

**A NETWORK SCIENCE BASED APPROACH FOR  
OPTIMAL MICROSERVICE GOVERNANCE**

2020-021

Project Proposal Report

De Silva N.

B.Sc. (Hons) Degree in Information Technology Specialized in Software  
Engineering

Department of Software Engineering

Sri Lanka Institute of Information Technology  
Sri Lanka

February 2020

**A NETWORK SCIENCE BASED APPROACH FOR  
OPTIMAL MICROSERVICE GOVERNANCE**

2020-021

Project Proposal Report

B.Sc. (Hons) Degree in Information Technology Specialized in Software  
Engineering

Department of Software Engineering

Sri Lanka Institute of Information Technology  
Sri Lanka

February 2020

## Declaration

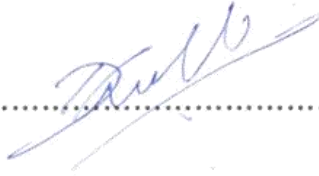
I declare that this is my own work and this proposal does not incorporate without acknowledgment any material previously submitted for a degree or diploma in any other university or institute of higher learning and to the best of our knowledge and belief it does not contain any material previously published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

Signature:

IT17006880	De Silva N.	N. De Silva
------------	-------------	-------------

The above candidates are carrying out research for the undergraduate Dissertation under my supervision.

Signature of the Supervisor: .....



Date: 23/02/2020 .....

## **Abstract**

One of the many advantages of using Kubernetes for microservice deployments is the ability to scale cluster resources based on traffic. This ability of Kubernetes to automatically scale cluster resources thereby enables developers and system administrators to reduce costs and make use of their deployed microservices more effectively and efficiently.

However, even though the autoscaling tools such as the Horizontal Pod Autoscaler (HPA) provided by Kubernetes help developers and system administrators to effectively utilize their microservice deployments as previously stated, they also possess some inherent drawbacks as well. The main drawback among them being that these autoscaling policies adopted are primarily local and rule-based auto-scaling techniques which do not consider the overall impact of the auto-scaling policy in order to truly optimize the existing resources in a particular deployment.

This research thereby aims to address this issue prevalent through aiming to provide a data science-based approach with the help of statistical and machine learning-based time series prediction measures to predict the load of microservice and thereby evaluate and determine centrality, based on identified dependency measurements to ultimately develop an improved auto-scaling policy which takes into account the global importance of a particular microservice.

**Keywords:** Auto-scaling, Kubernetes, Machine Learning, Microservices, Time Series

# Table of Contents

<b>Declaration.....</b>	<b>i</b>
<b>Abstract.....</b>	<b>ii</b>
<b>Table of Contents .....</b>	<b>iii</b>
<b>List of Figures.....</b>	<b>v</b>
<b>List of Tables .....</b>	<b>vi</b>
<b>List of Abbreviations .....</b>	<b>vii</b>
<b>1.0 INTRODUCTION.....</b>	<b>1</b>
<b>1.1 Background and Literature .....</b>	<b>3</b>
<b>1.2 Research Gap .....</b>	<b>7</b>
<b>1.3 Research Problem .....</b>	<b>9</b>
<b>2.0 OBJECTIVES .....</b>	<b>10</b>
<b>2.1 Main Objective .....</b>	<b>10</b>
<b>2.2 Specific Objectives .....</b>	<b>10</b>
<b>3.0 METHODOLOGY.....</b>	<b>11</b>
<b>3.1 Requirement Gathering.....</b>	<b>11</b>
<b>3.1.1 Past Research Analysis .....</b>	<b>11</b>
<b>3.1.2 Identifying Existing Systems .....</b>	<b>11</b>
<b>3.2 Feasibility Study .....</b>	<b>12</b>
<b>3.2.1 Technical Feasibility .....</b>	<b>12</b>
<b>3.2.2 Schedule Feasibility.....</b>	<b>12</b>
<b>3.2.3 Economic Feasibility .....</b>	<b>12</b>
<b>3.3 Requirement Analysis.....</b>	<b>13</b>
<b>3.4 System Analysis.....</b>	<b>14</b>
<b>3.5 System Development and Implementation .....</b>	<b>16</b>
<b>3.6 Project Requirements .....</b>	<b>18</b>
<b>3.6.1 Functional requirements .....</b>	<b>18</b>
<b>3.6.2 Non-Functional Requirements.....</b>	<b>18</b>
<b>3.7 Testing .....</b>	<b>20</b>
<b>4.0 PERSONEL AND FACILITIES .....</b>	<b>21</b>
<b>5.0 COMMERCIALIZATION .....</b>	<b>23</b>
<b>6.0 BUDGET.....</b>	<b>24</b>

<b>7.0 SUMMARY .....</b>	<b>25</b>
<b>8.0 REFERENCES.....</b>	<b>26</b>

**List of Figures**

Figure 1.1 MAPE Control Loop ..... 4

Figure 3.1 Load Prediction Component in the Proposed Governance Model ..... 14

Figure 3.2 System Overview ..... 14

## **List of Tables**

Table 3.1 Tools and Technology.....	17
Table 4.1 Personnel and Resources.....	21
Table 5.1 Budget .....	22



