Anomaly Detection in Microservice Systems Using Autoencoders

Manul de Silva
Computer System Engineering
Sri Lanka Institute of Information Technology
Malabe, Sri Lanka
it19197456@my.sliit.lk

Sam Daniel
Computer System Engineering
Sri Lanka Institute of Information Technology
Malabe, Sri Lanka
it19950532@my.sliit.lk

Manith Kumarapeli
Computer System Engineering
Sri Lanka Institute of Information Technology
Malabe, Sri Lanka
it19196084@my.sliit.lk

Sashika Mahadura
Computer System Engineering
Sri Lanka Institute of Information Technology
Malabe, Sri Lanka
it19003306@my.sliit.lk

Lakmal Rupasinghe
Computer System Engineering
Sri Lanka Institute of Information Technology
Malabe, Sri Lanka
lakmal.r@sliit.lk

Chethana Liyanapathirana
Computer System Engineering
Sri Lanka Institute of Information Technology
Malabe, Sri Lanka
chethana.l@sliit.lk

2022 4th International Conference on Advancements in Computing (ICAC) | 979-8-3503-9809-0/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICAC57685.2022.10025259

***Abstract*—The adaptation of microservice architecture has increased massively during the last few years with the emergence of the cloud. Containers have become a common choice for microservices architecture instead of VMs (Virtual Machines) due to their portability and optimized resource usage characteristics. Along with the containers, container-orchestration platforms are also becoming an integral part of microservice-based systems, considering the flexibility and scalability offered by the container-orchestration media. With the virtualized implementation and the dynamic attribute of modern microservice architecture, it has been a cumbersome task to implement a proper observability mechanism to detect abnormal behaviour using conventional monitoring tools, which are most suitable for static infrastructures. We present a system that will collect required data with the understanding of the dynamic attribute of the system and identify anomalies with efficient data analysis methods.**

Keywords— Microservices, Kubernetes, container, eBPF, Autoencoders, LSTM

I. Introduction

Micro-service has emerged as the most widely used system architecture for deploying applications in the recent past [1]. The rapid development of cloud technologies has been a massive catalyst for the growth in the adaptation of microservice architecture. The microservice architecture, along with the cloud-native deployment, has become the most popular distributed system design used for modern applications in the industry as it offers agility, reliability, scalability, and fine-grain control over the application elements in the system [2]. Leading tech companies such as Amazon and Netflix have already adopted this new microservice architecture [1]. Further, leading cloud service providers have also started providing managed services for containerized applications, which makes deploying microservices applications easier. Even though the microservice architecture and the cloud have eased the management of applications, the infrastructure has become very complex involving several virtual elements, which have become a considerable challenge for the observability of the microservice infrastructure [2]. The conventional monitoring tools are insufficient to build an observability mechanism due to their limitation in extracting advanced metrics such as in-kernel events, which are necessary for analyzing deeper problems in dynamic systems [3]. Along with that limitation, the existing monitoring tools are facilitated only to monitor host-related events, and they are not equipped to track the internal behaviours of each container in the micro-service systems. The Lack of proper data collection and analysis mechanisms is a huge threat to security, and it will be harder to identify other unknown underlying abnormalities within the system. Therefore, a proper observability mechanism is essential for the safety and management of such a dynamic and distributed system.

As a solution for the problems regarding observability in microservice systems, we are proposing an efficient anomaly detection system, which can collect the required advanced in-kernel parameters from the system by understanding the dynamicity of the system with the details extracted from the container orchestration platform and the container run-time of the microservice-based system. Since Kubernetes is the widely used container orchestration platform, our system is built based on the micro-service system implemented in Kubernetes. We have used eBPF tools to extract the advanced in-kernel metrics with less overhead efficiently. Along With the efficient data-collecting mechanism, another critical aspect of this anomaly detection system is the data analysis component which can identify abnormal behaviors efficiently. Recently, the data analysis sector has rapidly grown with the advancement in AI technologies. We have also equipped advanced data analysis

979-8-3503-9809-0/22/$31.00 ©2022 IEEE

methods involving Machine Learning to increase the efficiency of the anomaly detection system. The following section will provide an overview of the leading technologies involved in this research. Section III reviews previous projects that have addressed problems related to our research area. Section IV contains the methodology of the proposed system. The Methodology section will provide a deeper understanding of the system. And finally, section V has our proposed system's experiments and results.

