaches of academic integrity and  
plagiarism as grave and serious. I have read and understood the DCU Academic Integrity and Plagiarism  
Policy. I accept the penalties that may be imposed should I engage in practice or practices that breach  
this policy.

I have identified and included the source of all facts, ideas, opinions, viewpoints of others in the  
assignment references. Direct quotations, paraphrasing, discussion of ideas from books, journal articles,  
internet sources, module text, or any other source whatsoever are acknowledged, and the sources cited  
are identified in the assignment references.

I declare that this material, which I now submit for assessment, is entirely my own work and has not  
been taken from the work of others save and to the extent that such work has been cited and  
acknowledged within the text of my work. By signing this form or by submitting this material online I  
confirm that this assignment, or any part of it, has not been previously submitted by me or any other  
person for assessment on this or any other course of study.

By signing this form or by submitting material for assessment online I confirm that I have read and  
understood DCU Academic Integrity and Plagiarism Policy (available at:  
<http://www.dcu.ie/registry/examinations/index.shtml>)

Name(s):



Nigel Guven, Ikenna Festus, Shaun Carey, Michael Collins.

**Distributed Microservices Architectures: A Polyglot Future**

**Nigel Guven** [nigel.guven2@mail.dcu.ie](mailto:nigel.guven2@mail.dcu.ie)

**Festus Ikenna** [ikenna.ejike2@mail.dcu.ie](mailto:ikenna.ejike2@mail.dcu.ie)

**Michael Collins** [michael.collins38@mail.dcu.ie](mailto:michael.collins38@mail.dcu.ie)

**Shaun Carey** [shaun.carey26@mail.dcu.ie](mailto:shaun.carey26@mail.dcu.ie)

School of Computing,

Dublin City University.

**Abstract.** Microservices Architectures are widely discussed for their novel concepts and innovative perspective into a systems architecture which is loosely decoupled, highly scalable, and when provided with ancillary mechanics like DevOps, can be empowered with elasticity and automation. The potential for polyglot systems to further enhance microservices gains the ability to support heterogeneity and creativity. Polyglot microservices which are written in a subset of languages, independent of each other, enable businesses to select processes which satisfy their requirements. Microservices are known for their agility which allows small, self-sufficient software teams to take ownership of their specific products. The technological freedom provided by microservices allows for teams to find the best tool to match a requirement. However, there are also challenges associated with microservices, especially with adopting a polyglot approach. The possible ramifications of scaling too wide and utilising too many microservices, many of which may be polyglot could lead to disarray within a company. It is wiser to scale gradually and use a select subset of programming languages which suit a team’s skillset. There are investigations into two companies, one large and one small, which delve into their success with microservices and the ability to support polyglot functions. There is also a system-wide view of a sample microservices architecture which divulges into the composition of a microservices architecture.

**Key Terms:**

 Microservices  Microservices Architectures  Polyglot Microservices 

 Polyglot Persistence  Distributed Microservices  Polyglot Programming 

1. **Introduction**

In this paper, there will be a thorough examination of microservices and the push towards distributed polyglot systems, an understanding of the benefits and challenges of polyglot architectures, an examination of microservices architectural schemas and inspection of polyglot microservices from the standpoint of large and small businesses. Two key aspects are the concepts of polyglot persistence and the interdependence between suitability and sustainability of microservices concerning the stability of systems and businesses.

Distributed Microservices Architecture (MSA) is a relatively modern conceptual view of software development whereby a schema for an application is composed of a collection of highly granular functions that are loosely conjoined and separately deployable [1]. Each microservice is well maintained and robust in comparison to unary architectures such as monolithic applications whereby the entire codebase is accrued in a central location. Furthermore, microservices architectures can be polylingual in form which can enable a service in Java to communicate with another written in Python or C++ [2]. These communicate through one or more API Gateways which are language-neutral transfer structures connecting a whole system. Polyglot persistence is a term referring specifically to databases whereby a system may contain many databases, each of which has its specific functions and is entirely independent of the others [3].