## List of Abbreviations

Abbreviation	Description
ARIMA	Auto-Regressive Integrated Moving Average
AWS	Amazon Web Service
EMD	Empirical Mode Decomposition
HPA	Horizontal Pod Autoscaler
MAPE	Monitor Analyze Plan Execute
QoS	Quality of Service
SLA	Service Level Agreement
VM	Virtual Machine
VPA	Vertical Pod Autoscaler

## 1.0 INTRODUCTION

Kubernetes has become one of the most popular platforms for deploying microservices today, due to its ability to manage and orchestrate containerized workloads and services. The use of Kubernetes has therefore enabled developers the ability to move on from the traditional deployment methodologies such as physical servers and Virtual Machines and make use of the newer containerized deployment methodologies through the use of open-source tools such as Docker. Furthermore, since Kubernetes provides a vast array of features and services such as service discovery and load balancing, storage orchestration, self-healing, etc. [1], it has managed to become one of the leading container orchestration tools currently available.

Among the many services and that Kubernetes offers among its vast array of features, is the ability to automatically scale based on workloads. This process is known as autoscaling and refers to the automatic scaling of computational resources based on the required workloads. This autoscaling feature thereby enables developers to make use of optimal use resources while minimizing the incurred cost.

However, the above-mentioned autoscaling feature primarily adopts local and rule-based auto-scaling techniques to dynamically manage the number of microservice resources in a particular deployment. These rule-based techniques make use of a limited amount of infrastructure-level metrics such as CPU utilization to determine thresholds for autoscaling policies. However, doing so creates a lack of global awareness when making effective autoscaling decisions due to the use of a limited amount of metrics considered in autoscaling processes which in turn, may cause under or over-provisioning of resources and ultimately result in ineffective autoscaling. Therefore, in order to truly optimally utilize the existing resources in a microservice-based deployment, it may be necessary to have a global view of how each microservice is utilized.

This research aims to develop a solution to the above-described problem by aiming to adopt a data-science based approach with the use of statistical or machine learning-based time series prediction measures to develop an algorithm to predict future loads and thereby use these predicted loads to evaluate the centrality of a particular microservice, in order to produce a more effective autoscaling policy based on the global importance of a microservice.

## 1.1 Background and Literature

According to popular cloud providers such as AWS, auto-scaling is defined as a cloud computing service feature that allows AWS users to automatically launch or terminate virtual instances based on defined policies, health status checks, and schedules [2] whereas other publications such as [3] define autoscaling as a mechanism of dynamically acquiring or releasing resources to meet QoS requirements. However, one of the most in-depth descriptions of the definition of autoscaling is provided by the publication [4]. This publication clearly manages to summarize the key features of autoscaling, which can be listed as follows.

- Ability to scale out (addition of extra unused resources during increased demand) and scale in (removal of extra unused resources during reduced demand)
- Capability of setting rules for scaling in and out
- Automatically detect and remove unhealthy instances

Due to these features, autoscaling is widely adopted in a variety of cloud platforms as well in other technologies to scale a wide variety of resources ranging from VMs to other resources such as pods in orchestration tools such as Kubernetes. Additionally, due to the elastic nature which allows resources to be provisioned more conservatively, cloud providers can serve more customers with the same infrastructure.

Furthermore, the autoscaling process for the resources mentioned above can be performed using a variety of techniques, as described in [5]. However, among them, rule-based autoscaling is the technique most commonly used by many cloud vendors today [6] as well as in other orchestration platforms such as Kubernetes, evident through its HPA and VPA autoscaling tools.

Rule-based autoscaling involves defining the conditions under which capacity will be added to or removed from a cloud-based system, in order to satisfy the objectives of the application owner [6]. Therefore, for a rule-based autoscaling approach to be effective, the application provider has to specify upper and lower bounds, which are

usually defined through a performance metric such as CPU utilization. This approach to rule-based approach to autoscaling has therefore defined rule-based autoscaling as a more reactive approach to provision resources since the autoscaling process occurs when the defined thresholds and bounds set, are exceeded.

Another key characteristic that is present in most rule-based autoscaling policies is the adaptation of the MAPE (Monitor, Analyze, Plan, and Execute) control loop reference model used in various autonomic computing systems. This is reference model is primarily adopted in orchestration tools such as Kubernetes in the creation of guidelines of self-adaptive software systems [7].