II. Background

A. Microservices

Earlier, the software systems were built on monolithic architecture. Along with the drastic growth of the IT industry and the involvement of virtualization in the infrastructure, monolithic architecture was deemed insufficient to meet the modern requirements raised due to the rapid growth in the number of users and users' need for a higher quality of services. Along with the emergence of the cloud and the popularity of cloud-native architecture, microservice-based architecture has also evolved as a replacement for the monolithic architecture in software systems by solving the issues in a monolithic architecture. Many leading companies worldwide, including Microsoft, Amazon, Google, and Netflix, have started utilizing microservice architecture [1]. Microservices are atomic, autonomous services that work together to make up a whole distributed system, and they often communicate via an API (Application Programming Interface) [4]. The main characteristic of the microservice architecture is that the business behaviours of the system are divided into small services decoupled from each other to work independently to provide flexibility and scalability when developing and deploying the components of the system. With features such as flexibility, availability, and scalability, the microservice architecture has become well-suited for cloud-native environments. It has become a widely used architecture when deploying applications in the cloud.

B. Containers and Container Orchestration

A container is a remote executable unit with software components with all the necessary dependencies. Containers can be considered a more abstract version of VMs. The containers are more portable and lightweight when compared to VMs. The containers have become a common choice for application deployment in microservice architecture instead of VMs for their portability and optimized resource usage characteristics. Container orchestration has also become a standard part of modern architecture. Container orchestration is containers' automated management in containers' deployment, management, scaling, and networking of containers [5]. Container orchestration plays a huge part in providing dynamicity, scalability, and flexibility in modern microservice-based systems by enabling CI/CD and DevOps practices during development. Kubernetes, a container orchestration platform, has also become a prominent element in the contemporary microservice architecture for its various features to manage containers. Kubernetes has impressive features such as automated rollouts and rollbacks, horizontal scalability, and the capability of self-healing [6].

C. eBPF

eBPF is a recently emerged kernel technology that has made its path into the industry with the emergence of cloud-native observability. The basic functionality of eBPF is that it allows the user-written code to run without requiring additional modules or modifying the kernel source code. eBPF can be considered a lightweight, sandboxed virtual machine (VM) that resides in the Linux kernel, allowing programmers to run Berkeley Packet Filter (BPF) bytecode that uses specific kernel resources. Even though BPF started as a network packet filtering mechanism, eBPF has evolved as a tracing tool across all stacks of a Linux operating system. eBPF can trace all system calls and provide packet and socket-level views of all networking operations [7]. It is used in cloud computing for its use cases in security, observability, and network optimisation [8]–[10].

D. Anomaly Detection with ML

Anomaly detection is a data analysis process used to identify unusual behaviour that deviates from most of the data in a data set [11]. With the exponential growth of data due to the enormous increase in the IT industry, more sophisticated methods have emerged for analyzing more data. ML is a technology that has been used in many industries to leverage the power of data. Machine learning also can be utilized for anomaly detection purposes. ML is used in various anomaly detection applications such as system monitoring, fault detection, and fraud detection. Autoencoders is an unsupervised Ml algorithm used for dimension reduction and learning data set representations [12]. Autoencoder consists of two components. An encoder learns the underlying features of a process, and a decoder recreates the original form of the data using the elements in the reduced dimension state. When recreating the initial state, autoencoders will have reconstruction loss. Reconstruction loss can be used in anomaly detection systems as the dimension-reduced feature for identifying the anomalous instances with the anomalous reconstruction loss [12].

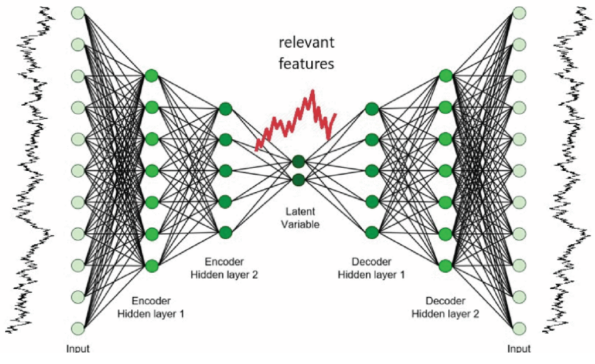


Fig. 1: Example of autoencoder function [11]

III. Related Work

In recent years, several kinds of research have focused on distributed systems observability and security issues. We will review some of the papers very close to the problem we are trying to solve, especially the research which focuses on the data collection in micro-service systems and the research which uses ML techniques to improve the security of micro-service systems.

