Migration aspects from monolith to distributed systems using software code build and deployment time and latency perspective

Alok Aggarwal, Vinay Singh
School of Computer Science, University of Petroleum and Energy Studies, Dehradun, India

Article Info

Article history:

Received Aug 27, 2023
Revised Jan 16, 2024
Accepted Feb 10, 2024

Keywords:

Containerization
Continuous-integration
Continuous-delivery
Fault tolerance
Microservices
Scalability

ABSTRACT

The transition from an error-prone, slower, and extremely high-volume legacy system like monolithic system to a faster, lighter, and error-free microservices based system is not always so simple. Microservices are independently deployable and allow for a better team autonomy. In this work, several migration efforts to migrate from a legacy based monolithic system to a pure distributed microservices based system has been tested and deployed in keeping two DevOps principles, the software code build and deployment time and latency in monolithic and microservices. Some real-time projects are considered to measure the performance and the time taken to execute the experiments. To measure the total build and deployment time and latency, Jenkins, Prometheus, and JMeter are installed which are industry-recommended softwares. It is observed that there is a total of 7 seconds taken to build and deploy at containers for 10 microservices whereas 10 monolith applications took almost 260 seconds to be built and deployed to the application server. While increasing more requests per second it is observed that upto 3000 requests per second, it impacted the response time of monolith applications but microservices stays the same. The main conclusion is that microservices are rarely impacted in response time with respect to requests per second.

Corresponding Author:

Alok Aggarwal
School of Computer Science, University of Petroleum and Energy Studies
Dehradun, India
Email: alok.aggarwal@ddn.upes.ac.in

1. INTRODUCTION

The current trend of the software industry is primarily focusing on achieving their new business goals faster, better, and error-free by adopting the new advanced systems which entail time-to-market. To achieve this goal, it simply requires recreating and careful selection of advanced, modernized, and sophisticated subsystems which could help to achieve their goal [1]. Subsequent variations to the system architecture will not only be extremely expensive and long but also difficult. Varying processes and switching to a new system frequently resulted in a lot of time and exertion, and many industrial process limitations make it not able to acclimate to changes without a gigantic volume of resources and complications. The transition from a monolithic system to a microservices based system is not always so simple [2]. Upholding the excellence of the software echo system from a monolithic or other legacy system can be tough. It has been observed that due to high volume and its complicated nature the software architects, developers, and project managers show their reluctance. But there is a motivation behind it since the big players in the market like Netflix, Google, Amazon, and more have been adopting the microservices infrastructure.

The microservice's architecture is very conducive to continuous development and deployment iterations. As the DevOps environment becomes more focused on rapid deployment, adaptability, and growth, microservices are the perfect tool to build versatile systems that allow for continuous improvement and scaling with little to no down time for the user [3]. By breaking down a large suite of features into discrete functions, where each runs as its own service that is not tied to any specific server or dependencies, developers are able to create a loosely coupled system of independent functionalities. In all, each microservice can do one or a few things very well. Microservices work well with agile development processes and satisfy the increasing need for a more fluid flow of information. Microservices are autonomously deployable and permit for superior group independence. Each microservice can be conveyed freely, as required, empowering nonstop advancement and quicker application overhauls. Particular microservices can be relegated to particular improvement groups, which permits them to center exclusively on one benefit or highlight. This means that teams can work autonomously without worrying about what is going on with the rest of the application. A typical process to migrate from a monolithic system to a microservices-based system involves the following steps: i) identification of business functional components, ii) prioritization of defective components, iii) identification of component's inter-dependencies, iv) identification of domain groups, v) creation of a user interface and program application programming interface (API), vi) migration of microservices to microservices, and vii) deployment of microservices to containerized environment.

In this work, migration from a legacy-based monolithic system to a pure distributed microservices-based system has been proposed. The proposed migration has been tested and deployed in keeping two DevOps principles; the software code build and deployment time and latency in monolithic and microservices. For the performance evaluation and the time taken, various real-time projects are taken. Jenkins, Prometheus, and Jmeter, the industry-recommended software, were installed. The total build and deployment time and infrastructure readiness software were calculated using Jenkins while Prometheus for latency.

The rest of the paper is organized as follows. Section 2 gives a brief of some of the related works done by the earlier researchers in the domain. Section 3 gives the details of the experimentation work. Experimental analysis and results of the microservices versus monolith-based systems are given in section 4. Obtained results are discussed in section 5 and section 6 concludes the work.

