

**STREAMLINING SOFTWARE RELEASE PROCESS  
AND RESOURCE MANAGEMENT FOR  
MICROSERVICES BASED ARCHITECTURE ON  
MULTI-CLOUD**

TMP-23-193

Project Proposal Report

M.R.A. Fadhil

B.Sc. (Hons) in Information Technology Specializing in Software  
Engineering

Department of Computer Science and Software Engineering

Sri Lanka Institute of Information Technology

Sri Lanka

March 2023

**STREAMLINING SOFTWARE RELEASE PROCESS  
AND RESOURCE MANAGEMENT FOR  
MICROSERVICES BASED ARCHITECTURE ON  
MULTI-CLOUD**

TMP-23-193

Project Proposal Report

B.Sc. (Hons) in Information Technology Specializing in Software  
Engineering

Department of Computer Science and Software Engineering

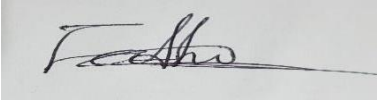
Sri Lanka Institute of Information Technology  
Sri Lanka

March 2023

## Declaration

I declare that this is my work, and this proposal does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any other university or institute of higher learning. To the best of my knowledge and belief, it does not contain any previously published material written by another person except where the acknowledgement is made in the text.

Signature

IT20784720	Fadhil M.R. A	
------------	---------------	-------------------------------------------------------------------------------------

The above candidates are researching the undergraduate Dissertation under my supervision.

Signature of the Supervisor: .....

Date: .....05/06/2022.....

## ABSTRACT

The deployment of microservices on Kubernetes clusters has become a popular approach for building scalable and resilient distributed systems. However, managing the resources of these microservices can be challenging, especially when dealing with varying workloads and resource demands. To address this challenge, we propose a resource management framework that combines machine learning and reinforcement learning algorithms to predict the resource load of microservices and automate the deployment strategy based on the prediction.

Our framework utilizes Docker and service mesh to deploy microservices on an Azure Kubernetes cluster, and uses Prometheus, Grafana, and Kiali to collect and analyze the time-series data of resource utilization. We propose a machine learning algorithm that predicts the resource load of microservices based on the collected data, and a reinforcement learning algorithm that optimizes the deployment strategy based on the prediction.

To evaluate the performance of our framework, we conducted concurrent performance tests using Gatling, JMeter, and Locust with large loads, and also performed throttling, error injection, and chaos testing. Our results show that our framework can effectively manage the resources of microservices and achieve better performance compared to traditional deployment strategies.

In conclusion, our proposed framework provides a promising approach for managing the resources of microservices deployed on Kubernetes clusters and has the potential to improve the scalability and resilience of distributed systems.

Keywords: Kubernetes, service mesh, monitoring, Docker, Performance , resilience , machine learning, optimization

## TABLE OF CONTENT

Declaration	3
Abstract	4
Table Of Content	5
List Of figures	6
List Of Tables	7
List Of Abbreviations	8
1.0 Introduction	9
2.0 Research Problem	14
3.0 Objectives	
3.1 Main Objective	15
3.2 Specific Objective	15
4.0 Methodology	
4.1 Requirement Gathering	16
4.2 Software Solution and System Analysis	17
4.3 Project Requirements	20
4.4 Testing	20
4.5 Feasibility Study	21
5.0 Personal and Facilities	23
6.0 Commercialization	24
7.0 Budget	25
References	26

## List Of Figures

Figure 1: Architecture of ARIMA Model	10
Figure 2: Architecture Prometheus matrices gathering for prediction	11
Figure 3: Architecture of DScaler	12
Figure 4: System Overview Diagram	17
Figure 5: Gantt Chart	22

## List Of Tables

Table 1: existing research on scalability, metrics and technology used	12
Table 1: tools and technologies	19
Table 3: Personal and facilities	23
Table 4: Budget	25

## List Of Abbreviations

VM	Virtual Machine
HPA	Horizontal Pod Auto Scaling
VPA	Vertical Pod Auto Scaling
AKS	Azure Kubernetes Service



## 1.0 INTRODUCTION

Containerization is a technology that allows applications to be packaged and run in lightweight, isolated environments. Instead of running a complete virtual machine (VM) with its own operating system (OS) on the top of virtualized hardware, containers provide an isolated environment for system resources (e.g., processes, file systems and networks) to run at the host OS level, this means that multiple containers can be shared with the same OS kernel, hence start much faster, and use just a fraction of the system memory compared to booting an entire virtualized OS [14].

Due to the popularity of microservices the need to scale and automate the deployment was increased. So, Google developed an open-sourced container orchestration platform called Kubernetes, also known as k8s. Kubernetes allows users to deploy and manage microservices in a distributed environment, make it a popular choice for building and deploying microservices based applications. One of the important features provided by Kubernetes is autoscaling in different ways such as automatically scale the number of pods running a microservice based on the metrics (Horizontal Pod scaling), automatically adjust the resource limit of individual pods according to the usage (Vertical pod scaling). Kubernetes scaling was made according to the workload, so auto scaling microservices applications can be not accurate since services are distributed.