Work done under kmon [13] mainly focuses on deriving the real-time data of specific attributes of micro-service systems such as latency, topology, and performance metrics with a low overhead using eBPF technology. They have structured their monitoring system into three levels to provide complete detail to the end-users; however, their scope is limited by the metric collecting mechanism. Viperprobe [14] is another research work that addresses the observability issues in the micros-service system. Viperprobe is an eBPF-based data collection framework for micro-service systems. Viperprobe leverages eBPF to increase efficiency and lower the performance overhead during the extraction of metrics in micro-service systems.

M. Kim et al., [15] introduce the algorithm MonitorRank for root cause detection in service-oriented architecture systems. MonitorRank algorithm uses an unsupervised ML model for ranking possible root causes. Historical and current time-series metrics were used as the input for the model. However, this research only attributes for host-level KPI and does not consider the in-kernel attributes of the system. MicroRCA [16] is another research work that addresses the observability issues in microservices. MicroRCA proposes an approach to identify the root causes of performance-related problems based on application and system-level metrics. MicroRCA uses the Distance-Based online clustering algorithm BIRCH for anomaly detection function based on the slow-response time of microservices. Then uses, the PageRank algorithm to identify the root causes.

IV. Methodology

This anomaly detection will have two central mechanisms to function efficiently as a whole system: an optimized data collection mechanism and an efficient data analyzing tool. This section will discuss the methodology of the proposed method in two significant subsections: data collection and data analysis.

A. Data-collection

The microservice systems have a dynamic attribute which makes it harder to monitor them using existing monitoring tools. An active data collection mechanism is needed to collect data from such a dynamic system. The information required to understand the dynamicity of the microservice system will be pulled out from the crictl utility, which is a command-line interface for CRI-compatible container runtime. With the collected information regarding the state of the microservice system, the collector program will run in the background and collect necessary data from the microservice system with the help of eBPF and other tracing tools for the data analysis mechanism to detect anomalies present in the system. We have selected the following parameters in Table 1 as the key parameters that can efficiently influence anomaly detection. But with the hyper-parameter optimization, the efficient parameters can be chosen at the end of building the ML model if we want to increase the accuracy further.

TABLE I: LIST OF PARAMETERS

Parameters	Description
cpu_usage	Average cpu_usage in a defined period
memory_usage	Average memory usage in a defined period
sys_calls	No of system calls made in a defined period
mmap_calls	No of mmap system calls made in a defined period
page_faults	No of page fault exception occurred in a defined period
block_req	No of block request made in a defined period
file_access	No of files opened in a defined period
PL_events	No of Process Level Events

1) State-identifier: This component of the data-collecting mechanism will be responsible for identifying the current status and the details of the dynamic features in the microservice systems. The crictl tool, a command-line interface for CRI-compatible container run-times, will extract the container and container-related information such as process id, pod name, and node name using CLI scrapping method using the python sub-process module.

2) Collector: The collector component will conduct the data extraction with the identified information related to the microservice system. bpftrace tool was used to extract the required parameters from the system. bpftrace is a high-level tracing language for eBPF, available in recent Linux kernels. It uses LLVM as its backend. Bpftrace extracts data from the Linux BPF system and existing Linux tracing capabilities: kernel dynamic tracing (kprobes), user-level dynamic tracing (uprobes), and tracepoints. The collector program is also responsible for storing collected data in a time-series database.

B. Data Analysis

The input data for the ML model will provide multivariate time-series data. And it would be inefficient to model the anomaly detection system without considering the temporal nature of the time-series data. LSTM (Long Short Term Memory model) is an artificial recurrent neural network (RNN) that is mainly targeted to handle sequential data [17]. Therefore, LSTM will be used to obtain the temporal component from each parameter's sequential time-series data to extract the temporal feature from the time-series data.

