EasyChair Preprint

№ 7748

Design and Verification of Cargo Management Processes Using Microservice-Based Software Architecture: a Case Study

Tolga Büyüktanïr, Handan Keçici May, Cunay Şebboy, Didem Yalçïn and Mehmet S. Aktaş

April 10, 2022

Design and Verification of Cargo Management Processes Using Microservice-Based Software Architecture: A Case Study

Tolga Büyüktanır, Handan Keçici May†, Cunay Şebboy†, Didem Yalçın†, Mehmet S. Aktaş**
*Computer Engineering Department, Yildiz Technical University, İstanbul,Türkiye
*{tolga.buyuktanir@std.,aktas@}yildiz.edu.tr
†R&D Office, MNG Kargo A.Ş. ,İstanbul,Türkiye
†{hmay,csebboy,dyalcin}@mngkargo.com.tr

***Abstract*—The rapid spread of electronic commerce increases the need for more sustainable, easily manageable, new integrations and possibly error tolerant, easily scalable and resource efficient software in the cargo and distributed structuring sector. In this study, microservice-based software architecture has been designed for cargo and distributed structuring sectors, and a prototype of it has been implemented. The performance of the prototype has been evaluated and superiority has been observed over the monolithic services.**

***Keywords*—cargo, microservice, docker container, kubernetes.**

I. INTRODUCTION

Logistics has become an integral part of the life in a constantly growing and changing world. As the use of electronic trade becomes widespread, the need for manned and unmanned transport increases day by day. For an organization providing service in this sector, it has become complicated to manage the process, to process and generate information from the data streaming from different resources and maintain the sustainability of the system. At the cargo transportation and distributed structuring sector, a cargo goes through many steps from its reception until its delivery to the customer. As shown in Figure 1, these steps may be either independent or dependent to each other.

Figure 1 includes the reception processes of cargo at the departure office in the first step, then delivery processes to departure hubs in the second step, the processes in destination hubs in the third step, and at the last step the transfer of the cargo to the destination branch office.

All these steps are managed by electronic applications and process monitoring is also is performed by these applications [1].

The cargo management and monitoring applications in today's cargo transportation and distributed structuring sector are generally verified by monolithic services (programming interface e.g.: WSDL based web services). When executed, such services do not respond to requirements of the sector in terms of various metrics such as scalability, performance, ease of portability, and easy maintainability.

Today, software architectures which can respond to these requirements are performed using microservice based software architectures instead of traditional monolithic systems. Systems developed through microservice based software architecture are freely distributable, easily scalable, having modular parts easily interoperable, easily portable, and easily maintainable systems.

While in monolithic services, the entire service should be updated and the refreshed version should go live in case any function is modified; in microservice based systems, this activity is being performed only at the microservice of the function to be changed. This is a solution that does not require the system to be stopped. As long as there is no change on a microservice communication interface, there is no need to modify other microservices which are related to that microservice [2]–[4].

The microservices are generally being run on application virtualization platforms. Application virtualization platforms (e.g.: Docker, Linux container etc.) are traceable, customizable, easily scalable, low-resource consuming virtualization environments. These platforms can be monitored in real time through the programming interfaces they provide and it is possible to detect immediate or potential attacks or errors on microservices working on the platform using collected data [5], [6]. The application virtualization platforms consume less resources than operating system virtualization platforms (virtual machines) and they run faster [7].

Each container in these platforms is a scalable unit and their proofing threshold desired from the distributed system. There are tools which allow real-time scalability and management and orchestration of containers in systems which use an pplication virtualization platform. Docker Swarm and Kubernetes technologies are examples of these tools [8].

In scope of this study; as depicted and detailed in Figure 1, we have performed a case study on designing and developing all the processes of transporting a cargo; based on a distributed architecture, capable of meeting the needs of the cargo transportation and distributed structuring sector using an innovative software architecture.

