Proposed Solution for Log Collection and Analysis in Kubernetes Environment

Josef Horalek, Patrik Urbanik, Vladimir Sobeslav(✉), and Tomas Svoboda

Faculty of Management and Informatics, University of Hradec Kralove, Hradec Kralove, Czech Republic
{josef.horalek,patrik.urbanik,vladimir.sobeslav,tomas.svoboda}@uhk.cz

Abstract. The aim of the paper is to design and verify a solution for collecting and analysing logs of a distributed application, which is operated as Software as a Service (SaaS) in the cloud environment in Kubernetes technology. Applications running in cloud environment are not monolithic in most cases, but consist of a large number of co-operating microservices. Providing logging for such distributed applications presents a complex issue, where to provide a comprehensive view of the application state, it is necessary to provide logging across all microservices representing the application. This paper first introduces modern approaches for application development using the technical means of virtualization, containerization and orchestration with an emphasis on Kubernetes technology. Next, approaches and analysis of application logging options are presented with the emphasis on the use of ELK and PLG stack technologies. Based on the analysis, a technical solution for logging applications in Kubernetes environment, operated in the form of SaaS, is proposed and verified.

Keywords: Kubernetes · container · virtualization · orchestrator · log · cloud · SaaS · PLG Stack · ELK stack

1 Introduction

Modern times place ever greater demands on applications in terms of computing power and the use of HW resources. This is mainly due to the increase in the number of users accessing these applications [1, 2] and using their services. A secondary reason, according to [3], is mainly the massive expansion of the Internet whose impact is the increasing number of users and connected devices. The above problem is solved by horizontal scaling, where the performance of HW resources is increased to run the application. Another solution is to use vertical scaling, where the application is run in multiple instances. These two approaches and especially the associated high financial and operation and maintenance costs are the main reasons for moving the applications in question to cloud solutions [4]. This is because cloud services make it very easy to scale, typically using GUI interfaces [5], and also the financial costs, since the application operator only pays

C. V. Phan and T. D. Nguyen (Eds.): ICTCC 2022, LNICST 473, pp. 9–22, 2023.
https://doi.org/10.1007/978-3-031-28790-9_2

for the resources it actually uses, without the need to purchase its own HW, including the provision of its management [6]. A major challenge in cloud-based application development nowadays is to ensure fast delivery of the target functionality. This requirement increases the demands on the human resources that develop the application. At the same time, with new functionalities, the amount of source code and thus the overall complexity of the application increases [7]. The need for frequent and fast delivery of the functionalities of a given application is the primary purpose of DevOps [8, 9]. Virtualization, containerization, and orchestrators are key technologies whose advent has been a catalyst for DevOps [10]. Despite the undeniable changes in the approach to software development, it remains a fact that developed applications have bugs in them whose manifestations and occurrences are random and unpredictable. It is imperative to monitor every application using logging, and at the level of application metrics and logs [11], where it must be taken into account that applications running in a cloud environment consist of a large number of cooperating microservices [12], but where logs are decentralized, as each microservice stores its logs separately. In order to provide a comprehensive view of the application behaviour, or a comprehensive view of the application logging, it is necessary to aggregate these logs from the individual microservices in one central location [13]. The above problem is addressed in several areas, influenced by sub-technologies, and mainly covers the use of ElasticSearch technology with subsequent provision of data for real-time analytics [14]. Logstash technology is often used for indexing and data normalization purposes [15]. Research in the area of downstream log analysis and the use of technical means for visualization is currently mainly focused on the use of Kibana [16]. All of these components allow communication with each other through APIs. Thanks to this architecture, any component can be replaced by another component provided that the new component supports the same interface [17]. At the same time, a major problem is the volatility of log information, e.g., due to restarts of compute nodes. A different view of cloud application monitoring concerns the architecture of a monitoring framework that is able to collect metrics not only from applications but also from system services. Metrics can be pushed from all types of services to the aggregator, where they are then streamed by processors [18].