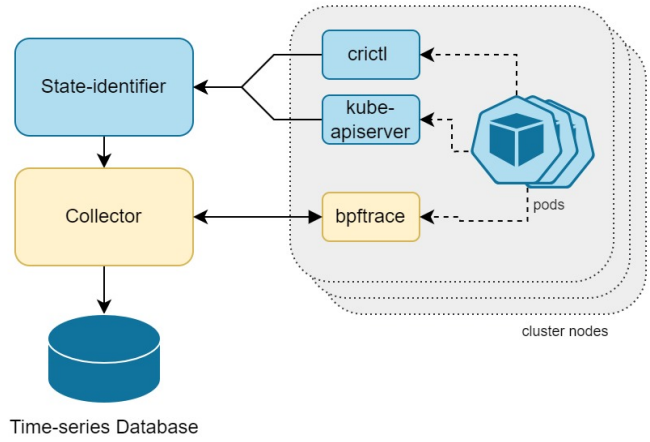


Fig. 2: Data collection mechanism overview

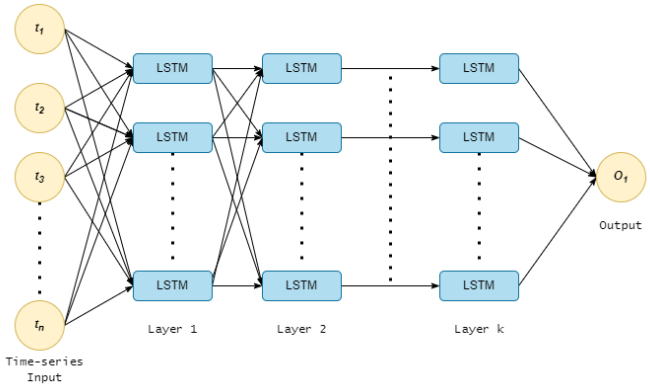


Fig. 3: Neural network of LSTM layers

LSTM uses a memory cell structure to store the information of the previous state of the data. A memory cell contains four components [18],

- Recurrent connection to the memory cell to maintain the state of the memory cell
- Forget gate to modulate the memory cell for controlling the persisting information regarding the previous state
- Output gate to Control the output from the memory cell
- Input gate to control the input data and memory cell connection

However, extracting the temporal feature for detecting outliers is not efficient enough since we have to consider the co-relations between the different parameters in the multivariate input data. Autoencoders use artificial neural networks to learn representations of the input data. The network imposes a bottleneck in the architecture, forcing a compressed knowledge representation of the original input. As a result, if there is a structure in the incoming data, it may be learned and compressed into a small, latent form [19], And the output equal to the input will be reconstructed again from the compressed latent form, with a reconstruction loss. Based on the reconstruction loss, we can detect the outliers [20].

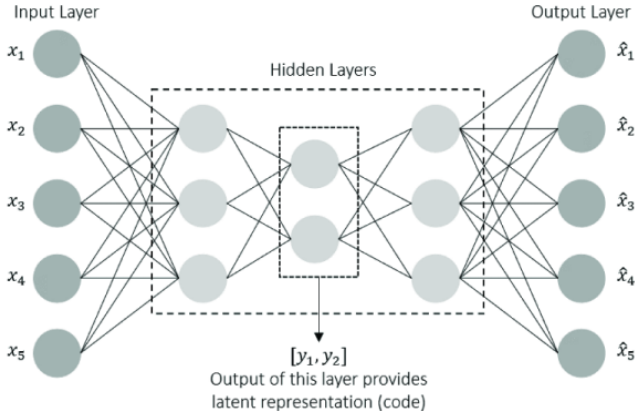
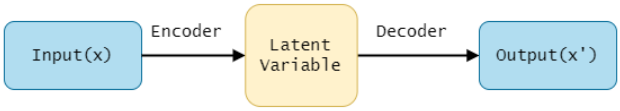


Fig. 4: General autoencoder network structure [19]



The Reconstruction loss *RL*(x) when the input data is x and the output data of the autoencoder is x's, is given by,

$$RL(x) = (x - x')^2 \tag{2}$$

We can calculate the reconstruction loss of the LSTM input fed into the autoencoder algorithm. The anomalous behaviours will be identified with a defined threshold value for the reconstruction loss, estimated during the training phase. The flow of the Data analysis process to identify anomalies is given in Fig. 5.

V. Experimentation and Results

For the development of the anomaly detection system, we provisioned an experimental setup in a cloud service platform. The experimental design consists of three virtual machines with four vCPUs with 6GB of memory. Kubernetes was used to orchestrate the containers in the microservice system. To simulate the microservice system, we deployed the Hipster shop web application consisting of 10-tier microservices. The Hipster shop is a demo application from google for benchmarking purposes. Hipster shop consists of a load-generating service to continuously sends HTTP requests to the front end to simulate the user flow. Along with the Kubernetes cluster, we deployed another VM for the time-series database to process the data sent by the data-collector program from all the nodes in the cluster.