The software architecture suggested in scope of this study is a microservice based, scalable software architecture which has an error proofing feature and which uses application virtualization platforms. A prototype application of suggested software architecture was verified for the cargo transportation

and distributed structuring sector. To reveal the availability and responsiveness of the prototype application, performance and scalability tests were performed and positive results were obtained.

In this case study, Chapter II explains literature review, Chapter III describes the problem, Chapter IV presents the suggested software architecture, Chapter VI gives details on the developed prototype of the application, Chapter VI explains the test that we have conducted on the prototype application. Chapter VII summarizes the results and experiences obtained at the end of this study and discusses future work.

II. Literature Review

The aim of traditional monolithic architectures and corporate applications is to provide a high integration among heterogeneous data sources [9], [10]. And high integration is achieved through tight-coupling. We observe a number of monolithic service oriented architecture based systems in different domains [11]–[24]. Maintaining a high integration during transfer of a monolithic system to a new environment becomes difficult. However, the biggest challenge results from systems which are scalable horizontally within the tight-coupling [25].

The microservice architecture was introduced to solve the limited scalability problem of monolithic services. The microservices are small applications which can be individually distributed, individually tested and individually scalable and which have the capability of distributed programming [26]. The microservices should be functional and autonomous according to their domains. So, as long as there is no change on the communication interface, the change in a microservice does not affect other services [2], [3]. Thus, a loosly-coupled, flexible, agile service with high cohesion [4].

The microservices are defined as small autonomous services. Based on the limits of business these services are developed as part of the functionality. When developing the code, one must avoid that it grows up and become messy. The development of the microservice does not require dependence on any technology. That means, microservices can be written in different programming languages. All communication is performed through the network. In case the input and output are not changed when a microservice is developed or modified, the other microservices in communication are not affected by this change [2].

While the microservice architectures provide many advantages, they also involve some challenges. Anytime an error occurs, its reasons should be found in a timely manner and it should be responded very rapidly. This case is called error isolation. The health status of microservices should be checked continuously and a very good logging mechanism should be established. Automation is necessary to take appropriate measures in case of a potential error in related services. The microservices should be written as independent as possible and they should not affect each other in case of any change. Testing microservices is the most challenging subject on this architecture. Although microservice architectures can be tested by automation tools, they increase test complexity [27]. The essential motivation of microservice architecture is that it solves the scalability problem in monolithic systems. However the main problem is experienced during a version update, running special versions and submitting the requests arriving to the services to other services [2], [3], [27], [28].

The microservices are generally run on cloud platforms with light-weight application virtualization technologies. These technologies are faster than other virtualization methods and consume negligibly less resources [7]. The most widely used one of light-weight virtualization platforms is the docker [29] platform. Docker involves features and tools which might tackle application challenges of the microservice architecture [2], [5], [6], [26]. Docker contributes to the microservice architecture in requirements such as error isolation, portability, convenience for automation and security [27], [28]. Additionally, docker's resource utilization can be managed and monitored in real time [30]. Large companies such as New York Times, arxiv.org, Uber, paypal and ebay use microservice architecture with docker. In fact, paypal has achieved a developer efficiency of %50 after switching to microservice architecture. Some applications have achieved %10-%20 performance efficiency [31].

While docker application virtualization environment involves several advantages, it also brings benefits on scalability. However, as the number of application virtualization environments increase, the utilized resource rate also increases. The resources should be used efficiently, if the resource utilization continues to rise. On the other hand, in case resources are not used anymore, the resources should be updated in terms of saving them. In order to keep such complicated situations under control, a solution not depending on application virtualization environment should be provided [5], [6].

An orchestration mechanism is needed for resource management, resource planning and service management of docker containers. The orchestration mechanism must be able to automate resource updates, define and plan scales, and service management processes such as load balancing and container isolation should be supported. At the market, there are orchestration mechanisms such as Docker Swarm [32], Kubernetes [33], Apache Mesos [34] and Cattle [32]; however, as Kubernetes is the most successful one in management of complicated services in larger architectures Kubernetes is preferred [8]. If it is for different needs in different sized architectures, other tools may be considered according to their performance and capacities.

