SBPAM:Secure Based Predictive Autoscaling Model For containerized application

Mohamed.I.El-Shenawy
Information Technology Department
Canadian International College
New Cairo, Egypt
moh.shenawy1983@gmail.com

Hayam Mousa
Faculty of Computers and Information
Menoufia University
Shibin El Kom, Egypt
hayam910@gmail.com

khaled M. Amin
Faculty of Computers and Information
Menoufia University
Shibin El Kom, Egypt
kh.amin.0.0@gmail.com

***Abstract*— During the past few years, the virtual technology used in the cloud has become unsuitable for service delivery, with the emergence of containers technology and the spread of its use throughout a wide range of cloud service providers because of the ease of use and provision of the resources used. The institutions have become widely seeking to develop this technology to suit the different needs to provide a good service for the end-user. In this paper, we use machine learning to improve the container orchestration process. Our approach focuses on getting containers cluster resources idle as much as can by scanning and clearing malicious and unwanted fake load to enhance the workers load then using machine learning models to predict loads in advance. Hence, the Auto-Scaler module begins to auto-scale the number of resources to meet the cluster's workload, leading to efficient use of resources.**

Keywords*— *Cloud computing, containers, autoscaling, virtualization, orchestration, machine learning.

I. INTRODUCTION

The world lives during the recent period in a state of panic and anxiety due to the spread of the Coronavirus "COVID19" in many countries of the world, which forced many countries to demand their citizens adhere to home isolation, suspend studies, close schools, and even impose curfews in many cases to limit its spread. The matter caused many sectors, especially the educational sector, to become confused. During this period, cloud computing has played a critical role in providing numerous services to most industries, including the educational sector.

Cloud computing (CC) may be a benefit show where computing administrations that are accessible remotely permit clients to get to applications, information, and physical computation assets over an organized, on request, through getting to gadgets that can be profoundly heterogeneous. In cloud computing, assets are leased based on requests and pay-per-use models from cloud suppliers.

"Cloud infrastructure consists of three service models. (SaaS): The provider's applications have been granted to run by consumers. However, they do not control cloud infrastructures such as operating systems, servers, or storage. (PaaS): there is no control over the cloud infrastructure, but consumers control deployed applications. (IaaS): "The underlying infrastructure such as virtual machines, operating systems, are managed by consumers." [1] see Fig. 1. Traditional virtualization hypervisors with heavyweights are used to support resource sharing in cloud computing [2] [3]. Using hypervisor in the upper layer of the host operating system decreases the performance of the virtual machine. Containers [4] are a suitable replacement for virtual machines that have grown in popularity among developers.

Containers use the kernel of the host operating system to isolate each container by enclosing it with the services it requires. The techniques that use containers offer the best performance, fast, isolation and elastic deployment, and powerful resource sharing. They have become widely used by organizations to deploy their differing workloads in the cloud. As a result, container orchestration platforms have risen, used to manage containerized applications' deployment. See Fig. 2.

Container orchestration [5] aims to manage container lifecycles, including automating container deployment, management, scaling, networking, and availability. The automatic scaling allows scaling up or down based on CPU or memory consumption. Automatic scaling ensures that the application is always available and that sufficient resources are available to prevent performance issues or outages. In this research, the proposed Auto-scaling System can predict the workload in advance and increase the capacity to reduce the application response time efficiently. The rest of the paper is organized as follows: Section Ⅱ gives a brief introduction of related work, Section Ⅲ introduces the proposed system, Section Ⅳ presents evaluation and experimental results, and Section Ⅴ drive the conclusion.

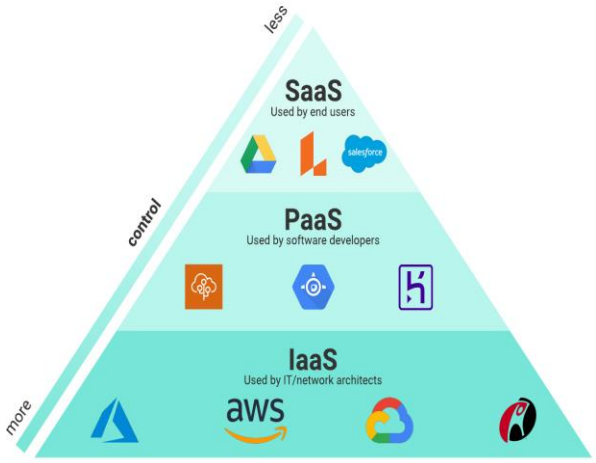

Fig.1. Three types of services provided by Cloud Computing providers [6]