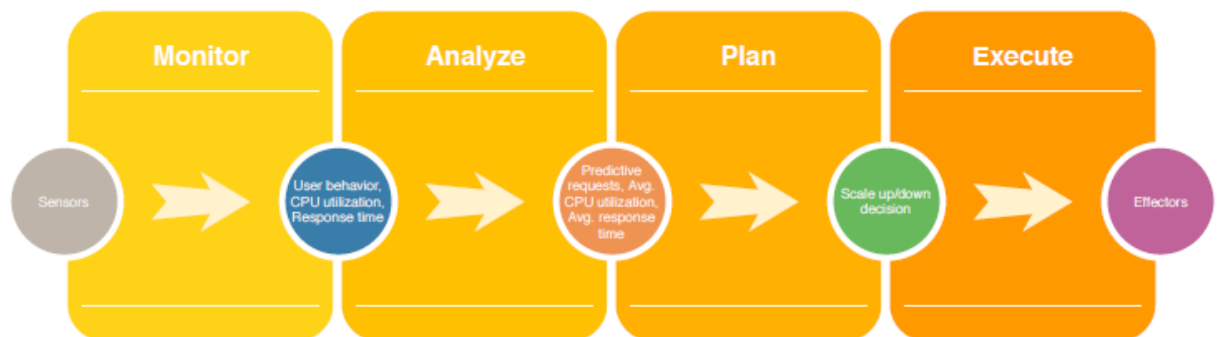


Figure 1.1 MAPE Control Loop [5]

However, there are some issues prevalent in the adaptation of this model along with the rule-based autoscaling policies adopted. These include issues ranging from the lack of adaptability to dynamic workloads faced, which result in under or over-provisioning of resources and loss in the QoS, to issues such as the response delay faced in resource creation. As a result, throughout the years, various research publications have proposed numerous approaches ranging from proactive autoscaling through resource prediction to other approaches such as performance modeling, in order to combat these prevalent issues.

Therefore, before moving on with the implementation of this research, it is vital a thorough analysis is conducted on some of the existing research carried out in order to solve some of the current issues found these local and rule-based (reactive)

autoscaling policies adopted in Kubernetes as well as other similar autoscaling technologies.

The research conducted by A. Zhao and his team [8], uses a resource prediction model based on Kubernetes container auto-scaling technology and makes use of a combination of ARIMA model as well as Empirical Mode Decomposition (EMD) to predict the resource usage and thereby proactively auto-scale the number of pod replicas. This research was primarily conducted aiming to solve the response delay found on the existing autoscaling strategy in Kubernetes as well as manages to provide a detailed insight into some of the current issues found in the Kubernetes HPA and its implementation of rule-based autoscaling policies. However, even though the use of this combination of the above-mentioned techniques proved to produce more accurate forecasting compared to the traditional ARIMA model used, the use of EMD can be quite an inefficient and time-consuming process, as discussed in publications such as [9].

The research conducted by H. Zhao and his team [10] makes use of a double exponential smoothing algorithm for predicting the resource consumption and thereby determining the predicted pods. However, a key point to note in the research conducted is the fact that the predictive algorithm developed, only makes use of the predicted values in the scaling-up process and does not utilize the prediction algorithm in the scaling down process.

Y. Meng and his team [11] makes use of a data science-based approach to predict resource utilization. This research proposes the use of CRUPA, a resource utilization prediction algorithm based on the ARIMA time series analysis model combined with docker container techniques. This prediction algorithm was quite successful and had a high prediction accuracy with an average forecast error of about 6.5%. However, a key fact noted in this approach used is that this model developed, is primarily based on microservice containers, and does not make use of the existing auto-scaling tools provided by Kubernetes for the scaling process.

Publications such as [12] propose a system architecture for Docker containerized applications with an auto-scaler based on a machine learning model. However, here

too, the proposed auto-scaler is primarily based on microservice containers and does not integrate with existing Kubernetes autoscaling tools similar to [11]. Other researches based on proactive containerized auto-scaling techniques similar to the previously mentioned researches stated include researches such as [13] and [14].

However, there have also been some other approaches proposed for improved autoscaling policies rather than the proactive solution-based approaches mentioned above, in order to solve the issues of rule-based autoscaling policies. Key researches include the dynamic multi-level autoscaling method proposed by the publication [15] which makes use of infrastructure as well as application-level monitoring data to dynamically specify thresholds for autoscaling. Other researches in this category include the heterogeneity-aware auto-scaling strategy [16] and the performance modeling-based approach for the rule-based auto-scaling technique proposed by the authors of [6].

Some other key related researches in this regard includes the framework for autoscaling proposed by Al-Sharif [17] used in the provisioning of sufficient VMs to address the changing resource requirements in cloud environments, the master-slave autoscaling architecture for containerized applications proposed by Kukade and Kale [18], the container-based elastic cloud platform named Do Cloud proposed by Kan [19] as well as the Platform Insights framework proposed in [20] which makes use of a combination of proactive and reactive models in order to scale resources.

From the above-mentioned researches, it is clear that even though the solutions for improved autoscaling strategies proposed by these researches were quite effective, they were only focused on providing effective solutions for the localized rule-based autoscaling policies which were primarily based on infrastructure level metrics such as CPU utilization.