III. Problem Definition

As electronic commerce is used more widely in cargo transportation and distributed structuring sector, a new need for integration emerges. A cargo passes through several steps from its reception until its delivery to the customer in the departure office, departure hub, destination hub, and destination office. When the software architecture where the complicated chain of transactions with high frequency are managed through a monolithic architecture, the system inadvertently grows up vertically.

Vertical growth involves challenges such as scalability, manageability, distribution, etc. Meanwhile, as the used services are very interdependent, a small change to be made on the service might affect the entire service. To deploy a change made on a small service component or a small integration on

the service to the live system, first the entire service should be stopped, and then it should be re-deployed and restarted.

Today, the communication between applications used in each process of a cargo is developed using Service Oriented Architectures (SOA). Such SOA based systems are used in transporting high scale cargoes. For example, the company MNG Kargo which offers service in the cargo transportation and distributed structuring sector has managed a total of 74,343,435 cargo deliveries on a SOA based software system in the first 10 months of 2019. It is envisaged that this number will increase as the electronic trade becomes popular.

However, there are challenges in these SOA based systems, because a monolithic system is used and it does not respond to the requirements of the sector and because the system is not easily manageable.

In a monolithic system, there is a resource allocated to the entire system. The unnecessary congestion on a service which is open to internal or external use overloads the entire system beyond control. For example; whenever an electronic trade web site having less cargoes performs many cargo tracking queries on the system, the load on the system might increase in an unbalanced way. In such a case, while the resources might be used for an unimportant operation, the response time of some important services might take longer and fail sometimes. Some marketplace customers use some cargo services periodically more intensely. In times of intense use, difficulties are experienced in ordinary operations. Lack of any quotas in the use of services for marketplace customers leads to extra intensity.

In the cargo transportation and distributed structuring sector, continuous new integrations and updates on some components of the monolithic service architecture are necessary. After the new integration and update, the entire system should be stopped and restarted after the update in order to deploy the software. The stopping of the entire system during deployment, hinders the whole running cargo distribution process. This is another problem to be discussed.

In scope of this research, a distributed system-based software architecture is suggested which might provide solutions to problems due to widely used systems based on monolithic software architecture in management of cargo and transportation sector processes. The software architecture providing a solution to above mentioned problems are explained in the next chapter with details.

IV. Proposed Software Architecture

Figure 1 depicts the processes for a cargo received from the sender until its delivery to its receiver. We summarize the main process steps in a cargo's life cycle as follows: 1) Reception of Cargo, 2) Measuring-Weighing-Barcode Labeling, 3) Special Services and 4) Loading. After the cargo goes through the first stage; it is brought in the second stage to the departure transfer center. We specify the life cycle steps in the departure transfer center as follows: 1) Picking Ring Transactions, 2) Sorting, 3) Fast Manifest Reading and 4) Line Vehicle Loading. At the next step the cargo arrives at the destination hub by line vehicles. We define the life cycle steps in the destination transfer hub as follows: 1) Sorting, 2) Fast Manifest Reading, 3) Loading to Branch Office Vehicle and 4) Routing Ring Operations. After the third stage of a cargo's life cycle, the cargo is transferred to the destination branch office by the branch office vehicle. We group the last stage of the processes at the destination branch office as follows: 1) Pre-delivery Preparation, 2) Delivery and 3) Solution Center/Customer Satisfaction.

Figure 1: Cargo Life Cycle

Our suggested distributed architecture including the specified cargo transportation life cycle microservices are shown in Figure 2. 4 rectangles drawn with black lines in the figure represent cargo processes. The microservices which are likely to be used beginning from each process are indicated within these rectangles. In the suggested architecture, each microservice should run on a light-weight application virtualization platform. The numbers indicated next to the microservices representing the subprocess show the utilization frequency of the microservice. At how many copy levels each container should run is specified based on the frequency of microservices which are depicted in the architecture in Figure 2.