1.1 Microservices, Containerisation and Orchestrators

Container virtualization is one of the main catalysts for designing applications using microservices. Microservices are built on two main pillars [19], the first is that a microservice addresses just one responsibility for a specific functionality and the second defines a microservice as a small application that can be deployed independently including independent scaling. The concept of leveraging microservices is key in the proper implementation of a DevOps approach. Distributed applications that are run in the cloud on a SaaS platform consist of many cooperating microservices [20]. In order to access microservices as packages of functionality, it is necessary to have a way to encapsulate the application and its configuration so that it can be migrated anywhere else. Container technologies or containerization are suitable candidates.

The principle of containerization is to encapsulate the application logic along with the configuration of the application into a minimized runtime environment that can then be easily deployed and operated [21].

Containers then do not need their own operating system, which results in lower hardware requirements and does not create memory and computational overhead. It is therefore possible to run more of them on hardware, and at the same time, they boot and restart much faster than traditional virtual machines. When using containers, the boot speed is 50× to 70× faster compared to standard operating systems [22]. Orchestrators are used to manage, deploy containers and support these processes [23].

Orchestrator is a system that provides an enterprise-level framework for container integration and management that simultaneously manages containers while allowing containers to be aggregated into a single entity, scaled, and comprehensively managed through their lifecycle. The Kubernetes orchestrator was developed by Google as an open source successor to an internal project called Borg and its successor project Omega [24, 25]. Several research teams are currently analysing the development of Kubernetes, including its features and deployment issues [26, 27]. To run the Kubernetes technology itself, it is necessary to have servers, their initial setup and subsequent management. The necessity of providing initial setup and subsequent management places great demands on the provision of hardware and especially human resources, to which end cloud providers offer managed Kubernetes cluster solutions that the customer uses as a cloud service. Currently, research in this area further focuses mainly on analysing the performance results of clusters [28], running applications in Kubernetes clusters [29], and ensuring high availability of applications [30].

1.2 Logging in Kubernetes

In order to provide logging in the Kubernetes environment, ELK stack and PLG stack technologies are mainly used nowadays [31]. ELK stack technology is a combination of Elasticsearch, Logstash and Kibana [32]. All these projects are backed by the main Elastic project. Elasticsearch is a database of data over which query-based search is implemented. Data is stored in indexes that are persisted to disk, over which queries are then executed. Elastic search then performs sorting and aggregation of the data. When data normalization and indexing is needed, the Logstash component is used. The normalized and indexed data is stored from Logstash to the Elasticsearch database. To visualize the data in the form of dashboards, the Kibana component in the ELK stack is used as a tool to support data manipulation [33].

PLG stack technology represents a combination of Promtail, Loki and Grafana projects. Currently, there is no research on the use of PLG stack for logging in Kubernetes. Loki technology represents the main component for dealing with persistence and querying log data. Compared to ELK stack, the approach to the problem of persistence and logging is different. The log data in Loki's submission is divided into two channels (index and blob). The indexes are used to store metadata about the log data. Blobs are pure logs, stored in their original format While in ELK stack the indexing of log files is done using Logstash technology, PLG stack provides the above functionalities through a single Loki technology. Promtail is a project primarily used for collecting data from servers. This collection is divided into three phases. The first phase is discovery, or discovering the targets from which data will be taken. The second phase involves the description of the collected information using labels. This extracted information is sent

to the Loki technology for further processing. Compared to Kibana, Grafana is a generic visualisation tool.

2 Problem Analysis

The examined solution used dedicated virtual servers of the cloud provider Microsoft Azure. In view of the increasing demands on CPU and memory, as well as the requirements for automatic scaling and dynamic creation of new services, it was decided to transfer the existing solution from virtual servers to Kubernetes based on a business analysis. In the context of the aforementioned migration of the runtime environment, there are also change requirements for the logging solution, which in the original solution relied on the stability of virtual machines and especially in the area of high availability.