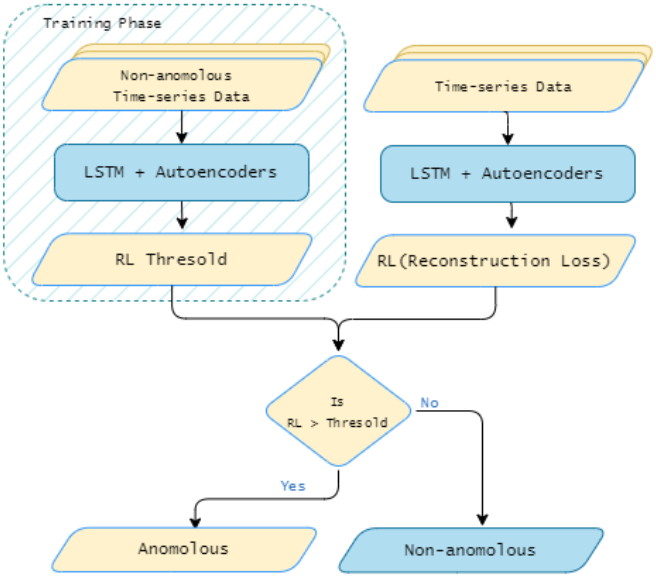


Fig. 5: Anomaly detection flow

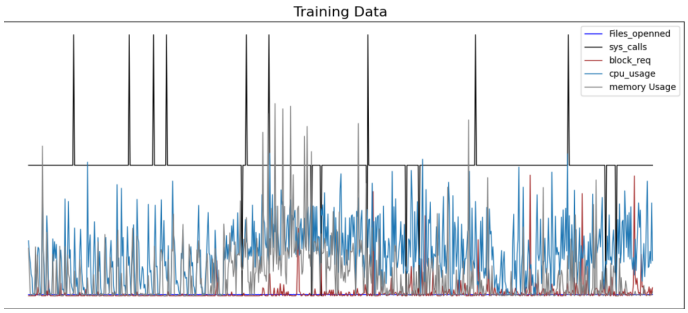


Fig. 6: Training data of a container

A. Experimental Data

The data collected during the non-anomalous condition was used for training the model. We used data that contains anomalous data points for validating the model. Fig. 6 and Fig. 7 contains the training and validation dataset

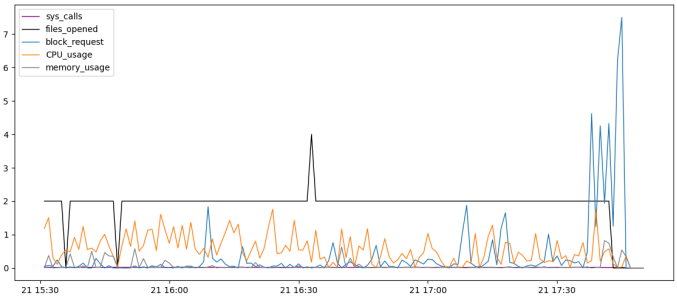


Fig. 7: Data with anomalous incident

B. Training and Validation

The non-anomalous time-series data were normalized and fed into the LSTM and auto-encoder function to estimate the RL threshold. Fig. 8 contains the distribution of RL of non-anomalous data. The 99.7th percentile of the RL is calculated as the threshold for detecting anomalies. Therefore the anomaly detection system will flag the inputs with an RL higher than the estimated threshold during the training phase.