Orchestration, load balancing, resource management and horizontal/vertical scaling should be done on microservices written to manage the cargo processes. An orchestration tool is required for these operations to be positioned in the part shown by a blue rectangle as depicted in Figure 2.

V. Prototype Application

To demonstrate the usability of the software architecture introduced in the previous chapter, a prototype application was developed. The developed prototype application selected as a case study was used to manage and monitor the processes of the company MNG Kargo operating in the cargo transportation and distributed structuring sector.

The architecture of the prototype application created for cargo transportation and distributed structuring processes at the company MNG Kargo is presented in Figure 3. In this architecture, all components of the company MNG Kargo are running on application virtualization environments as a microservice. Application virtualization environments can be run on multiple servers with different locations. Resource management, planning and service management of application virtualization environments are provided through Kubernetes.

Black rectangles depicted at the top of Figure 3 represent the applications used by the company MNG Kargo. Kubernetes

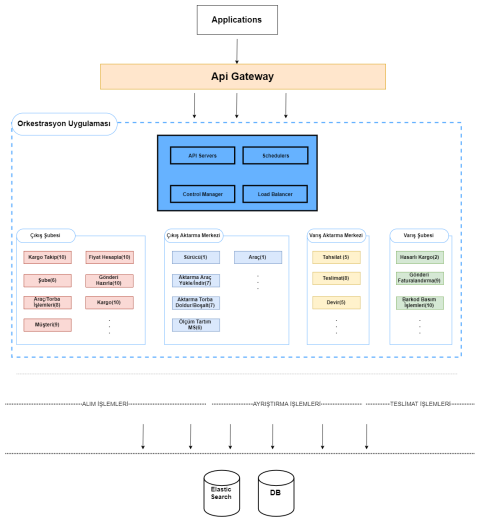

Figure 2: Suggested Application Architecture

cluster is shown by blue dotted lines in Figure 3. The requests from the applications are received by Kubernetes Ingress and directed to the services within the cluster and incoming responses are forwarded to the application. Load balancing is also done by Ingress.

Figure 3 depicts a server with 4 nodes. Within each node, there are docker containers. And there are microservices within docker containers. Colorful rectangles represent microservices. The rectangles include the subject of activity for the microservice and at the right-hand side of rectangles there is a numerical value indicating the importance of that microservice. The numerical value 10 indicates the mostly required microservice and the value 1 indicates the least required microservice.

Each node was established to manage a different process of the cargo. Node1 is used in managing the operations to receive the cargo from the customer, Node2 cargo for sorting operations and Node3 for managing the delivery processes of the cargo. Microservices established in Node4 are needed during these three processes. Therefore, these nodes shall be in communication with each other. Furthermore, all microservices shall be capable of linking to a single database and a distributed file system.

Management of resources for docker containers and nodes is performed by Kubernetes. The resource management is done by increasing or decreasing the CPU and memory amounts used by Docker containers and nodes. The resource management can be done by two methods, i.e. horizontal scaling (Horizontal Pods Autoscaler) and vertical scaling (Vertical Pods Autoscaler) by Kubernetes automatically. Scaling rules are specified, when Kubernetes Pods are installed. Kubernetes reads these rules every 30 seconds as default and if the rule applies, the horizontal scaling is performed immediately. A copy of Kubernetes Pod is created by horizontal scaling. Therefore, a copy of the docker container, thus a copy of microservice is created. Kubernetes follows the same way, when downwards scaling is performed. In vertical scaling, a process similar to horizontal scaling applies. Scaling rules are specified during setup. These rules are read once in 10 seconds and if the rule applies, then scaling is performed, and Kubernetes Pod is restarted.