In [13] proposes an intelligent workload factoring approach to optimize resource allocation in a hybrid cloud computing model. So, by identifying the challenges of workload factoring in hybrid cloud environments, where workloads can be distributed across multiple public and private clouds. The authors propose a solution that uses a workload characterization module to classify workloads into categories based on their resource requirements and performance characteristics. Then, a decision module uses this information to allocate resources optimally across different clouds, considering factors such as cost and network latency. The authors also propose a feedback mechanism to adjust the workload factoring strategy dynamically based on changes in workload and resource availability.

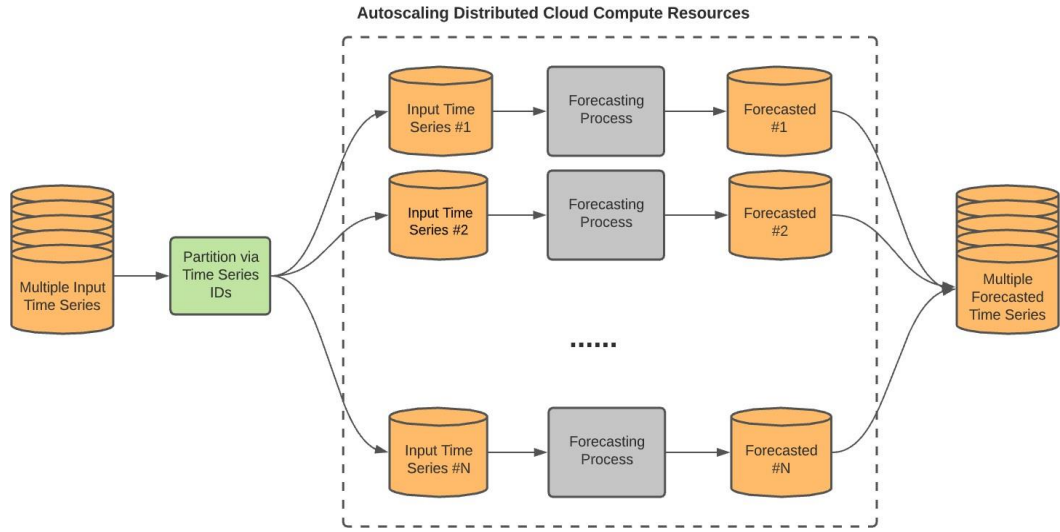


Figure 1: Architecture of ARIMA Model

In [15] proposes a deep learning-based autoscaling algorithm for Kubernetes, which is a popular open-source container orchestration platform. The proposed algorithm utilizes bidirectional long short-term memory (BiLSTM) neural networks to predict the future resource utilization of a Kubernetes cluster and dynamically adjust the number of replicas of containerized applications to optimize resource usage. The authors first trained the BiLSTM model on a dataset of historical resource utilization data collected from a Kubernetes cluster. The model was trained to predict future resource utilization based on past utilization patterns. Once the model was trained, it was deployed as a microservice within the Kubernetes cluster itself. The model continuously receives real-time resource utilization data from the Kubernetes cluster and predicts future utilization. The predicted future utilization is then used to dynamically adjust the number of replicas of containerized applications to optimize resource usage. so by, evaluating the proposed algorithm using a real-world dataset collected from a Kubernetes cluster running various microservices. The evaluation results showed that the proposed algorithm achieved better performance in terms of resource utilization and response time compared to existing Kubernetes autoscaling algorithms such as the default Horizontal Pod Autoscaler (HPA) and the Predictive Horizontal Pod Autoscaler (PHPA).