## 1.2 Research Gap

In the identification of the research gap, the research publication [21] can be stated one the main publications that manage to provide insight into some of the key issues present in existing rule-based autoscaling strategies (particularly with respect to Kubernetes), as well as the some of the influential factors that should be considered in the development of an optimal auto-scaling strategy for Kubernetes based deployments. This publication is therefore quite influential in the identification of the research gap this research aims at fulfilling.

According to this research publication, although existing rule-based auto-scaling methods may be suitable for cloud-based applications they may result in undesirable Quality of Service (QoS) or poor resource utilization with certain dynamic workloads. This publication then goes on to highlight the fact that ensuring a favorable performance in microservice-based applications governed by the existing reactive auto-scaling rules specifically is currently a challenging issue.

Publications such as [4] also clearly state some of the key challenges that need to be addressed in current autoscaling services. According to this publication, there is a lack of autoscaling studies focusing on the service-level of autoscaling and the use of service level metrics (e.g.: - transactions per unit time). Additionally, issues such as the lack of monitoring tools and aggregating metrics at the platform level and service level to support autoscaling decisions are also clearly described in this regard.

Furthermore, as described in the previous section, although many research publications provide various solutions in which to minimize issues present in rule-based autoscaling policies through the use of various measures, a majority of the proposed researches make use of a limited amount of infrastructure-level metrics (primarily metrics such as CPU utilization) in order to define autoscaling policies. This, in turn, may result in ineffective autoscaling since other service-level metrics that can help in the realization of a more holistic view regarding the importance of a particular microservice in a deployment are not utilized in the creation of the autoscaling policy.



Through analysis of these research publications, the apparent gap in research related to the creation and development of auto-scaling policies that make use of the global importance of a particular microservice along with the use of service-level metrics is realized.

This research thereby aims to fulfill this research gap with the inclusion of both service-level metrics and infrastructure-level metrics derived from monitoring solutions, for a load prediction based approach along with centrality based evaluation on the predicted loads, which will ultimately aid in the identification of key microservices in a particular deployment and thereby aid in the creation of a more globally aware autoscaling policy.

### 1.3 Research Problem

The auto-scalers of container orchestration tools such as Kubernetes follow a broadly accepted reference model named MAPE (Monitor, Analyze, Plan, and Execute) used in various autonomic computing systems [3,5,21]. This MAPE model uses monitoring-based mechanisms to analyze relevant thresholds and thereby scale resources.

However, this approach is rule-based and, at times, due to the dynamic nature of workload that microservices have to experience, the current rule-based auto-scaling mechanisms offered are unable to adapt to these workload intensities. This prevalent issue results in over-provisioning or under-provisioning of allocated resources and ultimately results in a lower Quality of Service (QoS) experienced by users.

Issues such the response delay caused during resource creation and initialization [8,22] and the difficulties posed in configuring rule-based auto-scaling due to lack of knowledge and expertise as well as the vast configuration space involved during this process which in turn make the selection of optimal parameters and variables virtually impossible [6], further complicates the creation of effective rule-based auto-scaling policies.

In addition, due to the dynamic nature of the workload faced by microservices, it becomes necessary to make use of a wide variety of metrics in order to maintain the required SLAs [4]. Furthermore, this process is quite inefficient and usually results in ineffective resource provisioning due to the fact that most auto scalers make use of a limited amount of infrastructure-level metrics such as CPU utilization to infer autoscaling decisions. This lack of inclusion of other service-level resource metrics prevents and hinders effective autoscaling decision making and does not provide the necessary insight in order to visualize and understand how each microservice is utilized. Evidence regarding the importance of using higher-level metrics in the creation of autoscaling policies can be seen from reading research publications such as [15].

## **2.0 OBJECTIVES**

### **2.1 Main Objective**

The main objective of this research is to develop an improved auto-scaling policy for a deployment, based on load prediction.

### **2.2 Specific Objectives**

The specific objectives for the research project are as follows.

- To develop an algorithm to predict the load of a microservice and determine centrality based on the predicted loads.
- To develop the proposed auto-scaling policy without considering factors such as the location of instance creation, based on the global importance of a particular microservice, measured using the centrality measures (such as degree, betweenness, etc.) applied to the derived co-dependency network.
- To evaluate the effect of centrality prediction-based mechanisms in the creation of an optimal deployment strategy for Kubernetes deployments.

## **3.0 METHODOLOGY**

### **3.1 Requirement Gathering**

Requirement gathering was primarily performed through the analysis of published research papers as a variety of online sources. A key focus was given on the identification of existing or similar systems developed and the methodology used.

#### **3.1.1 Past Research Analysis**

Past research analysis was primarily performed through reading research publications mainly focused on key areas such as resource utilization prediction, short term load prediction, time series analysis, proactive auto-scaling, cloud elasticity, centrality evaluation, and machine learning models.

The primary focus was given in the identification of the methodology used, tools used, experiments conducted, as well as the overall findings of the research with respect to load forecasting and resource utilization prediction.