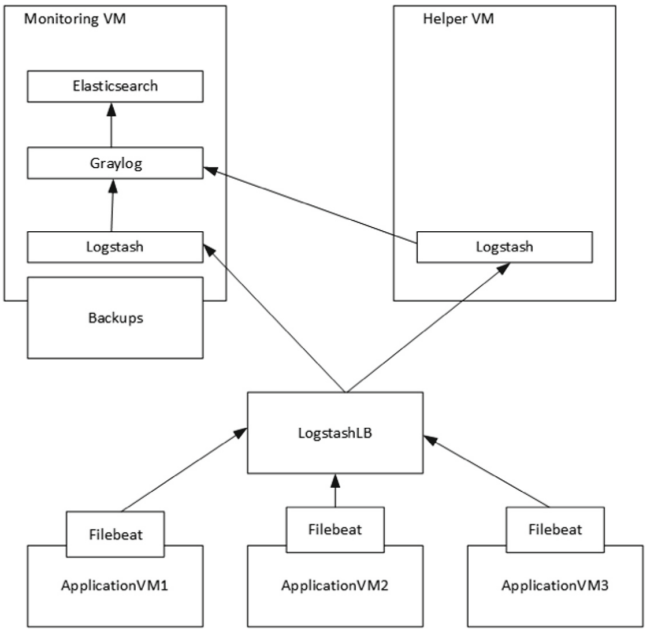

Fig. 1. Legacy logging solution

Figure 1 shows that only the Logstash component was optimized for high availability. The data on the virtual machine named monitoring was backed up once a day, and in case of failure of this virtual machine, a new one is started from this backup, and in the worst case scenario, one day of logs is lost. The original solution, ran on a single dedicated virtual server running all the necessary applications, including Elasticsearch, Logstah, Graylog. These components determined a minimum requirement of 64 GB RAM, 8 CPU cores and a 2 TB SSD drive for sufficient capacity to store the indexes. Logstash receives around 2,000 log lines per second, mediated by Filebeat, which collects this information from the output of docker containers running on all virtual machines. Thus,

high availability is out of the question in this solution. Minor failures of the Graylog tool to process incoming information are picked up by Logstash's retry mechanism. However, a failure of the entire monitoring server means an automatic loss of data.

2.1 Definition of Requirements

The requirements for a new solution can be divided into functional and non-functional requirements. From the analysis of the existing solution, the first functional requirement is the minimum number of processed log lines, set to 2 000. The second functional requirement is the possibility of alerting based on the information obtained from the logs. In the area of non-functional requirements, the main criteria for the new solution are stability and high availability. Outages of computing machines can be caused on the Azure side, for example, when moving a virtual machine running a Kubernetes worker node. Last but not least, the logging system must also provide decision support and support for easy issue tracking. This means that there must be the ability to perform event-specific queries over the data stored in the central repository, as well as statistical queries. Testing of the proposed will be carried out on an AKS cluster comprising three computing machines of type Standard_DS3_v2, i.e. machines having 4 CPUs and 14 GB of memory. The cost of the whole solution will be calculated on a running production cluster. The requirement for the **managed log volume** is defined with regard to the sustainability and development of the user base and the provided application portfolio specifies a threshold of 25,000 lines per second as a sufficient volume of processed logs. The system must be able to persist this volume of data and also be able to search over it. In the area of **high availability**, the requirements take into account the situation where a Kubernetes cluster is much more unstable than virtual machines. The system must therefore be prepared for virtual machine failure and must not lose data. At the same time, the system must be able to serve requests even if a node is unavailable, i.e. it must always appear to be fully functional externally. Requirements in the area of **Forensic Analysis and Statistical Queries**, the system must be able to provide support to developers as well as management. The developers will be particularly useful when they are looking for bugs in the application. Thus, the system must be able to provide the developer with data over which the developer will then be able to perform filtering and other refinements to the data they need. The system should also be able to perform statistical queries over the collected data for possible decision support. A typical example that will be tested is a query on the number of queries to an endpoint. In the area of **alerting**, given the number of running services in the Kubernetes cluster, the system must be able to provide support to the operations team for monitoring these services. This support should be represented by a message to some community channel if an error occurs in a service. Last but not least, the **cost** of the whole solution must be taken into account, which must be at most as expensive as the original solution was. The cost evaluation will be done on a long running Kubernetes cluster.

2.2 High Available PLG Stack Solution

Highly available Loki (used in version 2.1) solution represents for each part of the log processing and working process its own component always running in multiple replicas,

i.e. not only in one instance. At the highest level of abstraction, two log paths are addressed. The first is the write path that a log must travel from the moment it is recorded by the application until it is stored in some format. The second path is the reading path, which takes the log from the stored form to the visualized and filtered form required by the user. At a lower level of abstraction, these paths can then be broken down into the individual components that figure in these paths. A complete diagram of the cooperating components can be seen in Fig. 2. The individual components communicate with each other for maximum efficiency using an RPC implementation in the form of GRPC, an opensource RPC framework from Google.