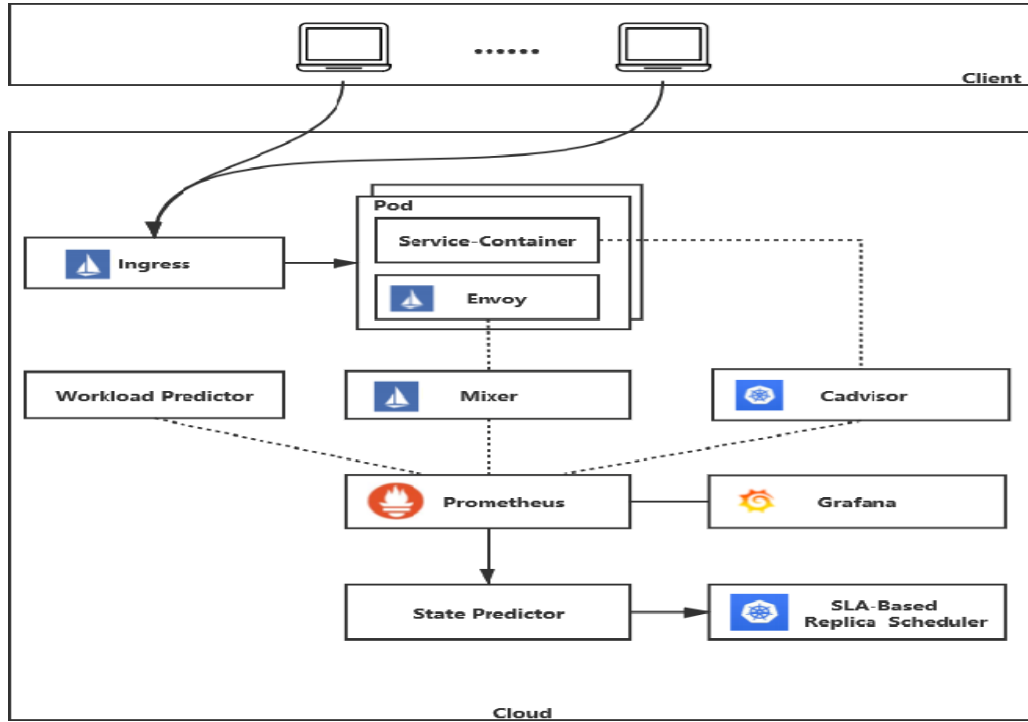


Figure 2: Architecture Prometheus matrices gathering for prediction.

In publication [2], proposes a new auto-scaling algorithm based on reinforcement learning (RL) for containerized cloud applications. The authors argue that existing auto-scaling approaches are often reactive and do not consider future resource usage, which leads to performance degradation and unnecessary resource consumption. To address these issues, the authors propose A-SARSA, an RL-based auto-scaling algorithm that takes both current and predicted future resource usage into account. A-SARSA uses the state-action-reward-state-action (SARSA) algorithm to learn an optimal policy that determines when to add or remove containers based on predicted future resource usage. The authors evaluate A-SARSA using several real-world benchmarks and show that it outperforms existing auto-scaling algorithms in terms of both resource utilization and application performance. A-SARSA achieves higher resource utilization by scaling proactively based on predicted future usage, and improves application performance by avoiding under-provisioning and over-provisioning of resources. Overall, the paper proposes an innovative approach to auto-scaling containerized cloud applications that leverages RL to learn and optimize resource allocation policies. The results suggest that A-SARSA can improve both resource utilization and application performance, making it a promising direction for future research in this area.

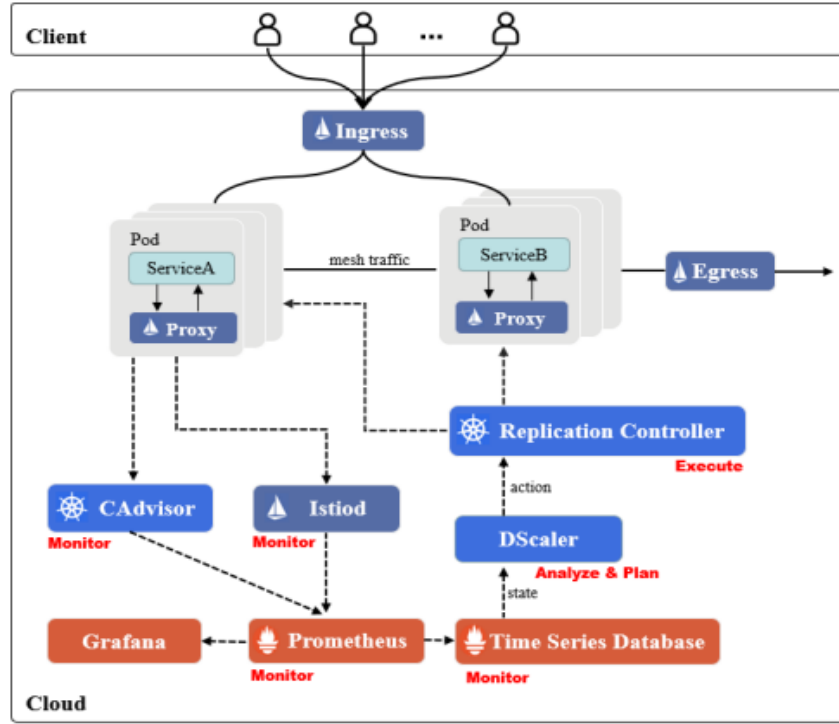


Figure 3: Architecture of DScaler

DScaler [1] was a horizontal autoscaler for microservice based application developed based on deep reinforcement learning. DScaler was developed for the usage of large scale and more complex microservice architecture applications to fix the scaling issues. DScaler uses a combination of deep neural networks and one of reinforcement learning algorithm called Q-learning algorithm to learn a policy for scaling microservices based system metrics. The proposed DScaler algorithm was tested with large e-commerce applications and with some rule based approaches. as a response DScaler has a good performance in resource utilization and response time.