With the establishment of new cloud computing technologies in the post-millennial age, microservices architectures are becoming increasingly important in the development of large-scale systems in which business and customer requirements must be fully realised. Migrating to the cloud, specifically through cloud-native architectures such as microservices, is a multi-dimensional problem and thus is non-trivial [4]. As applications expand in size and complexity, and requirements are fleshed out. Microservices architectures enable an organization to hold to these high standards, evolving their technology stack by refactoring outdated services wherein the whole application does not become influenced by any such changes [5].

The benefits and drawbacks of using a microservices architecture are to be analysed regarding the suitability of microservices to both companies and users, scalability in terms of utilising polyglot architectures and the benefits and challenges derived from that and how microservices can be compared to monolithic architectures. An examination of microservices architectural design is also initiated and discussed in terms of the number of services being utilised by systems and how polyglot languages may be applied.

Section 2 describes the literature review methodology. Section 3 covers the analysis of distributed microservices architectures and their capabilities regarding the polyglot schema. Suitability of polyglot microservices is discussed in association with businesses, scalability of microservices and possible effects produced as an output on the software development side. The benefits and challenges of using polyglot microservices architectures are placed in juxtaposition and there is also an examination in the types of microservice architectures currently being implemented. There are two case studies included with regards to how two companies Varidesk, a small e-commerce company selling innovative desk solutions and Amazon with a specific focus on their cloud computing platform, Amazon Web Services (AWS). Section 4 includes the constraints and limitations of this research document. Section 5 and 6 provide a further discussion on the subject and a definitive summarization of the study.

1. **Literature Review Methodology**

This research was formed based on a 4th Year B.Sc. in Computer Applications and Software Engineering assignment in the School of Computing, Dublin City University. A team of four students were allocated a research topic with the title: “Distributed Microservices Architectures: A Polyglot future”. Given that microservices are thoroughly utilised in many major software engineering environments, the relevance of this assignment is both highly educational and applicable to the work that the team of students might encounter in their future careers.

The first task was to identify white literature and extensions of grey literature where such documents could prove to be relevant, substantial and useful to the overall multivocal literature review (MLR). To retrieve key literature, significant strings had to be verified. The chosen strings were at first subsets of the topic title: "Distributed Microservices ", "Microservices Architectures", "Future of Microservices" and "Polyglot Microservices". Gaining information from these topics permitted an extension of informational strings to include: "Services Oriented Architectures", "Monolith to Microservices" and "Polyglot Persistence". A set of 30 articles, books and presentations were shortlisted as having been identified by the team as being crucial to the study and containing a high degree of interest. Provided with information about how to identify citable, substantial literature from an earlier lecture by Dr Vahid Garousi, the team were able to identify that each of the 30 knowledge sources could be supported given that they were published on one of the major Peer Review and Library websites including IEEE, ScienceDirect, ACM Library, O' Reilly, Springer, ResearchGate and Wiley. There were also some website articles utilised during our study and these were given careful examination before being deemed to have referential integrity. AWS and Microsoft documentation were also accredited in our research.

In gathering the full set of source documents for the literature review, those papers that had keywords, abstracts, and conclusions matching the search strings were deemed relevant. Publications were fact-checked by examining the authors to make sure that they were written by those with the relevant authority on those subjects. Furthermore, given that microservices architecture is a modern implementation of software systems, it was deemed necessary to examine the dates, references, and citations of the various articles to make sure that their content was contemporary with current articles to prevent the relay of information that might result in contradictions arising between two or more reviews. These problems are further explored in section 4.

1. **Analysis**

**A. Polyglot Systems - Benefits and Challenges**

There are many benefits and challenges to distributed microservices architectures and how they are implemented. Polyglot architectures describe a style of distributed microservices architectures which is greatly advantageous and adaptable to different environments whereby services can be written in a variety of languages which suit the system requirements and the skills of the development team. The aspect of polyglot microservices enables software companies to select features which match their specific requirements. Developers can empower their applications by utilising their adeptness in certain programming languages to enhance creativity and make debugging easier [6].

The main reasons for the transition from monolithic to microservices architectures are the beneficial characteristics with which the latter obtains. These characteristics include independence, language and platform variation and freedom. Independence specifically deals with the ability for each microservices architecture to be altered, replaced and improved. This comes with an enormous potential benefit for companies due to the vast quantities of available programming languages and frameworks, of which each component in a distributed microservices architecture can be written in one of many languages. In the words of Neal Ford, the software consultant credited with coining the term ‘polyglot programming’, ‘The times of writing an application in a single general-purpose language is over’ [7]. As a result of utilising polyglot architectures, developers are granted the freedom to choose the best tool or language to solve a certain task which reduces system build restrictions and increases workflow and creative energy around the company, especially in the development team [8].

