Streamlining Software Release Process and Resource Management for Microservice-based Architecture on the multi-cloud

Isuru Pathum Herath
Department of Computer Science and Software Engineering
Sri Lanka Institute of Information Technology
Sri Lanka
it20125516@my.sliit.lk

Samuditha Jayawardena
Department of Computer Science and Software Engineering
Sri Lanka Institute of Information Technology
Sri Lanka
it20074968@my.sliit.lk

Ahamed Fadhil
Department of Computer Science and Software Engineering
Sri Lanka Institute of Information Technology
Sri Lanka
it20784720@my.sliit.lk

Nuwan Kodagoda
Faculty of Computing
Sri Lanka Institute of Information Technology
Sri Lanka
nuwan.k@sliit.lk

Udara Srimath S.Samaratunge Arachchillage
Department of Computer Science
Sri Lanka Institute of Information Technology
Sri Lanka
udara.s@sliit.lk

***Abstract*— The software development process is more flexible with the concept of containerization in the microservice platform. This research is on three key components to resolve problems faced by the developers and DevOps teams in the IT industry. First, the development phase expects a fully automated software release process from to the deployment phase and then optimize processes tailored to Docker, and Kubernetes, in microservice-based applications. Then streamline the process and leverage the container orchestration technologies to monitor the main aspect of the development lifecycle through the multi-cloud deployment on demand of the growth of day-to-day releases on multi-regions. A centralized monitoring platform is developed to monitor the deployed applications and that provides comprehensive visibility regarding performance and health of microservices. At the stage of scalarization in microservices, Vertical Pod Autoscaling (VPA) and Horizontal Pod Autoscaling (HPA) are available approaches for resource allocation, and they require measuring the minimum and maximum resource limits. As a result, an intelligent resource allocation system is proposed using a combination of Convolutional Neural Networks (CNN) and Bidirectional Long Short-Term Memory (Bi-LSTM) algorithms to cater to dynamic resource allocation, optimizing scalability, and improving cost-efficiency. This research aims to achieve practical insights into the IT industry's automated deployment, managing, scaling, and monitoring of microservice-based applications through the mentioned components.**

Keywords—Kubernetes, Docker, Microservice, HPA, VPA, DevOps, IT, Containerization, Scaling, CNN, Bi-LSTM, multi-cloud, multi-region

I. INTRODUCTION

The microservice-based architecture is used by most companies to achieve their goals with the benefit of having high availability, scalability, and avoiding the problems of Monolithic architecture [1]. With the advancement of microservice-based architecture, containerization platforms are introduced, and Docker [2] is the most popular and highly used platform for containerization. Kubernetes was introduced to resolve the problem of container orchestration [3] and DevOps came to the IT industry as a main character to automate the complexity.

The primary goal of introducing DevOps is to increase the automation of the software delivery process [4]. Although Docker, Kubernetes, and Cloud environments solve many problems related to microservice application deployments, most companies face a lack of automation and optimization of their software release process. This problem causes a slowdown and untrust the human interaction in the software release process. As a solution for this problem, Continuous Integration and Continuous Delivery (CI/CD) and Continuous Deployment (CDT) [5] can be architect and optimized for the software release process according to architectural attributes such as performance, security, extendibility, reliability, and modifiability that is commonly feasible to use in most software companies. It is necessary to have good visibility to maintain and debug the applications or improve software performance. Even, though there are monitoring tools such as Jaeger, Prometheus, and Elastic Search, they are not well-developed with detailed dashboards to get summarized details. It would be intuitive to have a centralized platform to monitor all necessary statistics in one view and enhance usability.

Dynamically resource allocation is challenging when catering for enterprise-scale in different date and time ranges. As a solution, a combination of One Directional Convolutional Neural Networks (1D CNN) with a Bi-LSTM [6] is proposed to predict the necessary resource limits per application during specific days and specific time ranges automatically. This combined model can be used on sequences of data that have both spatial and temporal dependencies. Local patterns can be captured through the CNN layers, and longer-term dependencies can be captured through the Bi-LSTM layers. With this solution, resource allocation for HPA and VPA becomes easy because the Machine Learning (ML) algorithm gives the predicted value with the previous data.