Paper	Virtualization Technology	Monitored Metrics	Method	Technique
Zhang et al. [3]	VM	Request rate	Reactive	ARIMA
Li and Xia [4]	Container	CPU	Proactive	ARIMA
Rodrigo et al. [5]	VM	Request rate	Proactive	ARIMA
Xuehai Tang et al.[6]	Container	CPU	Proactive	Bi-LSTM
Ming Yan et al. [7]	Container	CPU, Memory	Hybrid	Bi-LSTM

Ming Yan et al. [7]	Container	CPU, Memory	Hybrid	Bi-LSTM
Imdough et al. [8]	Container	Request rate	Proactive	LSTM
Laszlo Toka et al.[9]	Container	Request rate	Proactive	LSTM
Arabnejad et al. [11]	VM	CPU	Reactive	RL
Fabiana et al. [12]	Container	CPU	Reactive	RL
Prachimutita et al.[10]	VM	Request rate	Proactive	ANN, RNN

Table 1: existing research on scalability, metrics and technology used.

Most of the research is based on the VM and container-based prediction for scale the resource load .so there are certain limitations on selecting the matrices for start the predictions. and all research is scaled based on the microservice architecture and deployed in and cloud-based cluster so there will be a cost that will be associated according to the resource load. So, through this component, first a prediction was made and according to the prediction by using a reinforcement algorithm, the predicted result will be optimized.

## 2.0 RESEARCH PROBLEM

Resource management and scaling are critical issues for microservices deployed on Kubernetes clusters. As the number of microservices and the volume of traffic increases, it becomes challenging to manage the resources effectively and ensure optimal performance. Traditional approaches to resource management, such as manual scaling, are not efficient and can lead to over-provisioning or under-provisioning of resources, which can impact the performance and cost of the system.

To address these challenges, there is a need for a resource management framework that can dynamically manage the resources of microservices based on the workload and resource demands. This framework should utilize modern technologies, such as machine learning algorithms, to predict the resource load of microservices and automate the deployment strategy based on the prediction.

Therefore, the research problem to be addressed in this research paper is: How to develop a machine learning-based resource management framework for microservices deployed on Kubernetes clusters? How to utilize machine learning algorithms to predict the resource load of microservices through time series data queried and propose a reinforcement learning algorithm to automate the optimized deployment strategy based on the prediction? How to evaluate the performance of the framework under different workloads and resource demands to ensure scalability, efficiency, and cost-effectiveness?

## **3.0 OBJECTIVES**

### **3.1 Main Objective**

this component aims to develop an optimal solution to Kubernetes cluster to allocate the resource requirements according to the demand that changes with time and complexity of the microservices. so as a first step set up the Istio service mesh in the Kubernetes cluster and get the time series matrices and logs from monitoring tools like Prometheus, Grafana and Kiali. By using prediction models and reinforcement learning, an autoscaling approach can be developed.

### **3.2 Specific Objectives**

1. Configure the Kubernetes cluster.
2. Configure the Istio service mesh.
3. Get the time series matrices of microservices using an API of Prometheus, Grafana and Kiali
4. Store those gathered time series data to a persistent database, so that can be use in the machine learning model to make the predictions.
5. Identify and apply a machine learning algorithm that can be used to predict the resource load of microservices through the time series data queried from the cluster.
6. Propose a reinforcement learning algorithm to automate the optimized deployment strategy based on the prediction.
7. Configure Kubernetes cluster health check.
8. Perform concurrent performance test with large loads (Gatling, JMeter, Locust).
9. Automating performance test and make as a stage in CI pipeline execution.
10. Propose a way to perform throttling, error injection and chaos testing.

## **4.0 METHODOLOGY**

### **4.1 Requirement Gathering**

Requirements are gathered for this component by referring several research papers , articles , publishes done in recent years and also by having a conversation with industry expertise. the knowledge to start the development and the configurations are gathered by following some Udemmy courses alongside by referring to the documentations and developing some practical's for test purpose to get the clear understanding on the tools and configurations.

#### **4.1.1 Past Research analysis**

Past research analysis are done by analyzing past research publishes on the area of Kubernetes, setting up Istio service mesh , Prometheus, Grafana and Kiali , existing prediction models that are used on resource management of Kubernetes , problems on storage allocation on Kubernetes , performance test with large loads by using frame works , tools that can be used on chaos testing.