There are challenges associated with utilising polyglot microservices in terms of a system's codebase, debugging and polyglot persistence. A code base that is polyglot in nature may be disbanded into separate codebases and this can result in system instability. Testing tools are required for these distributed codebases which can be time-consuming, especially when performing integration testing [9]. Monolithic codebases are beneficial in this circumstance given that every man-made document and file is contained in a single location [10].

Polyglot persistence is beneficial to a systems architecture in ensuring that databases are built so long as all system requirements are satisfied. While it is primarily useful for a multi-purpose application, having multiple separate databases comes with severe disadvantages which affect polyglot programming as a whole:

* Consistency is a major challenge because referential integrity must be ensured across all databases. Where one tuple references another and if there are duplicates of data, these must be deleted simultaneously [11].
* Security is further complexified where each database must have enforced access control and for many users, this can be difficult to achieve [12].
* Overheads in configuration, deployment and maintenance increase drastically with each additional database system being implemented which is in tandem with system codebases also [11].

A critical issue of deploying polyglot microservices is a result which leads to an over-exploitation of code deployment. Code churn is a term which refers to edits, modifications, comments and deletions of existing snippets of source code. A research document written for the 2010 International Symposium on Software Reliability Engineering has shown that there is a strong connection between high volumes of code churn and the number of defects discovered while testing, hence such situations that arise may delay the time to deployment [2].

As a result of service interdependence, a system makes use of programming languages more flexible and suited to certain circumstances. A discussion of polyglot architectures is not to be analysed without examining the distribution of programming languages and their distinct qualities. For example, a team may use a functional language like Erlang as one service for its power in concurrent programming [13]. A second service may be written in an object-oriented language like C# with its powerful Language-Integrated Query (LINQ) for database probing [14]. The diverse variety of languages currently in the industry and their paradigms can be used to the advantage of the programming team and to the system that they are designing. In addition to this, it has been stated that microservices cohabitate immensely with container management frameworks like Docker and Kubernetes. This means that an amalgamation of microservices and containers can further contribute to the polyglot future of microservices [15].

**B. Suitability in Business Terms**

While there is more than a substantial quantity of benefits of adopting microservices architectures and instilling polyglot functionality, there are also situations which arise where systems are unsuited for microservices architectures and there are some scenarios for which polyglot programming should be restricted. Applications which are entirely unsuited for microservices architectures are seen in small companies like Varidesk Inc. In an experience report written by two of their employees, they have reported on the challenges of migrating their monolithic architecture to microservices which include trying to reduce infrastructure costs and keeping build and release times low [16].

Smaller companies will often not have the required development budget for the upkeep of a monolithic application and may opt for microservices architectures instead. For businesses, microservices allow for more frequent delivery of service and enhanced performance in processes [6]. Therefore, the architecture would be suitable for business owners who prefer to get faster feedback or for adjusting their investments arbitrarily. This is one of the biggest reasons why larger companies are moving towards microservices like Amazon and Netflix where they both have the responsibility of satisfying both their customers and investors. Microservices allow them to have concisely focused teams that align with their business models and scalability [6].

A top-down view of a distributed microservices architecture allows for business owners to see exactly where their primary resources are allocated and if they need to shift resources from areas of low and high impact and vice versa. Another big reason why microservices suit a larger company, albeit is a reason that probably concerns smaller companies a lot less, is that business owners sometimes want to be able to offload work to a third-party without the risk of losing their property. Microservices allow businesses to segment work for non-important and non-restricted business functions without disclosing the important services [1].