The research is proposed to model by addressing the mentioned problems and solutions. The primary component of the proposed system is mentioned in the following section describing the solutions for the problems addressed in the above section.

1. Rearchitect a fully automated and optimized software release process according to architectural attributes such as performance, security, extendibility, reliability, and modifiability.
2. Develop a centralized monitoring platform to visualize monitoring tools such as Jaeger, Prometheus, and Elastic Search.
3. Use a reinforcement learning algorithm to predict the maximum and minimum resource limits for horizontal pod autoscaling based on time and date.

Section II comprises the background and literature of the related work and Section III discusses the methodology, and the overall architecture Section IV describes the results and outcome, and finally, the conclusion is available in Section V.

II. Background and Literature

The performance of the software release process is a big challenge in releasing software. The performance of Kubernetes deployment is measured and studied under Virtual Machines. However, the performance may slow due to the lack of memory [7], lack of storage, and lack of available Central Processing Unit (CPU). The available solutions of publications do not meet the necessary performance level to avoid the mentioned problems. The necessary packages and components are always downloaded from the internet during the jobs to build docker images. This process is time-consuming and uses more resources to execute. Furthermore, the extendibility is applied with automated configuration, initialization, and deployment with an algebraic approach. However, the available systems [7] are not optimized with Kubernetes and Docker-based automated software release process systems. The solution needed to become extendible and modifiable.

Allocating resources during the software testing phase needed to be optimized to achieve reliability [9]. The cost reduction [10] is also a gap between available paid versions. Attributes such as performance, security, and reliability are considered during optimization [11], but attributes such as extendibility, modifiability, and automation are missed to archive. Also, the publication [12] is focused on optimizing performance and reliability, but the system is not well-optimized with security, extendibility, modifiability, and automation. The available automations that are based on industrial architecture [13] are investigated with different microservice based architectures. However, the solution is based on a separate framework that only supports Java and cannot be applied to other programming languages.

Cloud monitoring is a very cost-effective thing for organizations. Therefore, many companies use open-source data monitoring and visualization tools. [14] However available open-source tools have limitations. As an example, Prometheus does not support application logs. The proposed centralized monitoring and visualization tool is supported to monitor metrics, logs, and traces in one dashboard. [17] Some existing commercial monitoring tools can monitor logs, metrics, and traces on the same dashboard, but they charge a very high price for that, the proposed tool uses the libraries of open-source tools and fulfils the requirement of centralized monitoring.

III. Methodology

The proposed architecture is optimized according to the architectural attributes to improve performance, security, extendibility, reliability, and modifiability. The collected data by the centralized monitoring platform is passed to the resource prediction system to predict resource allocation based on time ranges on different dates. Automated multi-cloud deployment feasibility is archived by giving the benefit of spreading the system over multi-regions by the proposed system as explained in Fig. 1.