## 4.2 Software solution and System analysis

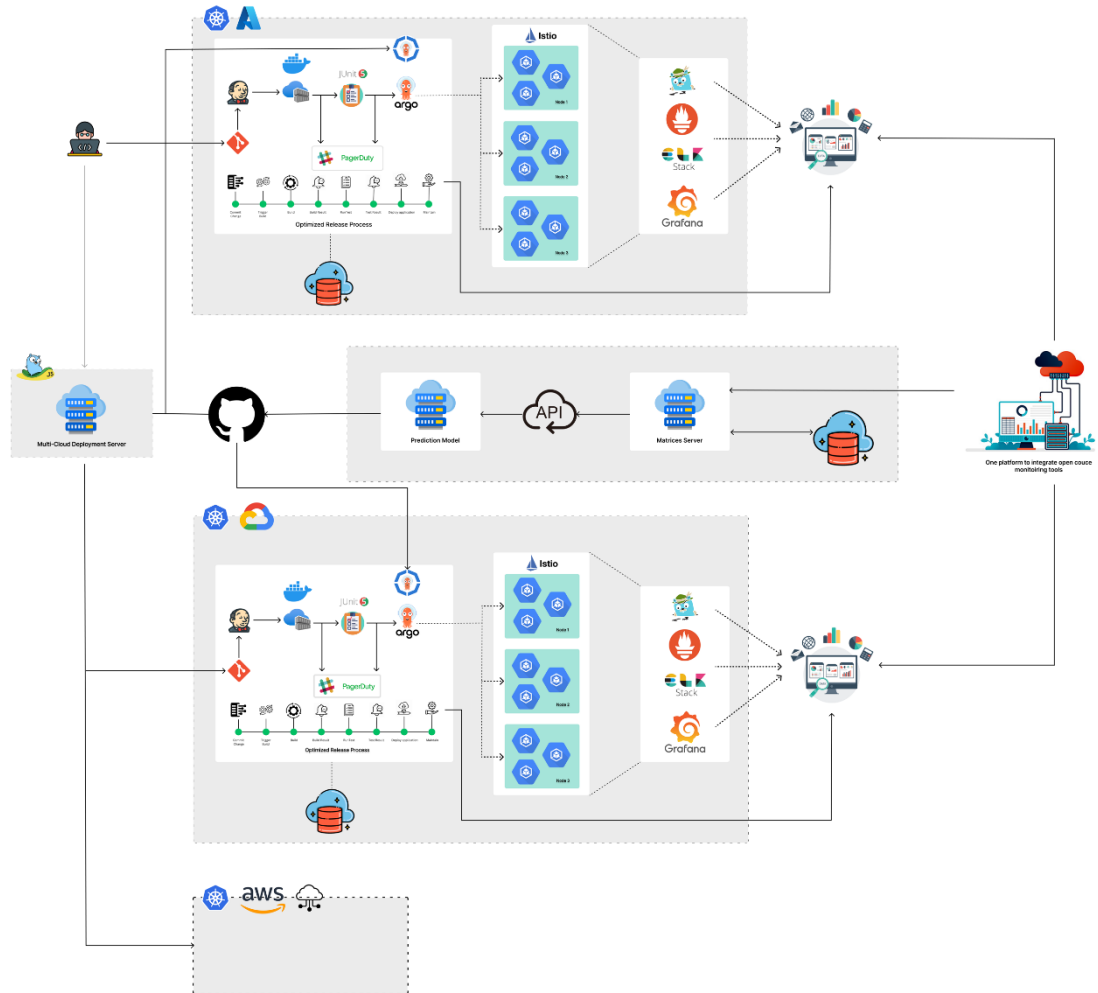


Figure 4: System Overview Diagram

in this component a helm chart was created with a set of Kubernetes yaml files, which define the resource requirement to run the application, such as deployment, services, and configmaps. So, the created helm chart was stored in a helm hub. So, from the repository the chart was retrieved and deployed to the Kubernetes cluster. By enabling the relevant configurations for Prometheus, Grafana and Kiali it can be used to query the data and stored in a persistence database for make the predictions.

When it comes to testing the performance of a Kubernetes cluster, tools like Gatling, JMeter, and Locust can be used to simulate various scenarios and test the clusters' ability to handle different loads. Automating performance tests and making them part

of CI pipelines is essential to guarantee performance and quality expectations and reduce downtimes. Datadog can be integrated with Gatling, JMeter and Locust to provide real-time visibility into the performance of your applications and infrastructure during the load testing.

Chaos monkey and Kube-monkey are open-source tools that simulate random failures in Kubernetes clusters.

Chaos testing is a technique that is used to test the resiliency of a Kubernetes cluster by deliberately injecting failure scenarios.