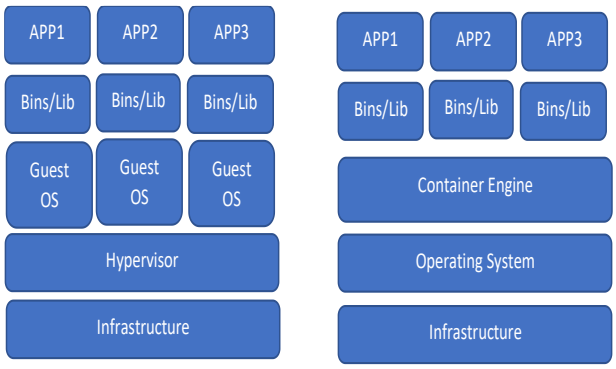

Fig 2. difference between virtual and containers

II. PROBLEM DEFINITION AND RELATED WORKS

A. *Problem definition*

We have a running system in a data center environment, and the system provides service for large-scale users. When there is a heavy load on the system, additional hardware resources are required to respond efficiently. Therefore, the system scales up in a moment of bottlenecks, but that requires some time to get the required resources from the host hardware, which affects the system response time and stability. We integrate machine learning to predict the future system resources usage based on the history of resources usage and scaling the system resources if needed. The system will get stable and always have the sufficient resources it needs.

B. *Related Work*

Over the past years, auto scaling the VMs and containers in the cloud infrastructure has been widely recognized as an exciting research topic in computer science, and different research groups work on different aspects. Meanwhile, cost-efficient resource provisioning based on real-time changes of workloads is critical in auto-scaling the VMs and containers in cloud environments to achieve the quality of service (QoS) [7].

Several related projects are highlighted here with their approaches to our problem statement. Zhang et al. [8] developed video surveillance using a container-based cloud platform; they use prediction to achieve fine-grained resource provisioning. Al-Dhuraibi. [9], proposed Elastic Docker, an autonomic controller powering the vertical elasticity of Docker containers autonomously; elastic Docker is used to scale CPU and memory containers. Moore et al. [10] proposed a hybrid elastic controller predictive and reactive to control scaling to improve cloud application performance. Matteo Nardelli et al. [11] propose Adaptive Container Deployment, a model for containerized application deployment using the Integer Linear Programming problem. Gandhi et al. [12] proposed an auto-scaling approach; they aim to scale cloud infrastructure resources without the intervention of application-level automatically. Lin et al. [13] developed an autoscaling system to monitor network traffic requests and HTTP response time to identify application performance in the cloud. Karl Mason et al. [14] used the Simple RNN method to accurately predict CPU utilization over short periods if there is a sudden change in CPU usage. Raouia Bouabdallah et al. [15] propose an automated resource provisioning method based on workload prediction using the Simple Exponential Smoothing method.

We found that most of the related work focuses on the application's performance without considering external issues that can affect the system autoscaling stability, so we focus on this work to enhance the system autoscaling with model scaling stability and reduce consumed resources.

III. PROPOSED SYSTEM

Application performance is a vital keystone for the success of any organization, therefore predicting the future workload of the running application is vital for getting their required hardware resources to work efficiently. So, we can get any application running efficiently by Appling security scanning before predicting its future workload and autoscaling the application to get its required resources; by applying this enhancement, we can decrease the required resources needed in the autoscaling process. Fig. 3 shows the autoscaling process.

A. *System architecture*

Many container orchestration frameworks can deploy and manage multi-tiered applications in a cluster. One of the most famous ones of the containers framework is Kubernetes [16]. Kubernetes is an open-source container-management system that automates the deployment, scaling, and management of computer applications. Google invented it, and the Cloud Native Computing Foundation now backs it up. Kubernetes integrates various container tools and runs containers in a cluster, frequently with images created with Docker, which is now deprecated in favor of containers. The architecture of Kubernetes is divided as the following.

Control-plane is the central node that manages the whole cluster, the workload and communication, and states between nodes. It also manages job scheduling such as starting, removing, deploy containers.

The workers nodes are a location that hosts containers deployment. The cluster workers node must run a container runtime such as Docker and the following services that handle the configuration and communications of these containers.

Kubelet is responsible for the workers nodes operational state. It manages the start, stops, and maintenance of application containers in pods. It continuously checks a pod's state and re-deploys if it is not in the desired state.

Kube-proxy combines a load-balancer and a network proxy in addition to networking operations. It routes traffic to the containers based on incoming request port number and IP address.