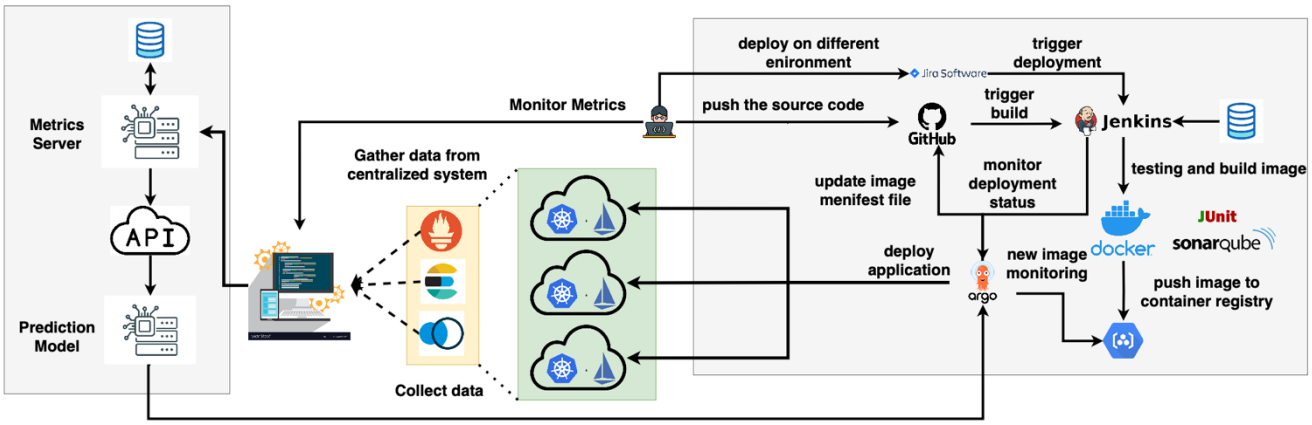

Fig. 1. Overview of the proposed architecture

A. *Fully automated software release process architecture & and optimization*

The fully automated software release process architecture is designed as a combination of two primary components. Table I explains the primary components and a brief description of each of the components.

TABLE I. The Primary Components of the Proposed Automated Architecture

Components	Description
CI Architecture	The optimized continuous integration architecture with all the components that need to create a fully automated CI process with open-source tools.
CD Architecture	The optimized continuous delivery architecture with fully automated CD for the whole software delivery process. All the tools are open-source tools to reduce the cost of the system.

a) Continuous Integration Architecture

Continuous Integration is the initial component of the DevOps lifecycle in the Software Development and Release process. Modern IT companies at the enterprise level are using Source Code Management (SCM) systems in better ways such as Git Flow, GitLab Flow, Bitbuckets Flow, and GitHub Flow. Table II describes the drawbacks of each mentioned SCM flow. According to the comparison in Table II, the [14] GitHub flow is selected as the SCM flow in this publication after considering the rich features that are currently available in GitHub SCM.

TABLE II. DRAWBACKS OF SCM FLOW

The flow of using SCM	Drawbacks
Git Flow	Complex and difficult to learn and not well-suited for teams that are new to SCM and different code segments for different branches
GitHub Flow	Lead to merge conflicts if multiple people are working on the same feature branch at the same time and are difficult to manage for large teams
GitLab Flow	Can be more complex than GitHub Flow and requires a CI/CD pipeline, which can be a barrier to entry for some teams
BitBucket Flow	Can be less flexible than other workflows and does not have a clear way to manage hotfixes

The available systems available in enterprise-level companies use manual triggering to run a pipeline. In the proposed system, GitHub triggers the Jenkins pipeline to build the Docker image and push it to the container registry. This auto-trigger solution may run the necessary pipeline automatically without any human interaction once the source code is pushed to the SCM.

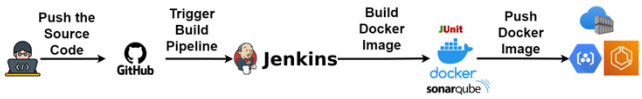

Fig. 2. CI Architecture

Fig. 2 describes the streamlined architecture from the code commit to the SCM and triggering the Jenkins pipeline to build the Docker image. Before the Docker image building, the necessary testing steps can be added using different tools with the requirement of the project or the relevant company. The docker image is pushed to the container registry once only the configured testing steps are completed.

The available solutions for the CI process are always downloading the necessary packages, dependencies, and necessary components during the application building. It takes a long time to build the application image, and this affects the performance of the build server. In the proposed system, there are separate data storages to store the downloaded packages, dependencies, and components. This reduces the downloading time and improves performance, modifiability, and reliability. Any of the new data storage can be added for new languages and integrated with Jenkins to improve the extendibility of the image-building process. Fig. 3 describes the optimization solution used in the CI architecture with the mentioned architectural attributes. Kaniko is used to build the Docker image without the Docker daemon. The Docker daemon is heavy and takes a long time to build the Docker image. Additionally, the Docker daemon required root access to build the Docker image using the Dockerfile. Using the Kaniko, can reduce the image build time and improve performance and security because it doesn't require to have root-level access.