In scaling software applications, monolithic architectures are unsuited for when a bottleneck arises (where the capacity of an application is limited by a single component [17].) With a microservice architecture, a system can scale its bottlenecked services by providing them with more subsidies and assistance through memory and processing power. Given the bounded nature of microservices, such a device does not require knowledge of the underlying system or of other microservices for that matter. The bounded context is beneficial to a company because it can combine functions into a single business capability which is then implemented as a service. This is seen with the FX Core system which handles trading and finance in Danske Bank, the largest bank in Denmark [18]. From the perspective of the developers, there is a sense of intimidation with regards to monolithic systems. A significant amount of time is required to become familiar with a codebase when new staff are employed. The time that is required to change code and the fear of breaking some dependencies may result in the wasting of the most precious resource available to a company: Time [19].

It would be extremely unsuitable to use microservices if there is no plan on utilising them in conjunction with DevOps – a set of processes that combine software development and technical operations [4]. Microservices can cause an explosion of moving parts with many intricacies and it is not adequate practice to implement microservices without efficient monitoring of automation and deployment. Furthermore, the construction of services that are too granular or that do not serve as efficiently as their requirements specifications can introduce both latency and the demand for increased management and communication for a development team to ensure that they understand the functionality behind each service and the system as a whole [6].

**Case Study 1: Amazon**

Amazon launched its cloud computing 'services on-demand' platform with the desire to create a reliable, scalable e-commerce presence, intending to lower prices for its retail customers. Amazon was essentially able to drive out the cost of their IT infrastructure by adopting cloud-enabled microservices [20]. Given that Amazon was initially a small company selling books online, they have driven their business through expansion in all product areas and geographies and are currently placed 5th in Fortune500 rankings for international companies (as of 2019)[21]. AWS supports massive scalability and reliability which match its growing business requirements. Like many other competitors, AWS provides services on demand which could be in AI, storage and even serverless computing functions such as AWS Lambda [22]. As a result of Amazon’s initial platform as the international leading e-commerce company, there is an ability to expand in all sectors, be it in employment, research or geographies. Amazon employs approximately 647, 500 people across all sectors which give it the flexibility it needs to supplement its scalability. In terms of software development, Amazon has assigned designated teams to coordinate focus onto a small subset of microservices on its cloud platform [20].

Companies which are using AWS for their IT solutions provide an initial investment to purchase, deploy and run services on AWS. According to the International Data Corporation, companies that have deployed on AWS have seen cumulative savings of as much as $2.5 million per application. This is a massive boon for companies that are deliberating to deploy their application on AWS. Furthermore, many organizations like Netflix which are renowned for their microservices architectures have cited that AWS offers the best scalability, time-to-market and pricing which are critical components of business models [20].

AWS offers SDK’s for seven different programming languages – Java, C#, Ruby, Python, JavaScript, PHP and Objective C [23]. While the level of support is challenging from the aspect of maintaining order and consistency across so many microservices, Amazon can support these polyglot microservices given that they have the requirements in delegated software teams and finance to administer subsets of microservices across their platform [24]. A further outcome generated is that Amazon is marketing their platform to a sizable portion of prospective companies whose product teams are more likely to choose AWS to deploy their application given the scalability and polyglot nature of AWS.

**Case Study 2: Varidesk Inc.**

Varidesk Inc. renamed as ‘Vari’ as of February 24, 2020, are an American e-commerce company and manufacturer of office furniture and equipment whose notable products are height adjustable desks. In 2019, they published a report of their recent experience in migrating from a monolithic application to a microservices-based architecture. They describe themselves as a small company and have discussed their issues in continuous integration and deployment (CI/CD) which are reported to not have scaled efficiently to meet enough business requirements [18].

The initial architecture which Varidesk had deployed was monolithic in style, where a Content Management System handled multiple responsibilities related to e-commerce and communications of an Enterprise Resource Planning (ERP) system. The initial system was found to be difficult to extend and customize. The company decided to redeploy its system as a collection of microservices on Microsoft Azure platform (choice of cloud provider was not biased at time of restructuring). The time required to redeploy accumulated up to a year and a half which resulted in 24 microservices composing the new distributed system.