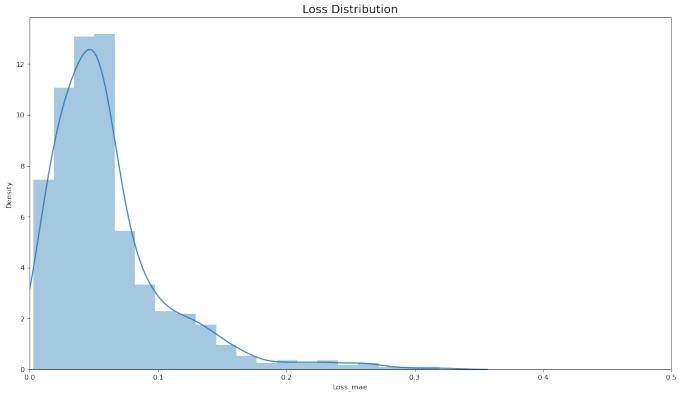


Fig. 8: Distribution of RL

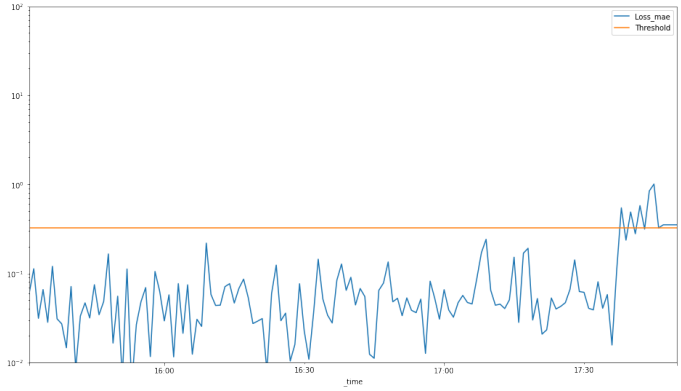


Fig. 9: RL value with thresold

Next, we selected time-series data that contains the anomaly incident where the workload has increased and caused the VMs to crash to validate the constructed model. Fig. 7 and Fig. 9 contain a sample container's time-series input data and the comparison of RL value with the estimated threshold value. As in Fig. 9, we can observe that the ML model could flag the anomalies with a false positive rate of 1.6 in the selected container. The training and the validation were done in all the containers across the cluster, and we achieved an average false positive rate of 1.16. We have provided the threshold and false positive details of the containers which had high false favorable rates in Table II

VI. Limitations and Future Work

Currently, the anomaly detection system only considers the performance and in-kernel metrics of the containers for anomaly detection. However, considering the application life-cycle events for the detection will help further reduce the false positive rate. In future work, this anomaly detection system can be improved by considering the circumstances,

TABLE II: RL AND FALSE POSITIVE DETAILS

Pod	*Threshold*	*False Positive*
coredns-565d847f94-w4pcvmstr	0.13346	10.83
kube-proxy-v29ctwkr1	0.15860	8.33
cartservice-6dbfb7f57-xpd5cwkr2	0.12654	2.50
kube-proxy-t7jhpwkr2	0.34982	2.50
cilium-operator-bc4d5b54-rdx5rmstr	0.18071	2.50
currencyservice-86f55694b9-gj6t8wkr3	0.12929	1.67
adservice-7dc899d57f-4jfw9wkr2	0.08437	1.67
kube-scheduler-mastermstr	0.19413	1.67
emailservice-7d677b875-8r2z2wkr3	0.07243	0.83
shippingservice-ff9646559-76tsnwkr3	0.18022	0.83
paymentservice-7946999b79-fq5pvwkr3	0.11368	0.83
checkoutservice-659797fc89-bhs7bwkr2	0.14170	0.83
kube-controller-manager-mastermstr	0.10142	0.00

such as deployment and scaling of the applications, and the consideration of service-level metrics to reduce the false positive rate and increase the accuracy.

VII. Conclusion

The increased complexity, the abstract, and the dynamic nature of microservices architecture make security a challenge in microservices compared to the existing monolithic architecture. Therefore, efficiently securing this architecture requires novel concepts ranging from unsupervised machine learning to low intrusive data collection techniques like eBPF, which is very efficient in collecting in-kernel metrics to identify anomalous behaviour in microservice environments. We have presented such a system that can collect advanced in-kernel parameters with the understanding of the dynamic nature of microservice architecture and detect anomalies by using RL value retrieved from the autoencoder function as the baseline for normal behaviour.

References

[1] J. Thönes, "Microservices," in IEEE Software, vol. 32, no. 1, pp. 116-116, Jan.-Feb. 2015, doi: 10.1109/MS.2015.11.

[2] C. M. Aderaldo, N. C. Mendonça, C. Pahl and P. Jamshidi, "Benchmark Requirements for Microservices Architecture Research," *2017 IEEE/ACM 1st International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering (ECASE)*, 2017, pp. 8-13, doi: 10.1109/ECASE.2017.4.