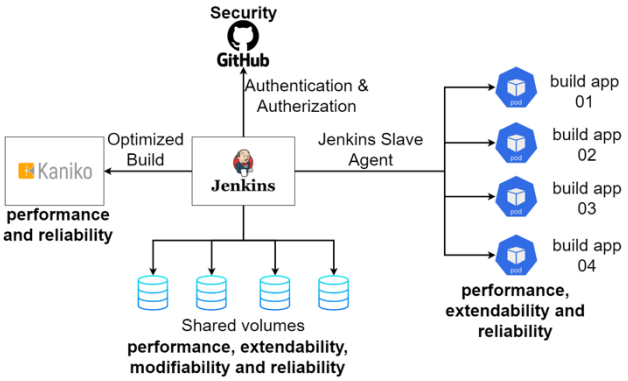

Fig. 3. Optimized Components for CI Architecture

In real-world scenarios, multiple building pipelines for different code repositories can exist simultaneously. Jenkins may take some pipelines into the queue or take a long time to run multiple pipelines. The proposed system uses Kubernetes pods as slave agents per each pipeline to run. Once the pipeline needs to run, there will be a pod created, and the pod as a slave to run the build pipeline. The solution reduces the build time and reliability by running multiple pipelines while improving the performance of the software release process. As a security enhancement, Jenkins is integrated with GitHub with Single Sign On (SSO). This doesn't need to create separate users manually in Jenkins as many of the companies do. Only required to have GitHub access and can control access levels with GitHub Teams. The proposed solution also reduces the time to manage Jenkins with SSO integration with GitHub.

b) Continuous Delivery/Deployment Architecture

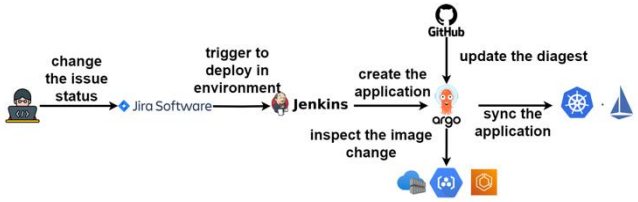

Fig. 4. CD Architecture

The Continuous Delivery is the other primary component in the software release process. As described in Fig. 4, the responsible team can change the status of the Jira ticket to deploy in the relevant environment. Additionally, this process can be automated to deploy in all the environments or selected environments without human interaction. The approval process is suggested as a best practice to avoid unforeseen. Jenkins starts the deployment process by retagging the image from the previous commit for the new deployment and pushes it to the container registry. The docker image is only built one time to reduce the build time and give the best performance to the CD process. Argo CD image updater monitors for new images identifies the application related to the newly tagged Docker image and updates the context file that includes the deployment details for the relevant application. The Argo CD Server syncs with the context file and deploys the new version of the application.

The proposed system automatically deploys the image to the development environment without any human interaction. Usage of the Argo CD Server and Argo CD image updater ensures deployment reliability. Furthermore, the proposed system uses rolling updates to ensure the availability of the system. All deployment uses the best practices of GitOps to ensure the fully automated software release process.

B. *Developing Centralized Data Visualization and Monitoring Tool*

Today, most companies use open-source monitoring and visualization tools due to cost considerations to manage microservices. However, when considering these tools, they have several limitations. DevOps Engineers must switch between various tools to monitor and visualize different data since none of the current tools can efficiently monitor all the required metrics altogether. Centralized data monitoring and visualization tools can overcome most of the limitations of the current monitoring tools.

a) Current Monitoring Tools Limitations

Jaeger is primarily used for distributed tracing of requests across multiple microservices and collects data such as the duration of each operation, the dependencies between different services, the service response time, and the error rate. However, it cannot collect data such as CPU usage, memory consumption, network traffic, or application logs.[17] Prometheus is another famous open-source monitoring tool. It can collect metrics and event data from different sources, but it cannot collect data such as the log, business, and user experience metrics. [18] Grafana is a visualization tool that helps to visualize and analyze the data collected by Prometheus. However, it cannot collect data such as log data, or business metrics. Also, Grafana does not provide inbuild charts or graphs to visualize important data. Kiali is a service mesh observability platform that provides real-time visibility into microservices and the Istio service mesh. Kiali collects data such as traffic volume, error rates, latency, and response times across microservices in real-time, allowing users to identify bottlenecks and troubleshoot issues quickly. However, Kiali cannot collect CPU usage and memory consumption, log data.