Issues quickly arose in the new system with having to reconcile with the outdated CI/CD pipeline:

* Reconciling dependencies among microservices: Each microservice had dependencies and to establish consistency and coordination, the build machines would have to support all the requirements for each microservice. A greater challenge was utilising polyglot services in .NET and NodeJS models which required the redesign of infrastructure.
* Keeping build/release times low: The build and release infrastructure became a bottleneck as each microservice had to deployed individually which caused a surge of requests for the two virtual machines which were responsible for the build and release functions.
* Keeping infrastructure costs low: To solve the build and release requests, the company derived a solution to scale horizontally by provisioning more virtual machines. This would lead to undesirable growth in costs per unit of infrastructure. Instead, it was decided to use Parallelised builds which were performed by Visual Studio Team Services (VSTS) which runs concurrently to provide multiple functions simultaneously [25].

The DevOps team decided to apply DevOps principles towards its infrastructure and CICD pipeline. The three main concepts that were applied include containerization, orchestration and infrastructure-as-code [18]. Containerization involves virtualization to deploy and run applications without the need for a dedicated VM. Two applications can run on the same system independently but isolated from one another with an underlying Operating System and access to the system libraries. The primary benefit of containerization is that it provides parallelization to be implemented. The orchestration was initiated through Kubernetes which provides container scaling and management [27]. Kubernetes managed to address the concerns regarding the cost of infrastructure by clustering the build/release VMs and providing automatic administration and monitoring of the two devices. Employing Kubernetes also provided Infrastructure-as-code mechanisms whereby the allocation of resources and containers instances could be instantly updated through a configuration file to handle efficient scaling [28].

As a small company, Vari has had great challenges in adopting microservices architectures with both the system and the business at risk of collapsing. However, their solutions to address the challenges have allowed the system to stabilise and with its new distributed microservices architecture, scaling can be managed efficiently. The company states that DevOps enabled them to overcome their challenges and accommodate a microservices-based architecture[18].

**C. Implementation of a Microservices Architectures**

To gain a broader scope into the benefits of utilising microservices architectures as opposed to monolithic architectures, there must be a brief description of the difference between these two architectures about code, apprehension, deployment, language and scaling [18].

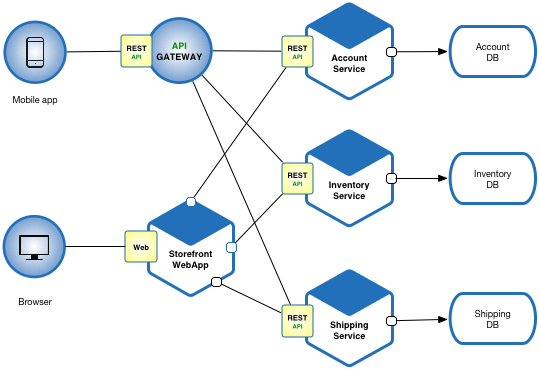
Concerning code, microservices architectures are equipped with multiple code bases of a possibly finite set. On the other hand, monolith architectures use a single code base for the entire application. This presents a serious problem for Monoliths in that they can become complex and difficult to maintain. An example of this can be seen with some companies in the space industry which utilise monolithic systems. Failure of a single component can compromise the entirety of a space shuttle, thus resulting in the loss of billions of dollars and even human lives [29].

In terms of deployment, monolithic architectures obtain complex deployments with scheduled downtimes and maintenance windows [30]. With microservices, there are multiple deployments, however, each block of time assigned for maintenance is greatly reduced with almost zero downtime. It is better to consolidate microservices into sets which can be managed efficiently and reduce problems such as chattiness.

Chattiness is a term associated with negative issues of microservices which must communicate through an API Gateway such as HTTP-based REST API’s. Microsoft Azure documentation states ‘A direct conversion from in-process method calls into RPC calls to services will cause a chatty and not efficient communication that won't perform well in distributed environments’ [31]. Chattiness is an effect of a distributed microservices architecture that contains low cohesion and tight coupling of services.

At this juncture, one might deduce that a distributed microservices architecture would be suitable for a project that might match the requirements of their business. To provide clarity of choice, a microservices architecture will be examined and the suitability of the system architecture will be analysed to provide insight into whether this design pattern is the right fit for a company. The microservices design pattern is ideal for medium to large-sized teams where there is an emphasis on scalability and productivity [32]. This means that if a team is looking to take advantage of CI/CD development flow and hypothesizing a means of integration of new developers into a project as soon as possible, then microservices would seem to be a good fit. It is also useful, given a degree of technology agnosticism, for teams looking to utilize emerging technologies.