When the nodes run at full capacity and a new pod (it may be considered as a new microservice within the docker) is to be created, cluster scaling (Cluster Autoscaler) is performed and the pod is instigated by creating a new node. However, these rules should also apply for this operation. For example, if there is a rule such as "Perform the operation, if 2 new pods are waiting in the queue" a new node is created when two pods are waiting in a queue. When node scaling is performed, the rules are checked every 30 seconds as the default setting. When any node waits for 10 minutes downscaling is performed.

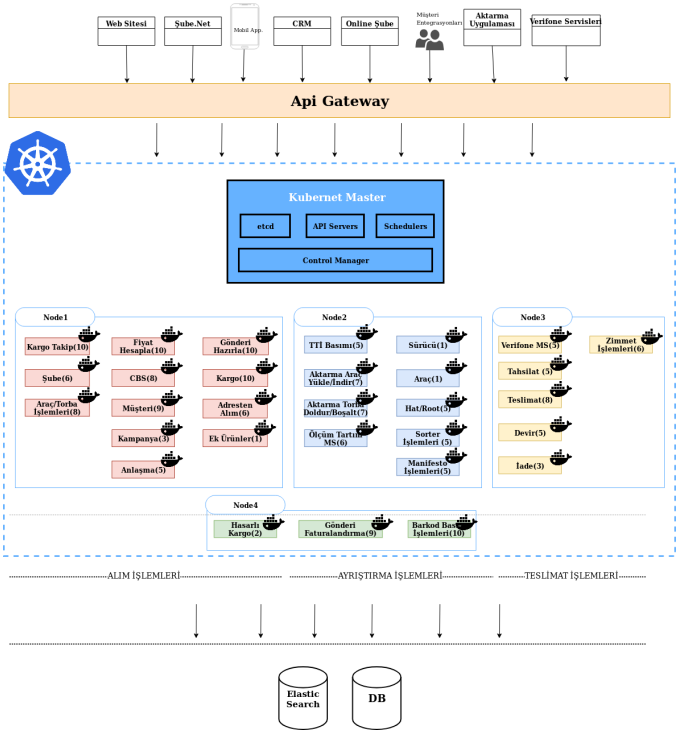

Figure 3: Prototype Application Architecture

VI. Performance Evaluation

In this chapter, experiments are explained for indicating the appropriateness of the suggested architecture in cargo transportation and distributed structuring sector. The experiments are for the performance of monolithic services and microservices. For the experiments, a flask application was written. In this application, $2^{1048576}$ mathematical operations are calculated, and the results of these operations are returned. Then, this application was run on the docker container and the images of containers were taken. This is the dummy microservice to be used in experiments.

In order to perform the experiments machines with 2 equal nodes and with a total 2vCPUs, 4 GB memory and 100 GB resource were leased from the service provider Digital Ocean and kubernetes orchestration tool were installed on them. To simulate a monolithic service, a docker container image within the dummy microservice which we have developed using flask was instigated on kubernetes. To test the monolithic system, Test-1 usage status was applied as indicated in Figure 4. To measure the simulated monolithic system performance,

2,4,8,16,32,64,128 and 256 requests were submitted asynchronously, and the response times were measured. The same performance measurement process was repeated by creating a copy of the container which includes the application as indicated in Figure 4 Test-2. Finally, as depicted at usage status in Figure 4 Test-3. The process was established so that there are 3 containers including the application and the response times to asynchronous requests were measured. The graphic showing response times to asynchronous requests in various numbers are provided for 3 different configurations in Figure 5.