#### **3.1.2 Identifying Existing Systems**

Existing systems were primarily identified through referring research publications as well as referring a variety of online sources. A key focus was given in the identification of the existing feature they offered as well as the potential drawbacks in the technology and methodology used.

## **3.2 Feasibility Study**

### **3.2.1 Technical Feasibility**

Technical feasibility was a key factor considered in the requirements analysis phase of this research project since this project mainly focused on the development of a load prediction algorithm using machine learning technology. A key focus was given in the identification of potential system requirements as well as the required tools and technologies that may be used in the development of the proposed prediction algorithm.

### **3.2.2 Schedule Feasibility**

The schedule feasibility was also a key factor considered throughout this research. A key focus was given in the identification of possible time periods and duration to develop and implement the proposed load prediction algorithm using the possible development tools and technology within an implementation period of about five months.

### **3.2.3 Economic Feasibility**

The economic feasibility was another key factor considered before the development of this prediction algorithm. A key focus was given in the identification of the possible costs that might be incurred in the development process, as well as the costs that may be incurred in the use of the planned development tools and technologies.

### **3.3 Requirement Analysis**

The requirement analysis phase was one of the keys phases in this research project since it enabled in the identification of a variety of factors that should be considered in the implementation process of this research.

During this process, the information gathered from the various sources during the requirement gathering phase was analyzed. As a result, the key factors related to the possible challenges that may be encountered as well as insight into the methodology and also a clear understanding of the use of possible and tools and technology were also able to be easily identified.

Furthermore, a clear idea of the scope of the proposed research, as well as the feasibility of the project, was also able to be identified during this phase.

Requirement analysis also helped in the determination of the existing research gaps as well as provide insight into the identification underlying research problem as the research.