The container combines the running application and related libraries and any other dependencies. The containers are placed in pods, and containers are accessed from the outside world by exposing the external IP address.

B. *Auto-scaler architecture*

The auto-scaling [17] issue is an old-style programmed control issue that aims to powerfully tune the resources and measure the resources assigned to reach a specific objective. In particular, it is regularly abstracted as a "MAPE control loop" [18], which continuously repeats itself over time. The auto-scaler uses a time-series database that stores the

monitored data so the analyzer can get that data in real-time for analysis.

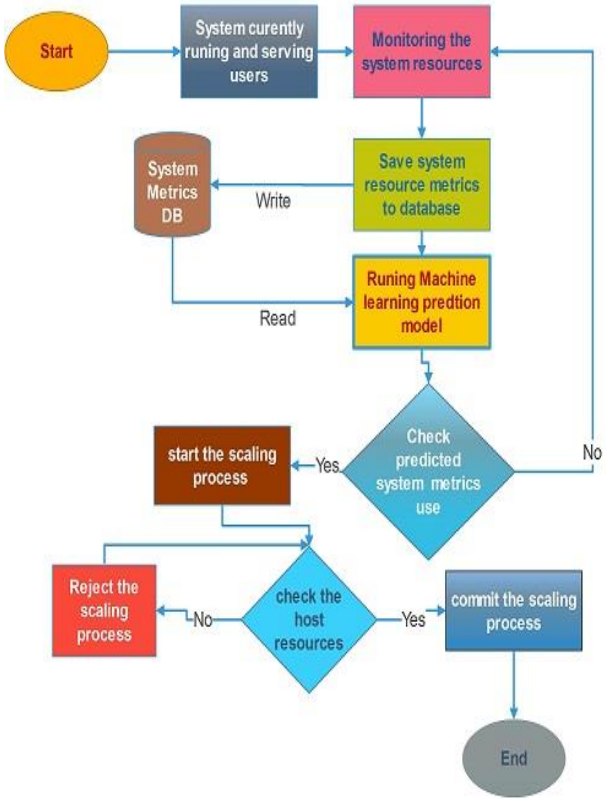

Fig.3. The autoscaling process.

Monitoring is the first phase in the loop, which continuously collects data needed during analysis and planning to regulate appropriate scaling actions. The monitored data is different for "HTTPS workload" network traffic and hardware resources such as CPU, memory, and storage. Many vendors offer to monitor apps in the market, for example, Prometheus [19] and CloudWatch [20].

Time series database is used for storing and serving time series through associated pairs of times and values, for example, influx-DB [21]. The primary purpose of adding TSDB to the auto-scaler is to properly maintain collected data as a historical record to improve its prediction accuracy; these records are used to train the prediction model.

Analyzer retrieves the most recent collected data from the database regularly and continuously uses a predefined window size to analyze it.

The planner uses the analyzer's output. It depends on the predicted workload instead of using the current workload to determine the number of replicas. It horizontally scales up or down container replicas in response to predicted workload. The predictable replicas (H estimated) is determined using (1)

$$H_{estimated} = \frac{L_{total\ predicted}}{L_{max\ containers}} \quad (1)$$

Where ($L_{total\ predicted}$) is the total predicted incoming workload, where ($L_{max\ container}$) is the maximum workload. We get the value of ($L_{total\ predicted}$) from the analyzer, but we get the value of ($L_{max\ container}$) by using stress tests during the development stage.

The executor starts after the planner decides the number of replicas, it is the turn to execute those commands by changing the number of container replicas.

C. Prediction model

In this section, we discuss the proposed model, which uses machine learning regression models. Any machine learning model must pass through five steps [22], as shown in Fig. 4. The first step is collecting the data from the time series database, as explained earlier. Then, data preprocessing uses EDA "Engineering Data Analysis" to clear and add feature data to the collected data to fit the model. This helps during model building. The next step is to build the model using a suitable machine learning model. In our case, we used the following ML regression models "XGboost, LightGBM, Random Forest, Decision tree. The fourth step is to train the model on the collected data to ensure that the model can give estimated actual values when testing and fitted in natural production. The final step is to test the model on unseen data and evaluate the model

performance.

Fig. 4. The overall machine learning process.

XG-Boost [23] XG-Boost is a distributed algorithm that boosts a gradient decision tree for computational speed. It was developed for machine learning.

LightGBM [24] LightGBM is a framework for building light gradient boosting machines. Microsoft has introduced this framework for machine learning. It gets ranking of decision trees for classification and regression

Random forests [25]Random forests are ensemble learning methods that involve training many trees. They then yield a class or mean or average regression of each tree..