b) Centralize data monitoring and visualization tool

In Fig. 4, the implementation process of a centralized monitoring tool is depicted, highlighting the collection of metrics, logs, time series data, and other relevant information from user applications. The diagram also illustrates the seamless transfer of data between the visualization and monitoring tools. Primarily, the proposed system creates a centralized platform capable of collecting extensive data, thus eliminating the need to switch between multiple tools. There are three open-source monitoring tools have been utilized, with Prometheus being the first one.

The utilization of Prometheus in the proposed solution is motivated by its numerous advantages, including high-quality data, queries, operations, and a wide range of libraries [19]. While Prometheus excels in monitoring and analyzing metrics, it does not support trace data. Consequently, Jaeger is incorporated into the proposed solution to address this limitation and enable efficient monitoring of trace data. ElasticSearch is capable of effectively monitoring log data. This inclusion is crucial to ensure comprehensive monitoring capabilities within the proposed solution.

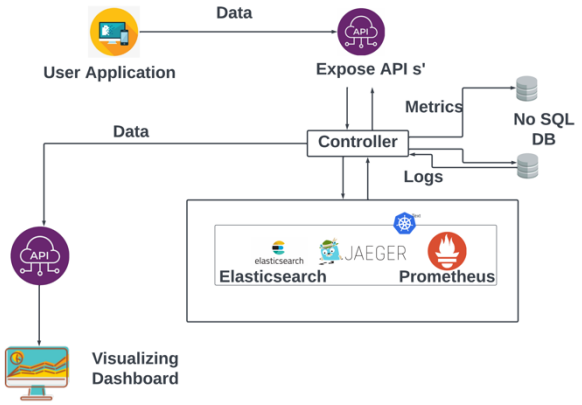

Fig. 4. High-level diagram of the monitoring and visualization tool

To establish the proposed solution, the first step involves hosting all the monitoring tools. By leveraging the cloud infrastructure, the monitoring tools can be efficiently deployed and managed. These tools expose their REST APIs, which can be accessed by making API calls to the endpoints provided by Prometheus, Jaeger, and Elasticsearch.

The proposed tool is designed to collect metrics, logs, and time series data from applications in a cloud environment. This tool utilizes APIs and libraries from open-source tools to gather data and expose these APIs to users. By utilizing these APIs, users can configure applications with the centralized tool. All the collected data is stored in MongoDB. Users can monitor and visualize metrics, logs, and time series data, eliminating the need to switch between different tools. The user can experience real-time updating of centralized cloud data visualization tools through the proposed solution.

C. *Reinforcement AI model to predict the resource allocation on demand*

a) Developing the metric server

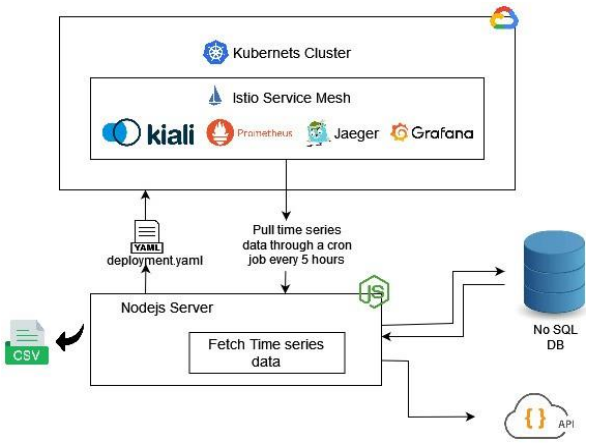

Fig. 5. Resource Prediction Server Architecture

The system architecture depicted in Fig. 5 illustrates the Node.js server utilized for fetching metrics from a deployed microservice application in a Kubernetes cluster. The Kubernetes cluster is configured with the Istio service mesh. The deployed applications under Istio are subsequently

monitored and visualized by the core metric server proposed as a centralized monitoring tool. The metrics such as CPU usage, memory usage, and network are obtained by using a PromQL query to identify CPU and memory utilization at the Kubernetes pod, and node levels within the cluster.