### 3.4 System Analysis

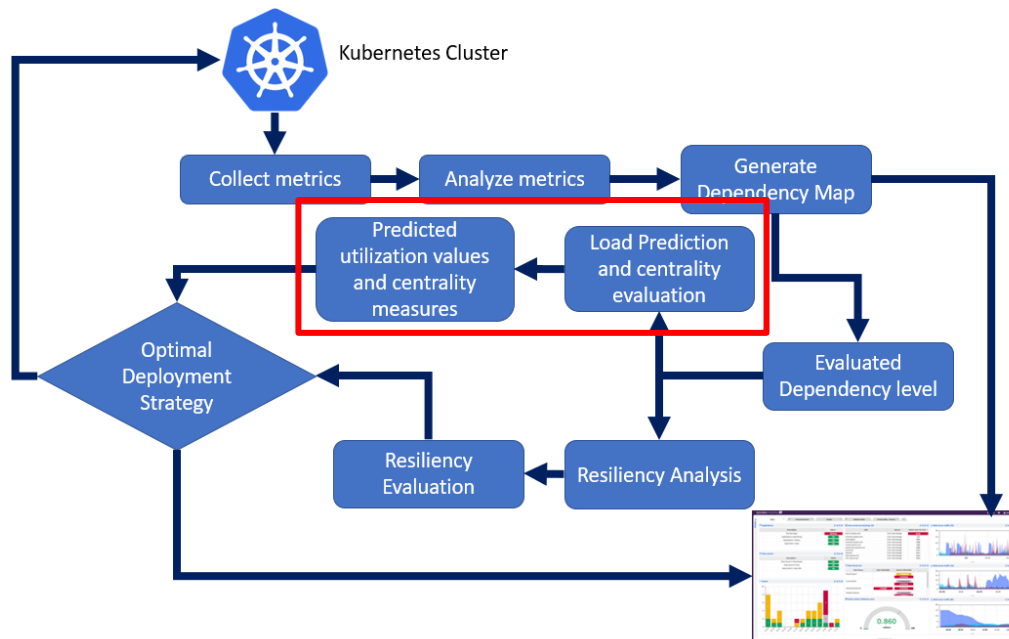


Figure 3.1: Load Prediction Component in the Proposed Governance Model

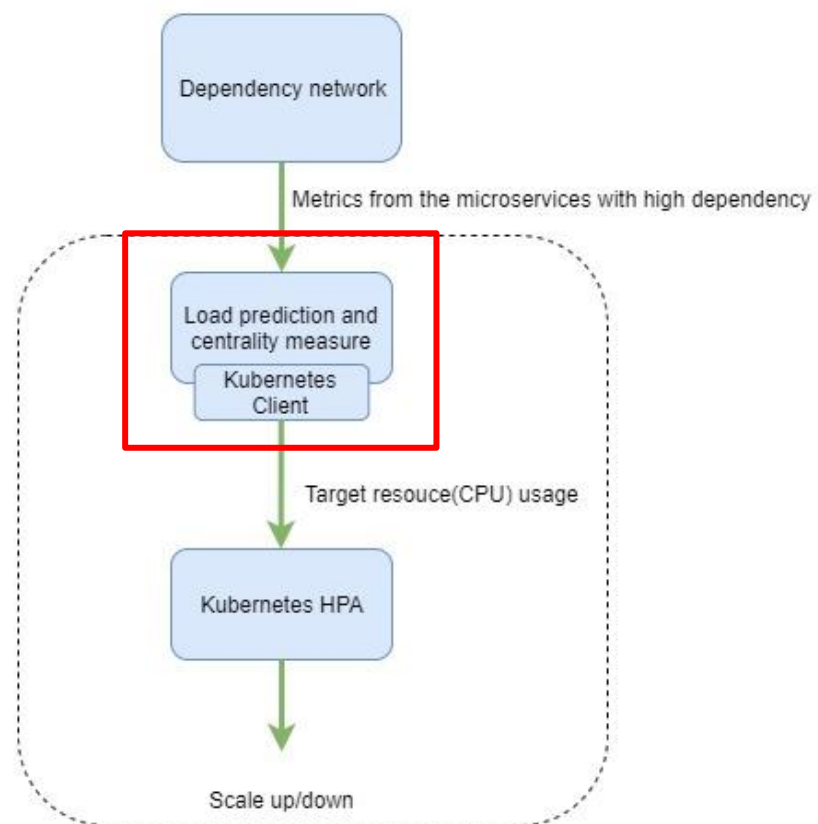


Figure 3.2: System Overview

The load prediction component will be integrated into the governance model proposed in the main research component, as given in Figure 3.1 above. Figure 3.2 describes a higher level, in-depth view of the proposed Load Prediction Component.

The load prediction component will be responsible for the retrieval of the metrics from the co-dependency network, prediction of future load based on metrics and identification and evaluation of the centrality measures of a microservice, based on the predicted loads. This component will make use of statistical and machine learning-based time series prediction measures in order to perform the above process.

In addition, the centrality of a microservice is planned to be evaluated based on the following types of centrality measures on the predicted loads.

- Degree centrality
- Closeness centrality
- Betweenness centrality
- Eigenvector centrality

Through evaluation of the above-mentioned centrality measures key microservices, the level of interdependency among microservices can be further analyzed. These key microservices identified using the evaluated centrality measures can be then used as inputs for the algorithm that determines the optimal deployment strategy for the given cluster. This component will thereby enable identification of key microservices within a cluster and aid in producing an improved autoscaling policy based on the global importance of a particular microservice.



### 3.5 System Development and Implementation

The steps in the development of the proposed load prediction component are as follows.

1. Obtain the list microservices that show a high level of dependency based on the proposed co-dependency network.
2. Identify the relevant pods in which the identified microservices are deployed.
3. Receive the resource usage metrics of the identified microservices.
4. Convert the metrics obtained into a time series.
5. Plot time series to identify the relevant time series components such as trend, seasonality, etc.
6. Convert time series to a stationary dataset.
7. Develop an algorithm using the prediction model to forecast (train dataset and forecast) the load on the obtained resource metrics.
8. Evaluate the centrality (degree, closeness, etc.) based on the predicted loads.
9. Forecasted utilization values for the identified microservices will be forwarded to the optimization algorithm and used in configuring the auto-scaling of the cluster.

The proposed implementation process will primarily make use of a data-science based approach in the development load prediction component. After the key microservices are identified with the help of the developed co-dependency network, the obtained resource usage and load metrics from the co-dependency network will be recorded in a temporary database within the cluster and converted into a time series model in order to perform the prediction process. Steps must also be taken to ensure the obtained data is properly cleaned so as to eliminate any irregularities in the collected data, and the relevant time series components could be easily identified. Here, the Python programming language, along with other open-source libraries and tools such as Jupyter Notebook and Anaconda, are expected to be used in performing the above-mentioned process. In order to ensure maximum accuracy is achieved in the prediction process, a variety of time series prediction models are also expected to

be evaluated, and the most suitable prediction model will be selected. Here too, machine learning libraries such as TensorFlow are expected to be used in the development of the prediction algorithm. Lastly, in order to evaluate the relevant centrality measures based on the predicted load, libraries such as NetworkX are expected to be used. In summary, a list of all primary tools, libraries, and programming languages that are expected to be used in this research is given in the table below.

Tools	<ul style="list-style-type: none"> <li>• Anaconda</li> <li>• Jupyter Notebook</li> </ul>
Programming Languages	<ul style="list-style-type: none"> <li>• Python</li> </ul>
Python libraries	<ul style="list-style-type: none"> <li>• Numpy</li> <li>• Scikit-learn</li> <li>• TensorFlow</li> <li>• Pandas</li> <li>• Matplotlib</li> <li>• NetworkX</li> </ul>

Table 3.1: Tools and Technology

### **3.6 Project Requirements**

#### **3.6.1 Functional requirements**

The primary functional requirements aimed at fulfilling during the implementation process in this research are as follows.