D. Integrating Auto-scaler System with prediction Model

We have applied the scaling on the Kubernetes cluster and integrated the machine learning prediction model to apply the autoscaling. The system resources are continuously monitored using a monitoring system and save the resources metrics to the database system to use the machine learning model. The monitoring module obtains the indicator data of the pod, and it then passes this acquired data to the prediction module for estimations. The prediction module provides the estimated number of pods and delivers it to the auto-scaler module for scaling. The auto-scaler module then scales the cluster based on the prediction given by the prediction module. As seen in Fig. 5.

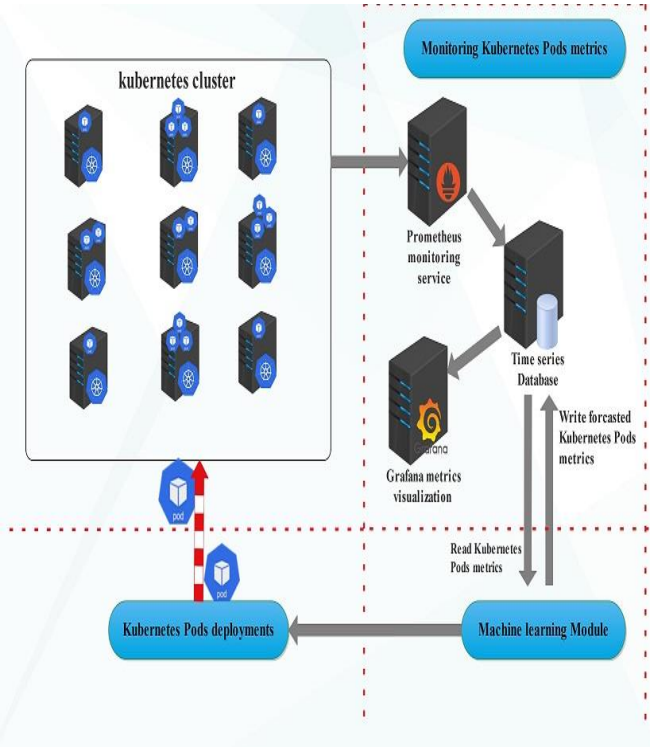

Fig. 5. Proposed Auto-scaler System Architecture.

E. Enhancing the application and cluster performance

All published applications constantly suffer from malicious and unwanted data, which makes the unwanted heavy load on back-end servers that host the published application.

To illustrate the effects of unclean traffic on autoscaling, we will discuss our real case. Our case consists of a web server hosted in a container cluster that serves our organization's online and local users, and the containers cluster exists in the on-premises data center. The issue was that there was unexpected traffic during the admission period of students, which led to a heavy load on the workers server, which hosts web server containers, as seen in Fig. 6.

We found that much traffic is not related to the actual source and has many requests per second by troubleshooting the issue. Also, by inspecting the traffic, we found much traffic related to the port scanner. This is a symptom of a DDOS attack. DDoS attacks are classified into four types: UDP flood, TCP SYN flood, Ping of Death, and Smurf attack [26]. DDoS attacks are classified into three categories: volume-based attacks, protocol attacks, and application-layer attacks. Although the auto-scaler model has scaled up the web app cluster to the maximum number of pods, the worker was fully loaded, and there is no available resource for responding to incoming real traffic from online or internal organization users. We tend to solve this issue by preserving the stability of the auto-scaler model and preventing DDOS from consuming the available workers resources.

We have to apply security filtration on multiple stages to filter incoming data. The target of a DDOS attack is to consume the available resources to block servers' responses to actual requests. To avoid that, we added a security layer on the cloud using cloud vendors like cloud flare [27], which has distributed Iaas overworld and let the DDOS traffic filtered out and free the local internet bandwidth from Fake traffic. The security profile applied to filter the incoming traffic by applying a web security challenge to filter fake service requests, apply filtration based on geographics income source IP location, block bots traffic, scan web request per second per source IP, and block the high source rate the result is cleaning 80% of fake incoming traffic. The next layer is the local security device which has applied another security profile as stated earlier. The security profile scans traffic vs. locally installed web application signature database. In addition, it applies the same security profile applied in the cloud except for the geographics location filter. The source comes from the Cloudflare IPs. The containers cluster workers resources become idle most time and available for actual load based on simple increased web server requests in the admission period. By applying traffic scanning, we show how malicious and unwanted data traffic affects the performance of applications. The Proposed Auto-scaler System Architecture is shown in Fig 5.