Moreover, Kiali enables the visualization of dependency mappings between the microservices, presenting request and response times, thereby facilitating the identification of any breakdowns occurring between microservices. The Node.js server retrieves all relevant endpoints exposed through the Istio service mesh. Upon fetching the required metrics using a scheduled cron job, they are stored in a NoSQL database along with corresponding timestamps. These time series data, along with the timestamps, are then utilized by the optimization server to generate an optimal solution. Subsequently, The Node.js server generates a CSV file from the retrieved NoSQL data, which aids in optimizing the deployment strategy of the Kubernetes cluster. Based on the solution provided by the optimization server, the deployment YAML file is updated and deployed on the cluster.

b) Developing the Optimization server

The objective of the optimization server is to predict the future workload on microservice deployment by using the historical data that was continuously fetched through the NodeJS server to create an optimal solution to microservice deployment and create an autoscaling [21] policy through the predicted resource allocation. The workload prediction is based on the resource utilization matrices that are fetched from the NodeJS server. Gathered resource utilization matrices perform Time series-based prediction on pod utilization metrics such as CPU utilization memory utilization and cluster level metrics, values are taken for a particular period of forecasting. The NodeJS server exposes an API that the optimal server can use to query the data. The data is applied for data preprocessing steps and training the data to create the prediction module to generate an optimal solution.

Various time series mechanisms are employed for predicting future data. However, in this research, the Bi-LSTM model was selected due to its capability to provide accurate predictions even with a limited amount of data. While ARIMA is a well-known time series method, it was utilized in this study to compare its accuracy with the Bi-LSTM model in predicting CPU and memory usage. The goal was to develop an autoscaling approach through vertical and horizontal auto-scaling methods. Conv1D constitutes a foundational element within CNNs, custom-tailored to analyze one-dimensional sequences, such as temporal data. Conv1D operates by sliding a series of adaptable filters across the input sequence, adeptly capturing intricate local patterns and distinctive features embedded within the data. Conversely, the Bi-LSTM stands as a derivative of the LSTM architecture within the realm of recurrent neural networks (RNNs). Bi-LSTM innovatively extends this capability by concurrently processing input sequences in both forward and backward directions, enabling holistic insight into past and future contextual information at every timestep.

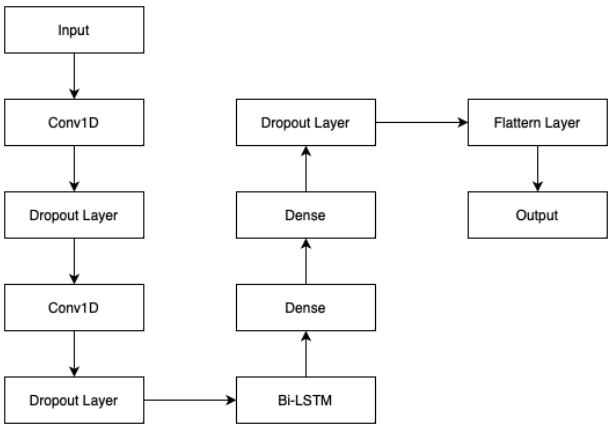

Fig. 6. The layers description of the CNN-Bi-LSTM model

The prediction process relies on a hybrid deep learning architecture that combines Convolutional 1D (Conv1D) layers with Bi-LSTM networks. This combination aims to accurately predict future data patterns. By integrating Conv1D layers with Bi-LSTM networks, our objective was to benefit from the Conv1D layers' ability to identify local patterns while also capturing extensive temporal dependencies through the Bi-LSTM networks. This hybrid approach was carefully selected to enhance the accuracy of forecasting CPU and memory usage.