2. RELATED WORK

A very minimal volume of published research work was found concerning migration from a monolithic-based system to a microservices-based system with respect to software code build and deployment time, space, decomposition, and latency time in monolith and microservices. Though there is some published research work on migration [4]-[19], and out of which few works use microservices-based migration approach [14]-[17]. Few authors have analyzed the concept of architectural decomposing and subsequently compared various refactoring methodologies which are recently been proposed. The methodologies are categorized by the primary decomposition procedure. The review produced a variety of strategies to decompose a monolithic application into independent services. Most approaches are only applicable under certain restrictions. A few earlier researchers analyzed pragmatic research on migration approaches and their certain practices towards the embracing of microservices by the software industry. Prominently, the various design patterns and work on various practices focus on targeting practitioners involved in the process of migrating their applications, the collected information on the executed various migration activities, and the real-time issues faced during the migration. These migration patterns indirectly support information technology organizations in planning their migration projects to be deployed in container-based environments more efficiently and effectively [20]-[25].

3. EXPERIMENTAL PART DESCRIPTION

Microservices must be designed to respond on a call from an API or a web service request to get data from the database and return it much more quickly. There must not be a dependency on third-party API or other heterogeneous platforms to help on horizontal or vertical scaling required in a typical development and must leverage continuous-integration and continuous-delivery pipeline. Experimentation using some real-time projects was conducted for software code build and deployment time and latency in monolith and microservices. Details of these experiments are given in sub-sections 3.1 and 3.2.

3.1. Software code build and deployment time

There are two factors in any version control, speed and agility. If coding is done at the local system, it often makes the application faster at local docker environment rather than deploying microservices on remote public server. Usually monolith applications are bulky in size and posses several of libraries with higher dependencies on the third party APIs which make the end build and deploy time very time-consuming. It means build mechanisms in monolith applications are a way lengthy, cumbersome, and vulnerable. The design pattern of microservices are that they take less time and consume fewer resources like CPU cycles, memory, and network bandwidth and are built to scale up and scale down.

This helps to speedup the application response time and several other backend applications running on kubernetes to significantly reduce the time for main microservice applications so that it can reach the market with *time-to-market*. In practice, there is a tendency to build and deploy microservices that are tightly coupled to the application they are written for. Minimizing dependencies between microservices should be a major focus of the design and training. Figure 1 shows a comparison on build and deployment time between microservices and monolith. Figure 2 shows real-time example considering build and deployment time of microservices. A script to build and deploy for microservices is shown in Figure 3.