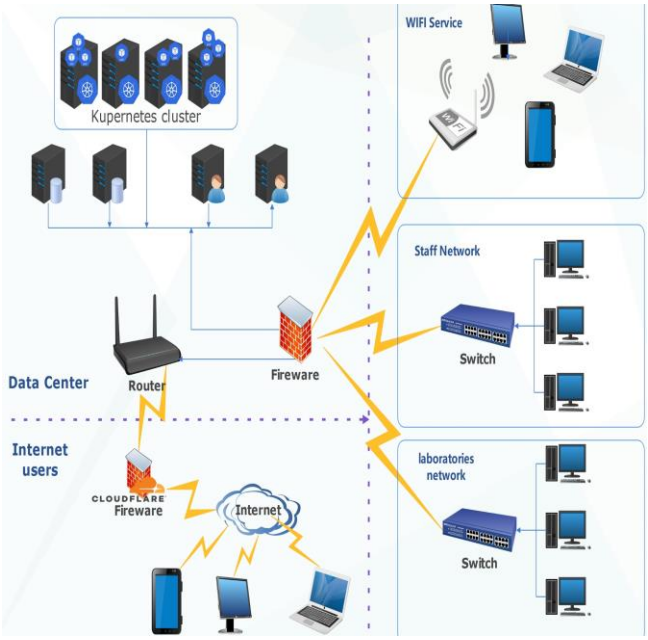

Fig. 6. Kubernetes cluster and other datacenter equipment's.

IV. PERFORMANCE EVALUATION

A. Evaluation environment settings

1) Test environment:

The test was done on Huawei 1288H V5 server with specs "28 CPUs x Intel(R) Xeon(R) Gold 5120 CPU @ 2.20GHz, 512 GB RAM ", There was 10 virtual machine host Kubernetes cluster, the Kubernetes cluster using ubuntu 20.04 LTS Linux operating system, The mentoring server was Prometheus version 2.25.0, Prometheus/alert manager version 0.21.0, node exporter version 1.1.1, For metrics visualization we used Grafana version 7.4.2, For security, I used Huawei firewall, Cloudflare.

2) Dataset

The dataset used in this paper is for our system resources usage history for a year. Data have been collected, and Preprocessing done on it. The data is collected every 5 minutes. It contains about 105k records. The dataset has been divided into standard train and validation and test as 70:20:10, respectively. We have cleaned missing data and

scaled it to 15 minutes. Also, we have added lagged feature for the history of the needed features. We have added the calendar's days, weeks, months, years based on the timestamp to enhance the model during training processes. However, instead of using a standard train and validation dataset, we used time-series cross-validation for its benefits. It Starts by using a small subset of data for training and forecast for the later data points, then the same forecasted data are included as part of the following training dataset and forecast the next data point. We used GapWalkForward from the TSCV library for validation. The prediction model then reads the data from the time-series database.

3) Evaluation metrics

The prediction model will be evaluated according to mean absolute error (MAE), root-mean-square error (RMSE), and NRMSE metrics [28] [29] as depicted in equations (2), (3), and (4).

$$MAE\,(y,\hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} |y_i - \hat{y}_i| \qquad (2)$$

$$RMSE\,(y,\hat{y}) = \sqrt{\frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} (y_i - \hat{y}_i)^2} \qquad (3)$$

$$NRMSE = \frac{RMSE}{Max_i y_i - min_i y_i} \qquad (4)$$

We test the model to predict the system usage for replica estimation calculation using unseen test data.

B. Evaluation results

in this section, we first evaluated our ML models and their results. The validation results are summarized in Table 1.

Table 1: Different ML algorithms results using the dataset of System resources usage.

Model	MAE	RMSE	NRMSE
Decision tree	0.818	2.521	0.025
LGBM	0.669	1.793	0.018
Random forest	0.497	1.778	0.017
XG-Boost	**0.476**	**1.588**	**0.015**

Table 1 and Fig. 7 shows validation results, the best result for MAE was 0.476 using XG-Boost, and the best result for RMSE was 1.588 using XG-Boost, and the best result for NRMSE was 0.015 using XG-Boost. So, based on the result we got, we will use the XG-Boost model in our autoscaling prediction model using security and without security modules.