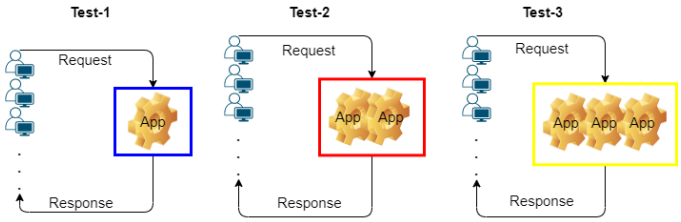

Figure 4: Test Cases Used for Experiments

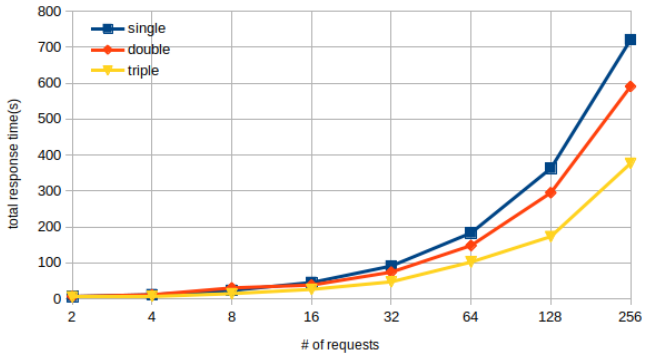

Figure 5: Experimental Results

VII. Results and Future Works

In scope of this study, a microservice based software architecture was suggested for the cargo transportation and distributed structuring sector. A prototype of suggested software architecture was verified, and some performance tests were done.

The effect of horizontal scaling and load balancing on performance was tried to be measured by the tests performed. We have observed that horizontal scaling has lowered the response time, as the number of asynchronous requests increased. Therefore, horizontal scaling has increased the performance, as the number of asynchronous requests increased.

Our future work shall include, artificial intelligence supported horizontal and vertical scaling, prevention of continuous scaling at overload such as ddos attacks and research and development activities which require learning from streaming data towards efficient use of resources.

Acknowledgement

We thank MNG Kargo Research and Development Center, which supported each step of this research during this case study.

References

[1] M. Khouja, "The evaluation of drop shipping option for e-commerce retailers," *Computers & Industrial Engineering*, vol. 41, no. 2, pp. 109–126, 2001.

[2] S. Newman, *Building microservices: designing fine-grained systems*. OReilly Media, Inc., 2015.

[3] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: yesterday, today, and tomorrow," in *Present and ulterior software engineering*. Springer, 2017, pp. 195–216.

[4] C. M. Aderaldo, N. C. Mendonça, C. Pahl, and P. Jamshidi, "Benchmark requirements for microservices architecture research," in *Proceedings of the 1st International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering*. IEEE Press, 2017, pp. 8–13.

[5] P. Hoenisch, I. Weber, S. Schulte, L. Zhu, and A. Fekete, "Four-fold auto-scaling on a contemporary deployment platform using docker containers," in *International Conference on Service-Oriented Computing*. Springer, 2015, pp. 316–323.

[6] M. Nardelli, "Elastic allocation of docker containers in cloud environments." in *ZEUS*, 2017, pp. 59–66.

[7] V. Gupta, K. Kaur, and S. Kaur, "Performance comparison between light weight virtualization using docker and heavy weight virtualization," *Mar*, vol. 2, pp. 211–216, 2017.

[8] I. M. Al Jawarneh, P. Bellavista, F. Bosi, L. Foschini, G. Martuscelli, R. Montanari, and A. Palopoli, "Container orchestration engines: A thorough functional and performance comparison," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–6.

[9] B. Gold-Bernstein and W. Ruh, *Enterprise integration: the essential guide to integration solutions*. Addison Wesley Longman Publishing Co., Inc., 2004.

[10] M. Chouinard, S. D'Amours, and D. Aït-Kadi, "Integration of reverse logistics activities within a supply chain information system," *Computers in Industry*, vol. 56, no. 1, pp. 105–124, 2005.

[11] M. S. Aktas, *Hybrid cloud computing monitoring software architecture*. Concurrency and Computation – Practice and Experience, Vol:30, Issue:21, 2019.