The steps that are followed in this component as follows,

- 1) create a Kubernetes cluster using a cloud provider such as Azure AKS using Free tier.
- 2) Configure Istio service mesh, Prometheus, Grafana and Kiali in cloud environment and deployed to AKS cluster.
- 3) Gather the data and logs from the metrics and store them in a persistent database that can be used to make the prediction using a machine learning model.
- 4) By using some popular and accurate time series prediction models like ARIMA and LSTM the prediction can be done, and the limit can be updated in the Kubernetes Vertical Pod Scaling (VPA).
- 5) Based on the predicted result, by using Deep Reinforcement Learning results can be optimized.
- 6) Cluster Health Check (CHC), would, based on a criterion, automatically detect all deployed services and pull their Health Check Status over /healthz API endpoint. Using a rolling time window, the CHC service would calculate the overall availability of the cluster and provide the status of the cluster health on demand – either success (HTTP 200) or a failure (HTTP 4xx-5xx).
- 7) Generate a status report for CHC.
- 8) Configure Gatling and JMeter to run the performance tests on the Kubernetes cluster.
- 9) Create an automation script to automate the execution of the performance tests, including commands to start the Gatling or JMeter test and to generate test reports.
- 10) Modify the existing pipeline to include the automation scripts for the performance tests.
- 11) Generate a test reports using Gatling and JMeter after each test run, including metrics such as response time, throughput, error rate and with graphs and charts.

- 12) Configure and setup Datadog agent on the where Gatling or JMeter is running. The agent collects and sends metrics and logs to Datadog.
- 13) Create a random attack through Chaos Monkey.
- 14) Analyses the resource performance through the Datadog dashboard.
- 15) Create alerts based on the metrics and logs in the Datadog dashboard and analyze them to identify performance issues.

The following technologies are used to develop the proposed system.

Programming Language	Java Go Node js Python
Tools and Technologies	Docker Kubernetes Istio JMeter TensorFlow Prometheus Grafana Azure Git GitHub

Table 2: tools and technologies

## **4.3 Project Requirements**

### **4.3.1 Functional Requirements**

- gather time series matrices and logs, structure the data and store it in a persistence database.
- Expose an API to fetch data, that includes all necessary details for creating a machine learning model.
- Develop a layer to deploy the optimal result to the cluster.
- A script to start the Gatling and execute the performance tests.
- The system should be able to display the cluster information through a monitor server.
- There must be a change in the performance when there was an attack for cluster.

### **4.3.2 Non-Functional Requirements**

The following are the non-functional requirements that are prioritized for the proposed component as follows.

- Availability
- Scalability
- Performance
- Reliability
- Maintainability
- Usability

## **4.4 Testing**

To make the process and functionalities of the research component follow up in the correct way and to reduce the bugs or errors that can occur during development, create a sample development workflow with configurations and deployment, so if there are any problems occur, it's easy to identify and fix, before developing the original workflow.

## **4.5 Feasibility Study**

### **4.5.1 Technical Feasibility**

The research component involves various tools and technologies that can be used to monitor, scale and make predictions, so technical feasibility was considered at the requirement analysis phase of the research.

#### **Knowledge in Kubernetes**

Microservices applications are deployed in Kubernetes cluster, so some concepts like deployment, services, nodes, pods, etcd, persistence volumes, persistence volume claim and horizontal and vertical auto scaling that covered in this component, with additional advanced concepts .so each member of the group need to understand the basic of Kubernetes to create a deployment. The overall research will mostly use Kubernetes with each member's components.

#### **Knowledge in microservices**

The overall tools and technologies are mainly come around with a microservice architecture deployment and monitoring. so, a standard microservice application will be deployed to the Kubernetes cluster, so develop a microservice based application. Knowledge is necessary.

#### **Knowledge in machine learning**

The main goal of this component to predict the resource load and create optimal result that can be assigned to the limit of the vertical auto sculling through gathered time series data.to create a model to optimize the result knowledge of machine. Learning is important, and in this component deep reinforcement learning algorithm also used to get more optimized solution.

## Knowledge on Monitoring tools

In the component time series matrices are gathered from tools like Prometheus, Grafana using an API call. And, to monitor the performance of the Kubernetes cluster we use Datadog, so the idea to make the relevant API calls and configurations an understanding is important.

### 4.5.2 Scheduled feasibility

The proposed research project needs to be completed within a time frame, so the group members following a grant chart to follow up the status of the research, where supervisors also can follow up the status of the completion.