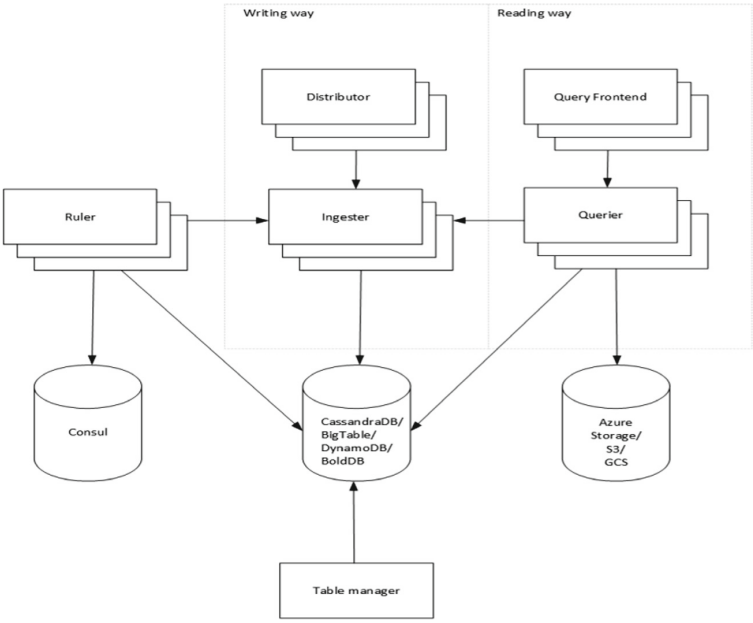

Fig. 2. HA Loki

2.3 Azure Kubernetes Services (AKS)

Azure Kubernetes Services is a managed service provided by Microsoft Azure cloud provider that takes care of the management of the running Kubernetes cluster, i.e. the operation and maintenance of the highly available master nodes on which the Kubernetes control plane runs. To make the logging ecosystem as resilient as possible to compute machine failures, it is necessary to ensure that there is always one pod of a given type running on a compute node, i.e., for example, each ingester runs on a different machine. This can be achieved using so-called anti-affinity. Anti-affinity places restrictions on Kubernet for scheduling individual pods.

For applications running in AKS as pods, logs will be collected from standard output. Thus, nothing special is needed in terms of application-level logging configuration. The

only major requirement for an application in terms of logging is to think about the correct labelling of the pod it is running in, so it is necessary to choose a mandatory set of labels for uniform search queries for all applications monitored by Loki. It is also advisable to consider the possibility that data from multiple Kubernetes clusters will be sent to a single Loki and possibly adapt the mandatory labels to this, i.e. introduce for example a label cluster that will describe the source cluster log information. A significant problem is where to store the chunks (logs in their textual, compressed form). AKS has two options, the first being mount storage, which in terms of Loki pods running inside a Kubernetes cluster, pretends to be a local disk. In terms of speed and cost, the optimal solution is to use Azure Files as local disks, even though there are real problems associated with Ingester component reboots, which in Kubernetes means that this disk is unmounted, and then remounted into a rebooted pod, which in extreme cases took up to an hour in tests. Another downside to this solution is that in the case of performance optimization, it is advisable to reach for the premium tier of Azure Files, which offers 5× the number of I/O operations, 4× the incoming traffic, and roughly the same outgoing traffic capability.