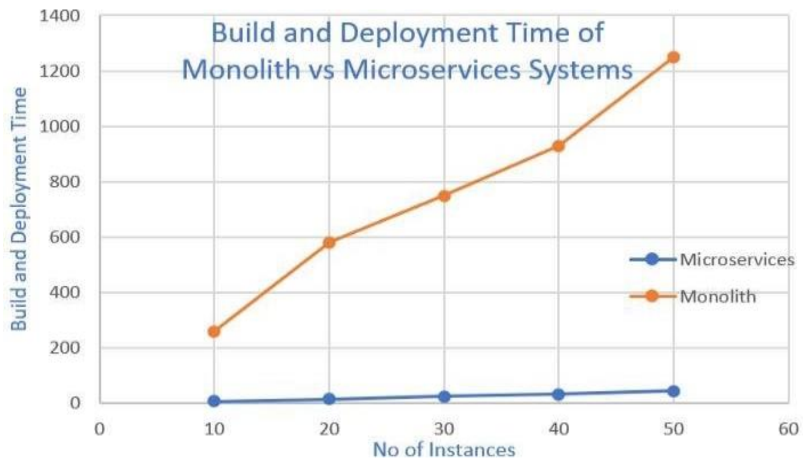
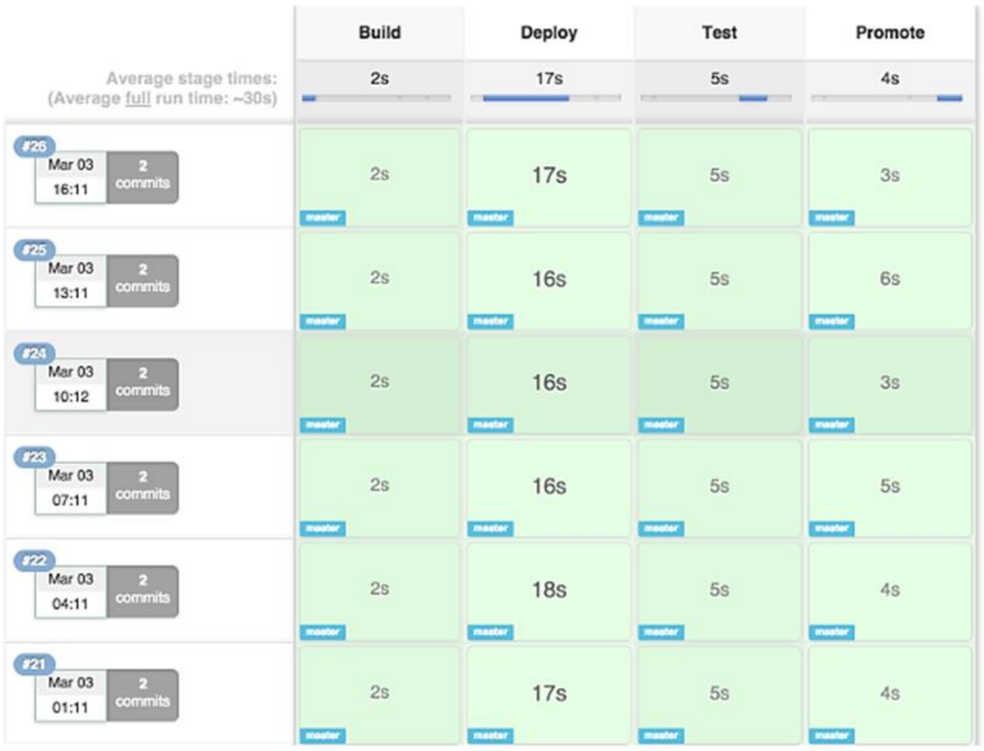
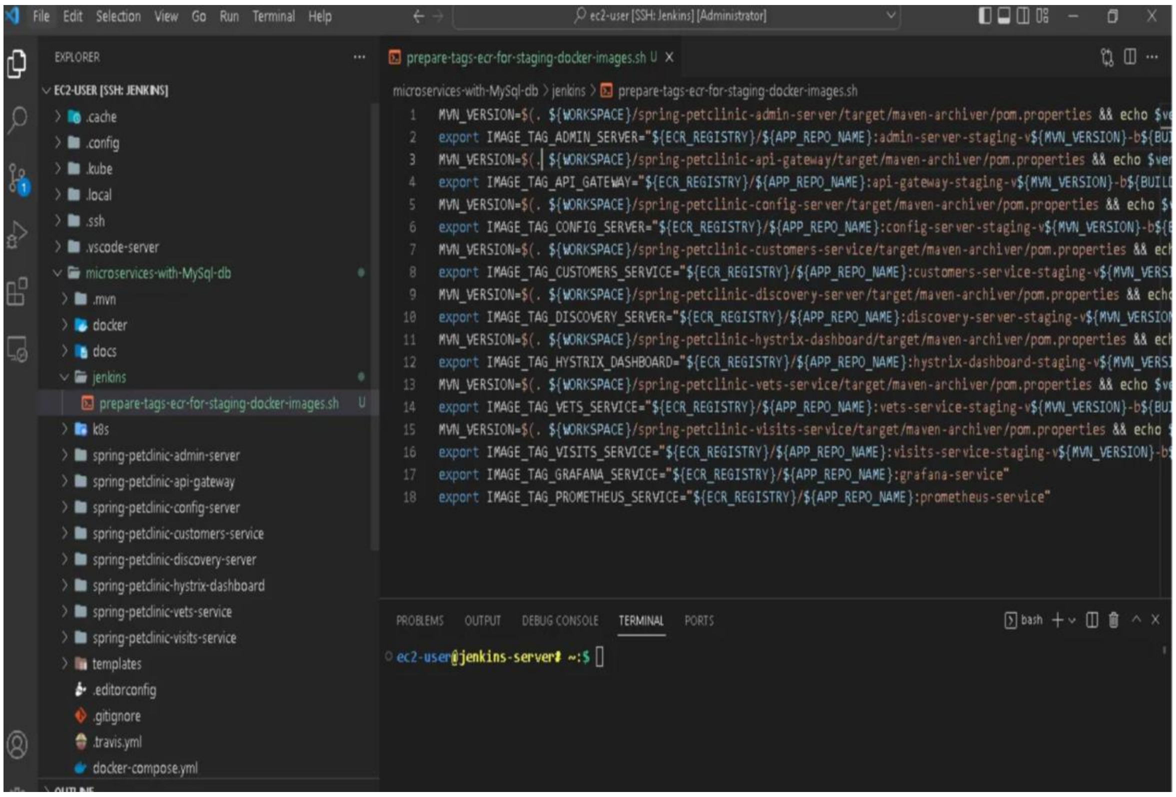