## 4.6 Timeline

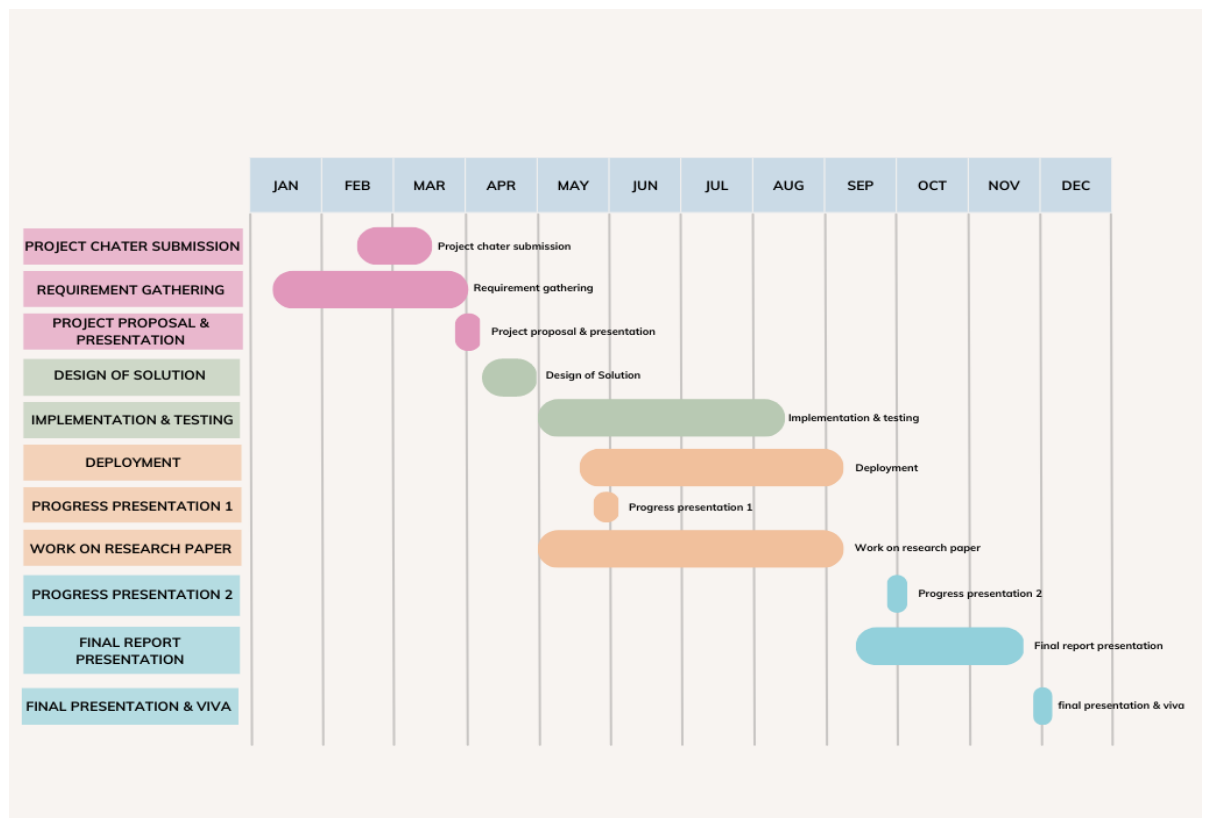


Figure 5: Gantt Chart

## 5.0 Personal and Facilities

Member	Task
M.R.A. Fadhil	<p>Config Kubernetes cluster</p> <p>Configure Istio service mesh in the Kubernetes.</p> <p>Configure monitoring tools.</p> <p>Gather the relevant time series matrices and create an API.</p> <p>Create a optimize prediction model using prediction algorithms and reinforcement learning.</p> <p>Configure performance tools with CI pipeline and data dog.</p> <p>Start chaos testing by creating random attack using Chaos Monkey</p> <p>Configure Kubernetes cluster health check and generate a report.</p>

Table 3: Personal and facilities

## 6.0 COMMERCIALIZATION

The commercialization potential for this research is high as it provides an efficient and automated way to automate and optimize release process. The solution could be marketed to companies and organizations that are using Kubernetes clusters for their applications and services. The automated release process and optimal autoscaling capabilities would allow them to reduce the time and resources required to manage their clusters and improve the performance of their applications.

Following are the advantage offered by the proposed system:

1. The automated release process can help organizations save costs by reducing the manual effort required for releases, minimizing the risk of errors, and reducing the need for additional resources.
2. By automating the release process, organizations can reduce the time required for each release, which can lead to faster time-to-market for their products.
3. The optimized architectural attributes of the software release process can help ensure that the releases are of high quality and meet the desired performance, security, extendibility, reliability, and modifiability requirements.
4. By adopting an automated and optimized release process, organizations can differentiate themselves from their competitors by offering faster and higher quality software releases.
5. As more organizations adopt the automated and optimized release process, it can become an industry standard, which can further enhance the commercialization of this solution.

Furthermore, the automated and optimized release process can help organizations save cost by reducing manual effort for the releases. The optimal architecture can be used to ensure the performance of the cluster using popular monitoring tools and the multi-cloud support would make it attractive to businesses that operate across multiple cloud providers. Autoscaling could also be extended to other areas, such as network management or resource allocation, which could increase its commercialization potential further.



## 7.0 BUDGET

The proposed system for the fully automated and optimized software release process is a software-based solution. So, for the implementation of the proposed system, there are no hardware components involved. There for no additional cost involved in the implementation process for the hardware. The proposed system was implemented in top of a cloud environment.so, as a solution Azure was used as a cloud environment to implement the proposed system. There for the primary cost will be the subscription based to Azure for the creation of the Kubernetes cluster and usage of virtual machines will be the primary source of cost.