Furthering this point, modularity is a core tenet when it comes to this design pattern [33]. On the most abstract level, a microservices oriented architecture consists of loosely coupled services which collaborate to provide a cohesive application to the user, while allowing developers to treat each microservice as effectively independent. This independence allows each of the services to be developed and maintained by small teams, removing considerable amounts of 'red tape' usually involved with larger teams and projects. The reduced size ensures the services are highly maintainable and testable.



***Fig 3.1*** *Example e-commerce platform developed with the microservices pattern [5].*

Figure 3.1 shows a sample microservices architecture for a fictitious e-commerce application which takes orders from customers on the front-end applications of multiple platforms (Web, Android, iOS etc.). The backend ‘black box’ side of the application verifies stocks, manages accounts and processes shipping information. Polyglot persistence is indicated in the diagram in which there are three different databases for inventory, accounts and shipping, all of which are wholly independent of one another [3]. The three services attached to each database can be written in numerous languages which as seen from above, utilise RESTful API’s to communicate over an API layer.

Each of the individual services is much smaller than if the application were to use a monolithic approach. Therefore, they can be individually developed and maintained by a smaller group of programmers. This also means that the services are easier to understand for developers and accelerates all aspects of the application from a production standpoint, thus increasing productivity.

Looking at the overall application, this solution allows for continuous delivery and deployment of large and complex applications. The approach as seen in the diagram aids improves maintainability, testability and deployment. Given that each service is managed by a small team (the ideal size of which is commonly referred to as ‘a team that can be fed by two large pizzas’) [6], allowing the product manager to organise the developmental effort around these multiple autonomous teams.

In addition to this, there are further benefits. A microservices approach increases fault isolation. For example, there might be a service which has a memory leak. With a microservices architecture, only the service in which a memory leak has occurred will be affected. This allows for the isolation of those services with the fault while still providing availability for the other non-affected components. If this problem were to occur within a monolithic application, the entire product would be at risk of failure. Modularity also eliminates the long-term commitment to any one technology stack [34]. When adding a new service, or even updating an existing one, the developers can change the software that they are using in a short period.

However, microservices architectures do not provide the ‘golden ticket’ away from any technological problems, they do have their own set of drawbacks. The predominant challenge with these types of systems is the added complexity of developing a distributed system. While the services themselves are easy to manage, developers must also create an inter-service communication mechanism that must be tested and maintained [35].

The inter-service mesh is at the heart of microservices architectures and hence is unavoidable. Developers must be able to make requests spanning multiple services and test those reliably. This requires strong communication between each team collaborating on a project and may become a problem later during the development process if not taken seriously. Other drawbacks include a lack of IDE support and increased complexity [36]. IDE’s are based around monolithic architectures and therefore do not have much built-in support for microservices architectures.

The complexity for microservices architectures is ‘N times M’ versus the N complexity of a monolithic application [37]. For instance, a monolithic application might run on a JVM(Java Virtual Machine or equivalent), for that same application to be run using microservices, there would need to be M times as many JVM’s where M is the number of services. If each service is run in a separate JVM (such as EC2 as is the case with Netflix), then the overhead is much higher.

1. **Research Limitations**

Due to the assignment-like environment that was assembled as part of the package with this research paper, along with the six-week deadline, it was assumed that time constraints were going to be our primary setback. Therefore, we had to come up with a solution that would aid us in our organisational capacity and limit the extent of our primary research to what we believed were the best sources which would be most beneficial to our research. Other ancillary documentation would provide us with information which could provide expansions into future research topics.

Much of this research had to be compacted, rather than expanded on, which limited the overall scope of the report. Given this factor, we couldn't complete our investigations and had to rely on research which has been done in the past. Subsequently, a great deal of information in this report had to be analysed for intellectual integrity, and conditions may have changed fairly since these investigations were undertaken. This research was carried out by a small group of undergraduate computing students. Whilst we were provided with tutorials on performing scholarly research, we still expected that our degree of expertise would not be up to the standard of more professional researchers, therefore carrying the risk that parts of the undertaken research may have inconsistent levels of credibility.