Figure 1. A comparison on build and deployment time between microservices and monolith



Average stage times: (Average full run time: ~30s)	Build	Deploy	Test	Promote
	2s	17s	5s	4s
#26 Mar 03 16:11 2 commits	2s master	17s master	5s master	3s master
#25 Mar 03 13:11 2 commits	2s master	16s master	5s master	6s master
#24 Mar 03 10:12 2 commits	2s master	16s master	5s master	3s master
#23 Mar 03 07:11 2 commits	2s master	16s master	5s master	5s master
#22 Mar 03 04:11 2 commits	2s master	18s master	5s master	4s master
#21 Mar 03 01:11 2 commits	2s master	17s master	5s master	4s master

Figure 2. Real-time examples considering build and deployment time of microservices


File Edit Selection View Go Run Terminal Help
ec2-user [SSH: Jenkins] [Administrator]
EXPLORER
EC2-USER [SSH: JENKINS]
.cache
.config
.kube
.local
.ssh
.vscode-server
microservices-with-MySql-db
.mvn
docker
docs
jenkins
prepare-tags-ecr-for-staging-docker-images.sh
k8s
spring-petclinic-admin-server
spring-petclinic-api-gateway
spring-petclinic-config-server
spring-petclinic-customers-service
spring-petclinic-discovery-server
spring-petclinic-hystrix-dashboard
spring-petclinic-vets-service
spring-petclinic-visits-service
templates
.editorconfig
.gitignore
.travis.yml
docker-compose.yml
prepare-tags-ecr-for-staging-docker-images.sh
microservices-with-MySql-db > jenkins > prepare-tags-ecr-for-staging-docker-images.sh
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
bash

```
MVN_VERSION=$(. ${WORKSPACE}/spring-petclinic-admin-server/target/maven-archiver/pom.properties && echo $ve
export IMAGE_TAG_ADMIN_SERVER="${ECR_REGISTRY}/${APP_REPO_NAME}:admin-server-staging-v${MVN_VERSION}-b${BU
MVN_VERSION=$(. ${WORKSPACE}/spring-petclinic-api-gateway/target/maven-archiver/pom.properties && echo $ver
export IMAGE_TAG_API_GATEWAY="${ECR_REGISTRY}/${APP_REPO_NAME}:api-gateway-staging-v${MVN_VERSION}-b${BUILD
MVN_VERSION=$(. ${WORKSPACE}/spring-petclinic-config-server/target/maven-archiver/pom.properties && echo $
export IMAGE_TAG_CONFIG_SERVER="${ECR_REGISTRY}/${APP_REPO_NAME}:config-server-staging-v${MVN_VERSION}-b${
MVN_VERSION=$(. ${WORKSPACE}/spring-petclinic-customers-service/target/maven-archiver/pom.properties && ech
export IMAGE_TAG_CUSTOMERS_SERVICE="${ECR_REGISTRY}/${APP_REPO_NAME}:customers-service-staging-v${MVN_VERSI
MVN_VERSION=$(. ${WORKSPACE}/spring-petclinic-discovery-server/target/maven-archiver/pom.properties && echo
export IMAGE_TAG_DISCOVERY_SERVER="${ECR_REGISTRY}/${APP_REPO_NAME}:discovery-server-staging-v${MVN_VERSION
MVN_VERSION=$(. ${WORKSPACE}/spring-petclinic-hystrix-dashboard/target/maven-archiver/pom.properties && ech
export IMAGE_TAG_HYSTRIX_DASHBOARD="${ECR_REGISTRY}/${APP_REPO_NAME}:hystrix-dashboard-staging-v${MVN_VERSI
MVN_VERSION=$(. ${WORKSPACE}/spring-petclinic-vets-service/target/maven-archiver/pom.properties && echo $ve
export IMAGE_TAG_VETS_SERVICE="${ECR_REGISTRY}/${APP_REPO_NAME}:vets-service-staging-v${MVN_VERSION}-b${BU
MVN_VERSION=$(. ${WORKSPACE}/spring-petclinic-visits-service/target/maven-archiver/pom.properties && echo $
export IMAGE_TAG_VISITS_SERVICE="${ECR_REGISTRY}/${APP_REPO_NAME}:visits-service-staging-v${MVN_VERSION}-b$
export IMAGE_TAG_GRAFANA_SERVICE="${ECR_REGISTRY}/${APP_REPO_NAME}:grafana-service"
export IMAGE_TAG_PROMETHEUS_SERVICE="${ECR_REGISTRY}/${APP_REPO_NAME}:prometheus-service"
```

```
ec2-user@jenkins-server ~:$
```