- Dependency metrics for prediction should be obtained from the co-dependency network developed
- Predictions should be made at least 24hrs ahead
- The algorithm should predict highly dependent microservices based on retrieved metrics identified using centrality measures
- Prediction should be based on a time series prediction model
- The auto scaler should be configured to scale based on the obtained predictions

#### **3.6.2 Non-Functional Requirements**

The following are the non-functional requirements that are primarily being focused on during this research.

- Availability – The proposed system should be able to function throughout the day without any restrictions.
- Efficiency – The proposed system should be as efficient as possible and make use of minimal resources in the prediction process such that it does not affect the performance of the overall cluster.

- Performance – The proposed system should be able to handle the vast amount of data it receives and be able to process the data without affecting the system performance.
- Interoperability – The proposed system should be able to interact and communicate with the other components proposed in this research and receive inputs as well as forward outputs to the desired components.

### **3.7 Testing**

The load prediction component will primarily be tested to ensure the forecasted time series model accurately captures key time series components and that the optimal set of parameters is given to the prediction model such that the forecasting errors are minimal as possible. Testing will be used to ensure the most suitable prediction model will be used in performing the prediction process in order to achieve the highest prediction accuracy, and the relevant centrality measures are accurately identified based on the predicted loads. Furthermore, since testing will be evaluated in a sample Kubernetes cluster, the auto-scaling process will also be evaluated to ensure that the autoscaling tool (HPA) is correctly configured to scale the number of pods based on the load prediction algorithm and the evaluated centrality measures. The metrics used for testing the load prediction component will be obtained from the load generated from the load generation tool used in the creation of the co-dependency network.

## 4.0 PERSONEL AND FACILITIES

Name	Key Tasks
De Silva N.	<ul style="list-style-type: none"><li>• Retrieval of the necessary metric data from pods in the cluster in order to perform the load prediction based on the developed dependency map.</li><li>• Development of solution in order to store the gathered metric data for time series analysis.</li><li>• Preparing and manipulating the extracted data in order to establish a time series. (data cleaning)</li><li>• Development of a time series data set based on the historical metric data retrieved, such that future loads could be predicted.</li><li>• Selection of the most appropriate time series forecasting model for load prediction.</li></ul>

Table 4.1: Personnel and Resources

	<ul style="list-style-type: none"> <li>• Development of the algorithm with the help of Machine Learning, for the time series using the selected model in order to predict future loads and determine centrality.</li> <li>• Finetune the developed model in order to get the most accurate prediction.</li> </ul>
--	---

## 5.0 COMMERCIALIZATION

This research is primarily designed as a supplementary tool for the main research component. However, this autoscaling model could also be developed as a standalone APM tool since it could be used to provide users a variety of benefits not seen in most autoscaling tools developed today. These benefits include the ability to:

- Integrate with monitoring solutions in order to retrieve load-based metrics and predict future loads based on the received metrics.
- Evaluate and provide solutions to visualize a vast variety of centrality measures based on the predicted loads.
- Automatically configure and auto-scale Kubernetes autoscaling tools based on inputs derived from the predicted load and centrality measures.

This tool will be primarily targeted for system administrators and cloud engineers who manage and maintain cloud-based microservice deployments. Due to the heavy competition in the APM market space which primarily includes a vast variety of opensource APM tools, the initial marketing strategy will be to adopt an opensource marketing strategy in order to easily enter the marketplace and capture users.

However, since this autoscaling tool is quite a unique tool that offers some useful benefits as mentioned above, adopting a freemium marketing strategy for this tool for users for additional supplementary features will be the most appropriate approach once this tool becomes more popular in the future.



## 6.0 BUDGET

The proposed prediction component will be primarily be developed using open source tools. Hence, based on this fact, as well as the information gathered during the requirements gathering phase, there is no cost expected to be incurred in the development process of this project.

However, some additional external costs are expected to be incurred in conducting this research as stated in table 5.1 given below

Internet use	1500 LKR
Publication costs	500 LKR
Stationary	500 LKR
TOTAL	2500 LKR

Table 5.1: Budget

**Note:** Azure Student subscription with \$100 of free credit for 12 months will be used for this project. Therefore, resource creation for the VMs needed for testing and creation of the Kubernetes cluster to be used for implementation purposes could be obtained without any additional cost and hence is not included in the budget.

## **7.0 SUMMARY**

The current implementation of MAPE based autoscaling techniques used in autoscaling tools in microservice orchestration tools such as Kubernetes are primarily rule-based and hence it becomes difficult to adapt to the dynamic workloads microservices experience resulting in ineffective scaling and a drop in QoS.

Furthermore, in order to effectively develop an autoscaling policy that optimally makes use of available resources and, in turn, minimize the effect of the above-mentioned problems, it becomes necessary to take into consideration a global view of each microservice and how it is being utilized.

This research, as described above, thereby aims to provide a solution to this problem through a statistical and machine learning-based approach to effectively predict load through metric analysis while incorporating centrality-based evaluation techniques to determine the centrality of a microservice so as to obtain a more holistic view of the utilization of each microservice and ultimately develop a more effective autoscaling policy.