We could not find many articles on business applications which discussed their experience with shifting to polyglot microservices. Although we did manage to find a case study for a small company in Vari, we deem that this case study alone is possibly not strong enough to provide evidential research of utilising microservices from the standpoint of a small company which was sourced all from within one document. We believe that there are potentially hundreds of companies who have documented their experience of utilising polyglot microservices but the fact that this information might not be made public remains to be seen.

1. **Discussions and Future Research**

Software architectures are repeatedly thrust into a cycle of constant redesign and redevelopment. For years, monolithic applications were the primary design choice for many businesses. With increasing demands for cheaper implementations of architectures and scalable modularized systems, new designs such as service-oriented architectures and microservices architectures have become the foundation for new methods which solve the problems of scalability and infrastructure cost reduction. The future looks promising for microservices in that new concepts are being effectuated in serverless computing and multi-cloud platform infrastructure. Businesses might in the future be able to push their entire application onto a cloud architecture without having to compensate for a single block of localised infrastructure. The ability to select the strongest features of various cloud providers can also be incorporated into a business application to supplement its systems.

For future discussion, we would have liked to discuss the differences between multiple microservices architectures currently in implementation by leading cloud platform companies such as Google Cloud Platform (GCP) and Amazon AWS and perform cost analysis on their systems. Furthermore, we would have preferred to focus discussion on the topic of utilising DevOps to coordinate the deployment of microservices architectures in greater detail as we have a great interest in this area. Any future research that examines the angle of polyglot microservices architectures should discuss the future of microservices and analysis of serverless computing and Function-As-A-Service (FAAS). There is even the possibility of microservices being used to empower applications with AI-As-A-Service functionality as can be seen with on-demand machine learning capabilities on some platforms like Firebase and Azure. We deduced that there was considerable overlap with other literature review topics, and we would have liked to discuss some of these intersecting areas while confining the discussion to the context of our research.

1. **Conclusion**

In this paper, we have provided an extensive examination of distributed microservices architectures and the approach towards their polyglot optionality. We have analysed the implementation of microservices from a polyglot perspective and how two companies, one large and one small, are utilising microservices architectures. We have also supplied a diagram of a sample system which utilises microservices. In terms of future research, we have issued possible areas of research for any interested readers of this document. We have analysed how large companies such as Amazon can scale and deploy with ease given their status as a powerhouse of technological developments. We followed this with a further case study into Vari, a relatively small company which recently disassembled their monolithic system into a microservices application. This provided a further interesting comparison given that the latter and former companies are both e-commerce companies.

A viewpoint of the polyglot nature of microservices received important results, given that companies wishing to adopt a microservices architecture should not scale too quickly or contribute too many polyglot services lest they endanger the organisation of their system and business. Upon further analysis, we discovered that DevOps principles are critical to the implementation of microservices architectures, given that DevOps processes provide a means to control the flow of production. It is most optimal to scale gradually, in conjunction with DevOps, and for development teams to provide initial research on their technology stack requirements so that they can choose which tools and languages best match their goals.

**References**

[1] Towards an Understanding of Microservices, Dharmendra Shadija et al., Proceedings of the 23rd International Conference on Automation and Computing, 2017.

[2] Gremlin: Systematic Resilience Testing of Microservices, Victor Heorardi, Shriram Rajagopalan et al, IEEE 36th  International Conference, 2016.

[3] Towards Automated Polyglot Persistence, Michael Schaarschmidt et al, Semantic Scholar, 2015.

[4] Microservices Architecture Enables DevOps: Migration to Cloud-Native Architecture, Armin Balalaie, IEEE, 2016.

[5] <https://microservices.io/>, Chris Richardson, Eventuate Inc., 2015

[6] Microservices from Theory to Practice: Creating Applications in IBM Bluemix, Shahir Daya & Shishir Narain, ITSO, 2015.

[7] <http://nealford.com/memeagora/2006/12/05/Polyglot_Programming.html>, Neal Ford, ThoughtWorks, 2006.

[8] Microservices: Yesterday, Today and Tomorrow, Nicola Dragoni & Saverio Giallorenzo, Springer, 2017.

[9] Microservices Validation: Mjollnirr platform case study, D.I Savchenko, 38th International Convention on Information and Communication Technology MIPRO, IEEE, 2015.