2.4 Alerting Solution

To support alerting, Loki comes with the Ruler component, which is able to analyse log information and trigger actions based on defined queries. To execute actions, a component that is familiar from the ecosystem around Prometheus is used. Specifically, it is the AlertManager component. This component takes care of deduplication, grouping and forwarding to the correct channel. The code sample shows a rule that will execute when the proportion of errors against all logs in a time-interval is greater than 5% and will execute once every 10 min. The handler is typically executed in a single replica and its theoretical unavailability does not matter. In the event that a large number of logs are expected to be evaluated over and alerting is a critical functionality, it is also possible to run Ruler in HA mode. In this mode, the Ruler needs information to access the hash ring through which individual Ruler instances exchange information.

```
1 groups:
2       - name: should_fire
3       rules:
4       - alert: HighPercentageError
5       expr: |
6       sum(rate({app="foo", env="production"} |= "error"[5m])) by (job)
7       /
8       sum(rate({app="foo", env="production"}[5m])) by (job)
9               > 0.05
10      for: 10m
11      labels:
12              severity: page
13              annotations:
14      summary: High request latency
```

3 Discussion and Results

The results were verified at the level of metrics collected by the Prometheus tool, for their visualization the Grafana tool was used, which also displays and visualizes logs.

3.1 Volume Test

In order to verify that the logging system can handle a relatively high load, a test was carried out in which 30 services were run in parallel, each of which will write approximately 900 records per second to its standard output, i.e. 27,000 processed lines per second in total, which is approximately 13 times more than the set lower limit. The service that takes care of writing the logs is a simple Node.js that outputs a sequence of numbers in an infinite loop, complete with the id of the running pod. During this test, Prometheus metrics of the components involved in the write path were monitored, from which it is possible to track how the system handles the load.

```
1 const process = require("process");
3 const ClientId = process.env["CLIENT_ID"];
4 (async function run() {
5 for await (let num of generateSequence()) {
6 console.log(‘${ClientId} - ${num}‘); }})();
7 async function* generateSequence() {
8 let counter = 0;
9 while (true) {
10 await tick();
11 yield counter++;16 }}
12 async function tick() {
23 return new Promise((res) => setTimeout(res, 1)); }
```

The Promtail component responded to the increase by increasing processor activity. It consumes about 2.5 CPU more to handle this amount of logs. The memory increase was essentially negligible compared to the CPU load. A substantial increase in network load can also be observed. As the number of processed rows increases, the amount of data required to be forwarded to the Distributor component also increases. This component reacted with a slight increase (0.1 CPU) in processor activity. The memory load remains constant. The increase in network load was significant. This was calculated as the result of the sum of the amount of data received and sent. Finally, a marked difference in memory usage can be observed on the Ingester component. This memory is used to temporarily store the logs so that queries can be processed in the shortest possible time. Based on the settings of the component, the data stored is 20 min old.

3.2 High Availability Test

In this test, a simulation of a machine failure was performed. This should verify the readiness of the system for problems of this type. The simulated outage was achieved by downscaling the nodepool by one machine, i.e. removing one node. Throughout the test, writes were performed at a rate of approximately 15,000 lines per second. To achieve this number of lines, the same program as in the previous test was used, only it will be run in fewer instances. The stability of the write was measured by a post-power promtail metric that tells how many log lines were discarded. First, the system will be put into an uncorrupted state, i.e., the number of replicas of the ingester com- ponents will be reduced to 1. Figure 3 captures this state will show the lost lines. Then the whole system will be restored to its original state followed by a simulated node failure. These two states will be compared against each other. In the first part of the graph, the lost log lines should be visible, while in the second part of the graph, no losses should occur. It was observed that a stable write of about 18,000 logs per second was in progress. At around 9:38 the system was put into a non-valid state by reducing the number of ingesters to 1. This resulted in a loss of logs and an increased number of Grafana query errors. At 9:45 the system was returned to a valid state and the graph shows that the error messages stopped appearing and the number of lost logs also dropped to 0. Around this time a local outlier can be observed on the graph showing the number of lines read. This reaches a threshold of 110,000 processed lines per second, which is successfully managed to clear. At about 9:49, the downscale of the number of worker nodes to 2 followed. This was reflected by reducing the number of Ingester instances and Distributor instances to 2, logically removing instances that were running on the removed compute node. This number is still valid for the system. Thus, there is no loss and Grafana can handle all queries, which implies that the log read path is fully functional. At approximately 10:01 the test was terminated.