[12] M. S. Aktas and M. Pierce, *High performance hybrid information service architecture*. Concurrency and Computation – Practice and Experience, Vol:22, Issue:15, 2010.

[13] M. S. Aktas and et al., *XML metadata services*. Concurrency and Computation – Practice and Experience, Vol:20, Issue:7, 801-823., 2008.

[14] M. Pierce and et al., *The QuakeSim project: Web services for managing geophysical data and applications*. Pure and Applied Geophysics, Vol:165, Issue: 3-4, pp. 635-651., 2008.

[15] G. Aydin and et al., *SERVOGrid complexity computational environments (CCE) integrated performance analysis*. The 6th IEEE/ACM International Workshop on Grid Computing, 2005.

[16] M. S. Aktas and et al., *iSERVO: Implementing the International Solid Earth Research Virtual Observatory by Integrating Computational Grid and Geographical Information Web Services*. Pure and Applied Geophysics, Vol:163, Issue: 11-12, pp. 2281-2296, 2006.

[17] G. Aydin and et al., *Building and applying geographical information system Grids*. Concurrency and Computation: Practice and Experience, 20 (14), 2008.

[18] S. Oh and et al., *Mobile Web Service Architecture Using Context-store*. KSII Transactions on Internet and Information Systems, Vol: 4, pp: 836-858, 2010.

[19] M. A. Nacar and et al., *VLab: collaborative Grid services and portals to support computational material science*. Concurrency and Computation: Practice and Experience, 19 (12), 2007.

[20] M. S. Aktas and et al., *A web based conversational case-based recommender system for ontology aided metadata discovery*. The 5th IEEE/ACM International Workshop on Grid Computing, pp:69-75, 2007.

[21] M. Aktas and et al., *Fault tolerant high performance Information Services for dynamic collections of Grid and Web services*. Future Generation Computer Systems, Vol:23, Issue: 3, 2007.

[22] M. S. Aktas, *A Federated Approach to Information Management in Grids*. International Journal of Web Services Research, Vol:7, Issue:1, 2007.

[23] G. Fox and et al, *Grids for real time data applications*. Parallel Processing and Applied Mathematics, Vol:3911, Book Editors: Wyrzykowski, R and Dongarra, J and Meye, N and Wasniewski, J, 2006.

[24] G. Fox, M. Aktas, and et al, *Real time streaming Data grid applications*. Distributed Cooperative Laboratories: Networking, Instrumentation, and Measurements, Book Editors: Davoli, F; Palazzo, S; Zappatore, S, 2006.

[25] W. Hasselbring, "Microservices for scalability: keynote talk abstract," in *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering*. ACM, 2016, pp. 133–134.

[26] J. Thönes, "Microservices," *IEEE software*, vol. 32, no. 1, pp. 116–116, 2015.

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

[28] D. Jaramillo, D. V. Nguyen, and R. Smart, "Leveraging microservices architecture by using docker technology," in *SoutheastCon 2016*. IEEE, 2016, pp. 1–5.

[29] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.

[30] T. Buyuktanir and et al., *Provisioning System for Application Virtualization Environments*. International Conference on Computer Science and Engineering (UBMK-18), Invited Book Chapter in a Book: Big Data Analytics for Sustainable Computing, Editor: H. Anandakumar, IGI Global, Global Publications, Pennsylvania, pp.1-15, 2018.

[31] S. Johnston, A. E. Liberman, and A. Iankoulski, "The journey to 150,000 containers at paypal," Feb 2018. [Online]. Available: https://www.docker.com/blog/containers-at-paypal/

[32] R. Smith, *Docker Orchestration*. Packt Publishing Ltd, 2017.

[33] K. Hightower, B. Burns, and J. Beda, *Kubernetes: up and running: dive into the future of infrastructure*. OReilly Media, Inc., 2017.

[34] D. Kakadia, *Apache Mesos Essentials*. Packt Publishing Ltd, 2015.