Figure 3. A pipeline script to build the staging docker images

3.2. Latency in monolith and microservices

The digital transformation or modernization of legacy applications from a monolith architecture to a microservice architecture is getting increasingly popular in recent times. However, to gauge the real benefits of this, migration needs to be evaluated depending on certain parameters. The objective of this evaluation is to benchmark one of the critical factors, the latency of the services, in two target environments. An evaluation technique can be used to create andanalyze a few sets of parameters like latency. It is required to benchmark the monolith and microservice-based systems. The measurement of the latency can be derived by differentiating the time factor right from initiating the HTTP request from the client application until the client application receives the HTTP response. The target number must be towards the lower side in terms of latency, which would signify the better performance of the system.

Figure 3 delineates a script to construct the arranging docker pictures labeled for the elastic container registry (ECR) registry and title it as “build-staging-docker-images-for-ecr.sh” and spare it beneath “Jenkins” envelope, as appeared over. The oversteps outline a normal stream of mechanized sending-in activity. It highlights how, with the proper apparatuses and setups in put, microservices can be persistently coordinated and sent with negligible manual intercession, guaranteeing quicker, more dependable discharges.

Figure 4 shows how the latency increased drastically in the monolith system for a range of 500 to 2500 HTTP calls per second which is very small and incapable in the contemporary business world where an average of trillion transactions are served every day. However, to the contrary in microservices-based systems, the variation in the latency is insignificant. Microservices come with inherent support to configuration for spinning up the instances on the fly to cater to the incoming load. The low latency is one of the significant attributes of any modern-day application and provides an edgeover monolith applications. The true nature of microservices is stateless and ultra-low latency microservices help to achieve the business objectives of the organization. The key factor in designing the low latency microservices is that developers should not introduce any blocking operations that take a significant amount of time. Also, the target approach should involve reducing unnecessary network trips, disk I/O usage, and any internal system calls [13].