IV. Result and Discussion

To evaluate the proposed system, there are multiple applications are used with multiple programming languages such as Java, Python, JavaScript, and C. The applications were built and deployed via the proposed system. The performance was increased due to the build time is not applicable for the deployment process in the proposed architecture. Furthermore, the queue time for the pipeline is reduced due to the Kubernetes slave pods without using the primary core and memory for the Jenkins server. The image-building time is reduced because of reduced downloads with separate data storage for CI. Table III describes the performance improvement of building the Docker image and push to the container registry. The performance is increased for all the tested programming languages as mentioned in Table III.

TABLE III. Performance improvement of CI Process

Language	Default Docker Image Build Time (sec)	Docker Image Build Time with Proposed System (sec)
Java	31.55	20.91
Python	20.19	15.66
JavaScript	25.67	18.87
C	19.78	14.12

The performance improvement of the deployment phase is described in Table IV. Due to the image re-tagging method, the rebuilt part is removed from the proposed system and that change reduced the CD time for the selected programming languages to test the results of the applied solution. The system performed fully automated as in the proposed system and all the builds and deployments are automated. The deployment for different environments may take the user trigger to identify the verified deployment.

TABLE IV. PERFORMANCE IMPROVEMENT OF CD PROCESS

Language	Default Deployment Time (sec)	Deployment Time with Proposed System (sec)	Performance Increase
Java	31.55 + 186.56	178.90	17.97 %
Python	20.19 + 110.78	105.24	19.64 %
JavaScript	25.67 + 130.76	110.11	29.60 %
C	19.78 + 100.45	98.89	17.73 %

As the result of the combination of selected monitoring tool APIs for the centralized monitoring tool, the proposed system can view all the necessary metrics and logs in a single platform instead of accessing different monitoring UIs to monitor the different types of metrics and logs. The solution reduces the accessing time and the monitoring time with a clear view to the end user with the powerful and user friendly UI. Metrics are gathered through a metrics server and based on the data, forecasting will be performed, and a deployment strategy will be created. Several models are trained to finalize the convenience and reliable model for the proposed system.

TABLE V. COMPARISON OF DIFFERENT MODELS

Model	MSE	RMSE	MAE
ARIMA	2.28907	1.51442	1.51805
Holt Winters	0.11957	0.34582	0.19297
BiLSTM	0.11241	0.33244	0.25719
CNN - BiLSTM	0.10100	0.31781	0.16651

The predictions were exported in CSV format and used to train the models. Table V shows that various models were trained and compared using a combination of Conv1D layers with Bi-LSTM networks to achieve more accurate predictions in less time. The combination of Conv1D and Bi-LSTM was chosen because it yields a low mean squared error and high accuracy, even with a limited amount of data.

V. CONCLUSION

This publication introduces a common architecture for fully automated and optimized software release processes under architectural attributes such as performance, security, extendibility, reliability, and modifiability. The proposed system is mainly based on open-source tools and technologies to reduce costs. Furthermore, the proposed system in this publication focuses on a centralized monitoring solution to gather all the metrics and logs in one single platform. The recourse prediction system is developed with a reinforcement model to learn and predict the resource usage on demand by learning the historical resource usage of the system per application. The overall objective of this publication is to propose a fully automated and optimized architecture for the software release process, monitoring all the metrics by a centralized dashboard and predicting and allocating resources on-demand for the Kubernetes pods by an AI model.

REFERENCES

[1] Grzegorz Blinowski, Anna Ojdowska, Adam Przybyłek, “Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation”, 18 February 2022.

[2] Luciano Baresi, Giovanni Quattrocchi, Damian Andrew Tamburri, “Microservice Architecture Practices and Experience: a Focused Look on Docker Configuration Files”, 6 Dec 2022.

[3] Josh Campbell, “The key differences between Kubernetes and Docker and how they fit into containerization”, [Online]. Available: https://www.atlassian.com/microservices/microservices-architecture/kubernetes-vs-docker, [Accessed 04 March 2023].

[4] Ionut-Catalin Donca, Ovidiu Petru Stan, Marius Misaros, Dan Gota, Liviu Miclea, “Method for Continuous Integration and Deployment Using a Pipeline Generator for Agile Software Projects”, 2022.

[5] Fernando Almeida, Jorge Simoes, Sergio Lopes, “Exploring the Benefits of Combining DevOps and Agile”, 2022.