Based on this test, we can say that the logging system is ready to run in the unstable Kubernetes environment due to its distributed nature and proper anti-affinity settings. This is a substantial improvement over the original solution, in which a failure would have meant momentary non-functionality and loss of logs. Furthermore, the power of Kubernetes in automating the whole process was shown here. After removing the compute machine from the cluster, Loki component instances that could not be deployed to the two remaining worker nodes disappeared, as deploying them to these machines would have violated the anti-affinity rules. However, the moment a newly started worker node appears in the cluster, Kubernetes automatically switches on all the missing components and everything runs again as it was originally.

Fig. 3. Logging in case of single node failure

3.3 Decision Support and Forensic Analysis

This test contained two parts. The first part focused on the ability of the system to provide relevant information, which is understandable and visualised, for the management of the company, on the basis of which the management must be able to decide where the application is able to go from here. The second part tested the ability of the system to provide relevant information to the developer, who should be able to trace the error based on this information. For the test, a REST service was used, which has three endpoints exposed, named for illustration Feature1, Feature2, Feature3. This service will then be called using a BASH script in a 5:3:1 ratio. At the same time, the application will log an error every 15th call to the Feature3 endpoint. From the results obtained, it can be seen that the primary functionality is Feature1, which represents 56% of all calls, which de-emphasizes the ratio in which the functionalities were called. Furthermore, a relatively high frequency of error logs can be observed from this dashboard, namely 0.062 errors per second. This value should be 0. Therefore, its elevated value could be a stimulus for further analysis of what is happening in the application. When the Feature3 endpoint is called, the current application state is 15, which triggers an error in the application. This test, by successfully verifying the distribution of calls to individual endpoints, proved that it is possible to extract useful information from the logs using LogQL for management, which can then be nicely visualized in graphs using Grafana. This visualization greatly helps to understand the data presented.

3.4 Alerting

In this test, the ability of the system, or rather the Ruler component, to alert when defined triggers occur in the application was verified. The same services as in the previous test were used for this test. For this service, it is known that every 15th call to the Feature3 endpoint an error is reported. This error was followed by an alert that is reflected by a message in the Microsoft Teams communication tool. Ruler itself only serves as a tool

that analyses the defined LogQL queries. If a query is evaluated as true, a notification is sent to the Alert Manager component. It is used to accumulate these notifications and then forward them to the selected communication channel. Alert Manager does not have a direct connector to Microsoft Teams. For this test, it is therefore necessary to choose an alternative solution and send the notification using the Webhook functionality. For this option, a target HTTP end-point is selected, to which a POST request is sent that contains the alert information in its body. Based on this information, Alert Manager then groups the alerts and sends them on if necessary. Microsoft Teams can receive messages using the Webhook mechanism. However, the exposed endpoint requires quite specific data, the structure of which can be seen in the Microsoft Webhook documentation.1 However, Alert Manager component cannot structure the data in this way. Therefore, it is necessary to use a proxy server that will receive data from Alert Manager and transform it into Microsoft Teams-compatible data. This component will be the open source project Prometheus-Msteams available from Github. 2 This setup is capable of sending alerts on errors from the application for the previous test. Even though the integration between Ruler component and Microsoft Teams was not straightforward, this test proved that alerting can work in a solution built on Loki project. By using the open source pro-projects AlertManager and Prometheus-Msteams, integration between the mentioned components was achieved. A big advantage is the definition of rules in a form that is similar to Prometheus rules. This greatly simplifies the operations team that already operates Prometheus to create these rules and use this functionality.

4 Conclusion

The proposed solution was rigorously tested and passed all tests successfully. Based on the test results, the solution can process more than 10 times the required lower bound and in one of the tests 110,000 lines were processed at one point. This solution is ready for future application growth and increasing number of processed lines. In terms of high availability, the solution passed a test in which a computing machine failure was simulated. Due to its distributed nature, the solution remained functional despite this outage and was able to handle 18,000 log lines per second without losing a single one. The moment the Kubernetes node in the cluster went down again, the system automatically returned to its original state without a single human intervention. The LogQL language is suitable for forensic and statistical log queries. The test was successful in demonstrating the capabilities of the system, how it will serve developers for log analysis as well as managerial positions as a possible sub-task for decision making. The data obtained by querying could be easily filtered or transformed into metrics. The response to bugs will be very fast thanks to the alerting support that was presented by one of the tests. Moreover, the rules for alerting are also written in LogQL and the user does not have to learn a new language or procedure to set up the alert. Furthermore, their definition is similar to Prometheus rule definition, making them easier for the operations team to write and maintain. Despite the fact that the solution could not be configured to write data directly to Blob Storage, but had to use the option of writing to Azure Files, the solution managed to beat the targets on the price test. Specifically, there was a 26% price reduction.