As we see from Fig. 8a, the uncleaned traffic directly affects the performance of the application and systems, where CPU average unitization is 81% , which affects

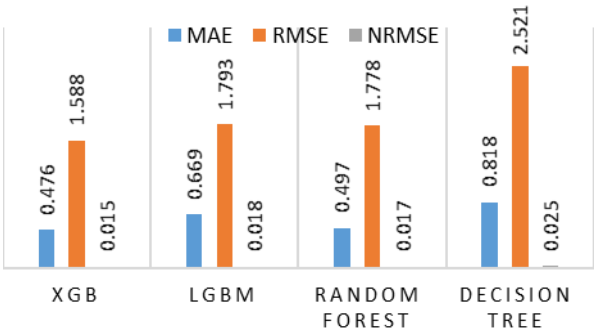

Fig 7. The predicted results regarding application pod

resource availability required to scale application running inside containers, which leads to not serving the end-users efficiently. But as we see in Fig. 8b, after we have scanned the incoming traffic to cluster and cleaning unwanted traffic, the CPU average unitization is 32 % , and this lets the auto scaler and prediction model can accurately predict the real resources needed by the application. Fig. 9 shows the difference between the CPU utilization in the two uncleaned and cleaned traffic cases.

The prediction model can do its work to archive the best performance. Hence, the application runs efficiently.

It can serve the end-users requests as shown in Fig.10, the auto-scaler start with 3 replicas then start to scale up the application containers to 9 replicas based on the predicted next load, then start increasing the replicas to full replicas based on predicted heavy CPU utilization, the replicas get to minimum replicas 3 after the load has ended. The predictor scale down the replicas step by step using the cooldown

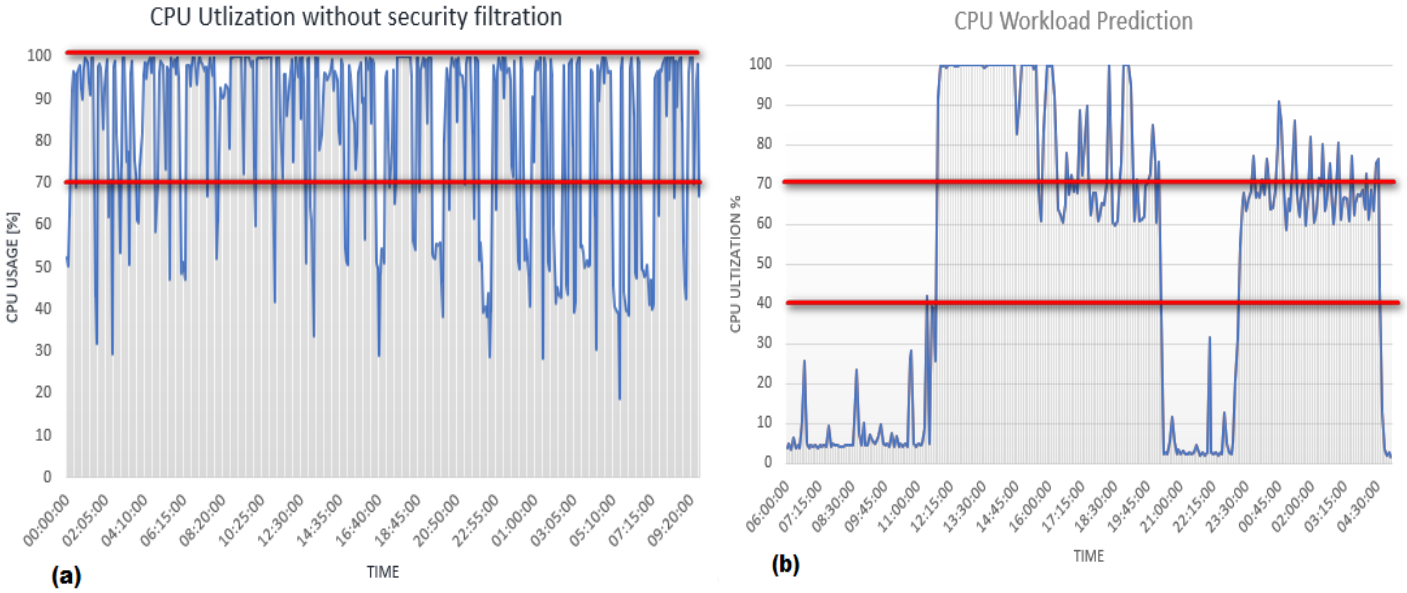

Fig. 8 a & b: the quantization with and without security filter applied

feature to not sudden down the replicas for system stability.