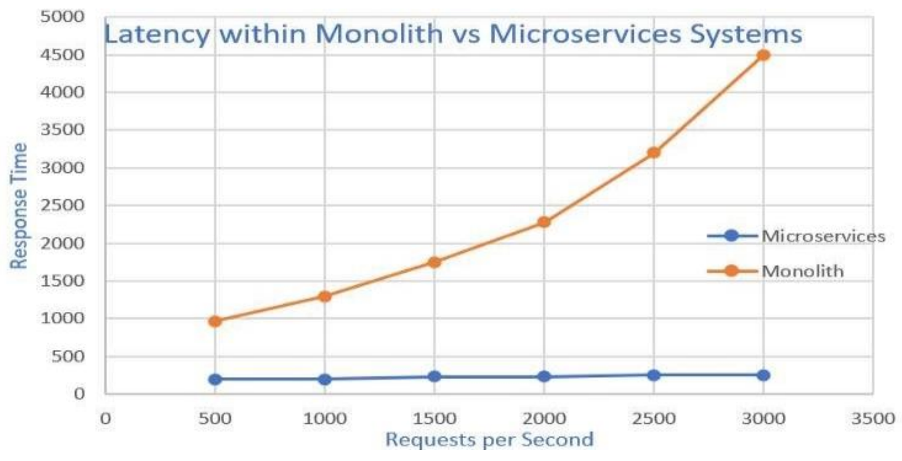


Figure 4. Latency statistics between monolith versus microservices based system

4. ANALYSIS OF THE EXPERIMENTATION

Jenkins, Prometheus, and Jmeter, the industry-recommended software, were installed. The total build and deployment time and infrastructure readiness software were calculated using Jenkins while Prometheus for latency. Ten tiny microservices were first created using Jenkins jobs, which were then executed. Ten microservices are found to take a total of 7 seconds to create and deploy at containers, while ten monolithic systems require about 260 seconds to develop and publish to the application server. It demonstrates how much faster and shorter the build and deployment times of microservices are than those of monolithic applications. When the number of microservices was extended to 20, 30, and eventually 50 later on, the build and deployment time was recorded as being 60 seconds, while when the same count was increased to monolith applications, the result was 1280 seconds. This demonstrates that having more microservices has no effect on build and deployment times and that microservices are suitable alternatives. With the help of JMeter, virtual users were created and requests were sent to microservices and monolith applications. Initially 500 requests per second were sent to monolith and microservices and it is observed that monolith applications took 1280 milliseconds whereas microservices took 122 milliseconds which is almost 10 times faster than monolith applications. The requests per second were increased gradually from 500 to 1000 and it is observed that microservices are taking almost the same amount of response time whereas monolith applications are taking 1420 milliseconds. While increasing more requests per second upto 3000 requests per second, it impacted the response time of monolith applications but microservices stays the same. It is clear that microservices are rarely impacted in response time with respect to requests per second.

5. RESULTS AND DISCUSSION

The program code construct and sending time and inactivity in solid and microservices are two DevOps standards that have been tested and conveyed in this work in an attempt to migrate from a bequest based solid framework to an impeccable distributed microservices based framework. Some real-time projects were taken to measure the performance and the time taken by doing experiments and it is observed that microservices are loosely coupled and due to its nature they are way faster and easily breakable over monolith systems. The measurement criteria was very simple based upon its build time and time-to-deploy which is faster and deployment time is very less due to their simple architecture and less resources and space. Very small real time projects were picked up with almost 50 independent microservices. These were built on continuous integration/continuous delivery/deployment (CI/CD) platform using Jenkins and after deployment on docker based Kubernetes and pivotal cloud foundry (PCF) containerized ecosystem, it was observed that only 90 to 95 seconds were totally consumed to build and deploy the microservices for 50 projects which includes the build time which is 20 seconds and deployment time including restarting of pods in Kubernetes after successful deployment. Whereas it has been observed that monolith systems are heavy in their volume and since they are so big in size and tightly coupled together that in case of any minor issue or bug the whole monolith application got failed during the build time and some of them were not ready to be deployed on servers due it its high demanding resource requirement and large space to get into repositories.

They almost consumed almost 1180-1260 seconds. Build and deployment time is way less in microservices compared to monolith.

It is observed that microservices architecture is better in all aspects. They provide faster build and deployment time. Their latency is less than monolith applications and their response time is faster than monolith applications. The architecture of microservices has to be designed to make the deployed application better scalable as they consume fewer resources which indirectly makes them to be infrastructure ready with respect to *time-to-market* approach.

6. CONCLUSION

Microservices are way more convenient to scale up and compile. From a software development cultural perspective, a small development team can also run a daily release schedule with microservices as they are easy to be automated with Jenkins and Kubernetes and can easily run on the daily scheduler. Scaling is only relevant over a certain amount of usage. If the business products get a million hits per second, microservices are better as they are completely sufficient to give a faster response time. Microservices are not the cure-all for all development problems, but they are strong candidates in current modern development because of their *time-to-market* feature. While microservices are a group of loosely coupled systems that collaborate to construct a bigger application, a monolithic application is a single, strongly coupled system. Compared to a monolithic architecture, microservices provide greater flexibility and scalability, but they can also be more difficult to create and manage. Microservices and monolithic architectures both assist developers in creating apps using various strategies. It's critical to realize that an application's complexity does not decrease with the use of microservices. Rather, the microservices architecture makes hidden complexity visible and makes it easier for developers to create, oversee, and grow massive applications. The improved scalability, agility, productivity, cost-efficiency, and fault tolerance are absent from monolithic architecture. Although they are inexpensive to create initially, monoliths are costly to scale. Conversely, microservices have a high initial cost, but because of their scalability, they end up being a more affordable option over time. The high availability of mission-critical business applications running in real time is ensured by the ability to adjust individual microservices without affecting the overall program.