[6] Shubo Zhang1, Tianyang Wu1, Maolin Pan1, Chaomeng Zhang2 and Yang Yu, “A-SARSA: A Predictive Container Auto-Scaling Algorithm Based on Reinforcement Learning”, 2020 IEEE International Conference on Web Services (ICWS).

[7] Yeddula Sai Dhanush Reddy, Padumati Saikiran Reddy, Nithya Ganesan, B. Thangaraju, “Performance Study of Kubernetes Cluster Deployed on OpenStack, VMs and BareMetal”, 30 August 2022.

[8] D. C Kalubowila, S. M Athukorala, B. A. S Tharaka, H. W. Y. R Samarasekara, Udara Srimath S. Samaratunge Arachchilage, Dharshana Kasthurirathna, “Optimization of Microservices Security,” December 2021.

[9] Adarsh Anand, Subrata Das, Ompal Singh, Vijay Kumar, “Testing resource allocation for software with multiple versions”, February 19, 2022.

[10] Zheng Liu, Huiqun Yu, Guisheng Fan, Liqiong Chen, “Reliability modelling and optimization for microservice-based cloud application using multi-agent system”, 13 March 2022.

[11] Z. Zhu, M. A. Naeem, and L. Li, "Automated Continuous Deployment at Facebook," in Proceedings of the 17th International Conference on Mining Software Repositories, IEEE Press, 2020, pp. 460-470.

[12] H. Guo, J. Chen, T. Li, Z. Li, L. Zhang, and J. Li, "An optimization model for software release management based on continuous delivery," Information and Software Technology, vol. 107, pp. 101-115, 2019.

[13] Spoorthi Jayaprakash Malgund, Dr Sowmyarani C N, “Automating Deployments Of The Latest Application Version Using Ci-Cd Workflow”, 2022.

[14] Yutsuki Miyashita, Yuki Yamada, Hiroaki Hashiura, Atsuo Hazeyama, “Design of the Inspection Process Using the GitHub Flow in Project Based Learning for Software Engineering and Its Practice", 6 Feb 2022.

[15] Meixia Yang, Ming Hung, “A Microservice-Based OpenStack Monitoring Tool”, 19 March 2020, doi:10.1109/ICSESS47205.2019.9040740.

[16] Lei Chen, Ming Xian, JianLiu, “Monitoring System of OpenStack Cloud Platform Based on Prometheus”,10-12 July 2020, doi:10.1109/CVIDL51233.2020.0-100.

[17] Abhishek Pratap Singh, “A Data Visualization Tool- Grafana”, Jan 26 2023.

[18] Andrea Janes, Xiaozhu Li, Valentina Lenarduzzi, “Open tracing tools: Overview and critical comparison”, Vol. 204, October 2023, 111793.

[19] Mahantesh Birje, Chetan Bulla, “Commerical and Open Source Cloud Monitoring tools: A Review”, January 2020, doi: 10.1007/978-3-030-24322-7_59

[20] Nhat-Minh Dang-Quang and Myungsik Yoo, “Deep Learning-Based Autoscaling Using Bidirectional Long Short-Term Memory for Kubernetes”, Applied Science – 11

[21] weave works, “Ensure High Availability and Uptime With Kubernetes HPA (Horizontal Pod Autoscaler) and Prometheus”, https://www.weave.works/blog/kubernetes-horizontal-pod-autoscaler-andprometheus

[22] Wei Fang; ZhiHui Lu; Jie Wu; ZhenYin Cao, “RPPS: A Novel Resource Prediction and Provisioning Scheme in Cloud Data Center”,2012 IEEE Ninth International Conference on Services Computing

[23] L. S. Hettiarachchi, S. V. Jayadeva, R. A. V. Bandara, D. Palliyaguruge, U. S. S. S. Arachchillage and D. Kasthurirathna, "Artificial Intelligence-Based Centralized Resource Management Application for Distributed Systems," 2022 13th International Conference on Computing Communication and Networking Technologies (ICCCNT), Kharagpur, India, 2022, pp. 1-6, doi: 10.1109/ICCCNT54827.2022.9984530.