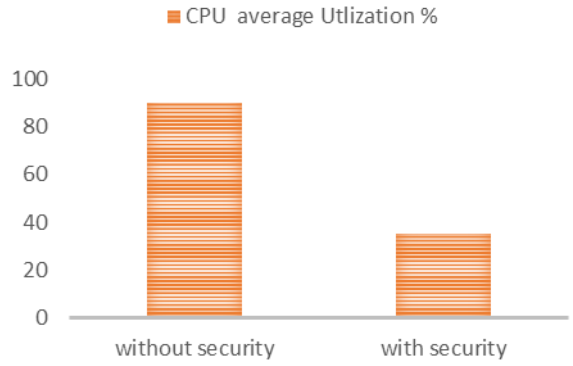

Fig 9. The resources utilization for two security cases.

The number of replicas usually cope with the CPU utilization, which means that the system scales up and down efficiently, saving the resources when available without causing bottlenecks in the system performance.

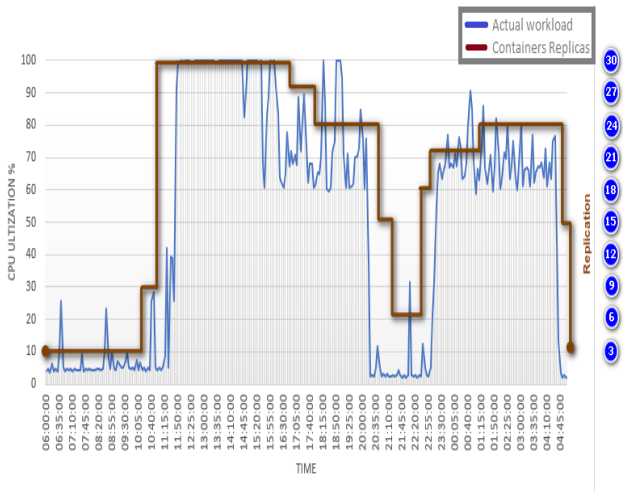

Fig 10. Number of replicas in response to CPU utilization

V. Conclusion and Future Work

The container has become the trend of the current deployment of most applications. While many cloud vendors widely use containers, this technology has many developments to meet the immediate need for elastic resource provisioning using autoscaling methods. We proposed in this paper a model to predict the future workloads for a containerized application and use an auto scaler model based on Kubernetes and machine learning. The prediction model used the XG-boost ML algorithm where it gets the best results of 0.015 using NRMSE. XG-boost used historical time data to predict the future workload accurately. In addition, experimental results that were proposed proven that as it can decrease hardware used resources from 81% to 32% by applying security scanning on incoming filtered data to container clusters to clean the traffic from unnecessary malicious traffic, which consumes many resources. XG-Boost model shows it get the best results for auto-scaler metrics and elastic speedup. For future work, we will try to use deep learning to forecast other types of resources like memory and network in addition to CPU. Apply integration between horizontal and vertical scaling and check its impact on applied systems.

References

[1] R. Chandramouli, M. Iorga, S. Chokhani, Cryptographic key management issues and challenges in cloud services, in: Secure Cloud Computing,Springer, New York, 2014, pp. 1–30.

[2] A. M. Joy, "Performance comparison between linux containers and virtual machines," in Computer Engineering and Applications (ICACEA), 2015 International Conference on Advances in. IEEE, 2015, pp. 342–346.

[3] Desai, Prashant. A Survey of Performance Comparison between Virtual Machines and Containers. INTERNATIONAL JOURNAL OF COMPUTER SCIENCES AND ENGINEERING. 2016. pp. 55-59.

[4] A. Celesti, D. Mulfari, M. Fazio, M. Villari and A. Puliafito, "Exploring Container Virtualization in IoT Clouds," 2016 IEEE International Conference on Smart Computing (SMARTCOMP), 2016, pp. 1-6.

[5] Casalicchio E. (2019) Container Orchestration: A Survey. In: Puliafito A., Trivedi K. (eds) Systems Modeling: Methodologies and Tools. EAI/Springer Innovations in Communication and Computing. Springer, Cham. 2019, pp 221-235.

[6] Lucidchart. 2021. *The Basics of Cloud Computing*. [online] Available at: https://www.lucidchart.com/blog/cloud-computing-basics

[7] P. R. Desai. A survey of performance comparison between virtual machines and containers. ijcseonline. org, 2016.

[8] Zhang H, Ma H, Fu, G, Yang, X, Jiang, Z, and Gao Y, Container based video surveillance cloud service with fine-grained resource provisioning, in Proceedings of 2016 IEEE 9th International Conference on Cloud Computing (CLOUD), pp. 758-765.