[10] Advantages and disadvantages of a monolithic repository: a case study at Google, Ciera Jaspan & Matthew Jorde, Proceedings of the 40th international conference on Software Engineering, ACM Library, 2018.

[11] Polyglot database architectures = Polyglot challenges, Lena Weise, Institute of Computer Science, University of Gottingen, CEUR, 2015.

[12] A Polyglot Persistence Approach for E-Commerce Business Models, Kristi Srivastava & Narendra Shekokar, ICIS, IEEE, 2016.

[13] Concurrency with Erlang, Steve Vinoski, Internet Computing, IEEE, 2007.

[14] Querying in C#: How language integrated query works, Mads Torgersen, 22nd Companion to the 22nd ACM SIGPLAN Conference on Object-Oriented Programming Systems and applications, OOPSLA, 2007.

[15] Performance Evaluation of Microservices Architectures Using Containers, Marcelo Amaral, 14th International Symposium on Network Computing and Application, IEEE, 2015.

[16] Overcoming Challenges with CI/CD pipelines when moving from Monolithic applications to microservices: An experience report from a small company, Virdroha Debroy & Seneca Miller, IEEE, 2019.

[17] The Architectural Implications of Cloud Microservices, Yu Gan & Christina Delimitrou, IEEE Computer Architecture Letters, 2018.

[18] From Monolithic to Microservices: An experience report, Antonio Bucchiarone et al, IEEE Software, 2018.

[19] <https://entrepreneur-resources.net/business-valuable-resource-time>, Dequiana Jackson, 2017.

[20] The Business Value of Amazon Web Services Accelerates Over Time, Randy Perry and Stephen D. Hendrick, IDC, 2013.

[21] <https://fortune.com/fortune500/>, Fortune500.

[22] Infrastructure Cost Comparison of Running Web Applications in the Cloud using AWS Lambda and Monolithic and Microservice Architectures, Mario Villamizar, Lorena Salamanca, Santiago Gil et al., 16th International Symposium on Cluster, Cloud and Grid Computing, IEEE, 2016.

[23] <https://aws.amazon.com/developer/>, Amazon Web Services Documentation.

[24] Polyglot Application Auto Scaling Service for Platform-as-a-Service Cloud, Seetherami R Seelam et al, International Conference on Cloud Engineering, IEEE, 2015.

[25] Building Lean Continuous Integration and Delivery Pipelines by Applying DevOps Principles: A Case Study at Varidesk, Vihodra Debroy & Seneca Miller, 26thACM Joint Meeting of European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ACM, 2018.

[26] Microservices Architecture in DevOps, Muhammad Waseem and Peng Liang, 24th Asia-Pacific Engineering Conference Workshops, IEEE, 2017.

[27] Kubernetes Microservice with Docker, Deepak Vohra, Springer Media, 2016.

[28] Infrastructure as Code: Managing Servers in the Cloud, Kief Morris, O’ Reilly Media, 2016.

[29] Comparative cost and utility analysis of monolith and fractionated spacecraft using failure and replacement Markov models, Gregory F. Dubos and Joseph H. Saleh, Acta Astronautica, Science Direct, 2011.

[30] A Reusable Automated Acceptance Testing Architecture for Microservices in Behaviour-Driven Deployment, Mazedur Rahman and Jerry Gao, Symposium on Service-Oriented System Engineering, IEEE, 2015.

[31] Communication in a microservice architecture, Sabah Shariq et al., Microsoft .NET Documentation, 2020.

[32] The Economics of Microservices, Andy Singleton, Cloud Computing, IEEE, 2016.

[33] Microservices Tenets, Olaf Zimmerman, Special Issue from Springer, 2016.

[34] Microservices-based Architecture for the NRDC, Vinh D. Lee et al., 13th International Conference on Industrial Informatics, IEEE, 2015.

[35] Service-Oriented Architecture: Analysis and Design for Services and Microservices, Thomas Erl, ACM Library, 2016.

[36] Towards a Taxonomy of Microservices Architectures, Martin Garriga, International Conference on Software Engineering and Formal Methods, Springer, 2017.

[37] Building Microservices Architectures for Smart Banking, Aneta Poniszewska-Maranda at al., International Conference on Intelligent Networking and Collaborative Systems, Springer, 2019.