Acknowledgment. The research has been supported by the Faculty of Informatics and Management UHK specific research project 2107 Integration of Departmental Research Activities and Students' Research Activities Support II. We would like to thank Mr. P. Kratochvil, a graduate of Faculty of management and informatics, University of Hradec Kralove, for the practical verification of the proposed solutions and close cooperation in the solution.

References

1. Morley, J., Widdicks, K., Hazas, M.: Digitalisation, energy and data demand: the impact of Internet traffic on overall and peak electricity consumption. Energy Res. Soc. Sci. **38**, 128–137 (2018). ISSN 22146296
2. Abbasi, A.A., Abbasi, A., Shamshirband, S., Chronopoulos, A.T., Persico V., Pescape, A.: Software-defined cloud computing: a systematic review on latest trends and developments (2019). ISSN 2169-3536
3. Tranos, E., Stich, Ch.: Individual internet usage and the availability of online content of local interest: a multilevel approach. Comput. Environ. Urban Syst. **79**, 101371 (2020). ISSN 01989715
4. Villamizar, M., Garces, O., Castro, H., et al.: Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. In: Sanchez, M., Gonzalez, O. (eds.) 2015 10th Computing Colombian Conference (10ccc). IEEE, New York (2015). ISBN 978-1-4673-9464-2, iSSN 2378-8216
5. El Kafhali, S., El Mir, I., Salah, K., Hanini, M.: Dynamic scalability model for containerized cloud services. Arab. J. Sci. Eng. **45**(12), 10693–10708 (2020). https://doi.org/10.1007/s13369-020-04847-2. SSN 2193-567X
6. Piraghaj, S.F., Vahid Dastjerdi, A., Calheiros R.N., Buyya, R.: A survey and taxonomy of energy efficient resource management techniques in platform as a service cloud. In: Handbook of Research on End-to-End Cloud Computing Architecture Design. Advances in Systems Analysis, Software Engineering, and High Performance Computing (2017). ISBN 9781522507598
7. Senapathi, M., Buchan, J., Osman, H.: DevOps capabilities, practices, and challenges. In: Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering (2018). ISBN 9781450364034
8. Maroukian, K., Gulliver, S.R.: Leading DevOps practice and principle adoption. In: 9th International Conference on Information Technology Convergence and Services (ITCSE 2020) (2020)
9. Balalaie, A., Heydarnoori, A., Jamshidi, P.: Microservices architecture enables DevOps migration to a cloud-native architecture. IEEE Computer, Los Alamitos (2016). ISSN 0740-7459
10. Elbert, C., Gallardo, E., Hernantes, J.; DevOps. IEEE Computer Society, Los Alamitos (2016). ISSN 0740-7459
11. Pi, A., Chen, W., Zhou, X., Ji, M.: Profiling distributed systems in lightweight virtualized environments with logs and resource metrics. In: Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing (2018). ISBN 9781450357852
12. Jash, S., Ganesh, R., Rachhadia, T.D., Shah, P.K.: A hierarchical approach to extract application logs with visualization in a containerized environment. In: 2019 International Conference on Computing, Power and Communication Technologies (GUCON) (2019)
13. Solomon, F.I.: Securing websites web applications using data analytics. In: 2019 International Conference on Computational Intelligence in Data Science (ICCIDS) (2019)