REFERENCES

[1] C. Kolassa, D. Riehle, and M. A. Salim, "A Model of the Commit Size Dist. of Open Source," *Proc. SOFSEM'13*, vol. 7741, pp. 52–66, 2013, doi: 10.1007/978-3-642-35843-2_6.

[2] Q. Hou, Y. Ma, J. Chen, and Y. Xu, "An Empirical Study on Inter-Commit Times in SVN," *Proc. S/w Engg. and Know. Engg.*," pp. 132–137, 2014.

[3] S. Vaidya, S. Torres-Arias, R. Curtmola, and C. Cappos, "Commit Signatures for Cen. Version Control Systems," *Proc. IFIP Adv. in Infor. and Comm. Tech.*, vol. 562, pp. 359-3732019, doi: 10.1007/978-3-030-22312-0_25.

[4] M. Clemencic, B. Couturier, J. Closier, and M. Cattaneo, "LHCb migration from Subversion to Git," *Journal of Physics: Conference Series*, vol. 898, pp. 1-4, 2017, doi: 10.1088/1742-6596/898/7/072024.

[5] V. Singh, M. Alshehri, A. Aggarwal, O. Alfarraj, P. Sharma, and K. R. Pardasani, "A holistic, proactive and novel approach for pre, during and post migration validation from subversion to git," *Computers, Materials and Continua*, vol. 66, no.3, pp. 2359-2371, 2021, doi: 10.32604/cmc.2021.013272.

[6] Á. M. Guerrero-Higueras et al., "Academic Success Assessment Through Version Control Systems," *Applied Sciences*, vol. 10, no. 4, p. 1492, Feb. 2020. doi:10.3390/app10041492

[7] A. Singh, V. Singh, A. Aggarwal, and S. Aggarwal, "Improving Business deliveries using Continuous Integration and Continuous Delivery using Jenkins and an Advanced Version control system for Microservices-based system," *2022 5th International Conference on Multimedia, Signal Processing and Communication Technologies (IMPACT)*, Aligarh, India, 2022, pp. 1-4, doi: 10.1109/IMPACT55510.2022.10029149.

[8] S. Mishra, S. K. Sharma, and M. A. Alowaidi, "Analysis of security issues of cloud-based web applications," *Journal of Ambient Intelligence and Humanized Computing*, vol. 12, no. 7, pp. 7051-7062, 2021, doi: 10.1007/s12652-020-02370-8.

[9] Y. Ma, Y. Wu, and Y. Xu, "Dynamics of open-source software developer's commit behavior: An empirical investigation of subversion," *SAC '14: Proceedings of the 29th Annual ACM Symposium on Applied Computing*, Korea, pp. 1171–1173, Mar. 2014, doi: 10.1145/2554850.2555079.

[10] O. Arafat and D. Riehle, "The Commit Size Distribution of Open Source Software," *2009 42nd Hawaii International Conference on System Sciences*, Waikoloa, HI, USA, 2009, pp. 1-8, doi: 10.1109/HICSS.2009.421.

[11] I. Zaikin and A. Tuzovsky, "Owl2vcs: Tools for Distributed Ontology Development" *in OWLED*, Citeseer, vol. 1080, 2013.

[12] D. I. Hillmann, G. Dunsire, and J. Phipps, "Versioning vocabularies in a linked data world," *International Journal of Semantic Computing*, 2014.

[13] L. Halilaj, I. Grangel, G. Coskun, S. Lohmann and S. Auer., "Git4Voc: Collaborative vocabulary development based on Git," *International Journal of Semantic Computing*, vol. 10, no. 2, pp. 167–191, 2016, doi: 10.1142/S1793351X16400067.

[14] A. Singh, V. Singh, A. Aggarwal, and S. Aggarwal, "Event Driven Architecture for Message Streaming data driven Microservices systems residing in distributed version control system," *2022 International Conference on Innovations in Science and Technology for Sustainable Development (ICISTSD)*, Kollam, India, 2022, pp. 308-312, doi: 10.1109/ICISTSD55159.2022.10010390.