[3] T. Weng, W. Yang, G. Yu, P. Chen, J. Cui and C. Zhang, "Kmon: An In-kernel Transparent Monitoring System for Microservice Systems with eBPF," *2021 IEEE/ACM International Workshop on Cloud Intelligence (CloudIntelligence)*, 2021, pp. 25-30, doi: 10.1109/CloudIntelligence52565.2021.00014.

[4] G. Kakivaya et al., "Service Fabric: A Distributed Platform for Building Microservices in the Cloud," in *Proceedings of the 13th EuroSys Conference, EuroSys 2018*, Apr. 2018, vol. 2018-January. doi: 10.1145/3190508.3190546.

[5] A. Khan, "Key Characteristics of a Container Orchestration Platform to Enable a Modern Application."

[6] "Kubernetes." https://kubernetes.io/ (accessed Jan. 26, 2022).

[7] E. Gershuni et al., "Simple and precise static analysis of untrusted Linux kernel extensions," in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, Jun. 2019, pp. 1069–1084. doi: 10.1145/3314221.3314590.

[8] P. Chaignon, K. Lazri, J. François, T. Delmas, and O. Festor, "Oko: Extending open vswitch with stateful filters," Mar. 2018. doi: 10.1145/3185467.3185496.

[9] "The design and implementation of hyperupcalls" *Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference*, https://dl.acm.org/doi/10.5555/3277355.3277365 (accessed Jan. 26, 2022).

[10] "Extension framework for file systems in user space," in *Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference"*, https://dl.acm.org/doi/10.5555/3358807.3358819 (accessed Jan. 26, 2022). .

[11] V. Chandola, A. Banerjee and V. Kumar, "Anomaly detection", ACM Computing Surveys, vol. 41, no. 3, pp. 1-58, 2009. Available: 10.1145/1541880.1541882 [Accessed 16 August 2022]. .

[12] O. I. Provotar, Y. M. Linder and M. M. Veres, "Unsupervised Anomaly Detection in Time Series Using LSTM-Based Autoencoders," *2019 IEEE International Conference on Advanced Trends in Information Theory (ATIT)*, 2019, pp. 513-517, doi: 10.1109/ATIT49449.2019.9030505.

[13] T. Weng, W. Yang, G. Yu, P. Chen, J. Cui and C. Zhang, "Kmon: An In-kernel Transparent Monitoring System for Microservice Systems with eBPF," *2021 IEEE/ACM International Workshop on Cloud Intelligence (CloudIntelligence)*, 2021, pp. 25-30, doi: 10.1109/CloudIntelligence52565.2021.00014.

[14] J. Levin and T. A. Benson, "ViperProbe: Rethinking Microservice Observability with eBPF," *2020 IEEE 9th International Conference on Cloud Networking (CloudNet)*, 2020, pp. 1-8, doi: 10.1109/CloudNet51028.2020.9335808.

[15] M. Kim, R. Sumbaly and S. Shah, "Root cause detection in a service-oriented architecture", *ACM SIGMETRICS Performance Evaluation Review*, vol. 41, no. 1, pp. 93-104, 2013.

[16] L. Wu, J. Tordsson, E. Elmroth and O. Kao, "MicroRCA: Root Cause Localization of Performance Issues in Microservices," *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, 2020, pp. 1-9, doi: 10.1109/NOMS47738.2020.9110353.

[17] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink and J. Schmidhuber, "LSTM: A Search Space Odyssey," in *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222-2232, Oct. 2017, doi: 10.1109/TNNLS.2016.2582924.

[18] R. Fu, Z. Zhang and L. Li, "Using LSTM and GRU neural network methods for traffic flow prediction," *2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, 2016, pp. 324-328, doi: 10.1109/YAC.2016.7804912.

[19] Q. Xu, Z. Wu, Y. Yang and L. Zhang, "The difference learning of hidden layer between autoencoder and variational autoencoder," *017 29th Chinese Control And Decision Conference (CCDC)*, 2017, pp. 4801-4804, doi: 10.1109/CCDC.2017.7979344.

[20] N. A. Mohamed and J. R. Cavallaro, "Design and Implementation of Autoencoder-LSTM Accelerator for Edge Outlier Detection," *2021 IEEE Workshop on Signal Processing Systems (SiPS)*, 2021, pp. 134-139, doi: 10.1109/SiPS52927.2021.00032.