## 8.0 REFERENCES

- [1] “What is Kubernetes,” Kubernetes. [Online]. Available: <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>. [Accessed: 13-Feb-2020].
- [2] AWS Auto Scaling", *Amazon Web Services, Inc.*, 2020. [Online]. Available: <https://aws.amazon.com/autoscaling/>. [Accessed: 20- Feb- 2020].
- [3] C. Qu, R. N. Calheiros, and R. Buyya, “Auto-Scaling Web Applications in Clouds,” *ACM Computing Surveys*, vol. 51, no. 4, pp. 1–33, 2018.
- [4] H. Alipour, A. Hamou-Lhajd and X. Liu, “Analyzing Auto-scaling Issues in Cloud Environments”, 2014
- [5] P. Singh, P. Gupta, K. Jyoti, and A. Nayyar, “Research on Auto-Scaling of Web Applications in Cloud: Survey, Trends and Future Directions,” *Scalable Computing: Practice and Experience*, vol. 20, no. 2, pp. 399–432, Feb. 2019.
- [6] A. Evangelidis, D. Parker, and R. Bahsoon, “Performance Modelling and Verification of Cloud-Based Auto-Scaling Policies,” *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2017.
- [7] S. Taherizadeh and M. Grobelnik, “Key influencing factors of the Kubernetes auto-scaler for computing-intensive microservice-native cloud-based applications,” *Advances in Engineering Software*, vol. 140, p. 102734, 2020.
- [8] Zhao, A., Huang, Q., Huang, Y., Zou, L., Chen, Z., & Song, J. “Research on Resource Prediction Model Based on Kubernetes Container Auto-scaling Technology.” IOP Conference Series: Materials Science and Engineering, 2019.
- [9] Yamin Wang, L. Wu and Shouxian Wang, "Challenges in applying the empirical mode decomposition based hybrid algorithm for forecasting renewable wind/solar in practical cases," 2016 IEEE Power and Energy Society General Meeting (PESGM), Boston, MA, 2016, pp. 1-5

- [10] H. Zhao, H. Lim, M. Hanif and C. Lee, "Predictive Container Auto-Scaling for Cloud-Native Applications," *2019 International Conference on Information and Communication Technology Convergence (ICTC)*, Jeju Island, Korea (South), 2019, pp. 1280-1282.
- [11] Y. Meng, R. Rao, X. Zhang and P. Hong, "CRUPA: A container resource utilization prediction algorithm for auto-scaling based on time series analysis," *2016 International Conference on Progress in Informatics and Computing (PIC)*, Shanghai, 2016, pp. 468-472
- [12] M. Imdoukh, I. Ahmad, and M. G. Alfaiakawi, "Machine learning-based auto-scaling for containerized applications," *Neural Computing and Applications*, Aug. 2019.
- [13] W.-Y. Kim, J.-S. Lee, and E.-N. Huh, "Study on proactive auto scaling for instance through the prediction of network traffic on the container environment," *Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication - IMCOM 17*, 2017.
- [14] T. Ye, X. Guangtao, Q. Shiyong, and L. Minglu, "An Auto-Scaling Framework for Containerized Elastic Applications," *2017 3rd International Conference on Big Data Computing and Communications (BIGCOM)*, 2017.
- [15] S. Taherizadeh and V. Stankovski, "Dynamic Multi-level Auto-scaling Rules for Containerized Applications," *The Computer Journal*, vol. 62, no. 2, pp. 174–197, Aug. 2018.
- [16] J. Sahni and D. P. Vidyarthi, "Heterogeneity-aware adaptive auto-scaling heuristic for improved QoS and resource usage in cloud environments," *Computing*, vol. 99, no. 4, pp. 351–381, 2016.
- [17] Z. A. Al-Sharif, Y. Jararweh, A. Al-Dahoud, and L. M. Alawneh, "ACCRS: autonomic based cloud computing resource scaling," *Cluster Computing*, vol. 20, no. 3, pp. 2479–2488, 2016.
- [18] Kukade P.P and Kale G, "Auto-scaling of micro-services using containerization", *Int J SciRes(IJSR)* 2015;4(9):1960–3.

- [19] C. Kan, "DoCloud: An elastic cloud platform for Web applications based on Docker," 2016 18th International Conference on Advanced Communication Technology (ICACT), Pyeongchang, 2016, pp. 478-483.
- [20] L.R. Moore, K. Bean and T. Ellahi, "A Coordinated Reactive and Predictive Approach to Cloud Elasticity." *CLOUD 2013*, 2013
- [21] S. Taherizadeh and M. Grobelnik, "Key influencing factors of the Kubernetes auto-scaler for computing-intensive microservice-native cloud-based applications," *Advances in Engineering Software*, vol. 140, p. 102734, 2020.
- [22] W.-Y. Kim, J.-S. Lee, and E.-N. Huh, "Study on proactive auto scaling for instance through the prediction of network traffic on the container environment," *Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication - IMCOM 17*, 2017.