14. Li, Y., Jiang, Y., Gu, J., et al.: A cloud-based framework for large-scale log mining through apache spark and elasticsearch. Appl. Sci. **9**(6) (2019. https://doi.org/10.3390/app9061114). ISSN 2076-3417. Accessed 09 Nov 2021
15. Lee, B.-H., Yang, D.-M.: A security log analysis system using Logstash based on apache elasticsearch. J. Korea Inst. Inf. Commun. Eng. **22**(2), 382–389 (2018)
16. Shonia, O., Topuria, N., & Kulijanovi, K. Collection and analysis of log data with cloud services. Bull. Georg. Natl. Acad. Sci. (2021)
17. Mfula, H., Nurminen, J.K.: Self-healing cloud services in private multi-clouds. In: 2018 International Conference on High Performance Computing & Simulation (HPCS), pp. 165–170. IEEE (2018)
18. Ramos, F., Viegas, E., Santin, A., Horchulhack, P., Dos Santos, R., Espindola, A.: A machine learning model for detection of Docker-based APP overbooking on kubernetes. In: ICC 2021 - IEEE International Conference on Communications (2021). ISBN 978-1-7281-7122-7
19. Pradhan, R., Dash, A.K.: An Overview of Microservices. Lecture Notes in Electrical Engineering, vol. 601. Springer, Singapore (2020)
20. Pahl, C., Jamshidi, P., Zimmermann, O.: Microservices and containers. Softw. Eng. **2020** (2020)
21. Srirama, S.N., Adhikari, M., Paul, S.: Application deployment using containers with auto-scaling for microservices in cloud environment. J. Netw. Comput. Appl. **160**, 102629 (2020). ISSN 10848045
22. Abdullah, M., Iqbal, W., Bukhari, F.: Containers vs virtual machines for auto-scaling multi-tier applications under dynamically increasing workloads. In: Bajwa, I.S., Kamareddine, F., Costa, A. (eds.) INTAP 2018. CCIS, vol. 932, pp. 153–167. Springer, Singapore (2019). https://doi.org/10.1007/978-981-13-6052-7_14 ISBN 9789811360527, ISSN 1865-0929
23. Zhang, Q., Liu, L., Pu, C., et al.: A comparative study of containers and virtual machines in big data environment. New York (2018). ISBN 978-1-5386-7235-8
24. Khan, A.: Key characteristics of a container orchestration platform to enable a modern application. IEEE Cloud Computing (2017). ISSN 2325-6095
25. Muddinagiri, R., Ambavane, S., Bayas, S.: Self-hosted kubernetes: deploying docker containers locally with minikube. In: 2019 International Conference on Innovative Trends and Advances in Engineering and Technology (ICITAET) (2019)
26. Dewi, L.P., Noertjahyana, A., Palit, H.N., Yedutun, K.: Server scalability using kubernetes. In: 2019 4th Technology Innovation Management and Engineering Science International Conference (TIMES-iCON) (2019). ISBN 978-1-7281-3755-1
27. Ferreira, A.P., Sinnott, R.: A performance evaluation of containers running on managed kubernetes services. In 2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom) (2019). ISSN 2330-2186
28. Vayghan, L.A., Saied, M.A., Toeroe, M., et al.: Deploying microservice based applications with kubernetes: experiments and lessons learned. In: 2018 IEEE 11th International Conference on Cloud Computing (CLOUD) (2018). ISSN 2159-6190
29. Vayghan, L.A., Saied, M.A., Toeroe, M., et al.: Microservice based architecture: towards high-availability for stateful applications with kubernetes. In: 2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS) (2019)
30. Pichan, A., Lazarescu, M., Soh, S.T.: Towards a practical cloud forensics logging framework. J. Inf. Secur. Appl. **42**, 18–28 (2019). ISSN 22142126
31. Jayathilaka, H., Krintz, C., Wolski, R.: Performance monitoring and root cause analysis for cloud-hosted web applications. Association for Computing Machinery, New York (2017). ISBN 978-1-4503-4913-0

32. Lamouchi, N.: Flying All Over the Sky with Quarkus and Kubernetes. In: Lamouchi, N. (ed.) Pro Java Microservices with Quarkus and Kubernetes, pp. 363–395. Apress, Berkeley, (2021). https://doi.org/10.1007/978-1-4842-7170-4_14 ISBN 978-1-4842-7169-8
33. Dooley, R., Brandt, S., Liang, K., Tanner, E. Experiences migrating the agave platform to a kubernetes native system on the jetstream cloud. In: Practice and Experience in Advanced Research Computing, 17 July 2021, pp. 1–4. ACM, New York (2021). https://doi.org/10.1145/3437359.346559. ISBN 9781450382922. Accessed 09 Nov 2021