[9]. Al-Dhuraibi, Y., Paraiso, F., Djarallah, N., Merle, P.: Autonomic vertical elasticity of docker containers with elasticdocker. In: Proceedings of the 2017 IEEE 10th International Conference on Cloud Computing, pp. 472–479.

[10] K.B. Laura R.Moore and T.Ellahi, "A coordinated reactive and predictive approach to cloud elasticity," in CLOUD COMPUTING 2013 : The Fourth International Conference on Cloud Computing, GRIDs,and Virtualization, 2013.

[11] Nardelli, M., Cardellini, V., Casalicchio, E.: Multi-level elastic deployment of containerized applications in geo-distributed environments. In: Proceedings of 2018 IEEE 6th International Conference on Future Internet of Things and Cloud. IEEE (2018).

[12] A.Gandhi, P.Dube, A.Karve, A.Kochut, and L.Zhang, "Adaptive, model-driven autoscaling for cloud applications," in 11th International Conference on Autonomic Computing (ICAC 14). Philadelphia, PA: USENIX Association, pp. 57–64.

[13] C.-C. Lin, J.-J. Wu, P.Liu, J.-A. Lin, and L.-C. Song, "EnglishAutomatic resource scaling for web applications in the cloud," in English Grid and Pervasive Computing, ser. Lecture Notes in Computer Science, J.Park, H.Arabnia, C.Kim, W.Shi, and J.-M. Gil, Eds. Springer Berlin Heidelberg, vol. 7861, pp. 81–90.

[14] Karl Mason, Martin Duggan, Enda Barrett, Jim Duggan, Enda Howley, Predicting host CPU utilization in the cloud using evolutionary neural networks, Future Generation Computer Systems,Volume 86,2018, Pages 162-173.

[15] Raouia Bouabdallah, Soufiene Lajmi, Khaled Ghedira," Use of Reactive and Proactive Elasticity to Adjust Resources Provisioning in the Cloud Provider", in 2016 IEEE 18th International Conference on High Performance Computing and Communications, 12-14 December, 2016.

[16] Kubernetes 2021 .Kubernetes. [online] Available at: https://kubernetes.io.

[17] E. Casalicchio and V. Perciballi, "Auto-Scaling of Containers: The Impact of Relative and Absolute Metrics," 2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W), 2017, pp. 207-214.

[18] M. Tahir, Q. Mamoon Ashraf and M. Dabbagh, "Towards Enabling Autonomic Computing in IoT Ecosystem," 2019 IEEE Intl Conf on

Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech), 2019, pp. 646-651.

[19] Prometheus.io. 2021. Overview | Prometheus. [online] Available at: https://prometheus.io/docs/introduction/overview/

[20] Amazon Web Services, Inc. 2021. Amazon CloudWatch - Application and Infrastructure Monitoring. [online] Available at: https://aws.amazon.com/cloudwatch/

[21] InfluxData. 2021. InfluxDB: Purpose-Built Open Source Time Series Database | InfluxData. [online] Available at: https://www.influxdata.com/

[22] Behera, Rabi & Das, Kajaree. (2017). A Survey on Machine Learning: Concept, Algorithms and Applications. International Journal of Innovative Research in Computer and Communication Engineering.

[23] Chen, T.; Guestrin, C. Xgboost: A scalable tree boosting system. In Proceedings of the 22nd ACM Sigkdd International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016; pp. 785–794.

[24] Ke, G.; Meng, Q.; Finley, T.; Wang, T.; Chen, W.; Ma, W.; Ye, Q.; Liu, T.Y. Lightgbm: A highly efficient gradient boosting decision tree. In Proceedings of the Advances in Neural Information Processing Systems,Long Beach, CA, USA, 4–9 December 2017; pp. 3146–3154.

[25] Breiman, L. Random Forests. Machine Learning 45, 5–32 (2001). https://doi.org/10.1023/A:1010933404324

[26] RivalHost. 2021. The 12 Types of DDoS Attacks Used By Hackers | RivalHost. [online] Available at: https://www.rivalhost.com/12-types-of-ddos-attacks-used-by-hackers/.

[27] cloudflare , https://www.cloudflare.com. (Online).

[28] Jie Zhang, Anthony Florita, Bri-Mathias Hodge, Siyuan Lu, Hendrik F. Hamann, Venkat Banunarayanan, Anna M. Brockway, A suite of metrics for assessing the performance of solar power forecasting, Solar Energy, Volume 111, 2015, Pages 157-175.

[29] Medium. 2021. MAE and RMSE — Which Metric is Better?. [online] Available at: https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d.