Expense Category	Cost Per Month
Azure Subscription	free
Internet use and web hosting	3500 LKR
Publication cost	2000 LKR
Stationary	1000 LKR
Total	6500 LKR

Table 4: Budget

The proposed system will be deployed using Azure free tier Azure offers, which provides \$100 free credits for a period of 12 months. So, the credits will be used for the creating Kubernetes cluster for testing and implementing the proposed system.

## REFERENCES

- [1] Zhijiao Xiao, Song Hu, "DScaler: A Horizontal Autoscaler of Microservice Based on Deep Reinforcement Learning " The 23rd Asia-Pacific Network Operations and Management Symposium (APNOMS) 2022, IEEE Press, 2020, pp. 460-470.
- [2] Shubo Zhang , Tianyang Wu, Maolin Pan, Chaomeng Zhang and Yang Yu "A-SARSA: A Predictive Container Auto-Scaling Algorithm Based on Reinforcement Learning " 2020 IEEE International Conference on Web Services (ICWS) , vol. 107, pp. 101-115, 2019.
- [3] Zhang, H.; Jiang, G.; Yoshihira, K.; Chen, H.; Saxena, A. Intelligent Workload Factoring for a Hybrid Cloud Computing Model. In Proceedings of the 2009 Congress on Services—I, Los Angeles, CA, USA, 6–10 July 2009; pp. 701–708. .
- [4] Fang, W.; Lu, Z.; Wu, J.; Cao, Z. RPPS: A Novel Resource Prediction and Provisioning Scheme in Cloud Data Center. In Proceedings of the 2012 IEEE Ninth International Conference on Services Computing, Honolulu, HI, USA, 24–29 June 2012; pp. 609–616.
- [5] Calheiros, R.N.; Masoumi, E.; Ranjan, R.; Buyya, R. Workload Prediction Using ARIMA Model and Its Impact on Cloud Applications' QoS. *IEEE Trans. Cloud Comput.* 2015, 3, 449–458.
- [6] Tang, X.; Liu, Q.; Dong, Y.; Han, J.; Zhang, Z. Fisher: An Efficient Container Load Prediction Model with Deep Neural Network in Clouds. In Proceedings of the 2018 IEEE Intl Conf on Parallel Distributed Processing with Applications, Ubiquitous Computing Communications, Big Data Cloud Computing, Social Computing Networking, Sustainable Computing Communications (ISPA/IUCC/BDCLOUD/SocialCom/SustainCom), Melbourne, Australia, 11–13 December 2018; pp. 199–206.
- [7] Yan, M.; Liang, X.; Lu, Z.; Wu, J.; Zhang, W. HANSEL: Adaptive horizontal scaling of microservices using Bi-LSTM. *Appl. Soft Comput.* 2021, 105, 107216
- [8] Imdoukh, M.; Ahmad, I.; Alfaiakawi, M.G. Machine learning-based auto-scaling for containerized applications. *Neural Comput. Appl.* 2019, 32, 9745–9760.
- [9] Toka, L.; Dobreff, G.; Fodor, B.; Sonkoly, B. Adaptive AI-based auto-scaling for Kubernetes. In Proceedings of the 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID), Melbourne, Australia, 11–14 May 2020; pp. 599–608.

- [10] Prachitmutita, I.; Aittinonmongkol, W.; Pojjanasuksakul, N.; Supattatham, M.; Padungweang, P. Auto-scaling microservices on IaaS under SLA with cost-effective framework. In Proceedings of the 2018 Tenth International Conference on Advanced Computational Intelligence (ICACI), Xiamen, China, 29–31 March 2018; pp. 583–588.
- [11] H. Arabnejad, C. Pahl, P. Jamshidi, and G. Estrada, “A comparison of reinforcement learning techniques for fuzzy cloud auto-scaling,” in Proc. 17th IEEE/ACM Int. Symp. Clust. Cloud Grid Comput. (CCGRID), 2017, pp. 64–73.
- [12] F. Rossi, M. Nardelli, and V. Cardellini, “Horizontal and vertical scaling of container-based applications using reinforcement learning,” in Proc. 12th IEEE Int. Conf. Cloud Comput. (CLOUD), 2019, pp. 329–338.
- [13] Hui Zhang, Princeton, Guofei Jiang; Kenji Yoshihira; Haifeng Chen; Akhilesh Saxena “Intelligent Workload Factoring for a Hybrid Cloud Computing Model”, 2009 Congress on Services – I
- [14] Arnaldo Pereira Ferreira, Richard O. Sinnott, “A Performance Evaluation of Containers running on Managed Kubernetes Services” , 2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)
- [15] Nhat-Minh Dang-Quang and Myungsik Yoo , “Deep Learning-Based Autoscaling Using Bidirectional Long Short-Term Memory for Kubernetes”, applied science – 11