[15] V. Isomöttönen and M. Cochez, "Challenges and Confusions in Learning Version Control with Git," *Proc. Comm. in Comp. and Infor. Sc.*, vol. 469, pp 178-193, 2014, doi: 10.1007/978-3-319-13206-8_9.

[16] B. Firmenich, C. Koch, T. Richter, D.G. Beer, "Versioning structured object sets using text based Version Control Systems, *In*

Proceedings of the 22nd CIB-W78, Melbourne, 27–30 June 2022.

[17] A. Kaur and D. Chopra, "GCC-Git Change Classifier for Ext. and Classification of Changes in S/we Systems," *Lect. Notes in N/w and Systems*, vol. 19, pp. 259-267, 2018, doi: 10.1007/978-981-10-5523-2_24.

[18] Anguita, A. López-Ruiz, R. J. Segura-Sánchez, and A. J. Rueda-Ruiz, "A Version Control System for Point Clouds," Remote Sensing, vol. 15, no. 18. MDPI AG, p. 4635, Sep. 21, 2023, doi: 10.3390/rs15184635..

[19] V. Singh, A. Singh, A. Aggarwal, and S. Aggarwal, "DevOps based migration aspects from Legacy Version Control System to Advanced Distributed VCS for deploying Micro-services," *2021 IEEE International Conference on Computation System and Information Technology for Sustainable Solutions (CSITSS)*, Bangalore, India, 2021, pp. 1-5, doi: 10.1109/CSITSS54238.2021.9683718.

[20] N. Nizamuddin, K. Salah, M. Ajmal Azad, J. Arshad, and M. H. Rehman, "Decentralized document version control using ethereum blockchain and IPFS," *Computers &amp; Electrical Engineering*, vol. 76. Elsevier BV, pp. 183–197, Jun. 2019. doi: 10.1016/j.compeleceng.2019.03.014.

[21] N. N. Zolkifli, A. Ngah, and A. Deraman, "Version Control System: A Review," *Procedia Computer Science*, vol. 135. Elsevier BV, pp. 408–415, 2018. doi: 10.1016/j.procs.2018.08.191.

[22] M. Waseem, P. Liang, and M. Shahin, "A Systematic Mapping Study on Microservices Architecture in DevOps," *Journal of Systems and Software*, vol. 170, p. 110798, 2020, doi: 10.1016/j.jss.2020.110798.

[23] da Costa and D. N. Nogueira, *Guidelines for Testing Microservice-based Applications*, Diss. Instituto Politecnico do Porto (Portugal), 2022.

[24] D. R. F. Apolinário and B. B. N de França. "A method for monitoring the coupling evolution of microservice-based architectures," *Journal of the Brazilian Computer Society*, vol. 27, no. 1, p. 17, 2021, doi: 10.1186/s13173-021-00120-y.

[25] S. Li *et al.*, "Understanding and addressing quality attributes of microservices architecture: A Systematic literature review," *Information and software technology*, vol. 131, p. 106449, Mar. 2021, doi: 10.1016/j.infsof.2020.106449.

BIOGRAPHIES OF AUTHORS

Alok Aggarwal is currently spearheading efforts as Professor (CSE), with University of Petroleum and Energy Studies, Dehradun (UK), India. Earned Ph.D. degree in Mobile Computing area from IIT Roorkee, Roorkee India in 2010, Master's degree (M.Tech.) in Computer Science and Engineering in 2001 and Bachelor's degree in Computer Science in 1995. Contributing over 20+ years in Teaching (CSE and IT) and as software engineer for 4 years. He has contributed 250+ research articles in various journals and conference procedings, authored 5 books with an H-index of 26. His areas of specialization are mobile computing, mobile ad-hoc networks, lightweight cryptography, and internet-of-things. He can be contacted at email: alok.aggarwal@ddn.upes.ac.in.

Vinay Singh is working as Sr. Manager Software and DevOps Engineering (Colorado, USA). Pursuing Ph.D. in CSE from UPES, India. Earned M.S. (CSIT) from Southern Arkansas University, USA. Total 16+ years of experience in Software development in IT and 2 years of experience as a lecturer at DIT University. Area of expertise is the version control system, DevOps, automation, data science, and IT migration. He can be contacted at email: vsbuild@mail.com.